会誌「情報処理」Vol.62 No.8（Aug. 2021）「デジタルプラクティスコーナー」

特集号招待論文

マルチコンテナオーケストレーションを用いた大規模コンテナ環境の設計と運用

坂下幸徳[1]

[1]ヤフー（株）

ショッピングサイトや地図サービスなどのWebサービスでは，利用者からのアクセス増減が激しく，バージョンアップによる機能追加やバグ修正も高頻度に行われる．このような特徴を持つWebサービスでは，機能単位に細分化しアプリケーションを分割するアーキテクチャのマイクロサービスが浸透している．これにより，Webサービス全体を変更せず特定のアプリケーションのリソースの増強やバージョンアップを行うことができる．この細分化されたアプリケーションの実行環境としてCPUやメモリなどのリソースの集約率の高さや増強のしやすさからコンテナが注目されている．しかし，マイクロサービスでは，機能単位に細分化されるため，コンテナ数が増加する傾向がある．この増加するコンテナを管理するためコンテナオーケストレーションKubernetesが普及している．Kubernetesは，コンテナのリソースを容易に増減させるスケール機能，障害のセルフヒーリング，Webサービスを無停止でバージョンアップするローリングアップデートなどを有している．このKubernetesの導入により，コンテナの管理負荷の軽減は見込める．しかし，複数Kubernetesを有する大規模コンテナ環境では，Kubernetes自身の管理やこれを構成するコンピュート・ネットワーク・ストレージのインフラ管理が複雑になり管理者の負荷が増加してしまう．本稿では，大規模コンテナ環境としてヤフー（株）にて運用しているKubernetes as a Serviceについて，アーキテクチャと運用実績から得られた知見を報告する．このKubernetes as a Serviceは，2020年12月時点で204,980個以上のコンテナ，860クラスタ以上のKubernetesが稼働している．このような大規模コンテナ環境により，コンテナを使わずVMのみでアプリケーションの実行環境を構築した場合と比較し，コンピュート・ネットワークリソースは約86.4%の集約効果があった．さらに，Kubernetesを使い複数Kubernetesを自律管理する方式にて開発し管理者間のコミュニケーションでのリソース調整を削減するインフラ構成とすることで，管理者の負荷の軽減効果があった．

1. 背景

インターネットを通じ利用されるショッピングサイトや地図サービスなどのWebサービスは，PC・スマートフォン・タブレットなど多くの情報端末からアクセスされ，社会を支える情報基盤の1つとなっている．このようなWebサービスでは，利用者からのアクセス数の増減が激しい．さらには，いかに短期間にアップデートを行い新機能追加やバグ修正を行うかが先進性と安定性を保つために重要である．このような特性を持つWebサービスの作りも，巨大な1つのサービスとして構成されるモノリシックな作りから，細かく独立したサービスに分割し，それらを結合しWebサービスを構築するマイクロサービス[1]へと変化している．このマイクロサービスの

特徴の1つである独立デプロイ可能性[2]により，他のサービスに影響を及ぼさずにアップデートを実施できるだけでなく，急なアクセス数の増減に対応すべく，特定のサービスのみスケールさせることが可能となる．以降，本稿では，細かく独立したサービスをアプリケーションとして示し，複数アプリケーションを結合したサービスをWebサービスとして記載する．

このようなWebサービスを構成するアプリケーションをスケールさせるためには，アプリケーションを動作させるコンピュート・ネットワーク・ストレージのインフラストラクチャ（インフラと略す）のリソースを柔軟かつ迅速に割り当てる必要がある．このインフラリソースの割り当てを柔軟にする技術として代表的なものに仮想サーバ（以下，VMと略す）とコンテナ[3]がある．これらを使い，アプリケーションごとに独立したVMやコンテナを作成することで，特定のアプリケーションの性能を増減させることが容易となる．VMとコンテナについて作成からアプリケーション起動の違いを述べる．VMはCPUやメモリを仮想化したマシンを作成後，OSセットアップやアプリケーションをインストールし実行する．これに対し，コンテナはホストOSのCPUやメモリをコンテナごとに割り当てた後，コンテナイメージをロードし実行するため，アプリケーションの起動までの時間が短くできる．このようなコンテナの特徴により，Webサービスでのコンテナ利用が普及している．

一方で，マイクロサービスにより細分化されたアプリケーションのコンテナ化が進むと，運用管理しなければならないコンテナ数が増加する．そこで，コンテナの運用管理を支援するために，コンテナオーケストレーションがある．代表的なコンテナオーケストレーションとして，Googleが開発・運用していたBorg[4]をベースに開発され2014年にオープンソースとして公開したKubernetesがある．

2020年時点で100以上のWebサービスを提供するヤフー（株）（以下，ヤフーと略す）では，多数のコンテナ化されたアプリケーションを運用管理するためにKubernetesをプライベートクラウド向けにサービス化しKubernetes as a Service（以下，KaaSと略す）として2018年8月より利用している．このKaaSでは，各Webサービスを開発・運用する部門ごとに独立したKubernetesを提供している．KaaSは，2020年12月時点で約 860クラスタのKubernetesが運用され，その上で稼働中のコンテナ数は約204,980個の規模へと成長した．

本稿では，ヤフーで運用するKaaSについて，設計と運用実績から得た知見について述べる．ヤフーのKaaSでは，年々増加するコンテナおよびKubernetesの数に負けないように管理作業を省力化すべく，Kubernetesを用いてKubernetesを管理する方式を開発し運用している．この方式のKaaSを2年4カ月運用している実績から，インフラリソースの集約効果と管理者の負荷軽減効果について述べる．

以下，第2章ではコンテナオーケストレーションと関連研究を紹介し，第3章にてヤフーのプライベートクラウドと管理者体制を述べ，第4章でヤフーにて運用しているKaaSについて設計を述べる．次に，第5章にて運用実績を示し，第6章にて考察を述べ，最後に第7章にてまとめる．

2. コンテナオーケストレーションと関連研究

本章では，コンテナとコンテナの運用管理を支援するコンテナオーケストレーションおよびKubernetesに関する自律管理，マルチテナント・マルチクラスタ構成，KaaSについて関連する技術を整理する．

2.1 コンテナと性能

コンテナは，ホストOS上の実行基盤であるコンテナランタイム上で実行され，ホストOSのカーネルを共有しつつアプリケーションのCPU・メモリ・ストレージを隔離する技術である．また，コンテナでは，実行するアプリケーションのバイナリファイルだけでなく，実行に必要なライブラリなどの実行環境をコンテナイメージとしてまとめ，これをコンテナランタイム上にロードし実行する．これによりアプリケーションの実行環境のセットアップの手間を削減し，短時間でのアプリケーションのセットアップと実行を実現する．コンテナランタイムの代表的なものにDocker[5]・containerd[6]・CRI-O[7]がある．W. Felterらの研究[8]では，ベアメタルサーバ（以下，ベアメタルと略す），Linuxの仮想化技術のひとつであるKernel Virtual Machine（以下，KVMと略す）[9]，コンテナランタイムのDockerの各環境にてベンチマークツールを使いCPU（Compression），メモリ（Stream），ネットワーク（TCP round-trip latency），ストレージ（Random read・write IOPS）を測定した結果が報告されている．この報告によると，CPU性能は，ベアメタルと比べDockerはわずか4%ほど遅く，KVMは22%遅い結果であった．メモリ性能は，ベアメタルと比べDockerでは性能差はなく，KVMはわずか3%遅い結果であった．ネットワークは，Dockerですべてのコンテナがホスト OS の仮想NICにブリッジ接続されNAT（Network Address Translation）のような接続となるためオーバヘッドがありベアメタルより2倍ほど遅く，KVMはベアメタルより1.8倍ほど遅い結果であった．ストレージ性能は，ベアメタルと比べてもDockerは性能差はなく，KVMは1/2ほど低い性能であった．つまり，Dockerは，ベアメタルと比べた場合，ネットワークで性能が劣るもののCPU・メモリ・ストレージに関しては性能劣化はごくわずかである．DockerとKVMと比べた場合，ネットワークでわずかに性能劣化があるものの，CPU・メモリ・ストレージに関しては，いずれも性能が良い結果である．

2.2 コンテナオーケストレーション

コンテナの運用管理を支援するソフトウェアとしてコンテナオーケストレーションがある．コンテナオーケストレーションは，複数台のコンテナランタイムをクラスタリングし管理するソフトウェアである．コンテナオーケストレーションの代表的なものにKubernetes・Docker Swarm[10]・Mesos[11]がある．CNCF（Cloud Native Computing Foundation）が実施したユーザサーベイの結果[12]によると各コンテナオーケストレーションの利用率は，2019年時点でKubernetesが83%，Docker Swarmが21%，Mesosが9%であり，Kubernetesを利用する組織が多い．また，Truyenらの報告[13]によるとコンテナオーケストレーションのうち，KubernetesとDocker Swarmが約3カ月ごとのペースと高頻度にバージョンアップされ機能追加やバグ修正が行われ活発化している．さらに，Kubernetesでは，3マイナーバージョンしかコミュニティではサポートされない．このため，バージョンアップを高頻度に行う必要があり，管理者の負荷は高い．

