Technology Reports (Special Articles) Kubernetes AWS Containers
Special Articles on Use of Public Clouds

Managing Multiple Kubernetes Clusters with a Cloud Orchestrator

DOCOMO Innovations, Inc. Masato Takada Yas Naoi

Kubernetes was originally created by Google developers, but is now developed as an open-source project by the CNCF. It is a system that enables users to operate containerized applications efficiently in any environment, and is currently used by many companies including industry leaders. Around 2018, a growing number of companies started using multiple Kubernetes clusters in individual projects, but at the time there were no tools for managing multiple Kubernetes clusters. Therefore, DOCOMO Innovations, Inc. developed an open-source Cloud Orchestrator to facilitate the centralized management of multiple Kubernetes clusters. This software is currently being used on DOCOMO's internal commercial systems, an example of which is described in this article.

1. Introduction

With the advent of Docker*1 in 2013, applications and system environments could be logically separated by using container virtualization*2 technology to virtualize containers at the OS level. This made it possible to deploy*3 and update containers in workloads*4 even faster than before. Docker, on the other hand, has had issues related to container management, scalability, and automatic recovery. Users have had to solve these issues before using each environment, but this can involve a lot of work.

Kubernetes [1], which has recently gathered a broad community of users, is an open-source solution

*1 Docker: Container-type virtualization software. A registered trademark of Docker Inc.
*2 Container virtualization: A computer virtualization technique where dedicated areas called containers are created on a single host OS, and the necessary application software is run within these containers.
*3 Deploy: Installing applications by placing them in their execution environments.

that aims to solve this problem. Kubernetes takes its name from the Greek word for "helmsman," reflecting its purpose as a tool for managing and automating the operation of containers. It provides developers with great benefits such as management of multiple containers, autoscale*5, and automatic recovery functions without having to prepare a framework*6 for containers on their own. Kubernetes has already become a de facto standard in the field of cloud computing. For example, Amazon Web Services (AWS)*7 uses Elastic Kubernetes Service (EKS)*8, Microsoft Azure*9 uses Azure Kubernetes Service (AKS)*10, and Google Cloud Platform (GCP)*11 uses Google Kubernetes Engine (GKE)*12. In addition to these managed services*13, there are also many other services provided by private cloud*14 vendors such as VMware and Red Hat.

At DOCOMO, some projects started deploying multiple Kubernetes clusters on their own systems from around 2017. However, when using multiple clusters*15, system operators were faced with issues of increased operating costs and variations in resource utilization between clusters. For this reason, we developed a Cloud Orchestrator (CO) as an open-source orchestration tool for managing multiple Kubernetes clusters on Drupal*16. This article describes the features of CO and presents some examples of its use within the company.

2. Kubernetes

2.1 Overview

While Docker is a useful tool for running on a single server, it suffers from a number of issues in large-scale environments consisting of multiple servers. Kubernetes is a container orchestration tool designed to manage containers for large-scale environments of this sort, and has become the de facto standard worldwide. Kubernetes evolved from Borg, which was originally developed by Google engineers. Having accumulated the container orchestration know-how that was used by Google, Borg became the progenitor of Kubernetes. In 2014, Kubernetes was open sourced in the Kubernetes project, which was transferred to the Cloud Native Computing Foundation (CNCF)*17 in 2016 for community-based development.

2.2 Architecture

As shown in **Figure 1**, Kubernetes consists of two types of nodes: a master node and worker nodes. In a cluster, applications and their setting values are managed in units called resource objects. A resource object is defined by a manifest file,*18 which the user works with when creating or updating the resource object. The constituent elements of Kubernetes are as follows:

1) Master Node

A node that manages an entire cluster and performs the roles of worker node management and pod management. A control plane is set as the default when deploying a cluster. The control plane is a component that controls the components that control a cluster and manages the cluster's internal state and configuration. The user operates the entire cluster by using a command line tool called kubectl (described later) to access the Application

*4 Workload: An indicator of the size of a system's load, such as the CPU utilization rate. In particular, in a public cloud environment, the workload may represent the system itself, including the OS and application code running on the cloud. In this paper, we use the term in this latter sense.

*5 Autoscale: A system that automatically adjusts the number of virtual servers on demand according to the quantity of resources required for processing at any given time, such as network traffic and CPU usage.

*6 Framework: Software that encompasses functionality and control structures generally required for software in a given domain. In contrast to a library in which the developer calls individual functions, code in the framework handles overall control and calls individual functions added by the developer.

*7 AWS: A cloud computing service provided by Amazon Web Services, Inc.

*8 EKS: A managed Kubernetes service in AWS.

*9 Microsoft Azure: A cloud computing service provided by Microsoft Corporation.

