# Multi・Hybrid Cloud Management Layerの提案

新宮 隆太[1,a)]　串田 高幸[1]

**概要**：現状のマルチクラウド, ハイブリッドクラウドの環境はベンダーロックインの影響で運用コストが高いという問題を抱えている. そこで本稿ではベンダーやオンプレミスの環境に依存しない抽象的で疎結合な構成管理ファイルとそれを用いて構成・管理を行うレイヤー Multi・Hybrid Cloud Management Layer を提案し, Google Compute Engine のインスタンスを建てる際に必要な設定項目数を Ansible と比較した. その結果より少ない項目数で構築を行うことができることが確認された.

## 1. はじめに

近年, コンピューティングリソースをインターネット上からアクセスすることができるクラウドが広く普及している. クラウドはインターネットを通して機能を提供する際に頻繁に用いられており我々の生活に不可欠である. クラウド基盤を提供するベンダー (例えば Google Cloud Platform(以下 GCP)[1], Amazon Web Services(以下 AWS)[2], Azure[3]) は数多く存在する. これらのベンダーの登場により開発者側もクラウドを用いることが容易になっている. また, 開発者から可用性や堅牢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るマルチクラウド [4] やパブリッククラウドとオンプレミスを組み合わせるハイブリッドクラウド [5] が注目され始めている.

しかしながら現状のクラウドにはベンダーロックイン [6] という問題点が存在する. ベンダーロックインとはベンダーの独自の設定やサービスに依存してしまい, 他ベンダーのサービスへの乗り換えが困難になることである. クラウド基盤を提供するベンダーは多くあるが提供しているサービスはほぼ同等であるにもかかわらず各ベンダーが共通化されていない独自の用語や設定項目をユーザーに押し付けて使用させている. これでは使用するクラウド基盤を変更する際に新しい独自の用語や設定項目を学習し直さなければならずコストが高くなる. また, マルチクラウドやハイブリッドクラウドの管理はベンダーロックインの影響を大きく受け, これらは運用する上で各クラウドサービスに詳しいエンジニアがそれぞれ必要であるためコストがより高くなる.

本稿ではこれらの問題を解決し自由にクラウドベンダーの変更, 組み合わせが行えるようにするためにベンダーやオンプレミスの環境に依存しない抽象的で疎結合な構成管理ファイル. その構成管理ファイルを用いて構成・管理を行うレイヤーである Multi・Hybrid Cloud Management Layer(以下 MHCML) を提案する.

## 2. 関連研究

今回関連研究を 3 つ取り上げる.

まず 1 つ目に CloudMF: Applying MDE to Tame the Complexity of Managing Multi-Cloud Applications[7] である. クラウドモデリングフレームワーク (CloudMF)[8] とモデル駆動型エンジニアリングを活用してマルチクラウドアプリケーションの複雑さを解決する手法を提案している.

しかしながら今回提案する MHCML はマルチクラウドだけではなくハイブリッドクラウドに対応している点, 構成を定義する言語を抽象化している点において違いがある.

2 つ目にマルチクラウド基盤上のアプリケーション構築自動化アーキテクチャの提案と評価 [9] 構成情報を抽象化することでベンダーに依存せずにマルチクラウド上で仮想マシンの構築とアプリケーションのデプロイを行うアーキテクチャを提案している.

ハイブリッドクラウド, マルチクラウドどちらにも対応しており API の抽象化をしている点において MHCML と同じであるが, 一部環境に依存している設定項目がある点や仮想マシン上の管理に焦点置いている点において違いがある.

