L-001

複数クラウドを利用したコンテナ動的分散配置システムの開発
Development of Dynamic Distributed Container Allocation System using Multilple Cloud Computing Systems

大平 剛[1)]　**神屋 郁子**[2)]　**下川 俊彦**[1)]
Tsuyoshi Ohira　Yuko Kamiya　Toshihiko Shimokawa

1　はじめに

近年，システムの運用の簡素化や拡張性の向上が重要視されてきている．さらに，サービスの環境構築のコード化が進んでおり，クラウド上にサービスの環境を構築することが容易である．これらに伴いクラウドを用いたシステムの運用が増えている．クラウド上に構築したシステムは，一般に耐障害性や耐負荷性能が高い．しかし，データセンタの停電やクラウドの内部システムの障害などにより，クラウド上で運用されているシステムがダウンしてしまう問題が起きている [1]．さらに COVID-19 の影響によるインターネットの利用増加に伴い，サーバダウンなどの問題も発生している．

近年ではこのような問題に対処するために複数のパブリッククラウドを組み合わせて利用するマルチクラウドやパブリッククラウドとプライベートクラウドを組み合わせて利用するハイブリッドクラウドなどの利用形態が注目されている．しかし，複数のクラウドを利用したサービスの構築には，クラウドごとに設定を行う必要があり構築が容易ではない．

2　研究目的

本研究の目的は，複数のクラウドを利用したサービスの構築を容易にすることである．

3　複数のクラウドを利用したサービス構築

本章では，複数のクラウドを利用したサービスを構築する際の問題点について指摘する．その後，その問題点を解決するための手法やシステムの要件定義，システムの設計について説明する．

3.1　複数のクラウドを利用したサービスを構築する際の問題点

複数のクラウドを利用してサービスを展開し構築する場合，利用するクラウドごとに運用するサービスの構築や設定，負荷に応じたサーバの増減設定を行う必要がある．そのため，クラウドごとに設定を行う必要があるという問題がある．

3.2　問題点を解決するための手法

本研究では，クラウドごとに設定を行う必要があるという問題を解決するために，複数のクラウド上に，一元的にサービスを展開し構築する手法を提案する．この手法を用いることで，クラウドごとに設定を行う必要がなくなる．そのため，複数のクラウドを利用したサービスの構築が容易になる．また，クラウド上に構築したサービスのサーバの CPU 使用率やメモリ使用率などの負荷に応じてサーバの数を動的に増減させることで，サービスの可用性の向上が期待できる．

3.3　要件定義

本研究で開発するシステムの要件定義は以下の通りである．

- 一元的に複数のクラウド上にサービスを構築
- サービスで用いているサーバの負荷やサービスの負荷に応じてサーバの数を動的に増減

3.4　複数のクラウドを利用したコンテナ動的分散配置システムの設計

本研究では，複数のクラウドを利用したコンテナ動的分散配置システム (以下，本システム) を開発する．本システムでは，単一のクラウド上に構築されるサービスを複数のクラウド上に展開し構築する．また，各クラウドに構築したサービスの負荷に応じてサーバの数を動的に増減させることでサービスの可用性を向上させる．

サービスの環境構築では，構成情報などをコード化したものを用いて構築されることが多い．本システムでは，コード化された環境構築の基盤として Docker Compose を用いる．Docker Compose は，単一のクラウド上にコンテナを用いた環境を構築する．そのため，本システムで構築されるサービスはコンテナ上のサーバで実行される．また，複数のクラウドを一元的に管理するためにクラウドの外部からコンテナを管理する必要がある．そのため，本システムでは Kubernetes を用いてクラウド上のコンテナを管理する．これにより，利用者は既存のサービスに大きな変更を施すことなく複数のクラウドを利用してサービスを構築できる．

本システムを利用するには，構築するサービスの情報と構築先のクラウドの情報が必要である．そのため，本システムでは構築するサービスの情報を記述した「サービス構築情報定義ファイル」とサービスを構築する Kubernetes クラスタの接続情報を記述した「クラスタ情報定義ファイル」を使用する．本節では「サービス構築情報定義ファイル」と「クラスタ情報定義ファイル」について説明する．