*10 AKS: A managed Kubernetes service in Microsoft Azure.

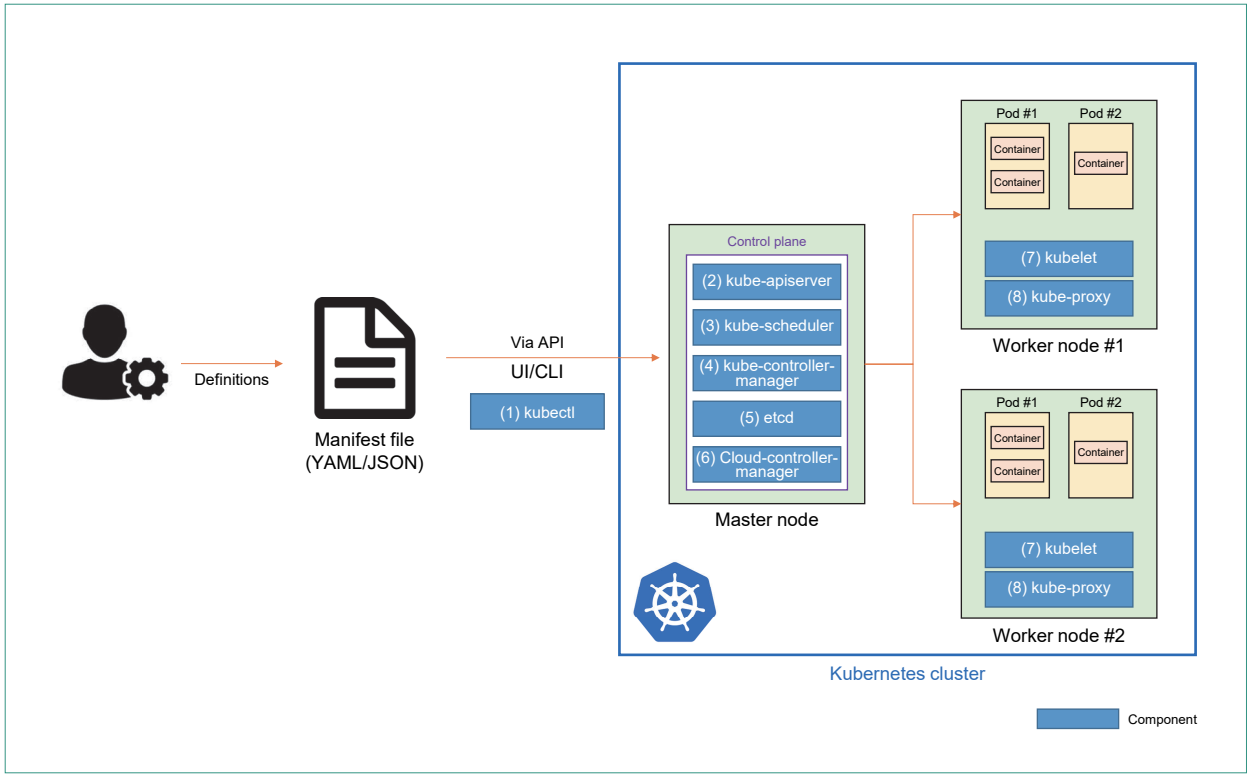

Figure 1 Kubernetes architecture

Programming Interface (API)*19 server provided by the control plane.

2) Worker Node

A node hosting a pod that stores the application containers. This includes components such as kubelet and kubeproxy, which are described later.

3) Pod

An execution unit in a Kubernetes application that encapsulates the application's container, storage, network information (network ID, IP address, etc.) and options for managing execution methods. It can also store multiple containers.

4) Manifest File

A file that describes the resource configuration, using JavaScript Object Notation (JSON)*20 or YAML (a recursive acronym for "YAML Ain't Markup Language")*21 as the file format. By declaring this file via the API, it is possible to manipulate resources in the cluster.

2.3 Components

The components shown in Fig. 1 are described below.

(1) kubectl

A command used by the user to send requests to kube-apiserver in order to create, update and delete resources.

*11 GCP: A cloud computing service provided by Google LLC.
*12 GKE: A managed Kubernetes service in GCP.
*13 Managed service: Cloud services whose resource provisioning, operation, etc. are mostly the responsibility of the cloud operator. Among cloud computing services, these are referred to as PaaS and SaaS, for example.
*14 Private cloud: Refers to an in-house cloud system configured in a corporation or organization, and provided to various in-house divisions or group companies. In contrast, open cloud services that do not restrict their services to certain users are called "public cloud" services.
*15 Cluster: A grouping of multiple servers as a single server group.
*16 Drupal: An open-source content management system, similar to WordPress and Joomla.
*17 CNCF: A project of the Linux Foundation that was created in 2015 to support the development of container technology and collaboration with the technology industry on the advancement of this technology.
*18 Manifest file: A configuration file that declares the functions and other items used by an application. A manifest file must be prepared for each application of every Kubernetes resource.