2.3 Kubernetesにおける自律管理

代表的なコンテナオーケストレーションの1つであるKubernetesに着目する．Kubernetesは，複数のコンテナランタイムをクラスタとしてまとめて管理する．さらに，Kubernetesでは，コンテナをPodという単位で管理する．Pod管理としてスケールアウト・スケールイン・スケールアップ・スケールダウン・セルフヒーリング・ローリングアップデートなどの特徴機能を有する．これにより，Pod数や利用するCPUやメモリなどのリソースを増減させることでPodにより構成されるWebサービスの性能増減，障害発生時の自動復旧，Webサービス無停止でのバ

ージョンアップといった自律管理が可能となる．自律管理は，Jefferey らの研究[14]により自律コンピューティングの概念が提唱され，G. Lanfranchi らの研究[15]により自律コンピューティングの管理方法として提案されている．この自律管理では，機器が正常に運用し続けるように常に状態を監視し，正常時の状態と異なる場合が検知された場合には，正常な状態に復旧するように自動で変更を行うことで自律化された管理を実現している．このような自律管理を Kubernetes では Reconciliation ループ[16]にて実現している．Reconciliation ループは Kubernetes が宣言的管理[17]を行うために利用されるフレームワークの中核となる処理であり，利用者が定義した望ましい状態と実際に稼働している状態の差分を定期的にチェックし，差分があれば望ましい状態になるよう調整処理を繰り返し実行することで，自律管理を実現している．以降，本稿では Pod とコンテナを明確に区別する必要がある場合を除き Pod をコンテナとして記載する．

リソースの増減では，利用者が定義したコンテナ数・CPU 数・メモリサイズとなるようにコンテナの作成・削除を行う．セルフヒーリングでは，障害などにより稼働していたコンテナが停止した際，利用者が定義した数のコンテナ数が稼働していない状態を検知すると，コンテナの再作成を行うことで障害回復を試みる．ローリングアップデートでは，利用者が定義したコンテナイメージのバージョンと稼働中のコンテナイメージのバージョンが異なることを検知すると，定義したコンテナイメージのバージョンにてコンテナを新たに稼働させ，古いバージョンのコンテナを削除する．これを繰り返し行うことで，Web サービスを停止させることなくバージョンアップを行う．さらに，Kubernetes は，コンピュート・ネットワーク・ストレージの各リソースを抽象化したモデルにより管理し，異なるベンダの機器の差異を吸収している．つまり，Kubernetes の管理者や利用者は，ベンダごとに異なる管理方法を習得する必要がなくコンピュート・ネットワーク・ストレージを扱える．

2.4　Kubernetes のマルチテナント・マルチクラスタ構成

複数の利用者が Kubernetes を利用する場合の構成について述べる．複数利用者が Kubernetes を利用するための構成としてマルチテナントとマルチクラスタの 2 つの構成がある．**図1**の左図に Kubernetes におけるマルチテナント構成，右図にマルチクラスタ構成について示す．

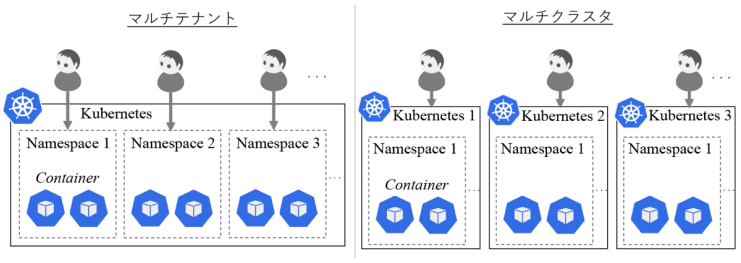


図1　Kubernetes におけるマルチテナントとマルチクラスタ

Kubernetes では Namespace というコンテナを実行する空間を論理的に分ける機能がある．マルチテナント構成では，1 つの Kubernetes 上に複数の Namespace を作り，各利用者にそれぞれ割り当てる構成である．マルチクラスタでは，利用者ごとに Kubernetes をそれぞれ割り当て

る構成である．Truyenらの報告[18]によると，マルチテナントは，コスト面・アプリケーションレベルでの独立性・コンテナイメージの共有などが良い点として挙げられ，セキュリティ面・管理が複雑化する点が悪い点として挙げられている．良い点として挙げられているコスト面に関しては，利用者がデプロイするアプリケーションのコンテナとは別にKubernetes自身および動作するために必要なコンポーネントであるkube-apiserver・kube-scheduler・kube-controller-manager・etcd[19]を，複数Namespaceにて共有することで，マルチクラスタ構成に比べコストを抑えられる．また，Kubernetesは，Namespaceごとにアプリケーションのコンテナが分離され別の利用者からのアクセスを禁止できるRBAC（Role Based Access Control）を備えているため，アプリケーションレベルでの独立性を確保できる．さらに，コンテナを実行する際，コンテナイメージをコンテナランタイムへダウンロードする必要があるが，複数Namespaceでコンテナランタイムを共有するため，あるNamespaceで一度コンテナイメージをダウンロードしていれば，他Namespaceで同じコンテナイメージを利用する際，再度ダウンロードする必要がなく，コンテナの起動が高速になる．一方，悪い点として挙げられているセキュリティ面に関しては，コンテナランタイムおよびこれを動作させるホストOSが複数Namespaceで共有となるため，悪意を持ったコンテナやバグなどによりホストOSのroot権限のような強力な権限が乗っ取られた場合，その影響範囲が広い点が挙げられている．管理面としては，すべてのNamespaceで共有のKubernetesとなるため，Kubernetesのバージョンが全Namespace内のコンテナが期待するバージョンに統一しないといけないという制約が生じる．また，複数NamespaceでホストOSが共通のため，性能面での独立性を保障するのが難しく，アプリケーションの構成が複雑化するとのことである．これに対しマルチクラスタ構成は，良い点としてはセキュリティ面・管理の簡素化・アプリケーションの独立性があり，悪い点としてはコスト面・コンテナイメージの非共有がある．アプリケーションの独立については，マルチテナント構成と同様にRBACを利用できるとともに各Kubernetesで別々のアカウントを設定できるため，マルチクラスタ構成であっても変わらない．また，マルチクラスタ構成のKubernetes上にマルチテナントを構成することも可能である．

2.5　Kubernetesの構築方法とKaaS

Kubernetesの構築方法について述べる．Kubernetesを構築する場合，パブリッククラウドが提供するKaaSを利用する方法と，プライベートデータセンタなどに独自に構築する方法の2通りがある．代表的なパブリッククラウドのKaaSとしてGoogle・Amazon・Microsoftが提供するサービスについて述べる．Googleでは，2014年にKubernetesを活用したGoogle Container Engine（GCE）の提供を開始し，2017年にGoogle Kubernetes Engine（GKE）[20]と改名しサービス提供している．GKEの利用者は，Google Cloudが提供するコンソールを用い，Kubernetesを構築し利用する．GKEでは，Google Cloudが提供している監視サービスやストレージサービスなどさまざまなサービスとインテグレーションされ，GKEおよびGKE上で実行されるアプリケーションのコンテナから利用できる．Amazonでは，Amazon Web Service（AWS）の1サービスとして2018年よりAmazon Elastic Kubernetes Service（EKS）[21]を提供している．利用者は，GKEと同様にAWSのコンソールを用いKubernetesを構築し利用する．さらにEKSでは，コンテナを実行するコンピュートリソースとしてVMのAmazon Elastic Compute Cloud（Amazon EC2）[22]とコンテナ向けサーバレスコンピュートAWS Fargate[23]を選択し利用できる．AWS Fargateは，VMとしてあらかじめ決められたCPUやメモリなどのリソースを割り当てるAmazon EC2とは異なり，コンテナの実行中のみCPUやメモリなどのリソースを割り当てるサーバレスアーキテクチャを提供し，Kubernetesの構築におけるコンピュートの設計を簡素化している．また，EKSは，GKEと同様にAWSが提供

している監視サービスやストレージサービスなどさまざまなサービスとインテグレーションされ利用できる．Microsoftでは，Microsoft Azureの1サービスとして2017年よりAzure Kubernetes Service（AKS）[24]を提供している．利用者は他2社のKaaSと同じく，Azureのコンソールを用いKubernetesを構築し利用する．AKSは，コンテナを実行するコンピュートリソースとしてサーバレスコンピュートAzure Container Instance（ACI）[25]を使い，コンテナが実行中のみコンピュートのリソースを割り当てることで，AWS Fargateと同様に，Kubernetesの構築におけるコンピュートの設計を簡素化している．このACIとAWS Fargateは，オープンソースのVirtual Kubeletプロジェクト[26]に基づき設計され，Kubernetes APIから操作できるノードの一形態となっている．また，AKSは他2社のKaaSと同じく，Microsoft Azureが提供している監視サービスやストレージサービスなどさまざまなサービスとインテグレーションされ利用できる．さらに，AKSは2019年よりWindows Serverコンテナの実行もサポートし，ASP.NETなどのWindows固有のライブラリを使ったアプリケーションもコンテナとして実行できる．これらGKE・EKS・AKSはすべてマルチクラスタ構成にて，利用者ごとにKubernetesが作られる．利用者は必要に応じ，提供されたKubernetes上にNamespaceを作りマルチテナントを構築し利用する．次に，プライベートデータセンタなどに独自にKubernetesを構築する方法について述べる．パブリッククラウドのKaaSとは異なり，プライベートデータセンタにて使用するコンピュート・ネットワーク・ストレージを準備し，それらを使いKubernetesを構築する．Kubernetesの構築を容易化するためのツールとして代表的なものにオープンソースのKubeadm[27]やKubespray[28]がある．また，Kubernetesの構築容易化やサポートなどを強化したベンダ提供のソフトウェアもある．代表的なものとして，2015年からKubernetesをサポートしたRed Hat OpenShift[29]，2017年から提供を開始したRancher[30]，2020年から提供を開始したVMware Tanzu Kubernetes Grid[31]がある．Red Hat OpenShiftとRancherは，各社がサポートするOSやストレージなどの製品などとのインテグレーションを行い統合的なサポートを特徴としている．VMware Tanzu Kubernetes Gridは，VMとKubernetesによるコンテナ環境を一元管理することを特徴としている．

