Technology Reports（特集）
Kubernetes AWS コンテナ
パブリッククラウド活用特集

クラウドオーケストレータを活用した複数Kubernetes管理

DOCOMO Innovations, Inc. 高田 雅人（たかだ まさと） 直井 康広（なおい やすひろ）

Kubernetesは元Googleの開発者によって開発され，現在，CNCFによりオープンソースとして公開されている．Kubernetesにより，ユーザはコンテナ化されたアプリケーションを，環境を問わず効率的に運用することが可能となり，昨今，先進的な企業をはじめとする多くの企業で活用されている．2018年ごろには1つのプロジェクトで複数のKubernetesクラスタを利用する企業も増えているが，一方で複数のKubernetesクラスタを管理するツールが存在していなかった．そこで，DOCOMO Innovations, Inc.では，複数Kubernetesクラスタの一元管理を可能とするクラウドオーケストレータをオープンソースとして開発した．本ソフトウェアは，現在，ドコモ社内の商用システム上ですでに活用されており，本稿ではその事例を解説する．

1. まえがき

2013年のDocker*1の登場は，コンテナ仮想化技術*2を活用したOSレベルでの仮想化がなされることで，アプリケーションとシステム環境が論理的に分離された．これによりワークロード*3におけるコンテナの素早いデプロイ*4やアップデートを可能とした．一方，Dockerはコンテナ管理，スケーラビリティ，自動復旧などに関する問題を抱えていた．利用者は各環境で使用する前に，これら課題を解決する必要があるが，多大な稼働が掛かっていた．

昨今，注目されているKubernetes［1］はこの課題を解決するために登場し，オープンソースソフトウェアとして提供されている．Kubernetesとは，ギリシャ語で操舵手という意味であり，その名が示すとおり，コンテナの運用管理や自動化を目的としたソフトウェアである．開発者は自前でコンテナ向けのフレームワーク*5を用意することなく，複数コン

＊1 Docker：コンテナ型仮想化ソフトウェア．Docker Inc.の登録商標．
＊2 コンテナ仮想化技術：コンピュータ仮想化技術の一種で，1つのホストOSの上にコンテナと呼ばれる専用領域を作り，その中で必要なアプリケーションソフトを動かす方式のこと．
＊3 ワークロード：CPU使用率などのシステムの負荷の大きさを表す指標．特にパブリッククラウドの分野では，クラウド上で実行されるOSやアプリケーションコードなどを含めたシステム自体を表すこともある．本稿では後者の意味で用いる．

テナの管理やオートスケール*6，自動復旧の機能など多大な恩恵を受けることができる．Kubernetesは，すでにクラウドの分野においてはデファクトスタンダードとなっており，AWS（Amazon Web Services）*7においてはEKS（Elastic Kubernetes Service）*8，Microsoft Azure*9においてはAKS（Azure Kubernetes Service）*10，GCP（Google Cloud Platform）*11においてはGKE（Google Kubernetes Engine）*12というサービス名でマネージドサービス*13が提供されており，さらにプライベートクラウド*14ベンダであるVMware，Red Hatなど多くの企業からサービスが提供されている．

ドコモ内でも2017年ごろから複数Kubernetesクラスタを利用するプロジェクトが出現し始めた．一方で複数クラスタ*15の利用に伴い，運用者は運用コストの増加やクラスタ間でのリソース使用率のばらつきの問題に直面していた．そのため，複数Kubernetesクラスタの管理を目的としたオーケストレーションツールとして，クラウドオーケストレータ（以下，CO）をオープンソースソフトウェアとしてDrupal*16上で開発した．本稿では，COにおける特長や社内の活用事例について解説する．

2. Kubernetes

2.1 概　要

Dockerは単一サーバ上で稼働するには便利なツールである一方，複数サーバで構成される大規模環境では多くの問題を抱えていた．このような大規模環境向けのコンテナ管理を目的としたコンテナオーケストレーションツールであるKubernetesが，世界的にデファクトスタンダードとして利用されている．Kubernetesは，元Googleエンジニアにより開発されたBorgが起源とされている．BorgはGoogle社内で活用されたコンテナオーケストレーションのノウハウを蓄積後，その中の主要機能がKubernetesから提供されている．Kubernetesは，その後2014年にKubernetesプロジェクトにおいてオープンソース化され，2016年にCNCF（Cloud Native Computing Foundation）*17に移管されて，コミュニティベースの開発が行われている．

2.2 アーキテクチャ

図1のとおり，Kubernetesはマスターノードとワーカーノードの2つから構成される．クラスタ内に存在するアプリケーションやその各種設定情報はリソースオブジェクトという単位で管理される．リソースオブジェクトはマニフェストファイル*18により定義され，ユーザはその新規作成や更新時に，このファイルにより操作する．Kubernetesを構成する要素は，以下のとおりである．

(1)マスターノード（Master node）

クラスタ全体を管理するノードで，ワーカーノード管理やポッド管理の役割を担う．クラスタのデプロイ時にデフォルトとして，コントロールプレーンが設定される．コントロールプレーンとは，クラスタを制御するコンポーネントを制御し，クラスタ内の状態や構成を管理するコンポーネントである．利用者は後述のkubectlを利用し，コントロールプレーンが提供するAPI（Application Programming Interface）*19サーバにアクセスし，クラスタ全体を操作する．