(2) kube-apiserver

This is responsible for the front end that provides the external API of a Kubernetes cluster.

(3) kube-scheduler

Monitors to check if a new pod has been assigned to a worker node. If not, it assumes responsibility for running the pod. Scheduling decisions are made in consideration of several factors, such as resource utilization and hardware/software/policy constraints.

(4) kube-controller-manager

A component that manages the status of worker nodes and pods via kube-apiserver. It takes responsive action when a node goes down, and manages pod replication.*22

(5) etcd

A key-value store*23 that stores all the Kubernetes cluster information. It can also be used as a Kubernetes data store*24.

(6) cloud-controller-manager

Manages objects specific to a cloud operator, such as nodes, routing, and storage.

(7) kubelet

An agent that runs inside worker nodes and guarantees the operation of each pod. It also monitors the content defined in the manifest file to make sure they match the container settings, and manages the execution environment*25 of nodes and containers.

(8) kube-proxy

Performs communication control (routing) between containers.

2.4 Supported Functions

Kubernetes not only manages containers on multiple servers, but also supports a number of painstaking aspects of operation, such as automatic scaling of containers and automatic recovery*26 in the event of a failure. Some key features are discussed below.

1) Network Load Balancing

When accesses are concentrated on some of the containers, the state of these containers is stabilized by distributing traffic to other containers.

2) Rolling Updates/rollbacks

To apply updates, the user only needs to change the state of the container in the manifest file. Kubernetes will then take care of applying this updated state (rolling update). If an application update fails, it can be easily reverted to its previous state by simply restoring the manifest file to its previous state (rollback).

3) Automatic Picking

For each task, it is possible to define which nodes should be executed, which resources should be used, and the priority for each task.

4) Automatic Repairs

If a container is stopped due to a failure or some other issue, the system automatically detects this state and restarts it.

2.5 DOCOMO's Challenges with Kubernetes

One project at DOCOMO has already been using several large Kubernetes clusters on commercial systems since around 2017. However, Kubernetes is only able to manage its own cluster, so the

*19 API: A specification describing the interface whereby software components can exchange information with each other.

*20 JSON: A data description language based on JavaScript object notation.

*21 YAML: A notation and processing format for describing data structures, similar to XML and JSON.

*22 Replication: In a database, a mechanism for providing redundancy and creating backups by replicating data to other servers.

*23 Key-value store: A storage system that manages records (data) as combinations of keys and values, unlike a conventional relational database.

*24 Data store: A system that stores data.

*25 Execution environment: A system environment that only supports execution processing and lacks systems required for software development (e.g., libraries and test environments).

*26 Automatic recovery: A system that automatically switches to a redundant standby system in the event of a system failure.

system operators had to manage multiple Kubernetes clusters individually. For example, to ascertain the status of containers across multiple clusters, it was necessary to access each individual cluster to check its status. Furthermore, since the clusters were divided according to their function, there was a large disparity in resource utilization between clusters. For example, while one cluster had to temporarily add instances*27 due to a shortage of resources, there were other clusters with resources that were sitting idle. These issues would not have arisen if load balancing*28 and job scheduling functions had existed so that tasks could be distributed among Kubernetes clusters.

When this issue came to light in around 2018, there was no open-source software or external tools capable of solving it. For this reason, we developed CO. From around 2019, other similar products became available from leading companies (such as Uber and Rancher) that have been using Kubernetes intensively.

3. What is CO?

3.1 Development Background

DOCOMO's CO project was launched in early 2018 on an open-source platform called Drupal. Our initial goal was to manage multiple accounts, primarily AWS. DOCOMO has a large number of accounts, and it is not uncommon for a single project to have multiple accounts for different purposes such as development, Quality Assurance (QA)*29 and commerce. The AWS website is basically divided into services by region*30, so even parts of the service cannot be checked on the same page if the service spans multiple regions. As a result, there were many cases where developers launched instances in various regions without the administrator's knowledge, and then failed to stop or delete these instances, which only came to the administrator's attention when viewing the billing information*31. In order to prevent such problems, CO provides a function for visualizing the list of resources under all registered account information on the same page, a function for automatically stopping a cluster based on resource utilization and time constraints, and a single sign-on*32 function. When we ran into Kubernetes issues within DOCOMO, we extended CO based on the features we were using for AWS and started applying them to Kubernetes clusters around the end of 2018.