このような中，ヤフーでは，特定のベンダ製品に依存せずパブリッククラウドと同様にKubernetesを構築し管理できるKaaSをゼットラボ（株）にて2016年より開発し，プライベートクラウドにて運用している．以降，本稿では特段の表記がない限り，ヤフーのKaaSを単にKaaSと略す．

3. ヤフーのプライベートクラウドと管理体制

本章では，ヤフーのWebサービスを支えるプライベートクラウドと，その管理体制について述べる．

3.1 プライベートクラウド

ヤフーでは，提供するショッピングサイトや地図サービスなどのWebサービスの多くが自社で運用するプライベートデータセンタにて稼働している．データセンタは2020年12月時点で日本国内5拠点・海外1拠点あり，災害などによりある拠点のデータセンタが稼働停止しても，主要なWebサービスが停止することなく継続運用できる．このデータセンタのうち，主となる日本国内2拠点を中心にプライベートクラウドを提供している．

プライベートクラウドでは，コンピュート・ネットワーク・ストレージを提供するInfrastructure as a Service（以下，IaaSと略す）を提供している．さらに，IaaS上にPlatform as a Service（以下，PaaSと略す），Function as a Service（以下，FaaSと略す）およびKaaSを構築し提供している．ヤフーのWebサービスの多くはプライベートクラウドに構築されたIaaS・PaaS・FaaS・KaaSのいずれかを選択し利用している．

IaaSはオープンソースのOpenStackを主に使用し，2020年12月時点で約200クラスタ以上のOpenStackを運用している[32]．OpenStackはVMとベアメタルを管理することができるが，ヤフーでは主にVM管理に使用している．VMを実行するハイパーバイザーとしてはLinuxが備え持つKVMを使用している．PaaSは，VMware Tanzu Application Service（旧：Pivotal Cloud Foundry）を使用し，2020年12月時点で約17,000個のアプリケーションが本番稼働している[33]．FaaSは，オープンソースのApache OpenWhisk[34]を使用し，2018年10月より運用している．KaaSは，第4章にて詳しく述べる基盤ソフトウェアを使用し，2017年7月にベータバージョンをリリース後，パイロット検証を経て2018年8月より本番運用している．また，KaaSは第2章に示すKubernetesのリリース間隔より短い間隔でバージョンアップをしている．短期間でのバージョンアップを繰り返すことで，利用者からの要望された機能追加やバグ対応に迅速に対応するだけでなく，Kubernetesの最新バージョンをサポートし続けている．Kubernetes v1.11よりサポートを始め，2020年12月時点ではKubernetes v1.18をサポートしている．

3.2　管理体制

KaaSの管理体制について述べる．**図2**にKaaSを管理するKaaS管理者および関連する組織体制について示す．

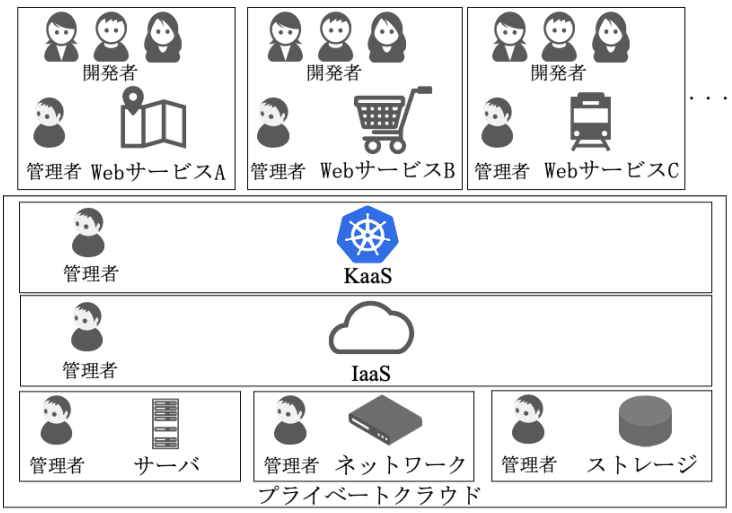


図2　KaaSの管理体制と関連組織

まず，KaaSを構成するIaaSおよびコンピュートであるサーバ，ネットワーク，ストレージの管理体制について述べる．サーバを管理するサーバ管理者は，物理サーバの管理を担当し，データセンタへのサーバ導入・設置から障害時の機器交換などを担当している．ネットワーク管理者

は，ルーター・スイッチ・ロードバランサーなどネットワーク機器の設置・配線からIPアドレスなどのネットワークのリソースを管理している．また，サーバやストレージなどの設置にてIPアドレスが必要となる場合は，ネットワーク管理者が調整し割り当てを行う．ストレージ管理者は，アプライアンスストレージやSDS（Software Defined Storage）の導入・設置からボリュームやテナントの割り当て，ストレージ起因の障害対応などを担当している．IaaS管理者は，主にOpenStackの管理を担当し，サーバ・ネットワーク・ストレージの各管理者と調整を行い，確保したリソースを使いOpenStackを設置・管理している．利用者がVMを利用する場合，利用者はIaaS管理者へVM提供依頼を出し，IaaS管理者がVMおよびこれに接続するネットワークやストレージも併せて提供する．また，各管理者は複数ベンダの製品を導入しているだけでなく，利用者からの要望に応えるためOCP（Open Compute Project），OpenStackの改良，ロードバランサーやストレージなどの開発も行い，複数ベンダ製品と自社開発のインフラによる混在環境を管理している．

次に，KaaSの利用者となるWebサービス部門について述べる．Webサービス部門は，ショッピングサイトや地図サービスなど顧客へ提供するWebサービスを開発し運用管理する部門である．Webサービス部門ではプライベートクラウドを使い，各種アプリケーションの開発とそれらを組み合わせたWebサービスの設計・構築を行う．さらに，Webサービスの運用管理についても，各Webサービス部門の管理者が行っている．KaaSを利用している場合，Webサービス部門にてアプリケーションをコンテナ化し，コンテナ化されたアプリケーションをKubernetes上にデプロイし運用する．Webサービス部門によっては，開発者が管理者を兼ねる場合もある．Webサービス部門の管理者は，自部門のWebサービスの管理者であり，プライベートクラウドにとっては利用者でありKaaS・IaaS・サーバ・ネットワーク・ストレージの深い知識を必ずしも有しているわけではない．そのため，障害時など必要に応じてKaaS・IaaS・サーバ・ネットワーク・ストレージの各管理者へ問合せを行い解決を行っている．また，プライベートクラウドでは，誤った操作による障害を防止するため，利用者には制限された権限のみ付与し提供している．KaaSの利用者に与える権限については4.5節と4.6節にて述べる．

最後に，KaaS管理者について述べる．KaaS管理者はIaaS管理者と調整し確保したリソースを使いKaaSを構築し管理している．KaaS管理者は大きくSRE（Site Reliability Engineering）[35]とCRE（Customer Reliability Engineering）[36]の業務に分かれる[37]．SREは，KaaS自身やUser Kubernetesのバージョンアップ・新機能検証・性能監視を含む健全性チェックなどKaaSの信頼性を担保する作業を担当している[38]．CREは，利用者からの問合せ対応に加え，コンテナやKubernetesに不慣れな利用者向けのトレーニングやKubernetesの活用事例などを共有する社内勉強会の主催などを行っている．KaaS管理者は2020年12月時点でSRE・CRE合わせて25人のチームである．KaaS管理者におけるインフラ管理の経験年数ごとの人数比率を**表1**に示す．

表1　KaaS管理者におけるインフラ管理の経験年数 （2020/12）

インフラ管理の経験年数	比率
0 年以上 5 年未満	56%
5 年以上 10 年未満	16%
10 年以上	28%

一般的にKubernetesの管理では，サーバ・ネットワーク・ストレージの横断的な知識に加え，コンテナやKubernetesの知識と幅広い知識が必要となる．しかし，表1に示すように，経験豊かで幅広い知識を持つ熟練の管理者は少なく，インフラ管理の経験の浅い管理者が多い体制にてKaaSを管理している．

4. KaaSの設計

本章では，KaaSの設計について述べる．KaaSの要件とアーキテクチャ，モデルを述べた後，KaaSを構築する際に使用するコンピュート・ネットワーク・ストレージについて設計思想を述べる．

4.1 要件

ヤフーでは，各Webサービス部門は独立し開発・運用管理している．さらに，マイクロサービス化によりコンテナの利用拡大が見込まれた．そのため，KaaSでは，以下の要件を満たす必要があった．

- 各Webサービス部門が独立した運用管理ができること
- セキュリティの観点から他Webサービス部門のリソースにアクセスできないように制御されていること
- Webサービスを止めることなくアプリケーションのアップデートやリソースの増減ができること
- データセンタのコスト削減のためリソースの集約化ができていること
- KaaSの利用数の増加に比例しKaaS管理者の負荷が増加しないこと

