Vol.2014-SE-183 No.12
2014/3/19

TOSCA を拡張したハイブリッドクラウド上のアプリケーション分散構築アーキテクチャの提案と評価

高木裕之[†1]　青山幹雄[†2]

異なるクラウド基盤を連携するハイブリッドクラウドが注目されているが，クラウド基盤ごとに API と構成管理方法が異なるためユーザの要求を満たすアプリケーションの分散構築が困難である．本研究では，クラウドアプリケーションの標準構成記述仕様である TOSCA を拡張し，異なるクラウド基盤上に分散配置を可能にする．そして，クラウド基盤ごとに異なる API を抽象化し，クラウド基盤によらない仮想マシン生成を実現する．分散構築自動化基盤アーキテクチャを提案する．提案アーキテクチャのプロトタイプを実装し，その有効性を評価する．

An Architecture for Automatic Building of Distributed Applications on a Hybrid Cloud Computing Based on TOSCA

Hiroyuki Takagi[†1]　Mikio Aoyama[†2]

Currently, hybrid clouds that integrate private clouds and public-clouds are attracting attention. However, construction of hybrid cloud is difficult due to the complicated configuration management and the difference of levels of abstraction off APIs for each cloud infrastructure. In this article, we extend TOSCA (Topology and Orchestration for Cloud Applications), a standard specification of configuration description for cloud application, in order to enable building of distributed applications on different clouds. We define a set of abstract APIs that encapsulate different APIs of each cloud, which enable to create virtual machines independent of cloud infrastructures. We propose an architecture for automatic building of distributed applications on a hybrid cloud computing. We developed a prototype of the architecture, and demonstrate the validity of the proposal architecture.

1. はじめに

クラウドコンピューティングの普及により，異なるプライベートクラウドやパブリッククラウドなどのクラウド基盤 [5]を連携するハイブリッドクラウドが注目されている．しかし，多数あるクラウド基盤ごとに API のシグネチャと抽象度，そして，クラウドアプリケーションの構成管理方法が異なる．そのため，ハイブリッドクラウドへユーザの要求を満たすアプリケーションの分散構築が困難である．

本研究では, TOSCA を拡張したハイブリッド上のアプリケーション分散構築アーキテクチャを提案する．TOSCA を拡張したクラウドアプリケーションの構成管理方法で統一し, クラウド基盤ごとの API を抽象化する. これにより，クラウド基盤によらない仮想マシンの生成を実現し，生成した仮想マシンへ自動的にクラウドアプリケーションの分散構築を実現する．

2. 研究課題

2.1 クラウド基盤ごとに API が異なる

クラウドアプリケーションの分散構築を行うためには，仮想マシンの生成が必要である．しかし，クラウド基盤ごとに API のシグネチャとその抽象度が異なる．

2.2 アプリケーションの構成管理方法が異なる

アプリケーションの構成管理方法はクラウド基盤ごとに異なるため，構築したいクラウドアプリケーションの構成を異なる基盤ごとに表現する必要がある．近年，異なるクラウド基盤間の構成管理方法を統一する仕様の TOSCA が標準化されているが，クラウドアプリケーションの分散構築に対応していない.

3. 関連研究

3.1 ハイブリッドクラウド [9]

単一 API のクラウド基盤によってパブリックやプライベート環境に構築されたハイパーバイザを制御しハイブリッドクラウドを構成する研究が行われている．しかし，ハイブリッドクラウドを構成する複数のクラウド基盤を単一 API に統一することは困難である．

3.2 フェデレーションクラウド [4]

フェデレーションクラウドとは，異なるクラウド基盤を連携するハイブリッドクラウドのモデルである．クラウド基盤ごとに信頼性のレベルやコストなどに関する差別化を行える．しかし，複数の異なるクラウド基盤を操作して仮想マシンの生成とアプリケーションの構築を行う必要があるため，クラウドアプリケーションの構築，運用が複雑となる．

3.3 TOSCA [7]

異なるクラウド基盤間でアプリケーションの構成を統一的に扱うための標準記述仕様である(図 1). サーバ, OS,

* †1 南山大学大学院　数理情報研究科
Graduate School of Mathematical Sciences and Information Engineering, Nanzan University.
†2 南山大学　情報理工学部　ソフトウェア工学科
Department of Software Engineering, Nanzan University.

ミドルウェア，アプリケーションをノードとして扱い，それらの関係に定義よりトポロジを表現する．また，TOSCA で用いるアプリケーションの実体やアプリケーションの構築に用いるスクリプトを管理，処理する TOSCA コンテナを必要とする．これを Service Template と呼ぶ．さらに構築手順を Plan として定義し，アプリケーションの自動構築を可能にする．

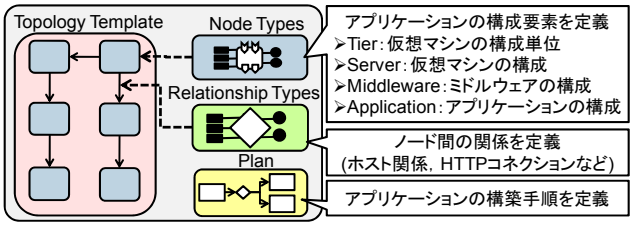

図 1 TOSCA の概要

Fig. 1 Overview of TOSCA

3.4 Chef[6)]

Ruby で実装されたシステム管理ツールである．サーバに対してミドルウェアやアプリケーションのインストール，ミドルウェアの設定，各稼働サービスの状態管理など，システム構築や運用作業を自動化するツールで，オープンソースソフトウェアとして公開される．