(2)ワーカーノード（Worker node）

アプリケーションコンテナを格納するポッドをホストするノードである．後述のKubelet，Kube-proxyなどのコンポーネントを有する．

(3)ポッド（Pod）

Kubernetesアプリケーションにおける実行単位

＊4　デプロイ：アプリケーションをそれらの実行環境に配置して展開すること．

＊5　フレームワーク：ある領域のソフトウェアに必要とされる汎用的な機能や基本的な制御構造をまとめたもの．ライブラリでは，開発者が個別の機能を呼び出す形となるが，フレームワークでは，全体を制御するのはフレームワーク側のコードで，そこから開発者が個別に追加した機能を呼び出す形となる．

＊6　オートスケール：ネットワークトラフィックやCPUの利用量など，その時々の処理に必要なリソース量に応じてオンデマンドで自動的に仮想サーバを増減する仕組み．

＊7　AWS：Amazon Web Services社が提供するクラウドコンピューティングサービス．

＊8　EKS：AWSにおけるマネージドKubernetesサービス．

＊9　Microsoft Azure：Microsoft社が提供するクラウドコンピューティングサービス．

＊10　AKS：Microsoft Azureにおけるマネージドkubernetesサービス．

＊11　GCP：Google社が提供するクラウドコンピューティングサービス．

＊12　GKE：GCPにおけるマネージドKubernetesサービス．

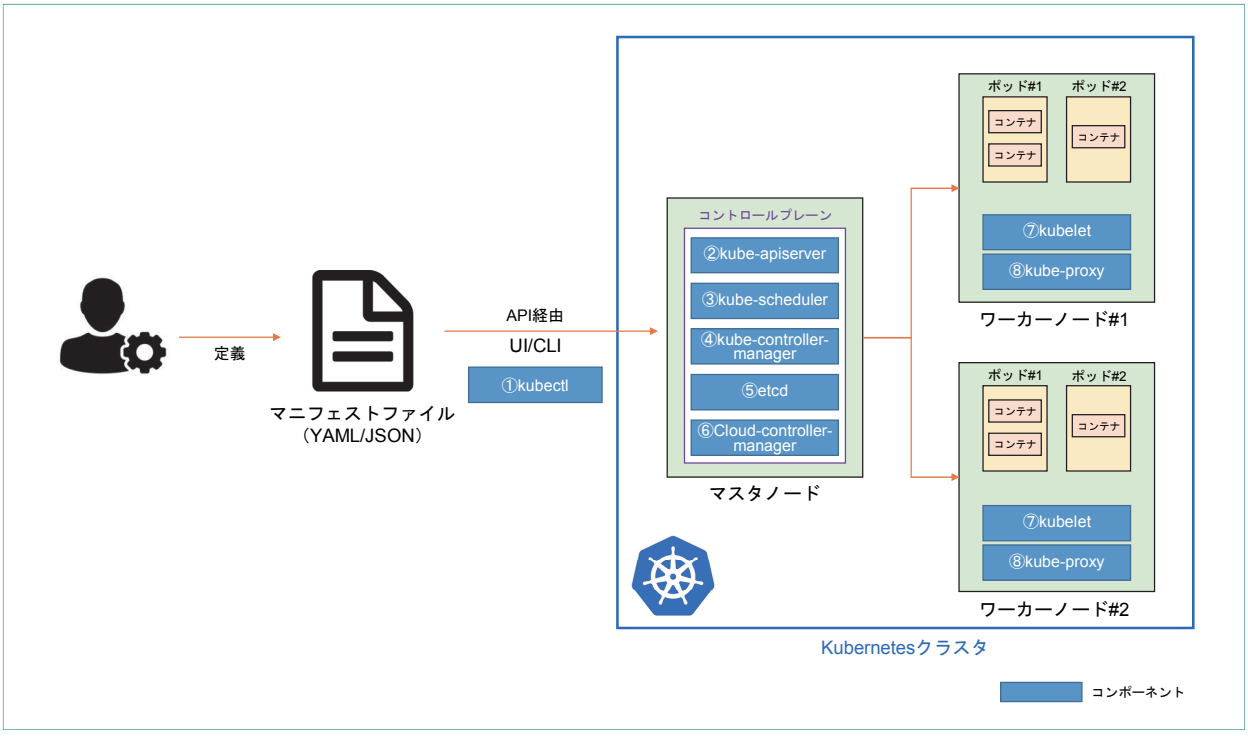

図1　Kubernetesのアーキテクチャ

で，アプリケーションのコンテナ，ストレージ，ネットワークIDやIPアドレスなどのネットワーク情報，実行方法を管理するオプションなどをカプセル化したものである．複数コンテナを格納することも可能である．

⑷マニフェストファイル（Manifest file）

リソースの構成が記載されたファイルで，ファイル形式はJSON（JavaScript Object Notation）[*20]やYAML（YAML Ain't Markup Language）[*21]を採用している．このファイルをAPI経由で宣言することで，クラスタ内のリソースの操作を可能とする．