次節より，上記の要件を満たすべく設計したKaaSについて述べる．

4.2 アーキテクチャ概要

KaaSは，第2章で述べたKubernetesの特徴機能を使い，コンテナだけでなくKubernetesを構成するコンピュート・ネットワーク・ストレージのリソースを自律管理し管理負荷を軽減する．**図3**にシステムの概略図を示す．

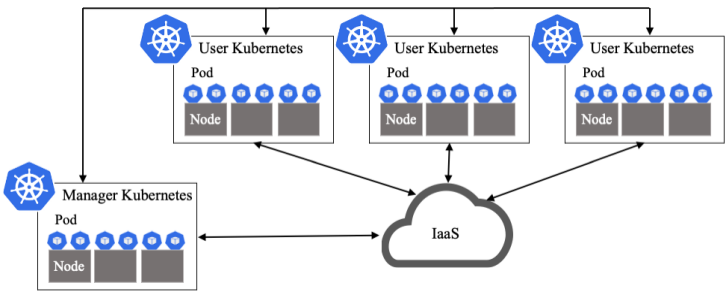


図3　KaaSのシステム概略図

KaaSでは，Webサービス部門が利用するKubernetesをUser Kubernetes，このUser Kubernetesを管理するKubernetesをManager Kubernetesと呼ぶ．Webサービスを構成するアプリケーションのコンテナは，User Kubernetes上で動作する．KaaSでは，マルチクラスタ構成にて各Webサービス部門へUser Kubernetesをそれぞれ提供する．これにより，各Webサービス部門が独立しUser Kubernetesを運用管理できる．

Manager Kubernetesは，KubernetesのAPIのモデル拡張機能であるCustom Resourceを使い，User Kubernetesをモデル化し管理する．モデルについては，4.3節にて述べる．KaaS管理者が，KaaSを管理する際に操作する基盤ソフトウェアは，このManager Kubernetes上でモデルと対になったコントローラにより構成され，これらはコンテナとしてManager Kubernetes上で実行される．

Manager Kubernetesは，IaaSのAPIを通じコンピュート・ネットワークのリソースを作成した後，このリソース上にUser Kubernetesを構築する．構築されたUser KubernetesとそのリソースはManager KubernetesのReconciliationループにより管理される．このような方式をとることで，Kubernetesが備え持つリソースの増減，セルフヒーリング，ローリングアップデートの特徴機能を活用し，Manager Kubernetesにて複数User Kubernetesを自律管理できる．

本アーキテクチャにより，Webサービスにてリソースの増減が必要となった時に割り当てられたインフラのリソースを増減させる．セルフヒーリングでは，User Kubernetesのノードに障害が発生した場合，Manager KubernetesにてUser Kubernetesのノード異常を検知すると，IaaSのAPIを通じ新規VMを作成した後Kubernetesのノードとして必要なソフトウェアをセットアップし，新規ノードとしてUser Kubernetesへ追加する．次に，障害ノードに新たなコンテナが配置されないようにした後，障害ノード上で動作中のすべてのコンテナを削除する．コンテナが削除されると，User Kubernetesのコンテナのセルフヒーリングにより正常なノード上にコンテナが再作成される．Manager Kubernetesは，障害ノード上のすべてのコンテナがなくなったことを確認した後，障害ノードをUser Kubernetesの管理から外し，障害ノードのVMを削除する．このようにManager KubernetesによりUser Kubernetes自体に障害が発生しても自動復旧する．ローリングアップデートもセルフヒーリングと同様の動きをする．つまり，セルフヒーリングの障害ノードを古いバージョンのノードと置き換えるとローリングアップデートの動きとなる．なお，ローリングアップデートはWebサービスに影響が出ない台数ずつアップデートすることで，Webサービスを無停止で全ノードをアップデートする．

4.3 モデル設計

Manager Kubernetesにて管理するUser Kubernetesのモデルについて説明する．**図4**の左図に作成されるリソース，右図にそれに対応するモデルを示す．

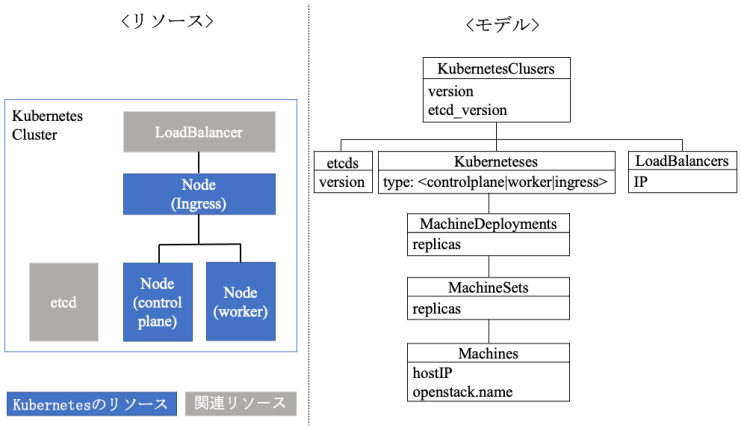


図4　User Kubernetesのモデル

各モデルには，対応するコントローラがそれぞれ存在する．コントローラがモデルに対応するリソースを監視し，Reconciliationループにて定義されたあるべき状態との差分を定期的にチェックし，違いがあればIaaSのAPIを通じリソースの作成・変更・削除などの処理を行う．このように，Reconciliationループでリソースを管理することで，KaaS管理者が手動によるIaaSの操作を行うことなくUser Kubernetesを自律管理する．

User Kubernetesを構成するリソースとしては，Control Plane・Worker・Ingressの3種類のノード用のコンピュート，Kubernetesのステータス情報を保持するデータベースetcd用のコンピュートおよび外部ネットワークとの通信で利用するロードバランサー用ネットワークのリソースがある．これらのリソースが図3に示すようにManager Kubernetesからのリクエストにより生成される．その後，生成されたControl Plane・Worker・Ingressのコンピュート上にKubernetesが構築され，これら3種類のコンピュートはKubernetesのリソースとして管理される．etcdとロードバランサーはManager KubernetesによりUser Kubernetesごとに紐づく関連リソースとして管理される．

次節よりKaaSをIaaS上に構築する際のリソースであるコンピュート・ネットワーク・ストレージについて設計思想を述べる．

4.4　コンピュート

Manager KubernetesがIaaSを通じ作成するUser Kubernetesのコンピュートについて述べる．IaaSとして利用しているOpenStackでは，VMとベアメタルの両方のコンピュートを管理できる．KaaSでは，VMをUser Kubernetesのコンピュートとして採用した．理由は，コンピュートの集約率の高さである．

コンテナを実行するコンピュートでは，4.1節の要件を満たすべく，いかにコンピュート上に性能劣化なく，より多くのコンテナを集約できるかが選定ポイントである．第2章で述べたようにコンテナ自体の実行性能に関してはコンピュートの重要なファクタであるCPU・メモリに関する性能劣化はベアメタルと比べわずかであるが，VMはベアメタルに比べて性能劣化する．一方，Kubernetesコミュニティ SIG-Scalabilityの報告[39]によると，1ノードあたり110Podを

超えると，コンテナランタイムの管理負荷が上がりコンテナの作成・削除・ステータス更新など管理操作の性能が劣化し始める．そのため，Kubernetesのデフォルト設定値では，1ノードあたり最大110Podに設定され100Pod以下での利用が推奨されている[40]．

そこで，コンテナの実行性能を落とさず，より多くのコンテナ数を稼働させるべく，**表2**に示す測定環境にて，ベアメタルとVMのKubernetesのノードをそれぞれ作成し性能比較を行った．VMは，1台の物理サーバ（Hypervisor）上に，CPU・メモリのオーバーコミットなしで2VMを作成する．本環境は，ベアメタル・VMともに同スペックのCPU・メモリ数のKubernetesのノードである．つまり，ベアメタルは最大110Pod，VM環境は最大220Pod（1VMあたり最大110Pod）が稼働可能である．これらのノードを割り当てたKubernetes v1.13.2にて性能測定を行った．

表2　測定環境

ベアメタル	CPU: 32 cores (Intel Xeon CPU E5-2620 v4x), Memory 64GBytes
VM	[Hypervisor] CPU 64 cores(Intel Xeon CPU E5-2620 v4x), Memory 128GBytes [仮想化基盤ソフトウェア] KVM [VM] 上記 Hypervisor 上に 2VM (1 VM あたり CPU 32 cores, Memory 64GBytes)

性能測定で利用するベンチマークツールsysbenchを1コンテナ／Podとして準備し，同時に実行するコンテナ数を10コンテナずつ増加させ実行し，96パーセンタイルのレイテンシを測定した．sysbenchでは，ストレージやネットワークを利用せずCPU負荷をかけるCPUテストを利用した．測定結果を**図5**に示す．