4. アプローチ

TOSCA を拡張して異なるクラウド基盤間でアプリケーション構築に用いる仮想マシンの構成管理方法を統一し，クラウド基盤の構成情報を管理可能にする(図 2)．このため，クラウド基盤によらない仮想マシンの生成を実現する抽象 API を定義し，ブローカによって仮想マシンの配置を行うブローカアーキテクチャを提案する．これにより，Service Template から仮想マシンの情報を抽出し，配置可能なクラウド基盤を選択する．そして，Plan で定義された構築手順に従い，ブローカの抽象 API を実行し，クラウド基盤によらないアプリケーションの自動構築を可能にする．

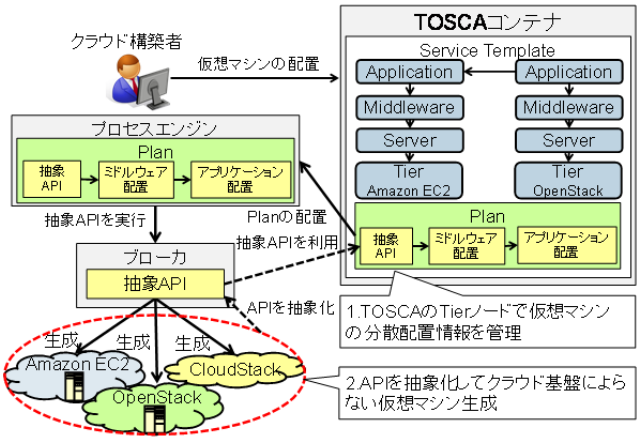

図 2 アプローチ

Fig. 2 Approach

5. 抽象 API の設計

5.1 クラウド基盤の API

異なるクラウド基盤ごとに API の抽象度は異なっている(図 3)．例として，Amazon EC2[1)]と OpenStack[8)]の仮想マシン生成で操作可能なサーバプロパティを示す．この例では，Amazon EC2 と OpenStack はともにインスタンスタイプを指定することでクラウド基盤に仮想マシンを生成する高水準 API を持っている．また，OpenStack では高水準 API に加えてインスタンスタイプの詳細を定義できる低水準 API を持ち，仮想マシンのリソース要求を詳細に定義することができる．そのため，高水準 API を分散構築自動化基盤で利用する抽象 API として定義すると，OpenStack などの低水準 API を持つクラウド基盤で，ユーザが要求する仮想マシンのリソース要求を過不足なく設定できなくなる．そのため，クラウド基盤ごとに異なる仮想マシンのリソース要求である Instance Type の要素の共通部分を仮想マシンの構成要素として定義する．そして，クラウド基盤ごとに抽象 API の前処理として，高水準 API を用いるクラウド基盤に対しては仮想マシンのリソース要求から Instance Type を導出する．低水準 API を用いるクラウド基盤に対しては仮想マシンのリソース要求から Instance Type を生成する．また，Amazon EC2 などの Instance Type は仮想マシンのリソース粒度が大きいほどコストが高くなる．そのため，ユーザのリソース要求を満たす最少の Instance Type を導出して仮想マシンの生成を行う必要がある．

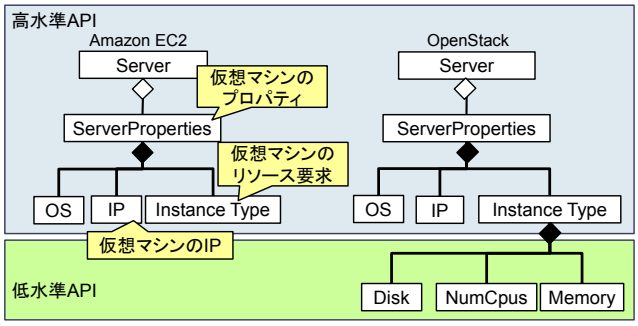

図 3 クラウド基盤の API モデル

Fig. 3 API Model of Clouds

5.2 抽象 API のデータモデル

図 4 に TOSCA の Service Template を拡張し，作成した抽象 API のデータモデルを示す．AmzonEC2 と OpenStack の Instance Type の共通要素からから仮想マシンの構成要素を Server プロパティとして定義した．また，クラウドアプリケーションの分散構築を可能にするため，仮想マシンの構成単位である Tier のプロパティに，クラウド基盤の構成情報を表現できるように拡張した．また，クラウドアプリケーションの構築に用いるスクリプト(以下構築スクリプト)やアプリケーションなどのノードの実体は Definition と共に CSAR (Cloud Service ARchive)ファイルに格納される．

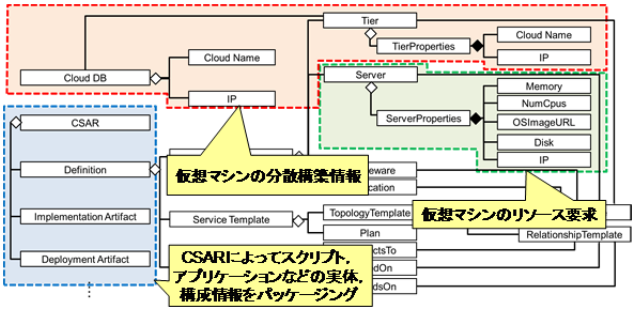

図 4 抽象 API のデータモデル
Fig. 4 Data Model of Abstract APIs

5.3 抽象 API の階層構造

抽象 API はクラウド基盤によらない仮想マシン生成を実現する．以下に抽象 API の階層構造を示す(図 5).