2.3　コンポーネント

図1にあるコンポーネントについて以下に解説する．

①kubectl

ユーザが利用するコマンドで，kube-apiserverに対してリクエストを送り，リソースの作成／更新／削除などを行う．

②kube-apiserver

Kubernetesクラスタにおける APIを外部に提供するフロントエンドの役割を担う．

③kube-scheduler

新規ポッドがワーカーノードに割り当てられているかどうかを監視し，もし割り当てられていない場合，ポッドを実行する役割をもつ．スケジューリングの決定は，リソース使用率，ハードウェア／ソフトウェア／ポリシーによる制約など，いくつかの点を考慮して行われる．

＊13 マネージドサービス：クラウドサービスのうち，リソースのプロビジョニングや運用の大半をクラウド事業者の責任で実施しているサービス．クラウドコンピューティングサービスのうち，特にPaaSやSaaSを指す．

＊14 プライベートクラウド：企業・組織が社内でクラウド環境を構築し，社内の各部署やグループ会社に提供するクラウド形態を指す．一方利用・提供する人の範囲が限定されず，オープンに提供されるクラウドサービスはパブリッククラウドと呼ばれる．

＊15 クラスタ：複数のサーバを1つのサーバ群としてグループ化したもの．

＊16 Drupal：WordpressやJoomlaと同様のオープンソースのコンテンツ管理システム．

＊17 CNCF：CNCFはLinux Foundationのプロジェクトで，コンテナ技術の発展と，その進化に関連するテクノロジ業界の連携を支援するために2015年に創設された．

＊18 マニフェストファイル：アプリケーションが利用する機能などを宣言した設定ファイル．すべてのKubernetesのリソースはそれぞれの用途に合わせたマニフェストファイルを作成する必要がある．

④kube-controller-manager

ワーカーノードやポッドの状態をkube-apiserver経由で管理するコンポーネントである．ノードがダウンした場合の対応や，ポッドのレプリケーション*22管理などを行う．

⑤etcd

Kubernetesのすべてのクラスタ情報を保管するキーバリューストア*23である．また，Kubernetesのデータストア*24として使用することも可能である．

⑥cloud-controller-manager

ノード，ルーティング，ストレージなどクラウド事業者固有のオブジェクトを管理する．

⑦kubelet

ワーカーノード内で動くエージェントで，各ポッドの動作を保証する．また，マニフェストファイルで定義された内容とコンテナの設定情報が合致していることの監視，ノードやコンテナの実行環境*25を管理する．

⑧kube-proxy

コンテナ間の通信制御（ルーティング）を行う．

2.4 提供機能

Kubernetesは，複数サーバでコンテナ管理するだけではなく，コンテナのオートスケーリングや障害時の自動復旧*26など，運用における骨の折れる点の数々をサポートする．いくつか重要な点を解説する．

⑴ネットワークの負荷分散

一部のコンテナへのアクセスが集中している場合，その他のコンテナへトラフィックを分散することでコンテナの状態を安定化させる．

⑵ローリングアップデート／ロールバック

ユーザは，マニフェストファイル上で，コンテナの状態を変更するだけで，Kubernetesがその状態に変更してくれる（ローリングアップデート）．仮にアプリケーションの更新に失敗しても，マニフェストファイルを以前の状態に戻すだけで簡単に以前の状態に戻すことも可能である（ロールバック）．

⑶自動ピッキング

タスクごとに実行するノードやリソース使用量，優先度を定義できる．

⑷自動修復

障害などでコンテナが停止しても，自動でその状態を検出し，再起動を行う．

2.5 ドコモのKubernetesにおける課題

ドコモのあるプロジェクトでは，すでに2017年ごろから複数の大規模Kubernetesクラスタを商用システム上で利用していた．しかし，Kubernetesは自身のクラスタのみ管理する機能しかなく，運用者は複数のKubernetesクラスタを個別に管理しなければならなかった．例えば，複数クラスタにまたがるコンテナの状況を知りたいとき，それぞれのクラスタにアクセスし，状況をチェックする必要があった．また，機能別にクラスタを分割していたことから，クラスタ間のリソース使用率に大きな偏りが発生していた．例えば，あるクラスタではリソースのひっ迫から一時的にインスタンス*27を追加しなければならない状態である一方，他のクラスタではリソース使用に空きが見られるといったことがあった．仮に，Kubernetesクラスタ間でタスクを分散するロードバランシング*28やジョブスケジュール機能が存在すれば，このような課題が発生することはなかった．

この課題が顕在化した2018年ごろには，これらの課題を解決するオープンソースソフトウェアや外部

＊19 API：ソフトウェアコンポーネント同士が互いに情報をやりとりするのに使用するインタフェースの仕様．
＊20 JSON：JavaScriptのオブジェクト記述に基づくデータ記述言語．
＊21 YAML：XMLやJSONと同様，データ構造を記述する記法，処理フォーマット．
＊22 レプリケーション：データベースにおいて，データを他のサーバに複製して冗長性やバックアップを実現する仕組み．
＊23 キーバリューストア：従来のリレーショナルデータベースとは異なり，レコード（データ）をキーと値の組み合わせで管理するストレージシステム．
＊24 データストア：データを保存するシステム．
＊25 実行環境：ソフトウェア開発用のライブラリやテスト環境など，開発に必要となるシステムを含まず，実行処理だけを提供するシステム環境．
＊26 自動復旧：システムに障害が発生した際に自動的に冗長化された待機システムに切り替える仕組み．