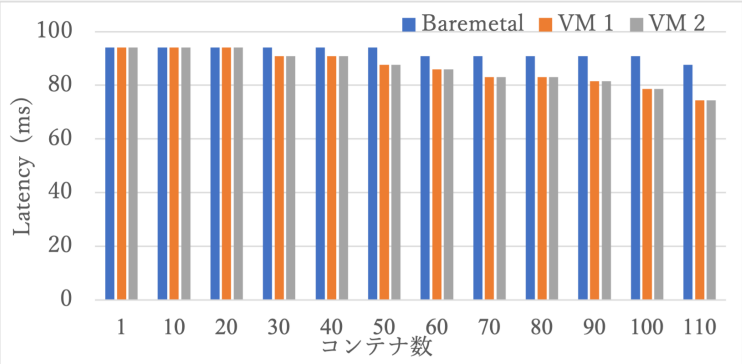


図5　ベアメタルとVMにおける実行コンテナ数における性能比較

図5に示すようにベアメタル上に110コンテナ，VM上に220コンテナ（VM1とVM2に各110コンテナ）が動作している状況では，VMの方がレイテンシが短く性能が良い結果であった．また，ベアメタルでは50コンテナ，VMでは60コンテナ（VM1:30コンテナ，VM2:30コンテナ）

から性能が徐々に向上している結果であった．これは，本測定環境にて使用しているCPUが熱や消費電力に応じて動作周波数を動的に引き上げるインテル ターボ・ブースト・テクノロジー[41]が有効であったため，これが影響していると推察する．

本測定結果より，VMの方がベアメタルに比べ1台の物理サーバ上で性能劣化なく動作させることができるコンテナ数は多く，集約率が高いと言える．必要なCPU数やメモリ量が小さいコンテナが数多く動作するような組織では，物理サーバ上にVMを多数構築しその上でコンテナを稼働させることで，1台の物理サーバあたりに稼働可能なコンテナ数は増加し，集約率はさらに向上する．一方，ベアメタルは，仮想化することができない特殊なハードウェアを備える場合や，大量のCPUやメモリを使用するアプリケーションのコンテナを少数動かす場合に有効であると考える．ベアメタルは仮想化のレイヤを挟まないため，VM管理が不要となり管理がシンプルになるメリットがあると考える．

ヤフーでは，CPUやメモリを大量に利用するアプリケーションは少なく，1台の物理サーバあたりのコンテナの集約率の高さからノードにVMを採用した．

4.5　ネットワーク

KaaSにおけるネットワークについて述べる．**図6**にKaaSにおけるネットワークについて示す．

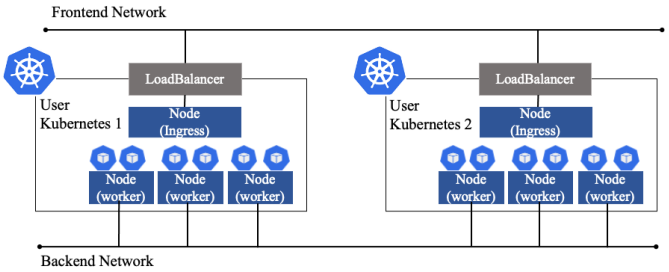


図6　User Kubernetesのネットワーク

Manager Kubernetesは，インターネットなど外部のネットワークに到達可能なフロントエンドネットワークから，User Kubernetesがネットワークアクセスを受けるアクセスポイントとしてロードバランサーを作成する．ロードバランサーはフロントエンドネットワークのアドレスを固定IPアドレスで割り当て作成する．このIPアドレスがDNSに登録されている．ロードバランサーには，ソフトウェアロードバランサーとハードウェアロードバランサーの複数種類を用意しており，Webサービス部門からの要求に応じ使い分けている．ロードバランサーには，L7ロードバランサーであるIngressノードがメンバ登録されている．

Ingressノードは，L7ロードバランサーであるオープンソースのnghttpx[42]をコンテナとして動作させ，負荷分散・SSL終端・仮想ホスティングの機能を提供している．Ingressノードで受けたネットワークアクセスは，対象となるアプリケーションのコンテナへ振り分けられる．User Kubernetes内のコンテナ間の通信は，コンテナネットワークとして，オーバレイネット

ワークを構築しUser Kubernetes内のみで有効なIPアドレスを各コンテナへ付与し通信を行う．このコンテナネットワークにより，コンテナ数が増大しても，フロントエンドネットワークやバックエンドネットワークのIPアドレスを消費しない．

各ノードのIPアドレスは，DHCPを用いVM間通信で使われるバックエンドネットワークのIPアドレスを動的に付与する．DHCPを利用することで，User Kubernetesの作成時やノード数を増減させる際，ネットワーク管理者とKaaS管理者間でIPアドレスの調整を行う必要がない．ただし，DHCPを使いノードのIPアドレスを付与すると，User Kubernetes 1とUser Kubernetes 2のノードが同じバックエンドネットワークの同一セグメントのIPアドレスが割り当てられるようになり，セキュリティの問題が発生する．つまり，User Kubernetes 1を利用しているWebサービス部門が別Webサービス部門の利用するUser Kubernetes 2へアクセスできてしまう．この問題に対し，利用者によるノードへのログインを禁止し，他Webサービス部門のノードへアクセスをできないようにした．KaaSにてノードにログインしなければならないユースケースは，ノード作成時や障害発生時であるが，いずれもKaaSにて実行されるため利用者やKaaS管理者がログインする必要がない．Manager Kubernetesのバグなどの想定外のケースによりログインせざるを得ない状況の場合は，これに対応できるKaaS 管理者のみがログインできればよく，利用者のログインを禁止しても問題はない．

このようにネットワークを組むことで，KaaSでは，User Kubernetes上のコンテナ数の増減やUser Kubernetesを構成するノードの増減があったとしても，KaaS管理者とネットワーク管理者が調整する必要なくIPアドレスを割り当てられる．

4.6　ストレージ

User Kubernetesのコンテナがデータを永続化するために利用するストレージについて述べる．**図7**にUser Kubernetesにおけるストレージについて示す．

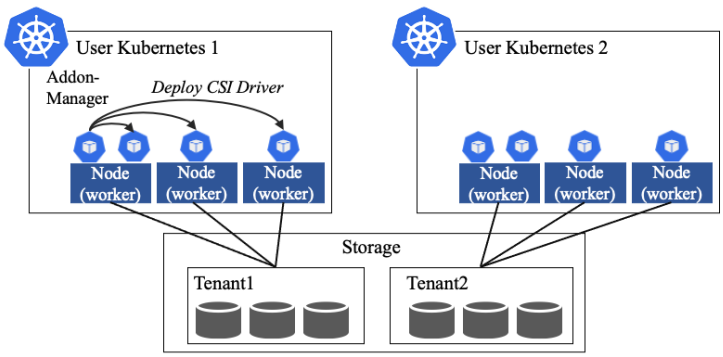


図7　User Kubernetes のストレージ

ストレージは，User Kubernetesから外部のストレージへアクセスし利用するため，Manager Kubernetesにて管理する必要がない．ただし，User Kubernetesからストレージにアクセスするためには，コンテナ向けストレージインタフェースの標準仕様CSI（Container Storage Interface）[43]に準拠したCSI DriverをUser Kubernetes上で稼働させる必要がある．このCSI Driverはストレージの製品ごとに必要であり，ストレージやCSI Driver自身もしく

はKubernetesのバージョンアップに応じてアップデートが必要となる．また，CSI Driverは，ストレージを利用するために必要不可欠なものではあるが，各Webサービス部門の利用者は自分たちが顧客へサービスするものではないため，できるならば管理したくない．そこで，KaaSではKubernetesが備え持つAddon-manager[44]をベースに安定性を向上させた独自Addon-managerを使いCSI Driverを各ノードへデプロイし管理する．これによりノードが新たに追加された際にAddon-managerにより自動でCSI Driverをセットアップするだけでなく，CSI Driverに障害が発生してもCSI Driverを自動復旧させる．さらにバージョンアップ時にも，古いバージョンのCSI DriverをAddon-managerが検知し自動でアップデートを行う．このようにAddon-managerを利用することで，利用者がCSI Driverを手動で管理する必要なくストレージが利用できるようになる．

次に，User Kubernetesが利用する外部のストレージについて述べる．ストレージの割り当てでは，ストレージ管理者との調整が発生する．金子らの研究[45]によると，ストレージの割り当てには，サーバ管理者とストレージ管理者の複雑な調整が必要となり管理上の課題となる．KaaSにおいても，同様にWebサービス部門・KaaS管理者・ストレージ管理者での調整は管理負荷を増加させる要因となる．これに加え，セキュリティの観点から，各Webサービス部門のデータは他Webサービス部門からアクセスできないよう制御する必要がある．

そこで，KaaSでは図7に示すようにストレージ内にテナントを作成し，各テナントを各User Kubernetesにそれぞれ割り当てる構成とする．利用者はストレージの初回の利用時にストレージ管理者へ容量・要望性能を伝え，ストレージ管理者は要求に応じストレージ内にテナントを作成し利用者へ提供する．利用者は，ストレージ管理者から連絡を受けたテナントへのアクセス情報（ストレージのIPアドレス，ユーザ名，パスワード）をCSI Driverに設定し利用する．Kubernetesでは，ストレージの操作はCSI Driverを通じ行われるため，ストレージ管理者がボリュームの生成元となるストレージプールをテナントに割り当てるだけで，利用者はストレージに直接アクセスすることなくボリュームの作成・削除およびコンテナへのマウント・アンマウントを自由に行える．しかし，ストレージが備え持つスナップショット・ミラー・クローンなどのレプリケーション機能においては，ベンダやストレージの種類によりアーキテクチャが異なる．そのため，ストレージの深い知識を有さない利用者は自部門のWebサービスが利用するボリュームの容量は判断できるものの，レプリケーション機能により，どれだけのSSDやHDDがストレージ内部で消費されるか正確に見積もることができない．これにより，場合によってはWebサービスで利用中のボリュームの性能劣化や新規作成ができなくなってしまうリスクがある．そのため，KaaSで使用するストレージでは，レプリケーション数などの制限をつけストレージ管理者にて管理している．