3.2 Architecture & Components

CO consists of three main parts: a user interface (including a portal function), various functional parts for purposes such as user management and cluster management, and a connector part for cloud and Kubernetes clusters.

As shown in **Figure 2**, CO provides portal functions. From this screen, it is possible to see at a glance the cloud information and Kubernetes clusters that are currently in use. In addition, by bringing up the cluster information, it is possible to visualize the current status of information in the form of tables and pie charts. The display can be customized from the management screen and can be freely changed by the user. In addition to this visualization function, the portal functions include many

*27 Instance: A virtual server that is made available on demand in cloud computing. A virtual server has a sporadic life cycle from start to finish. For example, it might start and finish only when a certain process takes place.
*28 Load balancing: A mechanism for distributing high-load processes across multiple servers.
*29 QA: In software development, the act of ensuring the quality of deliverables or providing quality assurance.
*30 Region: The region in which a data center providing cloud services is located.
*31 Billing information: Information about charges accrued through the use of cloud services.
*32 Single sign-on: The ability to log in to multiple services with a single account. Allows users to access all the functions to which they are entitled by only performing a single system authentication step.

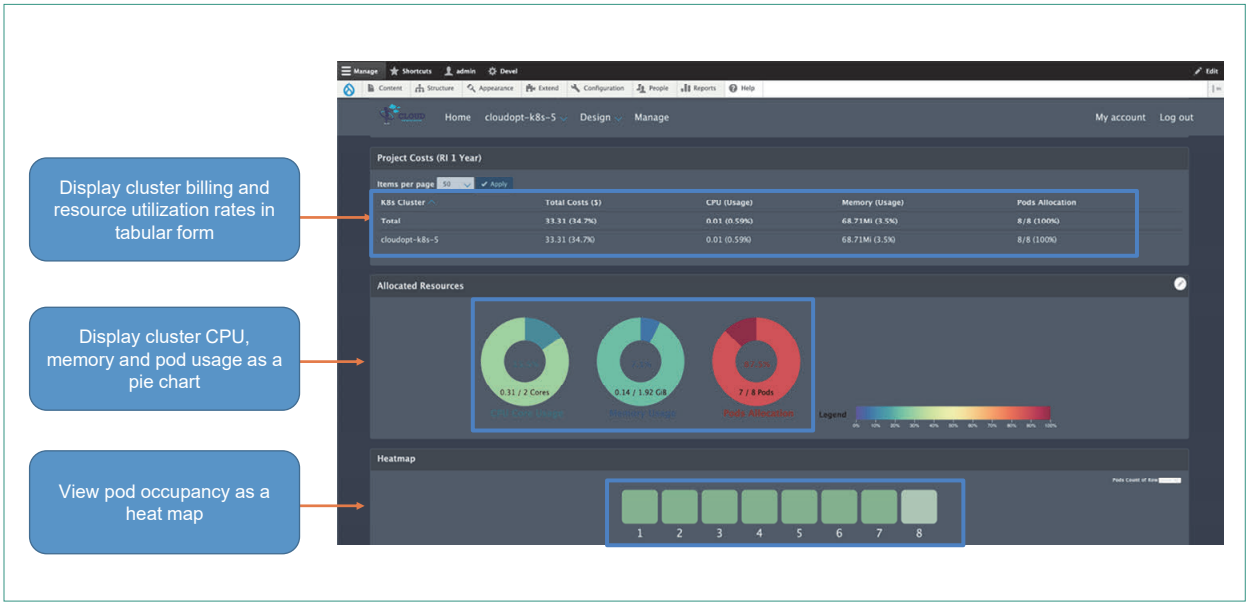

Figure 2 CO portal screen

other functions such as deploying tasks to be run on a cluster, and selecting clusters for deployment. Furthermore, since Drupal provides REST (REpresentational State Transfer) API*33 functions, it is possible to execute functions on the portal via the API.

As shown in **Figure 3**, CO provides many functions such as user management, cluster management, and job scheduling. Drupal's role*34 management features enable the definition of basic CRUD (Create, Read, Update and Delete)*35 functions for any function or service. By extending this functionality, CO can specify CRUD operations for all resource objects in a Kubernetes cluster. With this role management, even if different users deploy their own tasks on the same cluster, interference between these users can be prevented as long as each user is granted a role that can only affect his or her own tasks (i.e., the minimum necessary role). This idea is a fundamental feature underpinning the cost optimization discussed below.

The connector part uses APIs provided by various clouds and Kubernetes clusters, which we modularize and make available to users. The users can turn on these module functions and start receiving information periodically.

3.3 Key Features

This section describes the main functions of CO.

1) Single Sign-on

By making use of Drupal's single sign-on feature, CO can be linked to various clouds. Also, since Kubernetes clusters do not have user authentication functions, CO provides its own user authentication functions.