ツールは存在しなかったため，我々でCOを開発した．なお，2019年ごろから，Kubernetesをさかんに使用していた先進的な企業（UberやRancherなど）から類似製品が提供されている．

3. COとは

3.1 開発経緯

ドコモのCOプロジェクトは，Drupalというオープンソース上で2018年初頭に開始された．当初の目的は，主にAWSを中心とした複数アカウントの管理であった．ドコモ内には，多数のアカウントが乱立し，1プロジェクトでも開発，QA（Quality Assurance）[*29]，商用と複数アカウントをもつことは珍しくない．AWSのサイトは，基本的にリージョン[*30]ごとにサービスが分割されているため，同一サービスであっても，複数リージョンにまたがった場合，それらを同一ページにてチェックすることができない．そのため，管理者の知らないところで開発者がさまざまなリージョンにインスタンスを起動させ，それらが停止／削除されないまま，管理者が請求書情報[*31]を見て，その存在に気付くということが多々発生していた．COは，こういった問題を未然に防ぐため，登録されたすべてのアカウント情報配下にあるリソース一覧を同一ページにて可視化する機能，クラスタのリソース使用率／時間ベースでの自動停止機能，シングルサインオン[*32]の機能などを提供する．ドコモ社内でKubernetesの問題が発生した際，これらのAWS向けに使用していた機能を基にCOを拡張し，Kubernetesクラスタに適用し始めたのが2018年末ごろである．

3.2 アーキテクチャ＆コンポーネント

COでは，ポータル機能などのユーザインタフェース部，ユーザ管理やクラスタ管理などの各種機能部，クラウドやKubernetesクラスタのコネクタ部の大きく3つから構成される．

図2に示すとおり，COではポータル機能を提供し

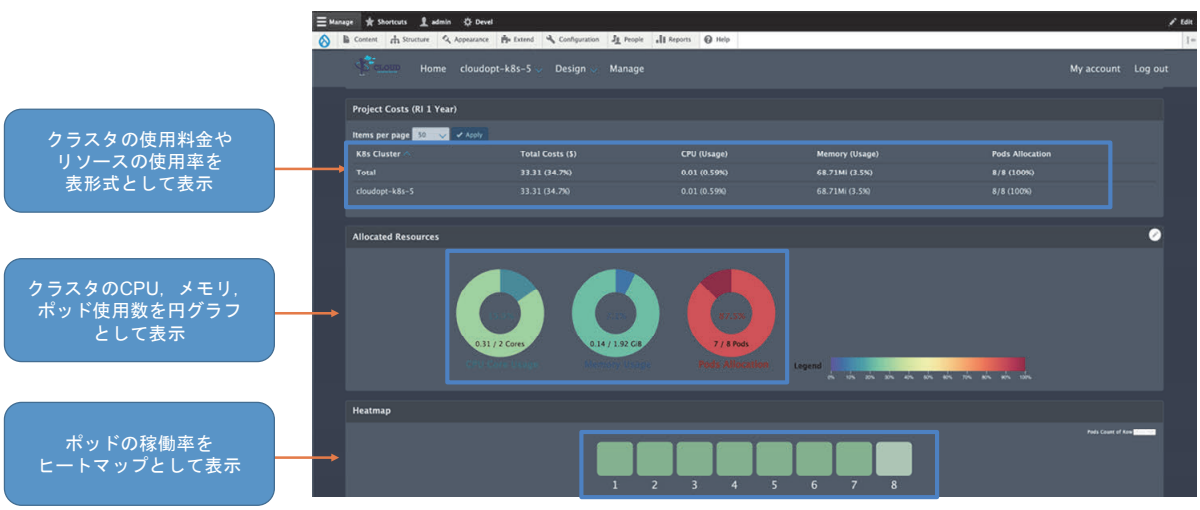

図2　COのポータル画面

＊27 インスタンス：クラウドコンピューティングにおけるオンデマンドで提供される仮想サーバ．ある処理が発生したときのみ仮想サーバが起動し終了するなど，仮想サーバの起動から終了までのライフサイクルは散発的である．
＊28 ロードバランシング：高負荷の処理を複数のサーバで分散して処理する仕組み．
＊29 QA：ソフトウェア開発において成果物の品質確保をする行為または品質保証をすること．
＊30 リージョン：クラウドサービスを提供するためのデータセンタが配置されている地域．
＊31 請求書情報：クラウド使用量を基にした請求書情報．
＊32 シングルサインオン：1つのアカウントで複数サービスにログインできること．あるユーザがシステムへの認証を1度行うことで，そのユーザに権限が与えられているすべての機能が利用できるようになること．

ている．画面上から，現在利用されているクラウド情報やKubernetesクラスタを一目で理解できる．また，クラスタ情報に行くと，現在の情報の状況を表，円グラフで可視化することができる．表示内容については，管理画面からカスタマイズが可能であり，ユーザが自由に変更可能である．また，可視化機能に加え，クラスタ上で動かしたいタスクのデプロイやデプロイ先のクラスタの選択など多くの機能を，このポータル機能を通して提供している．なお，DrupalでREST API（REpresentational State Transfer）*33機能を提供しているため，ポータル上の機能をAPI経由で実行することも可能である．