3.4.1　サービス構築情報定義ファイル

サービス構築情報定義ファイルは，サービスの構築に必要なコンテナの種類やコンテナ起動時の設定，コンテナの個数や増減に関する設定を記述するファイルである．サービス構築情報定義ファイルは，Docker Compose の定義ファイルにコンテナの増減に関する設定項目を追加したものである．そのため，Docker Compose を用い

1）　九州産業大学
Kyushu Sangyo University
2）　福岡女子大学
Fukuoka Women's University

表 1　コンテナの増減に関する設定情報

パラメータ	サブパラメータ	備考
spec	-	コンテナの増減に関する設定情報のオブジェクト
-	maxReplicas	コンテナの最大数
-	minReplicas	コンテナの最小数
-	metricsResource	コンテナが増減する際の基準として利用する指標

表 2　“target” 項目の設定内容

パラメータ	サブパラメータ	備考
target	-	メトリクスのしきい値に関する設定情報のオブジェクト
-	averageUtilization	メトリクスのしきい値
-	type	メトリクスの測定項目

て構築されている既存のサービスを本システムで利用することができる．

このファイルでは “spec” という項目をキーとしたハッシュの値にコンテナの増減に関する設定項目を指定する．コンテナの増減に関する設定項目を表 1 に示す．このコンテナの増減に関する設定項目のフォーマットは Kubernetes の定義ファイルである Kubernetes マニフェストを参考にしている．

コンテナの増減に関する設定項目の中で，“metricsResource” は更に子項目の設定が必要である．“metricsResource” をキーとするハッシュの値には，コンテナを増減させる際の指標となるメトリクスの種類とメトリクスのしきい値を指定する．メトリクスの種類は “name” パラメータに指定する．本システムでは，メトリクスの種類として “cpu” と “memory” に対応している．また，メトリクスのしきい値は “target” をキーとするハッシュの値に指定する．“target” 項目の設定内容を表 2 に示す．

ソースコード 1 にサービス構築情報定義ファイルの例を示す．ソースコード 1 は，Docker Compose の定義ファイルの例にコンテナの増減に関する設定項目を追加したものである．このサービス構築情報定義ファイルの例では，wordpress コンテナ全体の CPU 使用率の平均が 50% を下回るようにコンテナの数が 1 個から 10 個の間で変動するように定義されている．

ソースコード 1　サービス構築情報定義ファイルの例

```
version: "3"
services:
    db:
      image: mysql:5.7
      volumes:
        - db_data:/var/lib/mysql
      ports:
        - "3306:3306"
      restart: always
      environment:
        MYSQL_ROOT_PASSWORD:
            somewordpress
        MYSQL_DATABASE: wordpress
        MYSQL_USER: wordpress
        MYSQL_PASSWORD: wordpress
    wordpress:
      depends_on:
        - db
      image: wordpress:latest
      ports:
        - "8000:80"
      restart: always
      environment:
        WORDPRESS_DB_HOST: db:3306
        WORDPRESS_DB_USER: wordpress
        WORDPRESS_DB_PASSWORD: wordpress
      spec:
        maxReplicas: 10
        minReplicas: 1
        metricsResource:
          name: cpu
          target:
            averageUtilization: 50
            type: Utilization
volumes:
    db_data:
```

3.4.2　クラスタ情報定義ファイル

クラスタ情報定義ファイルは，Kubernetes クラスタの API Server のアドレスと認証情報を記述するファイルである．本システムでは Kubernetes の認証情報としてサービスアカウントトークンを用いる．

サービスアカウントとは，Kubernetes API によって管理されるユーザアカウントのことである．サービスアカウントは，Kubernetes クラスタ上のどのリソースに対してアクセスできるかを定義することができる．サービスアカウントの作成の使用したコマンドをソースコード 2 に示す．このコマンドでは，k8s-compose という名前のサービスアカウントに cluster-admin の権限を付与して作成している．

ソースコード 2　Master サーバの Kubernetes 環境構築コマンド

```
$ kubectl -n kube-system create
    serviceaccount k8s-compose
$ kubectl create clusterrolebinding k8s-
    compose --clusterrole=cluster-admin
     --serviceaccount=kube-system:k8s-
    compose
```

サービスアカウントトークンの取得に使用したコマンドをソースコード 3 に示す．このコマンドでは，k8s-compose の資格情報が保存されているオブジェクト名を取得した後，オブジェクト名を指定してサービスアカウントトークンを取得している．