*33 REST API: An API conforming to REST. REST is a style of software architecture developed based on design principles proposed by Roy Fielding in 2000.

*34 Role: A group of users that grants certain privileges to its members.

*35 CRUD: An acronym representing the four basic operations of software used to provide systems and services (create, read, update & delete).

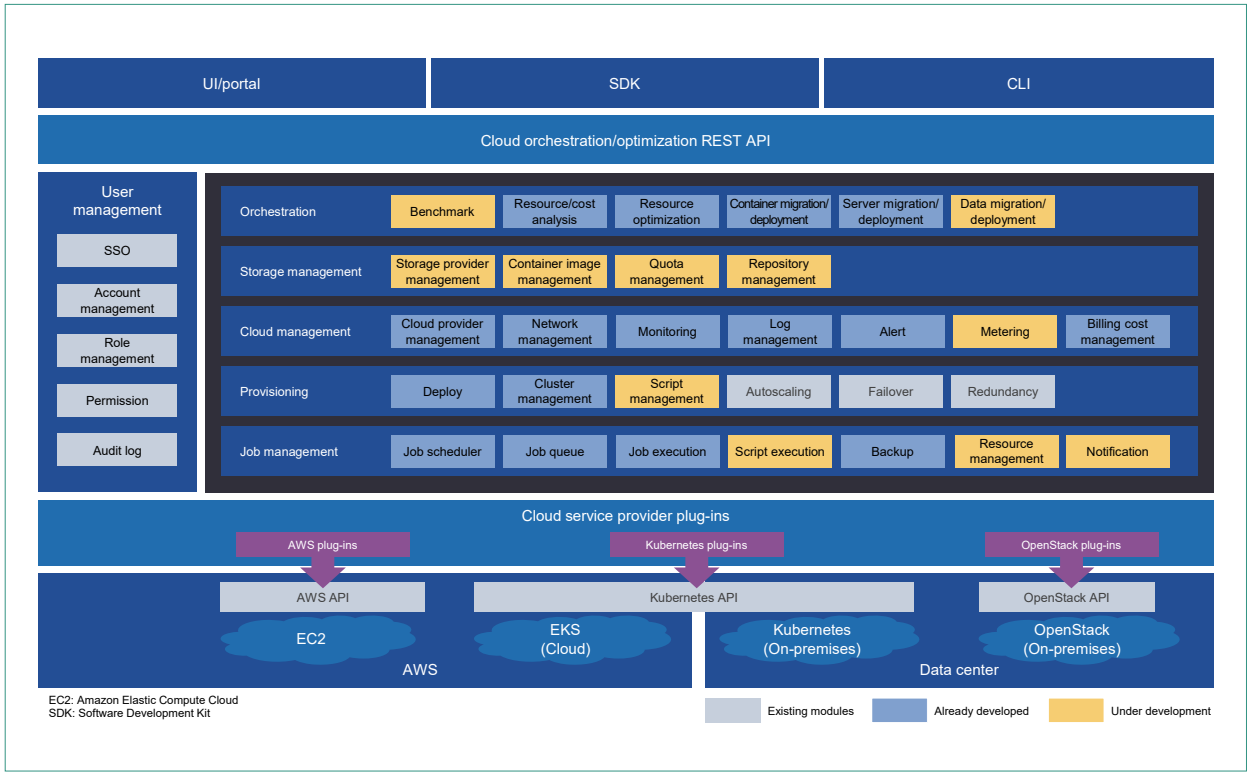

Figure 3 Cloud orchestration/optimization components

2) Resource Optimization

When CO manages multiple Kubernetes clusters, this function automatically performs load balancing between clusters. For example, when deploying any container, it automatically deploys it to the least occupied cluster based on the current resource usage situation. It is also possible for the user to select the deployment destination cluster beforehand.

3) Scheduling

CO includes time and resource based scheduling functions. For example, when a user wants to perform batch processing*36 at a particular time of day, such as late at night, the task's deployment time and end time can be set in advance. It is also possible to deploy low-priority tasks that run only when resources are available.

4) Cost Calculation

We specified our own logic to calculate the total instance cost of the master node and worker nodes based on the resource usage (memory, CPU, pod counts). This makes it possible to calculate the cost in Namespace*37 units.

5) Multi-cloud Management

Like public/private cloud solutions such as AWS, OpenStack*38 and VMware, Kubernetes clusters can also be managed in a unified manner. Users can check their resource usage status in a unified

*36 Batch processing: The automatic processing of data in batches without user interaction.

*37 Namespace: In Kubernetes, a virtual grouping that combines several different resources into a single unit.