図3に示すとおり，ユーザ管理，クラスタ管理，ジョブ管理，ジョブスケジューリングなどCOでは多くの機能を提供している．Drupalが提供するロール*34管理機能は，任意の機能やサービスに対して，基本動作であるCRUD（Create，Read，Update and Delete）*35機能を定義することができる．この機能を拡張し，COではKubernetesクラスタにあるすべてのリソースオブジェクトに対するCRUD操作を規定することができる．このロール管理により，同一クラスタ上に異なるユーザがそれぞれのタスクをデプロイしても，各ユーザに自身のタスクのみ操作可能なロールを付与，つまり必要最低限のロールを付与していれば，クラスタ上のユーザ間の干渉を防ぐことができる．この考えは，後述するコスト最適化を支える基本的な機能である．

コネクタ部では各種クラウドやKubernetesクラ

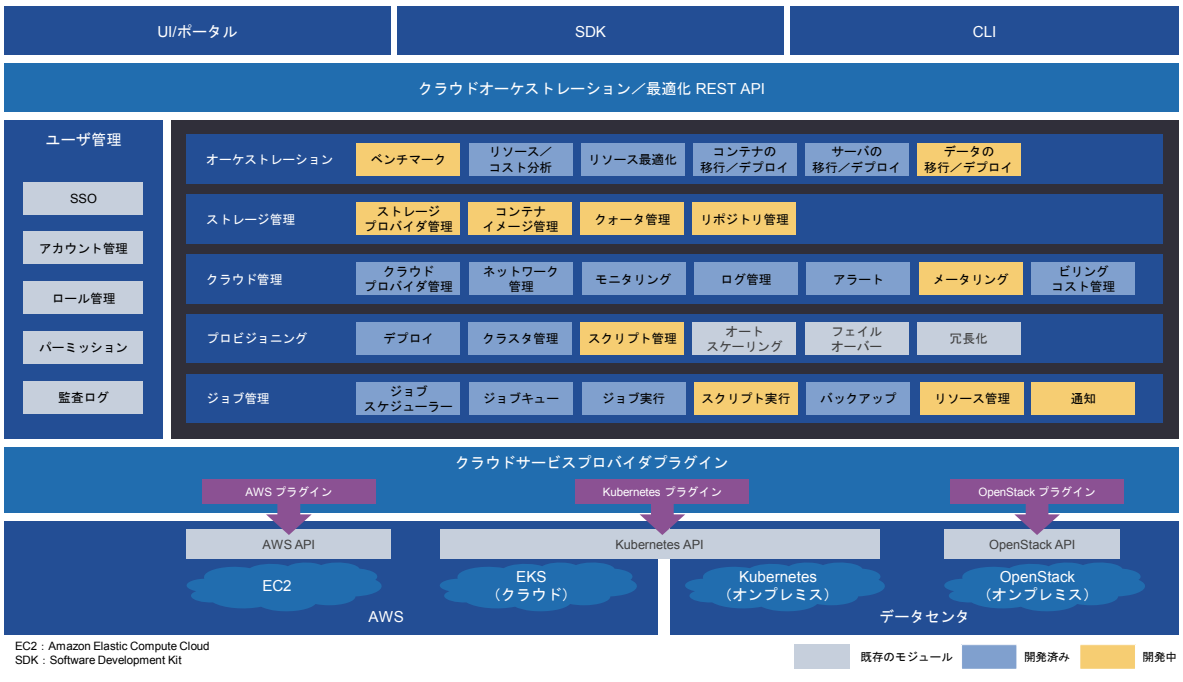

図3　クラウドオーケストレーション・最適化のコンポーネント

＊33 REST API：ウェブで用いられるソフトウェアアーキテクチャのスタイルの1つ．

＊34 ロール：ある特定の権限をユーザに付与するときのグループ（役割）のこと．

＊35 CRUD：システムやサービスを提供するソフトウェアにおいて，基本的な操作であるCreate／Read／Update／Delete（新規作成／読込み／書込み・更新／削除）の頭文字をとったもの．

スタが提供するAPIを使用しており，我々はそれらをモジュール化し，提供している．ユーザは，このモジュール機能をONにし，定期的に情報の取得を開始する．

3.3 主要機能

COにおける主要機能を解説する．

⑴シングルサインオン

Drupalのシングルサインオン機能を活用し，COは各種クラウドと連携ができる．また，Kubernetesクラスタはユーザ認証機能がないため，CO自体がユーザ認証機能を提供している．

⑵リソース最適化

COが複数のKubernetesクラスタを管理している場合，クラスタ間のロードバランシングを自動的に行う機能である．例えば，任意のコンテナのデプロイ時に，その時のクラスタのリソース使用率を基に，最も空いているクラスタを自動的に選択し，デプロイを行う．あらかじめ，ユーザ側でデプロイ先のクラスタを選択することも可能である．

⑶スケジューリング

時間とリソースベースのスケジューリング機能を有する．例えば，利用者が深夜帯などの特定時間帯にバッチ処理[*36]を行いたい場合に，タスクのデプロイ時間と終了時間をあらかじめ設定することができる．また，リソースが空いている時にのみ，優先度の低いタスクをデプロイすることもできる．

⑷コスト算出

マスターノードとワーカーノードのインスタンス費用を合計値とし，リソース使用率（メモリ，CPU，ポッド数）から算出するロジックを独自で規定している．これにより，Namespace[*37]単位でのコスト算出が可能となる．