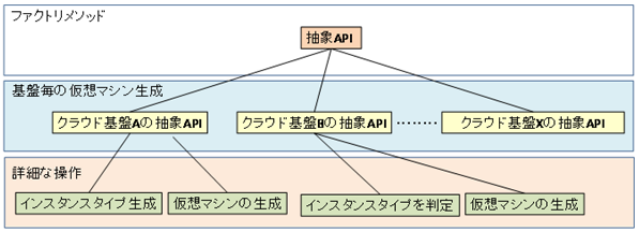

図 5 抽象 API の階層構造
Fig. 5 Hierarchical Structure of Abstract APIs

(1) ファクトリメソッド
ユーザからインスタンス名，サーバ名を受け取り，Service Template から仮想マシンのリソース要求を取得する.

(2) 基盤ごとの仮想マシン生成
ファクトリメソッド内部で呼ばれるハイブリッドクラウドを構成する異なるクラウド基盤ごとの仮想マシン生成 API. 仮想マシンのリソース要求を受け取り，指定したクラウド基盤に対して仮想マシン生成処理を行う.

(3) 詳細な操作
クラウド基盤にアクセスし操作する API 群. クラウド基盤ごとに複数の API を利用して，仮想マシンの生成に必要な操作を行う. 高水準 API を持つクラウド基盤に対しては，予め定義したインスタンスタイプの中から，ユーザのリソース要求を満たす最小リソースの Instance Type を導出し，仮想マシンの構築を行う. また， 低水準 API を持つクラウド基盤に対しては，リソース要求で指定されたリソースを持つ仮想マシンの Instance Type を生成し，仮想マシンを生成する.

6. 分散構築自動化基盤アーキテクチャ

アプローチに基づきアプリケーションを異なるクラウド基盤へ自動的に分散構築する，アプリケーション分散構築自動化基盤アーキテクチャを提案する.

6.1 ブローカによるハイブリッドクラウドの管理と操作

分散構築自動化基盤を実現するためには，ハイブリッドクラウドを構成する分散した異なるクラウド基盤の API を実行し，仮想マシンを生成する必要がある．また，ハイブリッドクラウドを構成するクラウド基盤はユーザごとに異なるため，ハイブリッドクラウドの構成変更に対応できる必要がある．そのため，ブローカにより異なるクラウド基盤に仮想マシンを生成するブローカアーキテクチャを提案する．これにより，分散するクラウド基盤の API の実行とハイブリッドクラウドの構成変更に対する影響を局所化できる(図 6)．この結果，ハイブリッドクラウドを構成するクラウド基盤によらない構成が実現できる.

提案した仮想マシンを生成する抽象 API をブローカで実装する. 抽象 API を用いた Plan をプロセスエンジンから実行することで，ハイブリッドクラウドにアプリケーションの自動分散構築を実現する.

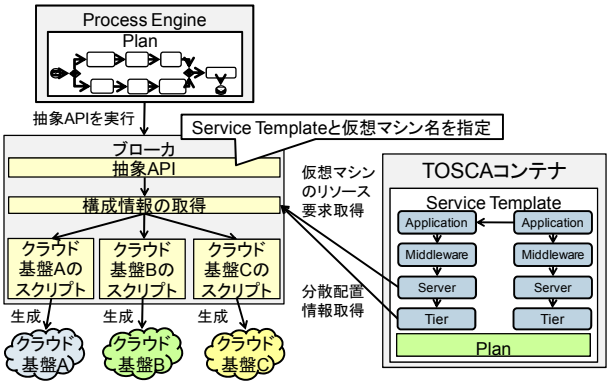

図 6 ブローカアーキテクチャ
Fig. 6 Broker Architecture

6.2 TOSCA コンテナの拡張

TOSCA 仕様ではアプリケーションの分散構築が考慮されていない. 分散構築を実現するため，参照実装の TOSCA コンテナにブローカとクラウド基盤の構成情報を管理する Cloud DB を実装して拡張する. クラウドアプリケーションの構築に必要な定義やアプリケーションの実体を配置する Deploy Manager と割り当て可能なクラウド基盤の情報を管理する Cloud DB を連携し，クラウド基盤の構成情報をブラウザ上に表示する．そして，仮想マシンの構成単位である Tier の情報を Service Template から抽出して，配置可能なクラウド環境へ仮想マシンを配置して，クラウドアプリケーションのインスタンスを生成する.

6.3 分散構築自動化基盤の機能

分散構築自動化基盤のユースケースを示す(図 7)．サービスカタログとは TOSCA 仕様のアプリケーションを提供するシステムである.

(1) Service Template の管理ユースケース
サードパーティが作成したクラウドアプリケーションを管理する. クラウドアプリケーションを構築したいユーザは管理されているアプリケーションを選択し，TOSCA コンテナへアップロードする.