*38 OpenStack: Cloud-infrastructure software that uses server virtualization technology to run multiple virtual servers on a single physical server. It can allocate virtual servers to different cloud services in use. OpenStack is open-source software.

manner even if they are working with different accounts or clusters.

3.4 How to Use CO

CO is an open-source Drupal project that can be used by anyone [2].

4. DOCOMO Case Study

4.1 Efficient Management of Multiple Clusters and Free Resources

1) Managing Multiple Clusters

CO is utilized in DOCOMO's commercial system. This system features multiple EKSs, with each cluster having more than 100 nodes. As shown in **Figure 4**, before the introduction of CO, administrators had to check the status of clusters and pods from the management screen of each cluster, and had to use a Command Line Interface (CLI)*39 to deploy containers, which made their work difficult.

However, after the introduction of CO, administrators no longer needed to check each cluster individually because CO obtains metrics*40 from every cluster. Also, when the management screen is used to register the details of a container and the Kubernetes cluster in which it is to be deployed, CO can perform the deployment automatically. If multiple Kubernetes clusters are specified as the deployment destination, then the system can automatically deploy containers to the selected cluster group and obtain metrics from each cluster. As mentioned above, in cluster selection, the user does not have to select a specific cluster, and can instead leave CO to use its resource optimization

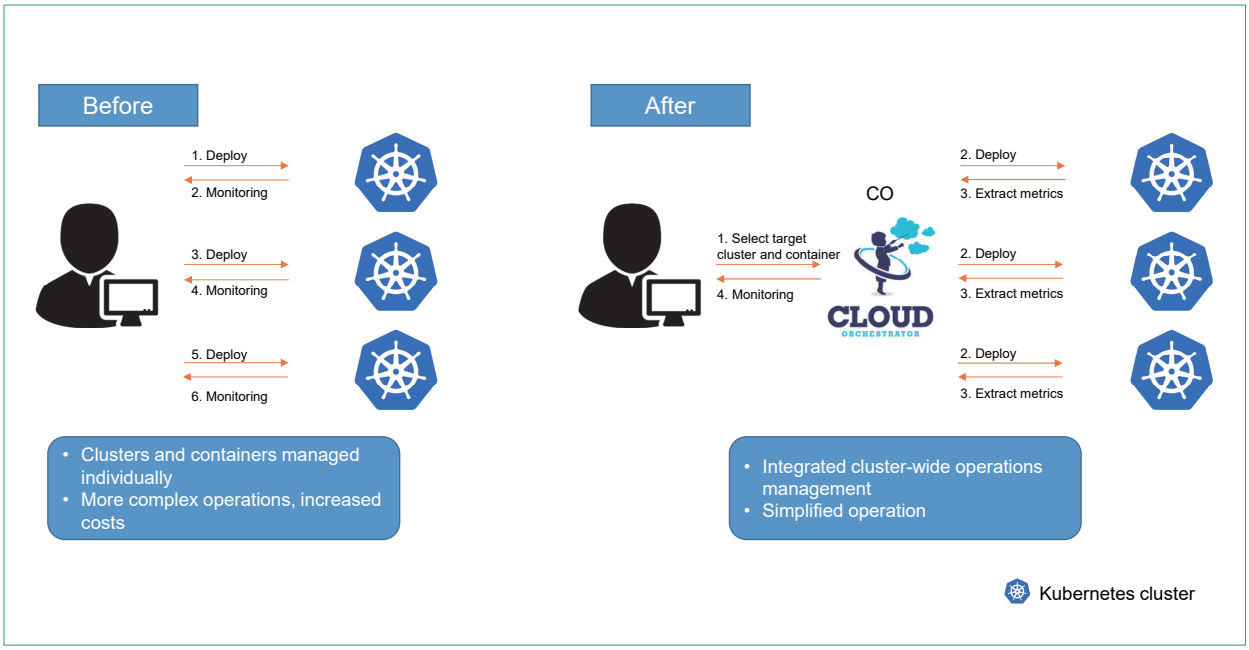

Figure 4 Changes resulting from CO adoption

*39 CLI: An operating method in which all instructions to a computer or software are given in the form of text.

*40 Metric: A quantitative item of information corresponding to the value of some parameter at a particular point in time, such as the CPU utilization rate, memory usage, or number of pods.

function to automatically select a free cluster.