⑸マルチクラウド管理

AWS，OpenStack[*38]，VMwareなどのパブリック／プライベートクラウドと同様に，Kubernetesクラスタも統一的に管理することが可能である．利用者は異なるアカウントやクラスタであっても，統一してリソース利用状況を確認することができる．

3.4 COの利用方法

COは，Drupalでオープンソースとして公開されているため，誰でも利用することができる［2］．

4. ドコモ事例

4.1 複数クラスタの管理と空きリソースの効率化

⑴複数クラスタの管理

COはドコモの商用システムで活用されている．このシステムの特長は，複数EKSを利用しており，1つのクラスタは100台以上のノードをもっている．**図4**に示すとおり，CO導入前，管理者は各クラスタの管理画面からクラスタやポッドの状態のチェックやCLI（Command Line Interface）[*39]経由でのコンテナのデプロイの実行など，その運用が煩雑化していた．

一方，COの導入後，COがすべてのクラスタからメトリック[*40]を取得するため，管理者は各クラスタへの確認が不要となった．また，管理画面上にてデプロイしたいコンテナとデプロイ先のKubernetesクラスタの情報を登録することで，COが自動的にデプロイを実施してくれる．仮にデプロイ先が複数Kubernetesクラスタであっても，クラスタを複数選択しておけば，自動的に選択されたクラスタ群にコンテナのデプロイを実施し，それぞれからメトリックを取得する．なお前述したように，クラスタ

＊36 バッチ処理：ユーザとの対話なしにデータをまとめて自動処理すること．
＊37 Namespace：Kubernetesにおいて，複数の異なるリソースを1つにまとめるための仮想的なグループの単位．
＊38 OpenStack：サーバ仮想化技術を用いて，一台の物理サーバを仮想的に複数のサーバのように動作させ，仮想サーバをユーザが利用するクラウドサービスごとに割り当てるクラウド基盤のソフトウェア．オープンソースソフトウェアとして提供されている．
＊39 CLI：コンピュータやソフトウェアへの指示をすべて文字によって行う方式．
＊40 メトリック：ある期間におけるCPUの利用率，メモリ使用量，ポッドの数といった定量的なデータ．

