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

M2013SE010　岡田　幸大

指導教員　青山　幹雄

1. はじめに

近年，複数のクラウド基盤を連携して運用するマルチクラウドが注目されている．そのため，複数のクラウド基盤を管理できるマルチクラウド管理ツールが必要となっている．また，クラウドの環境構築を自動化する構成管理ツールが提供されている．しかし，クラウド基盤ごとにクラウドを構成する要素が異なり，クラウドイメージの管理方法が異なるため複数のクラウド基盤上に同一方法で構成を定義してアプリケーションを配置することは困難である．

本稿ではマルチクラウド管理ツールである Scalr に基づき，同一方法で複数のクラウド基盤へクラウドの構成定義を可能にする．これにより，マルチクラウド上へのクラウドの構築とアプリケーションの配置を自動化する方法を提案する．

2. 研究課題

2.1. クラウド基盤ごとの構成要素の差異

クラウド基盤ごとに，クラウドを構成するために用いるAPIとその引数が異なる．マルチクラウド上にクラウドを構築するためには，各クラウド基盤の構成の定義に用いる構成を確認し，適切な値を適用して構成を定義する必要がある．

2.2. クラウドイメージの管理方法の差異

クラウド基盤ごとにクラウドを構築するためのイメージファイルの管理方法が異なる．そのためクラウド基盤ごとにアプリケーション配置の振る舞いが異なる．従って，任意のクラウド基盤上へのアプリケーションの自動配置が困難である．

3. 関連研究

3.1. Scalr[3]

Scalr は複数のクラウド基盤を管理するマルチクラウド管理ツールである．クラウドの構成をDBに定義することで，クラウドの構築を自動化できる．クラウドサービス全体の定義は Farm，クラウドの構成は Farm Role に定義する．アプリケーションの役割は Role に，対応するクラウドイメージは Role Images にそれぞれ定義する[4]．Farm を実行すると，定義された構成を抽出してクラウドを自動構築する．

3.2. Chef[5]

Chef はクラウドの環境構築を自動で行う構成管理ツールである．クラウドの環境を Recipe と呼ばれる Ruby ファイルで記述しておき，Recipe を実行することで，適用したサーバの環境を自動構築する．

4. アプローチ

クラウド基盤ごとに異なる構成要素とアプリケーションを配置する振る舞いを抽象化することで，同一のインタフェース上から各クラウド基盤に対応した構成を定義する．クラウドの構成は抽象 API を定義する．アプリケーション配置の振る舞いは，クラウド構築者によって選択されたアプリケーションに対応するイメージファイル(イメージ ID)と Recipe を抽出する．これにより，同一方法でマルチクラウドの構成定義と，アプリケーションの自動配置を可能にする(図 1)．

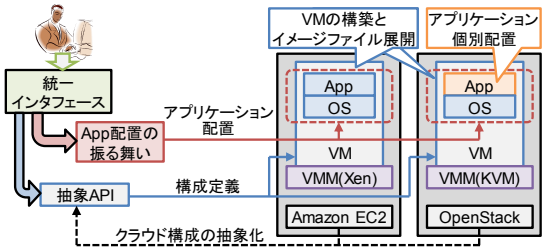

図 1 アプローチ

5. 構成の抽象化

5.1. クラウド構成の抽象化

クラウド基盤ごとに異なる構成要素を抽象化して定義する(図 2)．Scalr でクラウドを自動構築するためには，Farm Role にクラウドの構成を定義する必要がある．Farm Role には構築する対象のクラウド基盤やアプリケーションの役割を示す RoleID などを定義する．詳細な構成は Farm Role Settings に定義する．クラウドの構成を定義するための API 名は name に記述し，引数名は value に記述する．

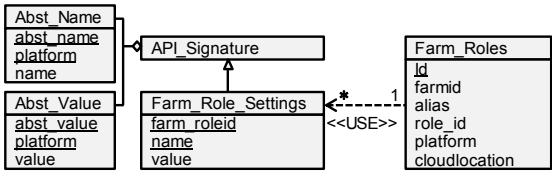

図 2 クラウド構成抽象化

5.1.1. API 名の抽象化

Farm Role Settings で定義される API 名は，各クラウド基盤で共通に用いられる API 名と，クラウド基盤ごとに異なる API 名に分類される．共通な API 名は全てのクラウド基盤の定義に利用できる．異なるAPI名は抽象化して定義す