(2) クラウドアプリケーション構築ユースケース

クラウドアプリケーションを構築するユーザは TOSCA コンテナにアップロードされたアプリケーションの構築依頼を TOSCA コンテナに行う．ユーザはアプリケーションを構成する仮想マシンごとにクラウド基盤を選択する．

(3) CSAR の処理ユースケース

サービスカタログからアップロードされた CSAR ファイル内に存在する構成情報やアプリケーション，OS イメージ，構築スクリプトを対応するストレージに格納する．

(4) CSAR 内の成果物を管理ユースケース

TOSCA Processer によって格納されたアプリケーションや OS イメージ，構築スクリプトの登録，検索，配置を行う．

(5) インスタンスの操作ユースケース

TOSCA 仕様によって作成されたクラウドアプリケーションのインスタンスの生成，検索，更新，削除を行う．

(6) Plan の実行ユースケース

TOSCA 仕様のクラウドアプリケーションの構築手順を定義した Plan の実行を行う．Plan の処理の中でインスタンスの操作が行われる．

(7) Implementation Artifact の実行ユースケース

作成した仮想マシンに対する操作を行う．仮想マシンの生成は，抽象 API によって行うため，仮想マシン生成用の構築スクリプトは必要としない．

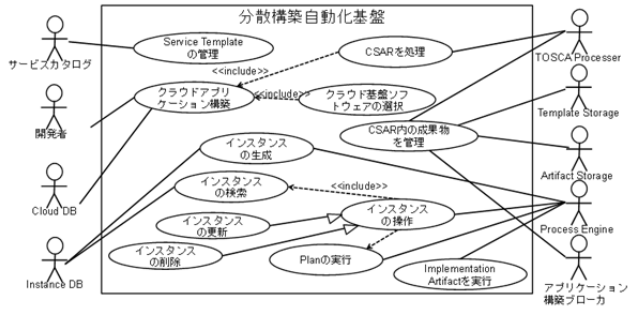

図 7 分散構築自動化基盤のユースケース

Fig. 7 Use Case of Automatic Hybrid-Cloud Building Platform

6.4 分散構築自動化基盤の構造

アプリケーション分散構築自動化基盤を示す(図 8).

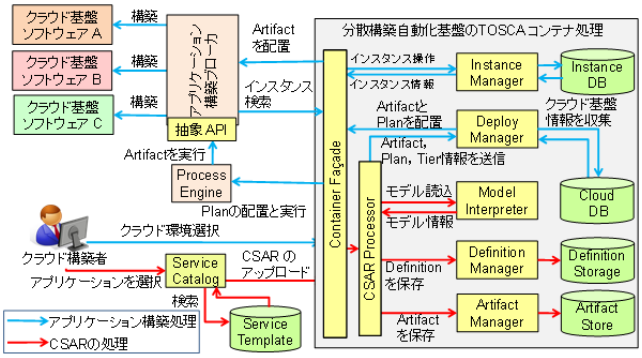

図 8 分散構築自動化基盤のアーキテクチャ

Fig. 8 Architecture of Automatic Hybrid-Cloud Building Platform

(1) サービスカタログ

サードパーティが TOSCA 仕様のアプリケーションを提供するシステムである．開発者はサービスカタログにアクセスし任意のアプリケーションを選択し，TOSCA Processor に渡す．

(2) クラウド構築者

ハイブリッドクラウド上へクラウドアプリケーションを構築するユーザである．

(3) Cloud DB

クラウドアプリケーションを配置可能なクラウド基盤ソフトウェアの名前，IP を管理しているデータベースである．クラウドアプリケーション構築する場合 Cloud DB に登録されているクラウド基盤ソフトウェアを仮想マシンごとに選択する．

(4) Instance DB

TOSCA 仕様のクラウドアプリケーションを構築することで生成されるアプリケーションのインスタンスを管理するデータベースである．Portable API によるインスタンスの生成，検索，更新，削除などの CRUD 処理が反映される．

(5) TOSCA Processer

CSAR ファイルの処理を行い，Definition の処理結果を基にファイル内の成果物を対応するマネージャに受け渡すシステムである．

(6) Definition Storage

Service Template や Node Type，Relationship Type などを含む Definition の管理するストレージである．開発者は Template Storage 内に格納されたアプリケーションのインスタンスを生成できる．

(7) Artifact Storage

TOSCA 仕様のアプリケーションに定義されたアプリケーション，ミドルウェア，OS や構築処理を行うための構築スクリプトを管理するストレージである．

(8) プロセスエンジン

TOSCA 仕様のアプリケーション構築手順を定義した Plan の実行環境で，BPMN[3]などを処理可能なプロセスエンジンを用いる．アプリケーション構築ブローカは，異なるクラウド基盤を操作するシステムである．プロセスエンジンとアプリケーション構築ブローカによりクラウド基盤によらないアプリケーションの自動構築を行う．

6.5 分散構築自動化基盤の振る舞い

CSAR アップロードの振る舞いについて説明する(図 9). クラウドアプリケーションを利用するユーザはサービスカタログなどから，Container Façade を用いて TOSCA コンテナに CSAR のアップロードを行う．TOSCA コンテナにアップロードされた CSAR ファイルは，CSAR Processer に渡される．CSAR Processer は Model Interpreter や Definition Manager，Artifact Manager と連携し，クラウドアプリケーションの構成情報に含まれる Tier の情報を抽出しアーティファクトの登録を行う．抽出した Tier の情報と Deploy

Manager に渡し，仮想マシンを割り当て可能なクラウド基盤の情報と Tier 情報のリストを，クラウドアプリケーションを構築するクラウド構築者のブラウザに表示する．