また，KaaSでは，許可されたUser Kubernetesのノードのみがストレージのテナントへアクセスできるように，ストレージのアクセスコントロールリスト（ACL）を更新する．しかし，このACLの更新には問題がある．ACLへ登録するノード識別子（iSCSIのIQNなど）はノードにログインしなければ情報を取得できない．しかし，4.5節に述べた理由により利用者はノードにログインできず，ノードが増減するたびにKaaS管理者がノードにログインしノード識別子を確認しストレージ管理者へ連絡する必要が出てくる．このような管理はKaaS管理者の負荷を増加させてしまう．そこで，KaaSでは，ノードの作成・削除時にノードからノード識別子を取得し，ACLを自動更新するコントローラを開発し，Addon-managerにてCSI Driverとともにセットアップしている[46]．これにより，人手を介さずにノード識別子を取得しACLを自動更新する．このような仕組みをとることで，KaaSでは利用者とストレージ管理者間でストレージの利用開

始時の1回のみストレージのテナントの割り当て依頼を行うだけで，その後は自由にボリュームを作成・削除できかつ他Webサービス部門からアクセスできないように制御されたストレージが利用できる．このKaaS向けのストレージは2020年8月より提供開始している．

5. 運用実績

本章では，第4章で述べたKaaSのリソース使用実績と運用実績について述べる．

5.1リソース使用実績

2020年6月および2020年12月時点での稼働しているコンテナの総数，KaaSにて管理されているUser Kubernetesの総数およびUser Kubernetesのノードの総数について**表3**に示す．

表3　KaaSのリソース使用量

	2020 年 6 月	2020 年 12 月
コンテナ数	約 129,350 個	約 204,980 個
User Kubernetes 数	約 680 クラスタ	約 860 クラスタ
ノード数	約 13,900 VM	約 27,900 VM

表3に示すように稼働するコンテナの総数は，6カ月で約1.6倍増加しており，ヤフーのWebサービスではコンテナ化が進んでいることが分かる．さらに，User Kubernetesの総数も6カ月で約1.3倍増加している．KaaSではマルチクラスタ構成にてUser Kubernetesを作成しWebサービス部門へ提供しているため，利用しているWebサービスが増えたとともに利用者数も増加していると言える．User Kubernetesのノードの総数は6カ月で約2.0倍増加している．1ノードあたりに稼働しているコンテナ数の平均値は，2020年6月時点では約9.31コンテナ／ノード，2020年12月時点は約7.35コンテナ／ノードである．1ノードあたりの稼働コンテナ数が下がっている．これは2020年6月から12月にかけ，大容量のメモリを要求するWebサービスが利用開始したためである．

また，2020年12月時点における各ノード種別とetcdのVM数の比率は，それぞれControl Plane 7.2%，Worker 70.6%，Ingress 10.7%，etcd 11.5%であった．

5.2　運用実績

図8にUser Kubernetes数とKaaS管理者数の推移を示す．

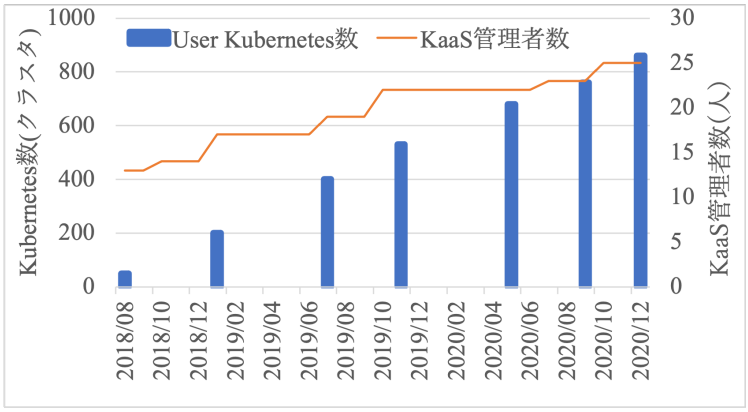


図8　User Kubernetes数とKaaS管理者数の実績

User Kubernetes数は運用開始直後から増加し続け，2020年12月時点では約17.2倍の約860クラスタが稼働している．また，第4章にて述べたKaaSの基盤ソフトウェアはバージョン1.12から運用を始め，2020年12月ではバージョン1.37が稼働している．つまり，平均1.12カ月ごとにバージョンアップしている．このバージョンアップでは，その都度，KaaS管理者により全User Kubernetesのローリングアップデートが実施される．つまり，ノードの連続稼働期間も約1.12カ月である．これにより，KaaSの新機能追加・バグ修正だけでなくKubernetesの最新バージョンへの追随やOSの最新パッチ適用が行われ先進性と安定性を保っている．

次に，利用者からKaaS管理者への問合せについて，月ごとの問合せ件数および問合せチケット内のコメント数について実績を**図9**に示す．なお，本グラフはKaaS管理者が問合せごとに作成しているチケットであるGitHub Issueの値であり，口頭やメールなどのコミュニケーションにて短期解決したものは含まない．また，問合せチケット内のコメント数は，問合せ内容の難易度を示す一指標として，調査時におけるKaaS管理者のメンバ間・IaaS管理者・サーバ管理者・ネットワーク管理者・ストレージ管理者・ゼットラボ（株）との技術的なやりとりのコメント数をカウントし，月ごとの中央値を算出している．

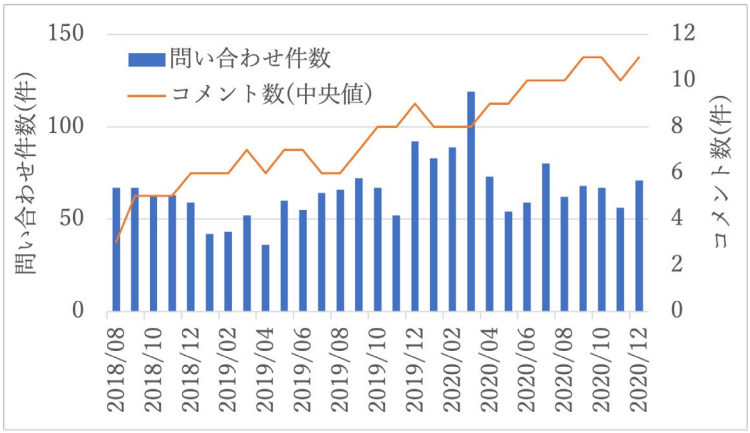


図9　KaaS管理者への問合せ件数とチケット内のコメント数

図9に示すように，各月でばらつきはあるが，平均65.55件／月の件数である．運用開始直後は67件，2020年12月時点では71件であった．また，2020年3月近辺の問合せが一時的に増加しているが，これは新たにセキュリティ区分の高いネットワークセグメント向けにKaaSを提供し始めたためである．

6. 考察

本章では，第5章の運用実績を用い，第4章の要件であったリソースの集約効果とKaaS管理者の運用負荷について考察を述べる．

6.1 リソースの集約効果

5.1節の運用実績から，KaaSを導入したことによるリソースの集約効果についてコンテナ化とマルチクラスタ構成の観点から考察する．まずは，コンテナ化による集約化効果について述べる．コンテナ化による集約率としてコンピュート・ネットワークは約86.4%の効果であった．Webサービスのマイクロサービス化により，アプリケーションを実行するコンピュートは，増減させる単位にあわせて設計される．そこで，アプリケーションをコンテナ化しなかったと仮定した際のコンピュートについて，表3の2020年12月時点のデータから算出する．最もシンプルな構成である1アプリケーション／VMの構成とした場合のVM数は約204,980台となる．これに対し，コンテナ化した場合は表3に示すように約27,900ノード（VM）である．つまり，アプリケーションのコンピュートにVMを使う場合と比べ，コンテナ化による集約率は約86.4%である．ネットワークについては，アプリケーションをVM上で稼働させる場合，最もシンプルな構成は1VMにつきバックエンドネットワークのアドレスを1IPアドレス割り当てる構成である．一方，コンテナの場合，1 コンテナにつき1IPアドレスとなる．しかし，KaaSではコンテナのIPアドレスは，4.5節に示すようにコンテナネットワークのアドレスであり，バックエンドネットワークのアドレスは消費しない．つまり，ノードのIPアドレスのみバックエンドネットワークのアドレスを消費する．そのため，バックエンドネットワークで必要となるIPアドレス数は，アプリケーションをVMで稼働させる場合は約204,980個，コンテナで稼働させる場合は約27,900個となる．つまり，コンテナ化することでバックエンドのIPアドレスの集約率はコンピュートと同じく約86.4%である．フロントエンドネットワークは，Webサービスのアクセスポイントとなるロードバランサーへ割り当てられるものであり，VMとコンテナによる消費するIPアドレス数の違いはない．同様に，ストレージに関しても，VMとコンテナによるボリューム数やサイズに差はない．