ることで，各クラウド基盤に対応した API 名を抽出する．

5.1.2. API の引数の抽象化

クラウド基盤ごとに API 名が異なると，適用する引数や抽象度が異なる．クラウド基盤ごとに異なる引数名を抽象化し，各クラウド基盤に同一の構成になる抽象 API を定義する．インスタンスタイプを例にして抽象 API を示す(図 3)．

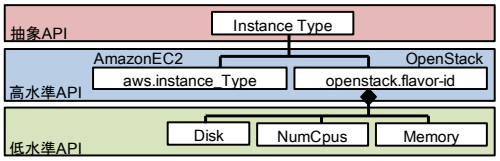

図 3 インスタンスタイプの抽象化

5.2. アプリケーション配置の振る舞いの抽象化

アプリケーション配置の振る舞いを抽象化し，同一方法で異なるクラウド基盤へアプリケーションを自動配置する．Amazon EC2[1]ではイメージファイルとして，あらかじめ様々なアプリケーションが配置されているので，Scalr の Role で定義されている OS とアプリケーションの役割から，対応するイメージ ID を取得できる．OpenStack[2]では多数のイメージファイルを管理することが困難なので，あらかじめベースとなる OS のイメージファイルのみを配置する．OS のイメージ ID とアプリケーションを配置する Recipe 名を抽出して Farm Role に定義することで，クラウドの構築とアプリケーションの配置の自動化を可能にする(図 4)．

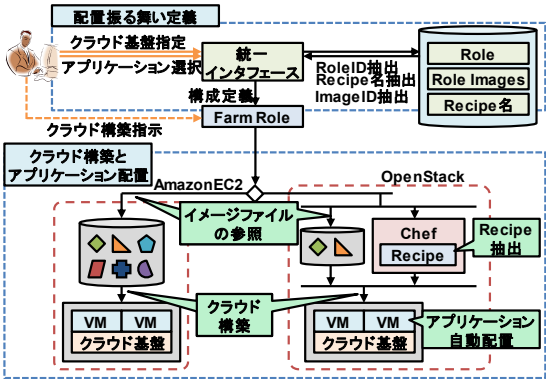

図 4 アプリケーション構成の抽象化

6. アプリケーション構築自動化アーキテクチャ

6.1. アーキテクチャのユースケース

提案する構築アーキテクチャのユースケースを示す(図 5)．アクタはクラウド構築者，Scalr，クラウド基盤，Chef である．クラウド構築者は Scalr を用いて新しい Farm の生成と Farm Role を登録する．Farm Role を登録するためには，クラウドの構成の定義や，配置するアプリケーションの指定をする必要がある．Scalr を用いて Farm を実行すると，Farm Role に定義された構成をもとに，クラウド基盤に対してクラウドの構築を指示する．また抽出した Recipe 名から，Chef に保存された Recipe を取得する．Recipe を適用することにより，自動的にアプリケーションを配置する．

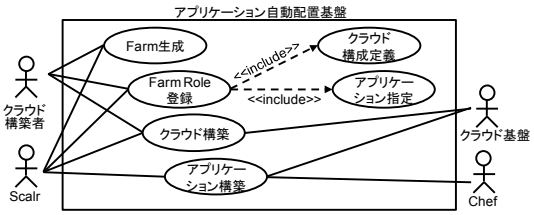

図 5 アーキテクチャのユースケース

6.2. アーキテクチャの構造

提案アーキテクチャの構造を示す(図 6)．Scalr はクラウド構築者からの指示に従ってクラウドを構築する．クラウドの構成を管理するための情報はデータベース上に保存され，新しいクラウドの構成を定義することで，クラウドの自動構築を可能にする．データベース上には，抽象 API とアプリケーションの役割に対応した Recipe 名を定義して作成する．Chef には環境構築をする Recipe を作成する．そして，クラウド構築者から選択された構成をもとに，各クラウド基盤に対応した構成を抽出して Farm Role に定義する Farm 生成プログラムを Scalr 上に作成する．Farm を実行すると，Farm Role に定義された構成を抽出する．抽出した構成を用いてクラウドを構築し，イメージ ID と Recipe によってアプリケーションを自動配置する．