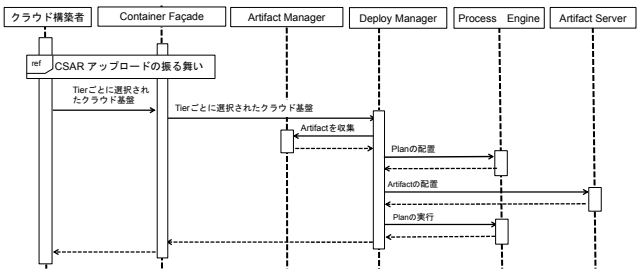

図 9 CSAR アップロードの振る舞い

Fig. 9　CSAR Upload

CSAR アップロードにより，ユーザのブラウザ上に Tier の情報が表示されている．表示される Tier ごとに割り当て可能なクラウド基盤を選択する(図 10)．Tier ごとに選択されたクラウド基盤の情報は Container Façade を通して Deploy Manager に渡される．Deploy Manager は受け取った Tier の情報とクラウド基盤の情報を Instance DB に送信しクラウドアプリケーションのインスタンスを生成する．そして Artifact Manager からアーティファクトの収集を行い，Process Engine やアプリケーション構築ブローカに Plan の配置と Artifact の配置を行う．その後，Plan の実行を行いアプリケーションの自動構築を行う．

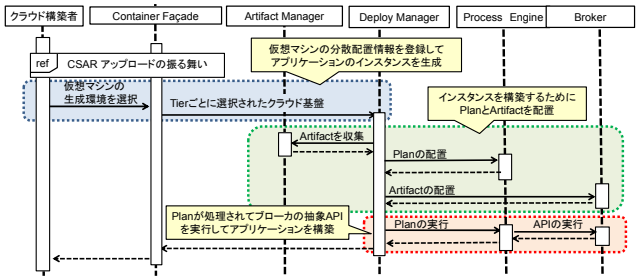

図 10 アプリケーション構築の振る舞い

Fig. 10　Behavior of Building Application

7. プロトタイプの構築

7.1 プロトタイプの目的

(1) 抽象 API の妥当性検証

異なるクラウド基盤からハイブリッドクラウドを構成する．そして抽象 API を実装し，クラウド基盤によらない仮想マシンの生成が可能かを検証する．

(2) ハイブリッドクラウドに対するサーバ構築の効率化検証

異なるクラウド基盤から構築したハイブリッドクラウドへ, TOSCA で記述したクラウドアプリケーションが分散構築可能か検証する．

7.2 プロトタイプの環境

プロトタイプのクラウド環境としてパブリッククラウドに Amazon EC2 を用い，プライベートクラウドに Openstack を用いた．この 2 つのクラウド基盤を用いる理由は仮想マシン生成に利用する API とその抽象度が異なり，Amazon EC2 がパブリッククラウドの IaaS 環境として最も普及していること，そして，OpenStack は TOSCA 仕様によって利用が推奨されているためである．この 2 つの異なる API の差異を吸収する抽象 API を実装し，アプリケーション構築ブローカによって管理する．また，プロトタイプではアプリケーションの構築をプロセスエンジンからではなく，クライアント側で構築手順を定義したスクリプトから実行する．

7.3 プロトタイプの機能

前提条件として, TOSCA コンテナにクラウドアプリケーションが登録され，各種 Artifact の配置が完了した状態とする．以下にプロトタイプのクラウドアプリケーション構築ブローカのユースケースを示す(図 11)．

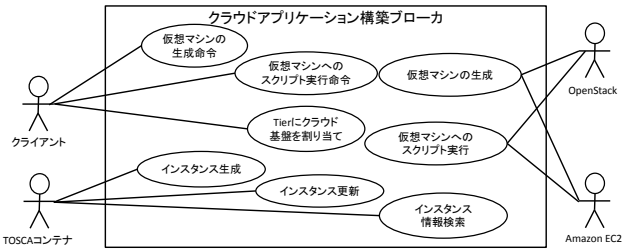

図 11 プロトタイプのユースケース

Fig. 11　Use Case of Prototype

7.4 プロトタイプの構成

作成したプロトタイプの構成を示す(図 12)．プロトタイプは，クライアントで実行された抽象 API に応じてアプリケーションの分散構築を実現する．

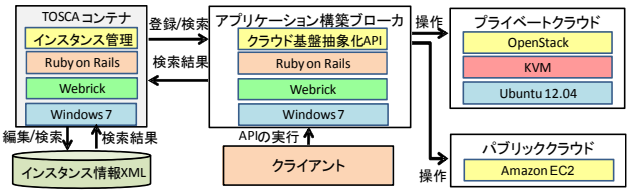

図 12 プロトタイプの構成

Fig. 12　Configration of Prototype

(1) アプリケーション構築ブローカ

Ruby on Rails で実装し WEBrick[10]サーバ上に動作している．アプリケーション構築ブローカは，仮想マシンの配置と仮想マシンへの構築スクリプト実行の抽象 API を Web API として公開している．抽象 API の引数でクラウド基盤の名前を受け取り，クラウド基盤名に応じた処理を実行することで OpenStack や Amazon EC2 に対する処理を実現する．また，仮想マシンへ構築スクリプトを実行する場合には，TOSCA コンテナで管理されるインスタンスの構成情報から構築スクリプト実行先の仮想マシンを判断し実行する．

(2) TOSCA コンテナ

インスタンスの構成情報を管理し，アプリケーション構築ブローカからの問い合わせでインスタンスの構成情報の検索や登録を行う．XML で定義されたアプリケーションのトポロジ情報を走査し構築スクリプトが実行される仮想マシンや Tier の情報を検索するができる．

(3) OpenStack(プライベートクラウド)