次に，マルチクラスタ構成によるリソースの集約率について述べる．第2章で述べたようにKubernetesでは，マルチテナントとマルチクラスタの2つの構成が取れる．KaaSではWebサービス部門ごとに独立したKubernetesが必要となるため，マルチクラスタ構成を採用している．しかし，マルチテナント構成とマルチクラスタ構成ではリソース量に差がある．仮に，マルチテナント構成により全Webサービス部門が1台の巨大なUser Kubernetesを共有し利用したと仮定する．この場合，1台のKubernetesとなるためControl Plane，etcdのリソースが集約される．つまり，2020年12月時点ではVM数の比率よりControl Planeとetcdの合計18.7%のコンピュートが集約できる．ネットワークについては，バックエンドネットワークのIPアドレス数はVM数に依存するため，コンピュートと同じ18.7%集約できる．フロントエンドネットワークのIPアドレスは，Webサービスのアクセスポイントとなるロードバランサーに割り当てられるもの

であり，必要なIPアドレスは変わらない．同様に，ストレージも消費量に差はない．このように，KaaSで採用しているマルチクラスタ構成は，マルチテナント構成に比べ，18.7%のコンピュートとネットワークのリソースを多く使用している．

以上のように，KaaSは，コンテナ化したアプリケーションを対象とすることで，コンピュート・ネットワークの集約効果があり，マルチクラスタ構成によってWebサービス部門の独立性を確保できる利点がある．もし，各Webサービス部門の独立性が必要ない組織であれば，マルチクラスタ構成ではなく，マルチテナント構成を取ることで，さらにコンピュートとネットワークのリソース削減が見込める．

6.2　KaaS管理者の管理負荷

5.2節の運用実績を用い，KaaS管理者の管理負荷について考察する．KaaS管理者は13人にて運用を開始し，2020年12月時点では25人である．2年4カ月間で約1.92倍増員している．しかし，単に管理者数が増員したことで，利用数が増加し続けるKaaSの管理が行えている訳ではない．

図10にKaaS管理者1人あたりが管理しているUser Kubernetesの平均クラスタ数の推移とその近似曲線を示す．2018年8月の約3.85 クラスタ／人から徐々に管理台数が伸び，2020年12月では約34.4クラスタ／人となっている．2年4カ月で，1人のKaaS管理者が管理できるUser Kubernetesの数が約8.94倍向上している．つまり，単に人数が増えたことで大量のUser Kubernetesを管理できているのではなく，KaaS管理者が管理可能な台数も同時に向上した結果である．さらに，3.2節に示すようにKaaS管理者は経験の浅い管理者が多く，一部のインフラ管理に長けた熟練の管理者に依存したものではない．

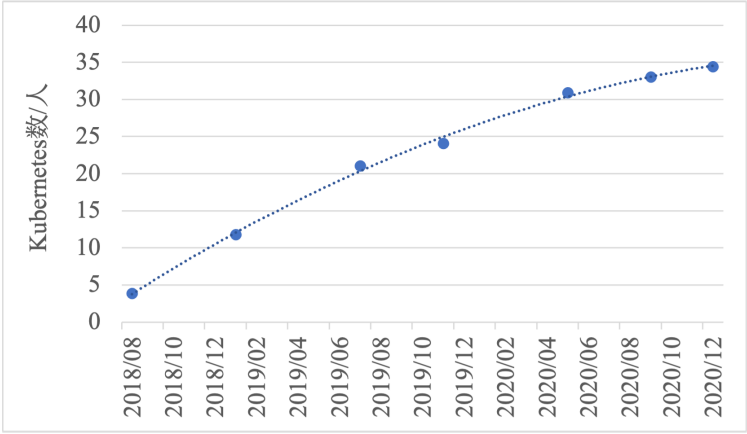


図10　KaaS管理者1人あたりが管理するKubernetesクラスタ数

次に，KaaS管理者におけるSREの作業に着目する．SREの主な作業の1つであるバージョンアップ作業を取り上げ，管理負荷の軽減効果について述べる．KaaSでは，5.2節に示したように平均1.12カ月ごとにバージョンアップし，新機能追加やバグ対応を行うことで先進性と安全性を保っている．このKaaSのバージョンアップでは，全User Kubernetesのコンピュートとネットワークのリソース再作成が必要となる．つまり，KaaSのアップデートにより，2020年12月ではUser Kubernetesで利用する約27,900台のノードがローリングアップデートにより，Webサー

ビスを停止させることなく自動で再作成される．ここで仮定として，もしKaaSを使わずに，全ノードの再作成をKaaS管理者が手動で行わなければならない状況を想定する．手動によるノードの再作成に要する時間を10分／台と仮定した場合，その作業時間は4,650時間（＝279,000分）となる．これは，KaaS管理者25人が全員で対応したとしても，1人あたり186時間かかり，1カ月の作業時間160時間（＝20（日）×8（時間））を超える．なお，この仮定では，ノードの再作成にIaaS管理者やネットワーク管理者，ストレージ管理者など他管理者との調整が発生しないことを前提としたが，調整が発生するような場合，さらに作業時間は伸びKaaS管理者の負担は増加する．KaaSの管理では，このバージョンアップ作業だけにすべての時間を費やすことはできない．これに対し，KaaSでは，宣言的管理によりKaaS管理者はUser Kubernetesの望むバージョンを指定するのみで全ノードがローリングアップデートされる．KaaSでは，この望むバージョンの指定は1コマンドの実行で指定でき，KaaS管理者の作業時間は1 User Kubernetesあたり1分以下である．2020年12月のUser Kubernetes数は約860クラスタであり，これのバージョンアップにかかるKaaS管理者の作業時間は860分以下である．つまり，KaaS管理者がバージョンアップを手動で行った場合と比べ，KaaSにより約99.7%もの作業時間が短縮され，管理負荷が大幅に軽減されている．

次に，KaaS管理者におけるCREの作業に着目する．CREの主な作業の1つである利用者からの問合せ対応について取り上げる．図9で示した利用者からの問合せの件数は，図8で示したUser Kubernetes数の増加数には比例していない．また，ヤフーではKaaS管理者のうちCREとして約半数の人員が対応している．つまり，KaaS管理者1人あたりが対応する問合せ件数は，2018年8月時点では約9.57件／人，2020年12月時点では約5.46件／人である．2年4カ月でKaaS管理者1人あたりが対応する問合せ件数は約42.9%削減している．問合せの内容に注目する．問合せ内容は，KaaSのサービス開始直後はKubernetesの使い方やKaaSの初歩的なバグに関するものが多くあった．その後，KaaSの利用が浸透するに従い，アップデート時の性能影響やコンテナ数増加に伴う高負荷時の挙動に関する問合せなど，徐々に高度な問合せ内容へとシフトしている．この問合せ内容の高度化に伴い，図9で示したKaaS管理者と他管理者との技術的なやりとりの回数が増加傾向にあり，複数人で対応しないと解決できない内容の問合せが増えている．このような高度な問合せへシフトしている主な要因は2つ考えられる．1つ目は，KaaSの利用者向けドキュメントの充足や社内勉強会の実施，さらには世界中でのKubernetesの利用者の増加に伴いドキュメントや事例などが増えたことで，利用者のKaaSおよびKubernetesへの理解度が高まったためと考えられる．2つ目は，KaaSのバージョンアップを高頻度に行うことで，機能改善やバグが早期に対策されているためと考える．このように，問合せ対応は量から質へと変化している．

以上のように，KaaS管理者の人員を増やすだけでなく，KaaSによる自律管理により管理者の作業負荷の軽減効果があった．さらに，利用者からの問合せは量から質へと変化している．特に，大規模なコンテナ環境を管理するKaaS管理者においては，KaaSのような自律管理により数に負けない管理手順を確立し時間を確保した上で，高度化する問合せに対応すべく，確保した時間を使い知見を深めることが，今後より一層重要になる．

7. まとめと今後の課題

ヤフーでは，年々増加するコンテナおよびコンテナの実行環境を管理するため，Kubernetesを使いKubernetesを管理するKaaSを開発し，2年4カ月運用してきた．このKaaSにより，2020年12月時点ではコンテナ数は約204,980個，User Kubernetes数は約860クラスタへと

成長した大規模コンテナ環境を，25人のKaaS管理者にて管理している．この大規模コンテナ環境により，コンテナ化したアプリケーションを多数運用できるようになったことで，コンテナ化せずVM上で稼働させたと仮定した場合と比較し，コンピュート・ネットワークは約86.4%の集約効果があった．さらに，KaaSはKubernetesを使いKubernetesを管理する自律管理方式により，KaaS管理者の負荷も軽減された．KaaS管理者の主な作業の1つであるバージョンアップ作業を取り上げると，約99.7%の作業時間の軽減効果があった．もう1つの主な作業である利用者からの問合せ対応については，2年4カ月の期間で問合せの量よりも質へと変化していることが判った．この変化に応えるために， KaaS管理者は今後より一層知見を深めることが重要である．この知見を深めるための時間を確保するためにも，大規模コンテナ環境では，KaaSのような自律管理方式の採用と各レイヤの管理者間で不要な調整を避けるインフラ設計が必要である．