ソースコード 3　Master サーバの Kubernetes 環境構築コマンド

```
$ SECRET_NAME=`kubectl -n kube-system
    get serviceaccount/k8s-compose -o
    jsonpath='{.secrets[0].name}'`
```

表 3　クラスタ情報定義ファイルの設定項目

パラメータ	サブパラメータ	備考
clusters	-	クラスタ情報のオブジェクト
-	name	クラスタ名
-	server	クラスタの API Server のアドレス
-	token	サービスアカウントトークン

```
$ kubectl -n kube-system get secret
    $SECRET_NAME -o jsonpath='{.data.
    token}'| base64 --decode
```

ソースコード 4 にクラスタ情報定義ファイルの例を示す．また，クラスタ情報定義ファイルの設定項目を表 3 に示す．

ソースコード 4　クラスタ情報定義ファイルの例

```
clusters:
- name: eks-osaka
  server: https://XXXXX.sk1.ap-
      northeast-3.eks.amazonaws.com
  token: XXXXX
- name: eks-test
  server: https://XXXXX.sk1.ap-
      northeast-1.eks.amazonaws.com
  token: XXXXX
```

4　複数のクラウドを利用したコンテナ動的分散配置システムの実装

本章では，本システムの実装に述べる．本システムは，サービス構築情報定義ファイルの変換機能，コンテナデプロイ機能を 1 つの Python アプリケーションとして実装した．

本システムの処理内容は以下のとおりである．

1. 定義ファイルの読み込み
2. サービス構築情報定義ファイルの変換
3. コンテナデプロイ

それぞれの処理について説明する．

4.1　設定ファイルの読み込み

この処理では，Python の YAML ライブラリを用いて 3.4.1 と 3.4.2 で述べた定義ファイルを読み込み，変数に格納する．

4.2　サービス構築情報定義ファイルの変換

この処理では，まず 3.4.1 で述べたサービス構築情報定義ファイルからコンテナの増減に関する定義を抽出し，コンテナの増減に関する部分の Kubernetes マニフェストに変換する．ここでは，予め Kubernetes マニフェストの雛形を用意しておき，サービス構築情報定義ファイルのコンテナの増減に関する定義の部分と結合することで Kubernetes マニフェストを生成している．次に，サービス構築情報定義ファイルからコンテナの増減に関する定義を削除する．これは，サービス構築情報定義ファイルにコンテナの増減に関する定義が入った状態では，Kompose が変換を行うことができないためである．そして，Kompose を用いてサービス構築情報定義ファイルから Kubernetes マニフェストを生成する．ここでは，システム内部でソースコード 5 のコマンドを実行し，Kubernetes マニフェストを生成している．

ソースコード 5　Kompose の変換コマンド

```
$ kompose convert -f docker-compose.yml
```

4.3　コンテナデプロイ機能

この処理では，4.2 で生成した Kubernetes マニフェストを Kubernetes クラスタにデプロイする．ここでは，3.4.2 で述べたクラスタ情報定義ファイルに記述されている Kubernetes クラスタの API Server のアドレスと認証情報を基に API Server と接続を行う．次に，Kubernetes マニフェストの種類によって Kubernetes API Server のエンドポイントが異なるため，Kubernetes API Server から Kubernetes マニフェストとエンドポイントの対応表を取得する．そして，システムで Kubernetes マニフェストの種類を判定し，適切な API Server のエンドポイントに POST リクエストで Kubernetes マニフェストを送信する．クラスタ情報定義ファイルに複数のクラスタの情報が記述されている場合は，すべてのクラスタに対して上記の処理を繰り返し実行する．これにより，Kubernetes クラスタ上で送信したマニフェストごとにリソースが作成され，コンテナを含む Pod がクラスタ上に配置される．

5　評価

本章では，本システムの評価内容，評価環境，評価結果，考察について述べる．

5.1　評価内容

本評価では，Docker Compose を用いて構築されているサービスを複数のクラウド上に展開し構築する．今回は，以下のサービスを複数のクラウド上に構築する．

- WordPress
- NextCloud
- LAMP 開発環境

評価方法は次のとおりである．初めに，サービス構築情報定義ファイルに評価するサービスのコンテナ情報を定義する．WordPress を用いた評価ではサービス構築情報定義ファイルに 3.4.2 のソースコード 1 を用いる．次に，クラスタ情報定義ファイルに Kubernetes クラスタの接続情報を定義する．その後，本システムを実行する．