アプリケーション構築ブローカから SSH 接続されることにより，仮想マシンの生成と仮想マシンに対する構築スクリプトの実行が行われる．

(4) Amazon EC2(パブリッククラウド)

アプリケーション構築ブローカから，Amazon API ツール [2]と SSH 接続により仮想マシンの生成と仮想マシンに対する構築スクリプトの実行が行われる．

(5) クライアント

アプリケーション構築ブローカにアプリケーションの構築依頼を行う．アプリケーションの構築手順を定義したスクリプトから抽象 API を実行する．

7.5 抽象 API の実装

図 13 に実装した抽象 API のフレームワークを示す．

```
class BrokeractionController < ApplicationController

def CreateVM
  instanceName = params[:IName]      ← 抽象APIの引数
  serverName = params[:SName]

  cloudName = GetParams(instanceName, serverName, "CloudName")
  cloudIP = Getparams(instanceName, serverName, "CloudIP")
  disk = GetParams(instanceName, serverName, "Disk")
  numcpus = GetParams(instanceName, serverName, "NumCpus")
  memory = GetParams(instanceName, serverName, "Memory")
  osimageURL = (instanceName, serverName, "OSimageURL")   ← Service Templateの定義

  if cloudName  == "AmazonEC2" then
     CreateAmazonEC2VM(cloudIP, disk, memory, numcpus, osimageURL)
  elsif cloudName == "OpenStack" then
     CreateOpenStackVM(cloudIP, disk, memory, numcpus, osimageURL)   ← 基盤ごとの仮想マシン生成
  else
   render :text => "#{cloudName} is not define"
  end
end
def ExeScript
```

図 13 抽象 API のフレームワーク

Fig. 13 Framework of Abstraction API

(1) 抽象 API の引数

抽象 API の引数は，クライアントが実行する WebAPI の引数として取得する．insetanceName はアプリケーションのインスタンス名である．また ServerName はアプリケーションのインスタンスに定義されているサーバ名である．

(2) Service Template の定義

クライアントが実行した WebAPI の引数である instanceName と serverName を基に，TOSCA アプリケーションのインスタンスに対応する Service Template から各種パラメータを取得する．cloudName は，指定された serverName の仮想マシンが生成されるクラウド基盤名，cloudIP は cloudName を持つクラウド基盤の IP である．disk，numcpus，memory，osimageURL は仮想マシンを生成するために必要なストレージ，メモリ枚数，メモリ容量，OS イメージである．

(3) 基盤ごとの仮想マシン生成

クラウド基盤へ仮想マシンの生成を行うメソッドである．cloudName のパラメータを基に分岐させ，ブローカのクライアントごとに利用しているクラウド基盤に対する仮想マシン生成メソッドを記述する．プロトタイプでは，AmazonEC2 と OpenStack を用いるため 2 つの仮想マシン生成メソッドを作成した．クラウド基盤を追加する場合は，対応する条件分岐を追加して仮想マシン生成メソッドを作成する．

7.6 クラウドアプリケーションのインスタンス管理

プロトタイプでは，インスタンスの管理に XML ファイルを用いた(図 14)． TOSCA コンテナに存在するアプリケーションのインスタンス名を定義し，Tier 情報にクラウド基盤情報を割り当てたときにクラウドアプリケーションのインスタンスが生成される．これにより，Tier 情報にクラウド基盤情報を持つ，定義したインスタンス名の XML ファイルが作成される．このインスタンスの XML ファイルに対してカスタマイズや，仮想マシン生成時に発生する情報を登録する．

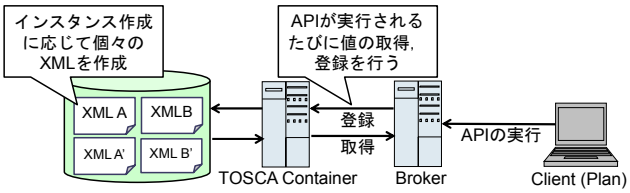

図 14 インスタンス管理

Fig. 14 Instance Management

7.7 インスタンスの情報検索

インスタンスの情報検索は，TOSCA コンテナに格納されるインスタンスの XML ファイルから行う(図 15)．

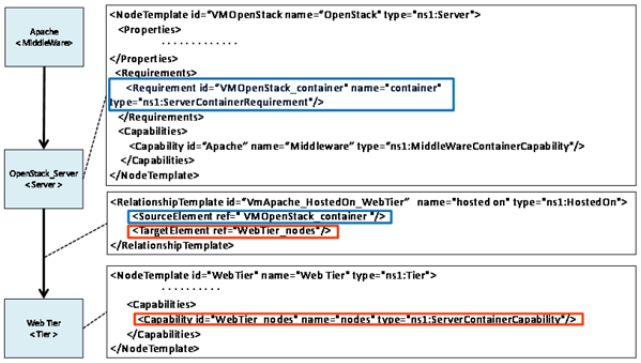

図 15 インスタンスの情報検索

Fig. 15 Instance Information Retrieval

仮想マシンの生成を行う抽象 API が実行されると，XML ファイルから Tier の情報を検索して割り当てられたクラウド基盤の情報を取得する．そして，そのクラウド基盤に対して仮想マシンの生成を行う．仮想マシンに対してミドルウェアやアプリケーションなどのインストールを行う構築スクリプトが実行されると，XML ファイルのトポロジ情報から，構築スクリプトが実行されるべきサーバの IP アドレスを検索する．その後，SSH を用いて構築スクリプトの実