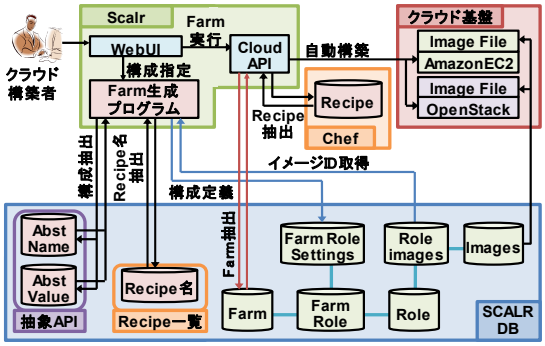

図 6 アーキテクチャの構造

6.3. アーキテクチャの振る舞い

提案アーキテクチャの振る舞いを示す(図 7)．クラウドの構築者は，Farm 生成プログラムを Web UI から利用する．クラウドサービス全体を定義する Farm を生成後，クラウドの構成を定義する Farm Role を登録する．クラウドの構成と構築する対象となるクラウド基盤，配置するアプリケーションを指定する．各クラウド基盤に対応した API 名とその引数を抽出する．また，Amazon EC2 であれば，選択した OS とアプリケーションの組み合わせとなるイメージ ID を抽出する．OpenStack であれば，ベースとなる OS のイメージ ID と，配置するアプリケーションを自動構築する Recipe 名を抽出する．抽出した構成を Farm Role に定義し，Farm

に登録する. 複数のクラウドを構築するには Farm Role の登録を複数回実行する. Farm を実行すると, 登録された Farm Role から構成要素を抽出し, また Open Stack であれば Recipe も抽出し, クラウドを自動構築する.

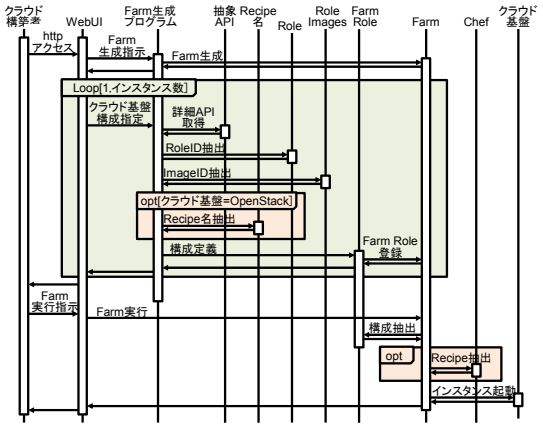

図 7 アーキテクチャの振る舞い

7. プロトタイプの開発

提案アーキテクチャのプロトタイプを開発した. プロトタイプ実行の前提作業として, Amazon EC2 のイメージファイル一覧が記録された Role を取得することと, Open Stack 上に Ubuntu12.04 のイメージファイルを配置し, Scalr のデータベースにイメージ ID を登録する.

7.1. プロトタイプの実行環境

プライベートクラウドは OpenStack 上に構築する. パブリッククラウドは Amazon EC2 上に構築する. クラウド管理ツールは Scalr を構築して利用する. 構成管理ツールは Chef を構築して利用する.

7.2. プロトタイプの実装

7.2.1. データベースの各要素の定義

(1) API 名の定義

クラウド基盤ごとに異なる API 名を抽象化して定義する. インスタンスタイプとセキュリティグループを抽象化して定義した. 抽象 API 名とクラウド基盤を指定することで, 各クラウド基盤に応じた API 名を抽出する.