最後に Topology and Orchestration Specification for Cloud Applications[10] である. Topology and Orchestration Specification for Cloud Applications(以下 TOSCA) は Organization for the Advancement of Structured Infor-

[1] 東京工科大学コンピュータサイエンス学部
CDSL, TUT, Hachiouji, Tokyo 101–0062, Japan
[a)] C0117155

mation Standards(OASIS) が策定したクラウド上で稼働するシステムのシステム構成を定義するための標準仕様である．YAML 形式 [11][12] での記述となっている．

これは一見同じ提案をしているように見えるが実際には違う．TOSCA はマルチクラウド環境上で動くアプリケーションの管理であるのに対して MHCML はクラウド環境自体の管理である．例えば TOSCA はマルチクラウド環境での MySQL のスキーマを管理するが MHCML ではそれは管理する対象ではない．

## 3. Multi Hybrid Cloud Management Layer の概要

MHCML はユーザーと各クラウド, オンプレミス環境の間に設置するマルチクラウド, ハイブリッドクラウドの管理と構築を行うレイヤーである．MHCML は管理を行うためのサーバーである Multi Hybrid Cloud Management Server(以下 MHCMServer) を必要とし, このサーバーに config ファイルを与えることにより各環境の管理を行う．これによりクライアント側は従来の問題であった複数環境の設定項目の差異を意識することなく管理を一括で行うことができる．

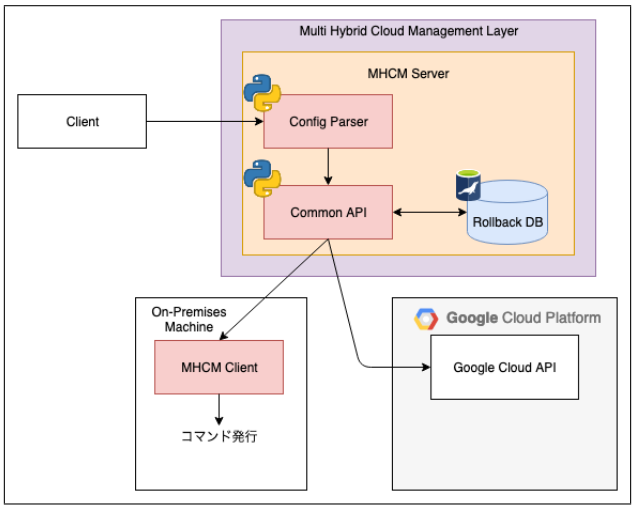


図 1　Multi・Hybrid Cloud Management Layer の概略図

### 3.1 Multi Hybrid Cloud Management Server

Multi Hybrid Cloud Management Server の構成例を図 3.1 に示す．MHCM Server は大きく 3 つに分けられる．まず 1 つ目に Config をパースする Config Paser，2 つ目に ConfigParser でパースした抽象化された構成情報を実際にそれぞれ環境にあった API へ投げる CommonAPI, 最後に構成情報の履歴を保存する RollbackDB である．これらを組み合わせて管理を行う．

クライアントから MHCM Server の Config Parser に対し 3.2 で後述する YAML 形式の抽象的な構成管理 config ファイルを送信する．それを Parser が解釈してクラウドベンダーが用意している API を抽象化した CommonAPI を通してオンプレミス環境に設置する MHCM Client, 各ベンダーが用意する API(例えば GoogleCloudAPI, AWSAPI) が処理を行う．これによりマルチクラウド環境, ハイブリッドクラウド環境のどちらにも適用することができる．また, このシステム自体の耐障害性を高めるために処理の途中で問題が発生した場合には RollbackDB に保存された直前の構成情報を元に Rollback を行う．

### 3.2 config ファイル

構成管理 config ファイルの例をソースコード 1 に示す．config ファイルは YAML 形式である．この例では GCP, AWS, CentOS7 をインストールしたオンプレミスマシンの 3 種類の環境のファイアウォールと仮想マシンを管理することを想定している．

1-16 行目の instances ブロックでは構成管理を行う環境の認証情報を記述する．28-27 行目の firewall ブロックでは apply-instance に指定された環境に対してファイアウォールの設定を行う．allow で指定された抽象的な用意されたキーワード (例えば ssh, web-server) を入力することにより ssh であれば 20/tcp を開放する．もし抽象化されたキーワードの中に目的のものが存在しない場合は custom-allow で直接指定することも可能である．29-38 行目の virtual-machines ブロックでは apply-instance で指定された環境に対して仮想マシンのインスタンスを立ち上げる．on-premises 環境にはこれを用いることができない．この例ではリージョンを指定せずに価格優先か低遅延優先かを指定することで最適なリージョンを自動的に導き出しそこにインスタンスを構築する．ストレージとメモリサイズは指定してもらいそれに一番近い構成の仮想マシンを用意する．