行を行う．この情報検索方法により，Service Template に定義された構成情報を取得してアプリケーションを構築する．

7.8 クラウド基盤への仮想マシンの生成と操作

仮想マシンの生成はクラウド基盤に SSH 接続することで実行する．そのため，ブローカはクラウド基盤の IP アドレスを登録しておく必要がある．プロトタイプでは，OpenStack の IP アドレスを基に SSH 接続を行い仮想マシンの生成を行う．Amazon EC2 では，ブローカで Amazon の API ツール認証を行っておき仮想マシンの生成を行う．また，各クラウド基盤とも仮想マシンの生成後に，互いが連携できるよう外部からアクセスが可能になるようにネットワークの設定を行う．

8. プロトタイプの適用

8.1 適用シナリオ

作成したプロトタイプを用いて OpenStack と AmazonEC2 に Apache をインストールし，作成した仮想マシンが外部からアクセス可能であるかを確認した．適用シナリオについて説明する(図 16).

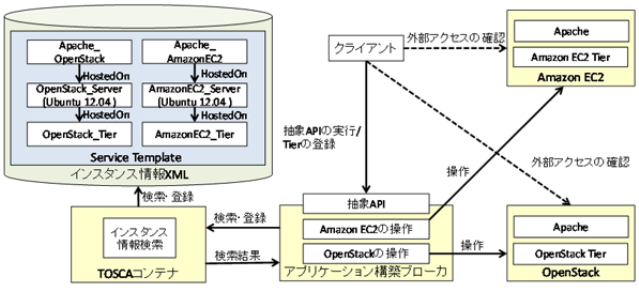

図 16 適用シナリオ

Fig. 16 Scenario

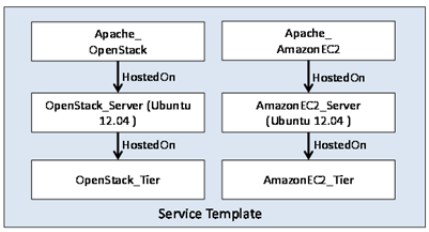

図 17 シナリオのサービステンプレート構成

Fig. 17 Example of Service Template

インスタンス情報 XML には，図 17 で示した構成の Service Template がアップロードされている．クライアントはブローカを通して Service Template の OpenStack_Tier と AmazonEC2_Tier のそれぞれに，OpenStack と Amazon EC2 を割り当てた．その後，抽象 API を呼び出し OpenStack 上に OpenStack サーバの構築と Apache の配置を行った．また，AmazonEC2_Tier 上に AmazonEC2_Server を構築し Apache の導入を行った．この後クライアントから，構築した仮想マシンに対して外部からアクセス可能かどうか確認した．

8.2 適用シナリオの振る舞い

作成したプロトタイプを用いて OpenStack と Amazon EC2 に仮想マシンの配置と Apache の配置し，各仮想マシンに外部からアクセスが可能か確認した．以下に仮想マシンの配置と仮想マシンへの構築スクリプト実行を行うシナリオの振る舞いを説明する(図 18).

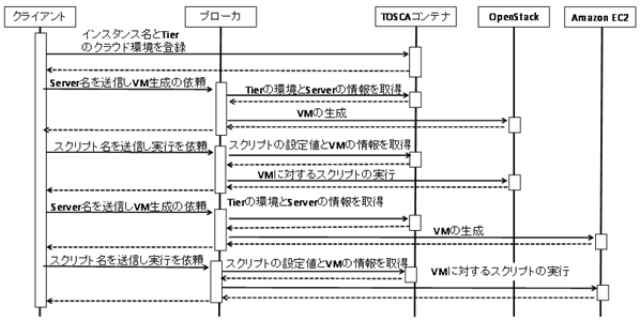

図 18 シナリオの振る舞い

Fig. 18 Behavior of Scenario

クライアントはインスタンス名と Tier ごとのクラウド基盤名をブローカに渡し，クラウドアプリケーションのインスタンス生成を依頼する．ブローカはクライアントから受け取ったインスタンス名と Tier ごとのクラウド基盤名を TOSCA コンテナに渡す．TOSCA コンテナは，受け取った値からクラウドアプリケーションのインスタンスを作成する．その後，仮想マシンの生成を行う抽象 API にサーバ名を指定し，仮想マシンの生成を依頼する．ブローカは TOSCA コンテナに対してサーバが属するクラウド基盤の検索依頼をする. TOSCA コンテナはトポロジ情報からサーバのクラウド基盤名を取得しブローカに返す．ブローカは受け取ったクラウド基盤名を基に仮想マシンの操作を行う．これにより，OpenStack 上に仮想マシンを生成する．その後，クライアントは Apache のインストールを行う構築スクリプト名を指定する．構築スクリプト名は，仮想マシンごとに一意に決まるように定義する．ブローカは受け取った構築スクリプト名から TOSCA コンテナに対して構築スクリプトをどのサーバに実行するのかを検索依頼する．TOSCA コンテナは受け取った構築スクリプト名を基にトポロジ情報からサーバの情報を検索し，検索結果をブローカに渡す. ブローカは SSH 接続を行い OpenStack に Apache のインストールを行う．同様の作業を Amazon EC2 に対しても行う．これにより，仮想マシンの生成と仮想マシンに対する構築スクリプトの実行及び，生成した仮想マシンへ外部からアクセス可能であることを確認した．