(2) インスタンスタイプの定義

クラウド基盤ごとにインスタンスタイプの引数名が異なるため抽象化して定義する. 各クラウド基盤によって異なるリソース量を定義することで, 同一のリソース量を持つようにクラウドの構成を定義する. なお OpenStack はリソース量を詳細に定義可能なため, Amazon EC2 で提供されるリソース量と統一する.

(3) Recipe の定義

アプリケーションを自動配置する Recipe を作成する. クラウド構築者によって指定されたアプリケーションの役割に対応する Recipe を抽出できるように定義する. Recipe を適用することでアプリケーションを自動配置する.

7.2.2. Web アプリケーションの作成

定義したデータベースを利用して, マルチクラウド上へクラウドの構成を定義してアプリケーションを自動配置する Web アプリケーションを PHP で作成した. アプリケーションの規模は 527(LOC)である.

(1) Farm の生成

生成する Farm 名を入力し, REST API を用いて新しい Farm を生成する.

(2) Farm Role の構成定義

クラウドを構築するための構成を選択することで, Farm Role に定義する要素を抽出する. 構築する対象のクラウド基盤, OS のバージョンや配置するアプリケーション, インスタンスタイプなどを選択する.

(3) Farm Role の登録

抽出した値を用いて, Farm に対して Farm Role を登録する. REST API を用いて登録する.

7.3. プロトタイプの構造

プロトタイプの構造を示す(図 8). Scalr は OpenStack と Amazon EC2 から提供される ID, パスワードを用いて認証し, Scalr から VM を管理可能な状態にする. また Chef と認証することで, Recipe を抽出可能な状態にする. Open Stack 上では VM を構築するために必要な仮想ネットワークを構築する. クラウドの構築者は, Scalr を用いて Farm の生成や FarmRole を登録する. 生成した Farm を実行することで, 定義された Farm Role の構成とイメージ ID, Chef の Recipe を抽出してクラウドの構築とアプリケーションを配置する. 構築したクラウドにアクセスし, アプリケーションが利用可能になる.

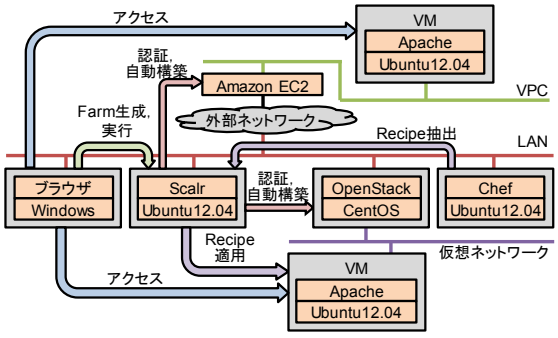

図 8 プロトタイプの構造

8. プロトタイプの適用

8.1. 例題の概要

プロトタイプを用いて Farm を生成し, 各クラウド基盤に Apache を配置する Farm Role を登録する. これにより, OpenStack と Amazon EC2 上に, Ubuntu12.04 と Apache がインストールされたクラウドを自動構築する.

8.2. クラウド構成の定義

作成したプロトタイプを用いて定義された Farm Role と Farm Role Settings の詳細を示す(表 1). Amazon EC2 では選択した OS とアプリケーションを示すイメージ ID を抽出した. OpenStack では選択した OS からベースとなる OS のイメージ ID を抽出し, アプリケーションの役割から Recipe 名を抽出した.

表 1 定義された構成

	クラウド基盤	Amazon EC2	OpenStack
Farm Role	Farm Role ID	77	78
	Alias	App-ec2	App-os
	Role ID	38355	38240
	Platform	ec2	openstack
	Cloud Location	ap-northeast-1	RegionOne
	Image ID	ami-7876c679	bafc5451-895d-4136-96b3-5873dcba8e3f
Farm Role Settings	scaling.enabled	1	1
	scaling.max_instances	1	1
	scaling.min_instances	1	1
	scaling.polling_interval	2	2
	system.timeouts.launch	9600	9600
	system.timeouts.reboot	9600	9600
	chef.bootstrap		1
	chef.environment		_default
	chef.log_level		auto
	chef.runlist		["recipe[app::default]"]
	chef.server_id		2
	aws.instance_type	t1.micro	
	aws.security_groups.list	["default","scalr.i7xe1381ccd.ip-pool"]	
	openstack.flavor-id		6
	openstack.ip-pool		83c9e17c-445f-45f6-a9f5-35a89dbe7d44
	openstack.networks		["dd566ff8-45a3-4983-89fa-ab8795e1f816"]
	openstack.security_groups.list		["default","scalr.i7xe1381ccd.ip-pool"]