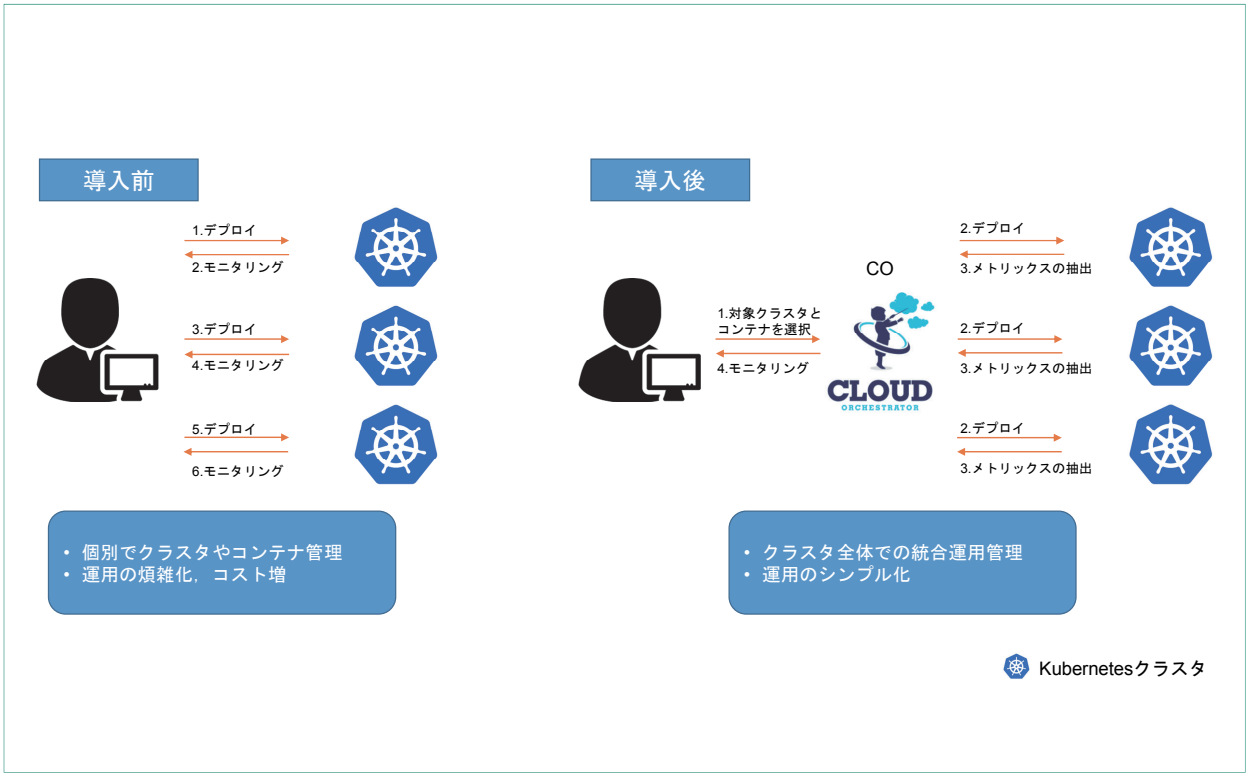

図4　CO導入における変化

選択において，ユーザは特定のクラスタを選択せず，COが自動的に空いているクラスタを選択するリソース最適化機能を使用することも可能である．

⑵空きリソースの効率化

図5に示すとおり，COは時間とリソースベースのスケジューリング機能を有している．ドコモの商用システムでは，システム全体のリソース使用率がユーザの利用に比例する傾向があった．具体的には，ユーザが良く利用する日中帯は80%以上と高い使用率になる一方で，深夜帯になるとその使用率は20%近くまでに落ち込むといった具合である．空いたリソースを利用するため，管理者はCO上で，デプロイしたいコンテナとスケジューリング方法と必要なパラメータを登録しておけば，COは登録されたスケジューリング方法に従って，コンテナのデプロイを実施する．このケースにおけるコンテナは，リアルタイム性を要求せず，他のコンテナより優先度が低いことが多く，クラスタのリソース使用率のバースト＊41時には，優先的にこれらのコンテナが削除される仕組みとなっている．この商用システムでは，時間ベースのスケジューリング機能により，深夜帯においてシステムログの集計や機械学習モデルの更新を実施している．またリソースベースのスケジューリング機能により，あるクラスタのリソースがひっ迫した場合，一時的にリソースを追加するのではなく，そのタスクの一部を他のクラスタに自動的に分散することで，システム全体のリソースの均等化を図っている．

＊41　バースト：ネットワークトラフィックが一時的に増大するなどして，瞬時的に複数の信号が，装置内の一定箇所に集中すること．

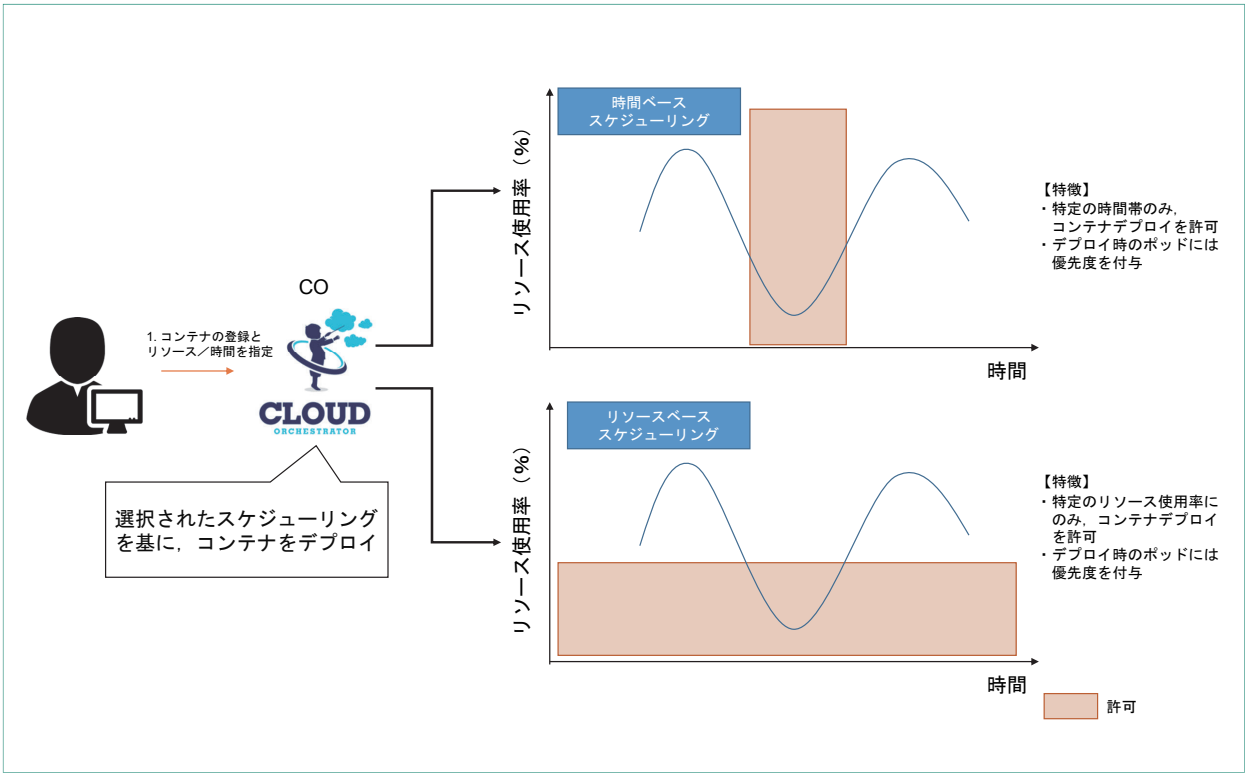

図5　スケジューリング機能

4.2　アプリケーションとシステムレイヤの分離

図6(a)に示すとおり，ドコモ内の各プロジェクトはそれぞれがAWSアカウントを作成し，その中でアプリケーション開発やシステム構築を行っている．この形式では全プロジェクトでAWSの知識の獲得，アカウントやシステム環境の管理，セキュリティポリシーの理解が必要で，サービス展開までに膨大な稼働が掛かってしまう．さらに，セキュリティ専門家の不在によるセキュリティリスクの増大，全体最適ではないことによるクラウドコスト増といったさまざまな問題が発生する．