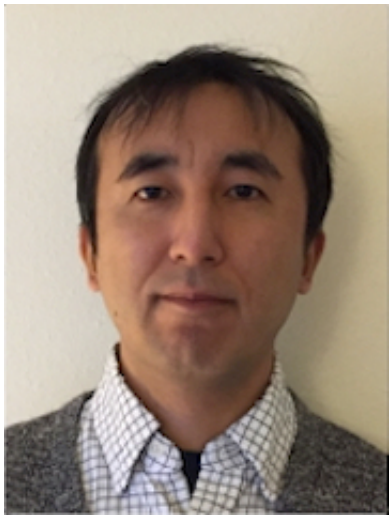
今後の課題は，リソースのさらなる集約化に向けて，各Webサービス部門の独立性を維持しつつ，マルチテナント構成のようにKubernetes 自身の管理で必要なコンポーネント用の集約およびパブリッククラウドのサービスで利用が始まっているサーバレスコンピューティングの活用によるさらなるリソース削減である．また，KaaS管理者のさらなる負荷軽減に向けて，KaaS管理者がKubernetesを直接管理するシーンをさらに削減すべくWebサービス部門のアプリケーション開発・運用管理に連動したアプリケーション視点での自律管理方式の実現である．

参考文献

1）Lewis, J. and Fowler, M. : Microservices, https://martinfowler.com/articles/microservices.html (参照 2021-02-03)
2）Newman, S.（著），島田浩二（訳）：モノリスからマイクロサービスへ―モノリスを進化させる実践移行ガイド，オライリー・ジャパン (2020).
3）Soltesz, S., PÖtzl, H., Fiuczynski, M. E., Bavier, A. and Peterson, L. : Container-based Operating System Virtualization : A Scalable, High-performance Alternative to Hypervisors. In Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, EuroSys '07, pp.275–287 (2007).
4）Verma, A., Pedrosa, L., Korupolu, M. R., Oppenheimer, D., Tune, E. and Wilkes, J. : Large-scale Cluster Management at Google with Borg, Proceedings of the European Conference on Computer Systems, Bordeaux, France, pp.1-17 (2015).
5）Mouat, A. : Using Docker, O'Reilly Media, Inc. (2015).
6）containerd Community : containerd, https://containerd.io/ (参照 2021-02-03).
7）CRI-O Community : cri-o, https://cri-o.io/ (参照 2021-02-03).
8）Felter, W., Ferreira, A., Rajamony, R. and Rubio, J. : An Updated Performance Comparison of Virtual Machines and Linux Containers, 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Philadelphia, PA, pp.171-172 (2015).
9）Kivity, A., Kamay, Y., Laor, D., Lublin, U. and Liguori, A. : kvm : the Linux Virtual Machine Monitor, Proceedings of the Linux Symposium, Vol.1, No.8, Dttawa, Dntorio, Canada, pp.225-230 (2007).
10）Docker Inc. : Swarm Mode Overview, https://docs.docker.com/engine/swarm/ (参照 2021-02-03)
11）Mesos community : Apache Mesos, http://mesos.apache.org (参照 2021-02-03)
12）CNCF Survey : Use of Cloud Native Technologies in Production Has Grown Over 200%, https://www.cncf.io/blog/2018/08/29/cncf-survey-use-of-cloud-native-technologies-in-production-has-grown-over-200-percent/ (参照 2021-02-03)
13）Truyen, E., Van Landuyt, D., Preuveneers, D., Lagaisse, B. and Joosen, W. : A Comprehensive Feature Comparison Study of Open-source Container Orchestration Frameworks, Applied Sciences, Vol.9, No.5, p.931 (2019).

14）Jeffrey O. Kephart and Devid M. Chess : The Vision of Autonomic Computing, IEEE Computer Society, pp.41–50 (2003).
15）G. Lanfranchi, P. D. Peruta, A. Perrone and D. Calvanese: Toward a New Landscape of Systems Management in an Autonomic Computing Environment, IBM SYSTEMS JOURNAL, Vol.42, No.1, pp.5–18 (2003).
16）Hausenblas, M. and Schimanski, S. : Programming Kubernetes, O'Reilly Media, Inc. (2019).
17）Kubernetes Community : Declarative Management of Kubernetes Objects Using Configuration Files, https://kubernetes.io/docs/tasks/manage-kubernetes-objects/declarative-config/ (参照 2021-02-03).
18）Truyen, E., Van Landuyt, D., Reniers, V., Raque, A., Lagaisse, B. and Joosen, W. : Towards a Container-Based Architecture for Multi-Tenant SaaS Applications, Proceedings of the 15th International Workshop on Adaptive and Reflective Middleware, ARM 2016, New York, NY, USA, Association for Computing Machinery (2016).
19）Kubernetes Community : Kubernetes Components, https://kubernetes.io/docs/concepts/overview/components/ (参照 2021-02-03)
20）Google Cloud : Google Kubernetes Engine, https://cloud.google.com/kubernetes-engine (参照 2021-02-03)
21）Amazon Web Service : Amazon Elastic Kubernetes Service, https://aws.amazon.com/eks/ (参照 2021-02-03)
22）Amazon Web Service : Amazon EC2, https://aws.amazon.com/ec2/ (参照 2021-02-03)
23）Amazon Web Service : AWS Fargate, https://aws.amazon.com/fargate/ (参照 2021-02-03)
24）Microsoft Azure : Azure Kubernetes Service (AKS), https://azure.microsoft.com/en-us/services/kubernetes-service/ (参照 2021-02-03)
25）Corey Sander : Fast and Easy Containers : Azure Container Instance, https://azure.microsoft.com/en-us/blog/announcing-azure-container-instances/ (参照 2021-02-03)
26）Virtual Kubelet Project : Virtual Kubelet, https://github.com/virtual-kubelet/virtual-kubelet/ (参照 2021-02-03)
27）Kubernetes Community : Kubeadm, https://kubernetes.io/docs/reference/setup-tools/kubeadm/ (参照 2021-02-03)
28）kubespray Community : Deploy a Production Ready Kubernetes Cluster, https://kubespray.io/ (参照 2021-02-03).
29）Red Hat : Red Hat OpenShift, https://www.redhat.com/en/technologies/cloud-computing/openshift/ (参照 2021-02-03)
30）Rancher Labs : Rancher, https://rancher.com/products/rancher/ (参照 2021-02-03)
31）VMware : VMware Tanzu Kubernetes Grid, https://tanzu.vmware.com/kubernetes-grid (参照 2021-02-03)
32）奥村　司：月間800億PVを支えるIaaS基盤の舞台裏（構築編）, https://techblog.yahoo.co.jp/entry/2020102130034499/ (参照 2021-02-03)
33）水落啓太：ヤフーのPrivate PaaSチームにおけるSREの取り組み, https://techblog.yahoo.co.jp/entry/2020121130052943/ (参照 2021-02-03)
34）OpenWhisk community : Apache OpenWhisk, https://openwhisk.apache.org (参

照 2021-02-03)
35）Beyer, B., Jones, C., Murphy, N. R. and Petoff, J. : Site Reliability Engineering, O'Reilly Media, Inc. (2016).
36）Rensin, D. : Introducing Google Customer Reliability Engineering, https://cloud.google.com/blog/products/gcp/introducing-a-new-era-of-customer-support-google-customer-reliability-engineering (参照 2021-02-03)
37）藤江貴司：クラスタ数530以上，大規模Kubernetesを運用するエンジニア組織の作り方，https://techblog.yahoo.co.jp/entry/20191211786995/ (参照 2021-02-03)
38）勝田広樹：Kubernetes as a Serviceを2年稼働させて，行ってきた改修と知見紹介，https://techblog.yahoo.co.jp/entry/20200203804048/ (参照 2021-02-03)
39）Shyam Jeedigunta and Maciek Różacki : Kubernetes Scalability A Multi Dimensional Analysis, https://static.sched.com/hosted_files/kccna18/92/Kubernetes%20Scalability_%20A%20multi-dimensional%20analysis.pdf (参照 2021-02-03)
40）Kubernetes Community : Considerations for Large Clusters, https://kubernetes.io/docs/setup/best-practices/cluster-large/ (参照 2021-02-03)
41）Intel Corporation : インテル ターボ・ブースト・テクノロジー 2.0，https://www.intel.co.jp/content/www/jp/ja/architecture-and-technology/turbo-boost/turbo-boost-technology.html (参照 2021-02-03).
42）Nghttp2 Community : nghttpx—HTTP/2 proxy-HOW-TO, https://nghttp2.org/documentation/nghttpx-howto.html (参照 2021-02-03)
43）Container Storage Interface Working Group : Container Storage Interface (CSI) Specification, https://github.com/container-storage-interface/spec (参照 2021-02-03)
44）Kubernetes Community : Addon-manager, https://github.com/kubernetes/kubernetes/tree/master/cluster/addons/addon-manager (参考 2021-02-03)
45）金子　聡，中島　淳，坂下幸徳，敷田幹文：大規模高信頼データセンタ向けVM 作成自動化方式の提案と事例による評価，情報処理学会デジタルプラクティス，Vol.7, No.2, pp.195-204 (2016).
46）坂下幸徳，飯田　諒：ヤフー／ゼットラボのステートフルアプリケーションへの挑戦，https://techblog.yahoo.co.jp/entry/2020082530014927/ (参照 2021-02-03).



坂下幸徳（正会員）ysakashi@zlab.co.jp
2003年北陸先端科学技術大学院大学修士修了．2015年同大学博士（情報科学）．2003年（株）日立製作所入社，主任研究員．2018年ヤフー（株）／ゼットラボ（株）入社，研究開発に従事．

採録決定：2021年3月31日
編集担当：浜　直史（（株）日立製作所研究開発グループ）