2) Efficiency of Available Resources

As shown in **Figure 5**, CO has a time-based and resource-based scheduling functions. In DOCOMO's commercial system, the resource utilization rate of the entire system tended to be proportional to the user's usage. Specifically, the utilization rate can easily reach 80% or more during the daytime, when users are most likely to be using the system, but drops to around 20% late at night. In order to use the available resources, the CO administrator registers the container to be deployed, the scheduling method, and the necessary parameters, and CO then deploys the container according to the registered scheduling method. The containers in this case do not require real-time performance and often have a lower priority than other containers, so a mechanism is employed whereby the removal of these containers is prioritized during bursts[*41] of cluster resource usage. In this commercial system, a time-based scheduling function is used to aggregate the system logs and update the machine learning models late at night. In addition, when the resources of one cluster are overwhelmed, the resource-based scheduling function automatically distributes some tasks to other clusters to equalize the resources of the entire system instead of temporarily adding resources.

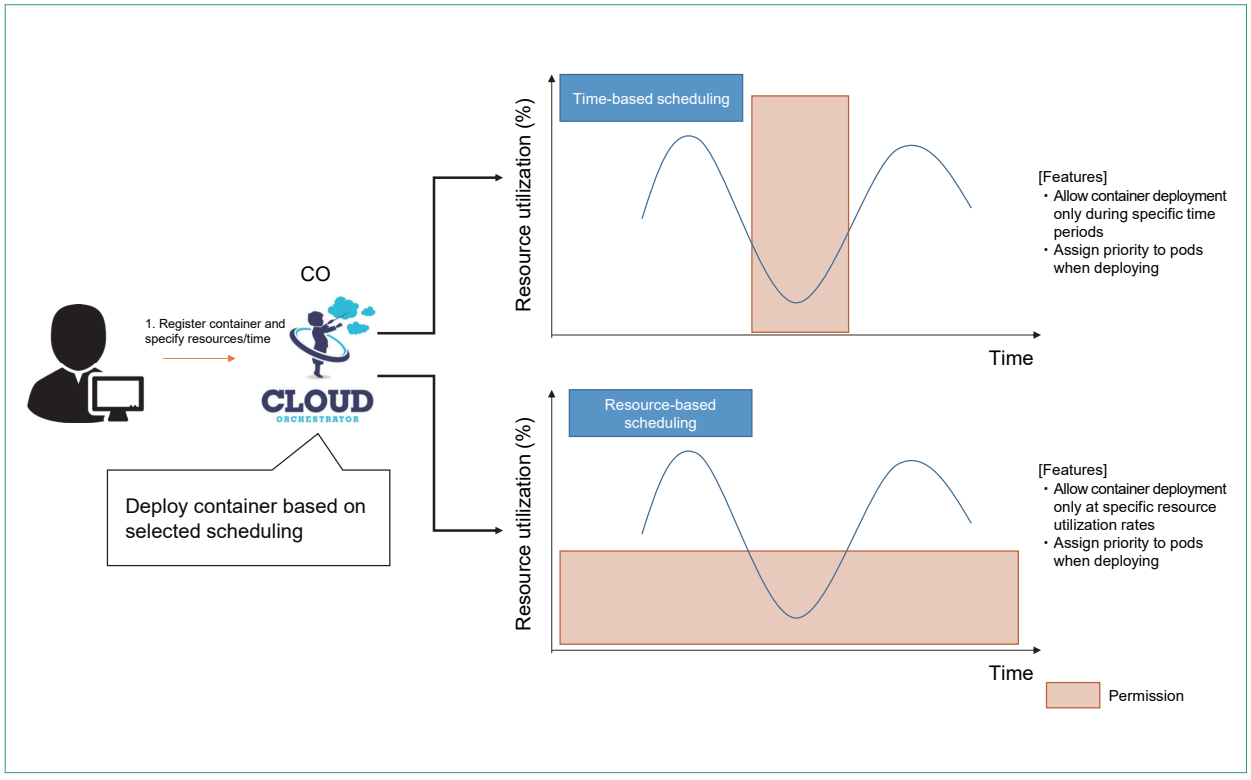

Figure 5 Scheduling function

*41 Burst: A momentary concentration of multiple signals at a fixed point in a device, caused by a temporary increase in network traffic, for example.

4.2 Separation of the Application and System Layers

As shown in **Figure 6** (a), each of DOCOMO's projects creates its own AWS account in which to develop applications and build systems. With this model, every project needs to be aware of AWS, manage its own accounts and system environments, and understand the security policies. As a result, deploying services can take a large amount of time. It also gives rise to various other issues, such as increased security risks due to the absence of security experts and increased cloud costs due to non-global optimization.