9. 評価

9.1 抽象 API の評価

(1) 抽象 API の機能要求の実現性

ハイブリッドクラウドにアプリケーションを分散構

築するためには，サーバのリソース要求を満たす仮想マシンを生成し，外部からアクセスできる環境構築が必要である．これに対して，抽象 API は Service Template にリソースの要求を設定した構成情報から仮想マシンの生成を実現し，外部からアプリケーションを利用，アプリケーション間で連携するための外部アクセス可能な IP アドレスの割り当てを実現する．

(2) 抽象 API の妥当性

仮想マシンの生成を行う抽象 API の引数をクラウドアプリケーションのインスタンス名とサーバ名に抽象化し，抽象 API 実行後に Service Template から詳細なパラメータを取得する．詳細なパラメータは低水準 API の引数に対応しているため，OpenStack では詳細なインスタンスタイプを生成でき，Amazon EC2 ではユーザの要求を満たす最小のインスタンスタイプを導出する．Amazon EC2 はインスタンスタイプに応じてコストが発生するため，最小のインスタンスタイプを割り当てることでコストを抑えることができる．この結果，クラウド基盤ごとに異なる API のシグネチャとその抽象度を吸収し，クラウド基盤によらない仮想マシンの生成ができるため，抽象 API は妥当である．

9.2 ハイブリッドクラウドに対するサーバ構築の効率化

TOSCA により異なるクラウド基盤間でアプリケーションの構成定義を統一した．そして，ブローカにより異なるクラウド基盤へ透過的に仮想マシンの生成が行え，クラウド基盤ごとの仮想マシンを生成するスクリプトが不要になった．また，Plan からアプリケーションの自動構築が可能である．この結果，従来のクラウド基盤ごとに異なる API を操作しアプリケーションを構築する方法と比べ，サーバ構築業務の効率向上が期待できる．

9.3 提案アーキテクチャの妥当性

ブローカにより，ハイブリッドクラウドを構成するクラウド基盤の仕様変更や，異なるクラウド API を持つクラウド基盤の追加に対する変更を局所化できる．この結果，クラウドアプリケーションを利用するユーザごとに異なるクラウド基盤を組み合わせ，ハイブリッドクラウドを容易に構築できる．

10. 考察

TOSCA を用いたハイブリッドクラウド上でのアプリケーション分散構築の自動化基盤アーキテクチャにより，クラウド基盤によらないアプリケーションの構築が可能になる．この結果，従来のハイブリッドクラウド構築ではクラウド基盤を統一するため最適な SLA を持つクラウド基盤の選択が困難であったが，提案アーキテクチャによりユーザ自身が運用するプライベートクラウドとは異なるパブリッククラウドやコミュニティクラウドからハイブリッドクラウドを容易に構築することができる．

作成したクラウドアプリケーションは仮想マシン間で連携できるように外部アクセスが可能な設定にされている．クラウドアプリケーションのテスト環境として利用する上では問題ないが，運用する場合はアクセス制限などのセキュリティの設定を行う必要がある．

11. 今後の課題

11.1 異なるクラウド基盤間の設定を同期

クラウド基盤では，仮想マシンのインスタンスタイプやセキュリティグループなど，様々な要素をあらかじめ設定することが可能である．保守や運用で設定値の差異によって混乱が生じないように異なるクラウド基盤間で設定を同期する必要がある．

11.2 クラウドアプリケーションのインスタンスを移植

ユーザの要求するクラウド環境を構築するために，クラウドサービスの停止や安価な新規サービスの提供に対して，構築したクラウドアプリケーションのインスタンスが容易に移植できることが望ましい．そのため，インスタンス運用時に記録したデータなどを抽出し，異なるクラウド環境へ自動的にインスタンスを構築する機能を追加する必要がある．

12. まとめ

TOSCA による構成管理方法の統一と，クラウド基盤ごとに異なる API の差異を吸収する抽象 API を実装したブローカにより，クラウド基盤によらないアプリケーションの構築を可能にした．この結果，Plan によって，ハイブリッドクラウドへクラウドアプリケーションを自動構築できる．提案したアーキテクチャのプロトタイプを実装し，抽象 API の妥当性，ハイブリッドクラウドに対するサーバ構築の効率化，提案アーキテクチャの妥当性を評価した．

参考文献

1) Amazon, Amazon EC2 (Amazon Elastic Compute Cloud), http://aws.amazon.com/jp/ec2/.
2) Amazon, Amazon EC2 API Tools: Developer Tools: Amazon Web Services, http://aws.amazon.com/developertools/351.
3) T. Allweyer, BPMN2.0, Bod,2010.
4) R. Buyya, et.al., Cloud Computing: Principles and Paradigms, WILEY,2011, pp. 393-411.
5) 林 雅之, オープンクラウド入門, impress R&D, 2012.
6) M. Marschall, Chef Infrastructure Automation Cookbook, Packt Publishing, 2013.
7) OASIS, Topology and Orchestration Specification for Cloud Applications Version 1.0, Nov. 2013, http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.html.
8) OpenStack, Open Source Cloud Computing Software, http://www.openstack.org/.
9) B. Sotomayor, et.al., Virtual Infrastructure Management in Private and Hybrid Clouds, IEEE Internet Computing, Vol.13, No. 5, 2009, pp.14-22.
10) D.Thomas, et.al., Agile Web Development with Rails (Pragmatic Programmers), Pragmatic Bookshelf, 2006[前田修吾(監訳), Rails によるアジャイル Web アプリケーション開発,オーム社,2006].