この課題を解決するために，COを境にアプリケーションとシステムレイヤで分離することを考えている（図6(b)）．現在は構想段階であるが，この方式にすれば，アプリケーション開発者はサービスのコンテナ化を条件に，COより下のレイヤの管理や面倒なセキュリティ管理が不要になり，サービス開発に注力することが可能となる．一方で，システム管理者は，コンテナ化によりアプリケーションの中身を意識する必要がなくなる．システム管理者がクラスタを集中管理することにより，全体リソースの効率化やセキュリティリスクの低減が図られ，最終的には全社的なクラウドコストの削減が期待される．

ただし，この方式では，複数のサービスが同一のクラスタで動くことになるため，重い処理を行うコンテナは他のサービスに影響を及ぼす可能性がある．また，アプリケーション開発者はクラウド費用を意

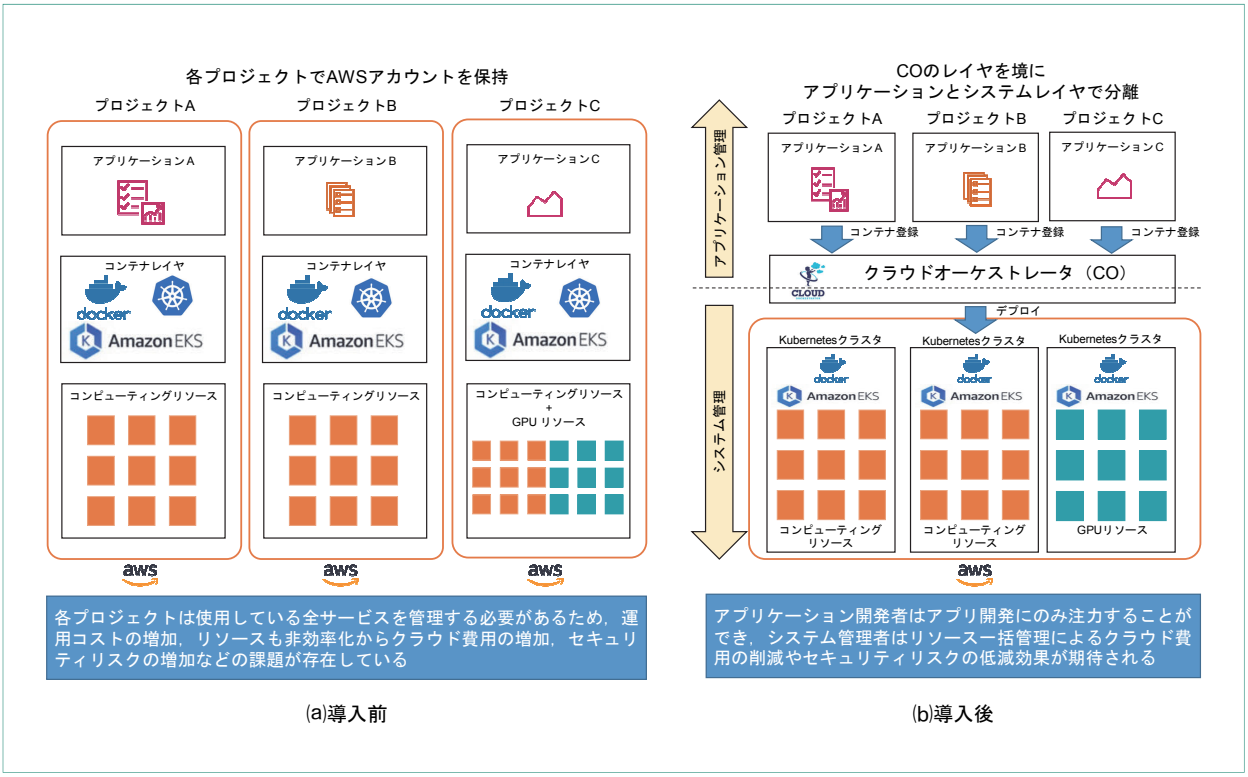

図6　COによるアプリとシステムレイヤの分離

識することがなくなるので，非効率な処理を実行する可能性もある．この点は，KubernetesのResource quota[*42]やLimit range[*43]の機能を使用することに加えて，COではKubernetesクラスタから得られるCPU，メモリ，実行ポッド数などのリソース使用率を基にしたコスト算出機能を利用して，アプリケーション開発者に利用料を通知する．これにより，アプリケーション開発者にコスト意識をもたせることができる．

現在，サービスイノベーション部と連携して，上記方式の検証を進めている段階である．この方式を満たすためには，CO側の機能が不足しているため，引き続き，これらの更新をしていく予定である．

5. あとがき

本稿では，複数Kubernetesを管理するCOについて解説した．ドコモ社内で蓄積された知見を活かして，クラウドの利用をさらに効率化できるよう，今後もCOを改善したいと考えている．

文　献

[1] Kubernetesホームページ．
https://kubernetes.io/
[2] Drupal：“Cloud.”
https://www.drupal.org/project/cloud

＊42 Resource quota：コンテナへのリソースの割当て量の設定値．
＊43 Limit range：コンテナからのリソースリクエストを制限する最小値と最大値の範囲を決める設定値．