全体を通して一貫している特徴としては項目の抽象度が高い点である．この抽象度の高さによりユーザーはベンダーによる項目名の差異を意識することなく構成を定義することができる．

ソースコード **1** example-config.yaml

```
instances:
  "gcp-instance-1":
    instance-type: "gcp"
    account:
      credential_file_path: "./MHCML-TEST-831
          febb010b5.json"
  "aws-instance-1":
    instance-type: "aws"
    account:
      name: "hoge"
      password: "password"
  "on-premise-centos7":
    instance-type: "on-premise"
    os: "centos7"
    account:
      name: "hoge"
      password: "password"

firewall:
  allow:
    - "web-server"
    - "ssh"
  custom-allow:
    - "8001/tcp"
  apply-instance:
    - "gcp-instance-1"
    - "aws-instance-1"
    - "on-premise-centos7"

virtual-machines:
    region-primary: "price" # "price" or "
        latency"
    storage-size-gbyte: 50
    memory-size-mbyte: 1024
    target-os: "debian"
    network:
      name: "network-instance"
    apply-instance:
      - "gcp-instance"
      - "aws-instance-1"
```

## 4. 実装

今回, MHCML の ConfigParser と CommonAPI はともに Python3.8 で実装を行い, VMware ESXi 上のコア数:1 RAM:1GB ストレージ: 30GB の Ubuntu Server 18.04 LTS をインストールした仮想マシンで上で稼働させた. ソースコードはリポジトリ[*1]から確認することができる.

### 4.1 ConfigParser

ConfigParser はクライアントから何かしらの手段で送られてきた抽象化された構成情報が記述されている YAML 形式の config ファイルを読み抽象度を落として後述する CommonAPI へ渡す役割を持っている. 今回, ConfigParser は Python モジュールとして実装を行った. PyYaml を用いて YAML を Python オブジェクトにパースし, CommonAPI へ Python オブジェクトを渡している.

### 4.2 CommonAPI

CommonAPI は ConfigParser から渡された構成情報を Rollback DB へ保存する. その後それを元に各ベンダーが用意した API, もしくは MHCM Client が用意する API を叩く役割を持っている. 今回, CommonAPI は GCP と MHCM Client にのみ対応している.

## 5. 評価

今回, GCP の GCE インスタンスの構築をセクション 4 で説明した環境で MHCMServer と RedHat が開発を行っているオープンソースの構成管理ツールの Ansible で行い, 必要な記述項目数の比較を行った. 構築する GCE インスタンスは f1-micro タイプで 50GB のストレージを持つものを想定する. 実際の GCE インスタンスを建てるためには disk と network が必要となる. 今回はそれらを作成した上でインスタンスを作成する.

Ansible の構成管理ファイルは MHCMServer と同じ YAML 形式である. Ansible の構成管理ファイルは 34 項目となった. それに比べ MHCMServer は 17 項目で同じ実装を書くことができる. また Ansible でマルチクラウドのために同等の構成の別ベンダーのクラウドサービスを使う場合更に項数は単純計算で 2 倍の 68 項目だが MHCMServer は instances に認証情報を記述するのみで対応ができるため 24 項目の記述で構築が可能になる.

また, ベンダーに依存した項目数を比較すると Ansible が 8 項目あるのに比べ MHCMServer は project_id の 1 項目のみである. この結果から MHCML はより簡単にマルチクラウド環境を構築できることがわかる. しかしながら柔軟性は細かい設定ができる Ansible の方が高いと言わざる負えない.