クラウド上にコンテナが配置されたかどうかはソースコード 6 のコマンドを実行して確認する．

ソースコード 6　稼働中のコンテナを確認するコマンド

```
$ kubectl get pods
```

5.2 評価環境

本評価では，Amazon Web Service の大阪リージョンと東京リージョンで EKS クラスタを構築し，評価を行った．この評価では，各リージョンに 2 台の Node サーバで構成された EKS クラスタを構築し，利用した．EKS クラスタで用いた Node サーバのスペックを表 4 に示す．

表 4 Node サーバの構成

OS	Amazon Linux 2
CPU コア数	2
メモリ	4GB
ディスク	8GB

5.3 評価結果

本評価の結果，WordPress と NextCloud は複数のクラウド上にサービスが展開し構築されたことを確認した．しかし，LAMP 開発環境についてはサービス構築情報定義ファイルの変換に失敗したためクラウド上にコンテナを展開できなかった．

5.4 考察

本評価の結果，一部のサービスを除き複数のクラウド上にサービスが展開し構築された．この結果より，本システムを用いることで複数のクラウドを扱う際に運用が煩雑であるという問題について解決に向けて見通しを得ることができた．

LAMP 開発環境のサービス構築情報定義ファイルの変換が失敗した理由として，Docker と Kubernetes でサポートしているストレージの種類が異なることが考えられる．LAMP 開発環境の定義ファイルには，PHP コンテナがホスト OS 上のディレクトリをマウントする設定が記述されている．この設定は，コンテナ内部とのファイル共有が容易に行えるため Docker を用いた開発環境でよく用いられる．Kubernetes は，複数の Node サーバ上にコンテナが自動的に配置されるため，ホスト OS 上のディレクトリをストレージとすることができない．このような設定に対しては，サービス構築情報定義ファイルの変換時にディレクトリ内のデータをボリュームに変換するなどの対策が必要である．

6 結論

本章では，本研究のまとめと今後の課題について述べる．

6.1 まとめ

近年，サービスの環境構築のコード化が進んでおり，クラウド上にサービスの環境を構築することが容易である．また，サービスの冗長性を高めるために複数のパブリッククラウドを組み合わせて利用するマルチクラウドなどの利用形態が注目されている．しかし，複数のクラウドを利用したサービスの構築には，クラウドごとに設定を行う必要があるという問題がある．

本研究では，複数のクラウドを利用したサービスの構築を容易にすることを目的とする．この目的を達成するために，単一のクラウド上に構築されるサービスを複数のクラウド上に展開し構築するシステムの開発を行った．このシステムは，一元的に複数のクラウド上にサービスを展開し構築する．これによりクラウドごとに設定を行う必要が無くなり，複数のクラウドを利用したサービスの構築が容易になる．今回開発したシステムでは，コード化された環境構築の基盤である Docker Compose を用いる．Docker Compose は，単一のクラウド上にコンテナを用いた環境構築を行う．そのため，本システムではコンテナ上でサーバを実行する．本システムでは，環境構築のコードが記述された Docker Compose の定義ファイルを利用する．また，複数のクラウドを一元的に管理する必要があるため，クラウドの外部からコンテナの管理が可能な Kubernetes を用いてクラウド上にコンテナを配置する．これにより，利用者は既存のサービスに大きな変更を施すことなく本システムを用いて複数のクラウド上にサービスを構築できる．

本システムの評価では，Docker Compose を用いて構築されているサービスを複数のクラウド上に構築する．評価の結果，一部のシステムを除き複数のクラウド上に構築することができた．以上のことから，複数のクラウドを利用したサービスの構築を容易にすると見通しを得た．

6.2 今後の課題

本研究の今後の課題として，5.4 節で述べたサービス構築情報定義ファイルの変換が失敗する問題がある．この問題の解決するために，サービス構築情報定義ファイルの変換時にホスト OS 上のディレクトリをボリュームに変換する機能を実装する必要がある．また，Docker Compose を用いて構築されている，より多くのサービスを対象として複数クラウドに展開し構築できるか調査する必要がある．

参考文献

[1]Summary of AWS Direct Connect Event in the Tokyo (AP-NORTHEAST-1) Region
https://aws.amazon.com/jp/message/17908/