To address these issues, we considered separating the application and system layers at the boundary of CO (Fig. 6 (b)). Although this idea is currently only at the conceptual stage, it would eliminate the need for application developers to manage layers below CO and deal with cumbersome security management as long as their services are containerized, allowing them to concentrate their efforts on service development. On the other hand, containerization means that system administrators no longer need to be aware of the content of an application. Centralized management of clusters by system administrators is expected to improve the

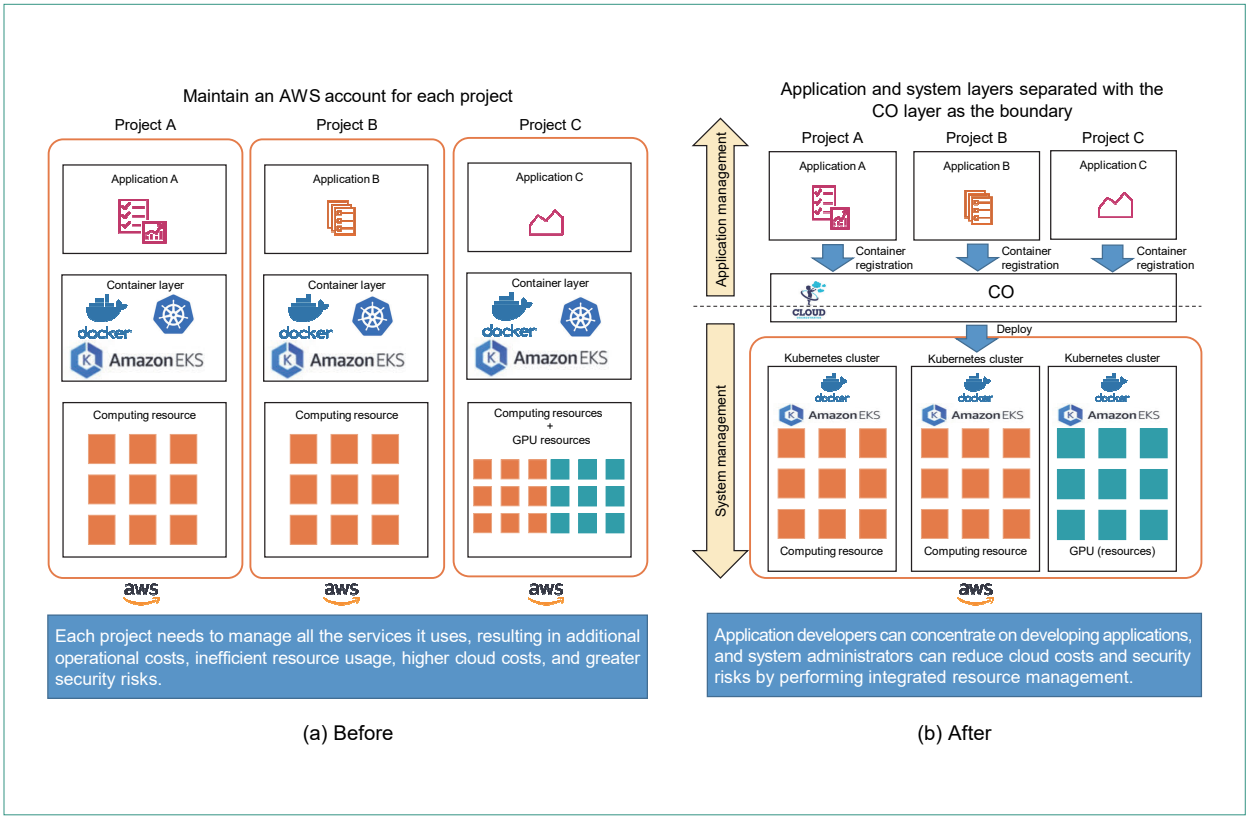

Figure 6 Separation of application and system layers by CO

overall efficiency of resource usage and reduce security risks, ultimately bringing down cloud costs company-wide.

However, with this approach, multiple services can run on the same cluster, and containers that perform heavy processes may affect other services. In addition, application developers will no longer be aware of cloud costs, so they may run processes that are less efficient. In this regard, in addition to using the Resource Quota*42 and Limit Range*43 functions of Kubernetes, CO uses a cost calculation function based on information obtained from the Kubernetes cluster to notify the application developer of fees accrued through the use of resources such as CPUs, memory, and execution pods. This can make application developers more cost-conscious.

We are currently in the process of validating the above method in cooperation with the Service Innovation department. There are not enough functions on the CO side to satisfy this approach, so we will continue to update them.

5. Conclusion

We have described a CO system that manages multiple Kubernetes clusters. We hope to continue to improve CO to make use of the knowledge accumulated by DOCOMO to further streamline the use of cloud services.

REFERENCES

[1] Kubernetes Web site.
https://kubernetes.io/
[2] Drupal: "Cloud."
https://www.drupal.org/project/cloud

*42 Resource Quota: A setting that indicates the amount of resources allocated to a container.
*43 Limit Range: A setting that determines minimum and maximum values with which to limit resource requests from a container.