## 6. 議論

MHCMServer の config ファイルでは抽象度を上げてベンダーごとの設定項目の違いを吸収することに重点を置いている. しかし現在は virtual-machines ブロックのメモリサイズ, ストレージサイズをユーザーに指定して貰う必要がある. そこで config ファイルの virtual-machines ブロックに使用用途を記入する項目を作成しそれに合わせたメモリサイズとストレージサイズをレコメンドする手法を確立して更に抽象度を上げるというものを検討している. また, 各ベンダーが用意する API を利用する際に用いる認証情報の保管についても検討事項が存在する. 現在, 認証情報の安全性は考慮されていない. そこで MHCMServer のアプリケーション側でアカウント認証を挟み保存することを

[*1] https://github.com/homirun/MultiHybridCloudManagementLayer

検討している．更に VM イメージの指定は関連研究 [9] のように管理サーバー側でイメージを持っておいて各環境から参照させる手法を取り入れることを検討している．更に，セクション 5 の通り設定項目の抽象化を行っている代わりに構築柔軟性が現状のものに比べて低い事が分かっている．そのため抽象度をできるだけ下げずに細かい設定をできるような構成管理ファイルの記述方式を検討している．これにより更に簡単に管理と構築が行えるようになる．

## 7. 終わりに

本稿では構成管理情報を記述する config ファイルの抽象度を上げてベンダーに依存しない項目で構成することで簡単にマルチクラウド・ハイブリッドクラウド環境の管理と構築を行える Multi・Hybrid Cloud Management Layer を提案した．Multi・Hybrid Cloud Management Layer では推論や各項目の抽象化によって記述するために必要な事前知識と項目数の減少へ繋がった．更にクライアントアプリケーションを用意することによりオンプレミス環境でも利用することができるようになりマルチクラウドだけではなくハイブリッドクラウドに対応できるようになった．

しかしながら課題点がまだ存在している．一番大きな問題点は構成管理ファイルの抽象度を上げれば上げるほど柔軟性が失われていく点である．これを解決するために今後も研究・調査を続けていきたい．

### 参考文献

[1] LLC, G.: Google Cloud Platform の概要，`https://cloud.google.com/docs/overview/?hl=ja` (2019.12.26).

[2] Amazon Web Services, I.: アマゾン ウェブサービス (AWS)，`https://aws.amazon.com/jp/` (2019.12.26).

[3] Corporation, M.: Microsoft Azure, `https://azure.microsoft.com/ja-jp/` (2019.12.26).

[4] CloudFlare, i.: What Is Multicloud? — Multicloud Definition, `https://www.cloudflare.com/learning/cloud/what-is-multicloud/` (2019.12.7).

[5] Mell, P. M. and Grance, T.: SP 800-145. The NIST Definition of Cloud Computing, Technical report, Gaithersburg, MD, United States (2011).

[6] Satzger, B., Hummer, W., Inzinger, C., Leitner, P. and Dustdar, S.: Winds of Change: From Vendor Lock-In to the Meta Cloud, *IEEE Internet Computing*, Vol. 17, No. 1, pp. 69–73 (online), DOI: 10.1109/MIC.2013.19 (2013).

[7] Ferry, N., Song, H., Rossini, A., Chauvel, F. and Solberg, A.: CloudMF: Applying MDE to Tame the Complexity of Managing Multi-cloud Applications, *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pp. 269–277 (online), DOI: 10.1109/UCC.2014.36 (2014).

[8] Ferry, N., Chauvel, F., Rossini, A., Morin, B. and Solberg, A.: Managing Multi-cloud Systems with CloudMF, *Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies*, NordiCloud '13, New York, NY, USA, ACM, pp. 38–45 (online), DOI: 10.1145/2513534.2513542 (2013).

[9] 幸大岡田，幹雄青山：マルチクラウド基盤上のアプリケーション構築自動化アーキテクチャの提案と評価，第 77 回全国大会講演論文集，Vol. 2015, No. 1, pp. 125–126 (2015).

[10] for the Advancement of Structured Information Standard, O.: OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC, `https://www.oasis-open.org/committees/tc\_home.php?wg\_abbrev=tosca` (2019.12.25).

[11] for the Advancement of Structured Information Standard, O.: TOSCA Simple Profile in YAML Version1.3 (2019).

[12] Ben-Kiki, O., Evans, C. and Ingerson, B.: Yaml ain't markup language (yaml$^{TM}$) version 1.1, *yaml. org, Tech. Rep*, p. 23 (2005).