9. 評価

9.1. アプリケーションの自動配置

Amazon EC2 上と OpenStack 上に配置したアプリケーションにアクセスできることを確認した. これにより, 定義した抽象 API の妥当性を確認した.

9.2. アプリケーション配置の時間削減

アプリケーション配置の振る舞いを抽象化し, マルチクラウド上へのアプリケーションの配置を自動化した. これにより, アプリケーションが利用できるようになるまでの時間を短縮できた. 手作業は 3 分 10 秒から 1 分に短縮し, 削減率は 68%になった(表 2).

表 2 手動作業の削減率

作業内容	従来方法	提案方法	削減時間	削減率
Farm 生成	1 分 30 秒	15 秒	-30 秒	33%
Farm Role 登録		45 秒		
アプリケーション構築	1 分 40 秒	0 秒	-1 分 40 秒	100%
合計	**3 分 10 秒**	**1 分**	**-2 分 10 秒**	**68%**

9.3. Farm Role 生成のプロセスの削減

抽象 API を用いることで, 同一方法でマルチクラウドに対応した構成定義が可能になった. そのため, 複数のクラウド基盤に構築する際の作業プロセスを削減した. 同一構成でクラウドを構築する場合は, Farm Role の定義回数は 1 回のみで, マルチクラウド上へ自動配置が可能になる.

10. 考察

10.1. 同一構成による設定の誤りの削減

提案アーキテクチャでは, 同一方法でマルチクラウド基盤上にアプリケーションを配置することを可能にした. そのため, マルチクラウド基盤に同一の構成で定義する際に, 構成定義の設定の誤りの削減につながる.

10.2. アプリケーション数の増加の対応

マルチクラウド上に配置するアプリケーション数が多くなる場合や, アプリケーションの配置に必要な設定項目が増加する場合, アプリケーション配置時間が手作業に比べてさらに短縮できると考える.

11. 今後の課題

11.1. クラウド構成の変更管理

提案アーキテクチャはマルチクラウド上に同一方法で構成定義ができるが, 構成の変更管理はできない. クラウド構築者によって, 変更管理を可能にする必要がある.

11.2. 定義可能な構成の追加

プロトタイプではネットワーク構成や外部ストレージの利用の定義が行えない. そのため抽象 API を拡張することにより, 妥当性を確認する必要がある.

11.3. 柔軟なクラウドの構成定義

マルチクラウド上のアプリケーション配置は, 同一の構成に限定される. クラウドの構成と配置するアプリケーションの制約がない柔軟な定義を可能にする必要がある.

12. まとめ

本稿では, クラウド基盤ごとに異なる構成やアプリケーション配置の振る舞いを抽象化して定義した. これにより, 同一方法でクラウドの構成を定義し, マルチクラウド基盤上へアプリケーションの配置を可能とするアーキテクチャを提案した. 提案アーキテクチャのプロトタイプを構築し, 同一方法で Amazon EC2 と OpenStack に対応した構成を抽出して定義した. 定義した構成を用いて, マルチクラウド基盤上にクラウド構築とアプリケーション配置を自動化した.

参考文献

[1] Amazon, Amazon EC2 (Amazon Elastic Compute Cloud), http://aws.amazon.com/jp/ec2.

[2] K. Jackson, OpenStack Cloud Computing Cookbook, Packet Publishing, 2012.

[3] Scalr Enterprise Cloud Management Platform, http://www.scalr.com.

[4] Scalr Wiki, https://scalr-wiki.atlassian.net/wiki/display/docs/Home.

[5] 吉羽 龍太郎ほか, Chef 実践入門, 技術評論社, 2014.