Chef を応用した Kubernetes のコンテナ管理方法の提案と評価

M2015SE016　土本 大貴

指導教員　青山 幹雄

1. はじめに

サーバ仮想化を実現する新たな方法としてコンテナ型仮想化である Docker が注目を集めている. しかし, コンテナの配置, 操作には, ホスト OS を経由する必要がある. これに対処するためコンテナ管理ツールである Kubernetes が提案されている. しかし, 設定ファイルを作成するためには, Docker と Kubernetes 双方のモデルを理解する必要がある.

本稿では Chef と Kubernetes の管理モデルを対応づけ, Chef で作成した設定ファイルを Kubernetes で利用可能にするコンテナ管理方法を提案する.

2. 研究課題

本研究では, Chef と Kubernetes を連携させ, コンテナ管理を容易にし, 効率化するための方法を提案する. 具体的には以下の 2 つを研究課題とする.

(1) Kubernetes と Chef の適用範囲の明確化
(2) Chef で Kubernetes の設定ファイルを生成する方法の提案

3. 関連研究

3.1. Kubernetes[3, 4]

Kubernetes は複数のホストにまたがって Docker[2] コンテナを配置し, スケジューリングするコンテナ管理ツールである[5]. Kubernetes 独自の概念として Pod, Replication Controller, Service の 3 つがあり, サービスとして提供されている[5].

3.2. Chef[1, 7]

Chef はサーバ構築の自動化を行う構成管理ツールである[1, 7]. 各構成要素の役割や環境を記述した構成ファイルを作成し, 実行手順をコードとしてまとめた Recipe でサーバ構築を行う. Chef には Infrastructure as Code[6]の考え方があり, 一度作成したコードは再利用できる.

4. アプローチ

本研究では, 構成管理ツールである Chef の構成要素を利用し, Kubernetes の設定ファイルを生成するコンテナ管理方法を提案する(図 1).

クライアントは Chef の構成ファイルに Kubernetes の設定を記述し実行する. Chef では, クライアントの要求を基に Kubernetes で使用するための設定ファイルを生成し, Kubernetes の Master サーバに設定ファイルを送信する. Master は Chef から送信された設定ファイルから要求プロセスを実行し, Node サーバにサービスを適用する. これにより, 管理者が Chef の考え方で各構成ファイルにリソースを記述し, Kubernetes のコンテナ管理を容易にする.

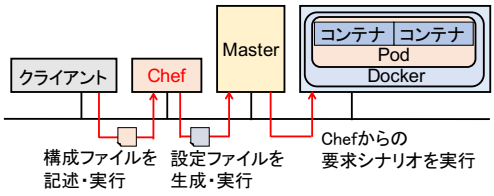

図 1 アプローチ

5. 提案方法

5.1. コンテナ管理方法

コンテナ管理を実現するための 4 つのプロセスを示す(図 2).

(1) クライアントは Chef の各構成ファイルに設定を記述し, 構成ファイルを実行する.
(2) Chef は Kubernetes のコンテナ管理に必要な設定ファイルを生成し, Master に送信する.
(3) Master は設定ファイルの情報を取得し, Node に対して Kubernetes のサービスを行うコンテナ管理を行う要求プロセスを実行する.
(4) Node は Master の要求プロセスを実行する.

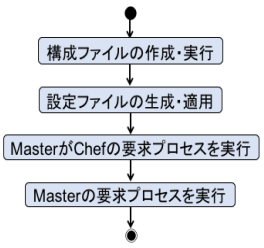

図 2 コンテナ管理方法

5.2. 提案アーキテクチャ

提案アーキテクチャの構成を示す(図 3). 以下の 4 つのプロセスを行うコンテナ管理方法を提案する.

(1) クライアントは各構成ファイルに設定を記述し,

Node の構成ファイルを実行する.

(2) Node から Role と Environment の構成ファイルを実行する. そして, Recipe に各構成要素で設定した値を集約する.

(3) template の構成ファイルに Recipe に集約したハッシュ値を代入する. 設定ファイルを生成し, Kubernetes の Master に送信する.

(4) Master は設定ファイルの情報を取得し, Node に対して Kubernetes のサービスを行うコンテナ管理を実現する.

本研究は Kubernetes の Pod, Replication Controller, Service の 3 種類のサービスを Chef と Kubernetes を連携して実行する.

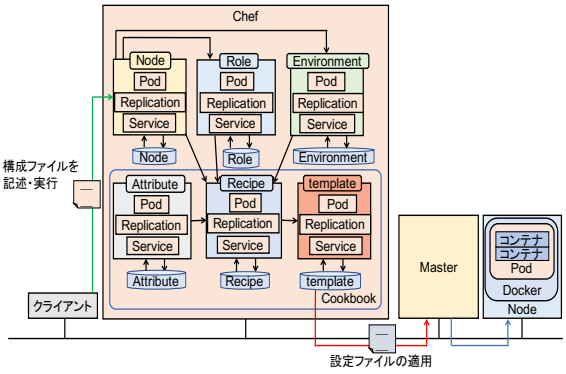

図 3 提案アーキテクチャ

6. 構成管理モデル

6.1. Chef と Kubernetes の構成管理モデル

Chef の各構成要素と Kubernetes の設定ファイルのフィールドの構成管理モデルとを対応づけで, Chef の構成管理モデルで Kubernetes のコンテナ管理を実現する.

Chef の各構成ファイルは Attribute という構成情報を定義する. 各構成要素に役割や環境など記述できる. 役割や環境は Role や Environment ごとに記述する要素が異なる. また, 各構成要素に優先度の関係がある.

Kubernetes では, 設定ファイルの各フィールドによって Pod などのサービスの役割や環境を記述する要素が異なる.

6.2. Kubernetes のフィールド管理モデル

Kubernetes で Pod, コンテナ配置などのサービスを提供するために必要な設定ファイルのフィールド管理モデルを Pod, Replication Controller, Service の 3 種類のサービスごとに作成する. 例として Pod のフィールド管理モデルを示す(図 4).

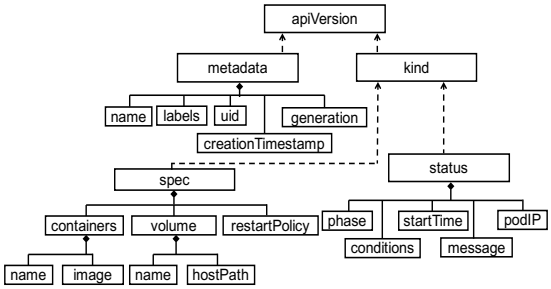

図 4 Pod のフィールド管理モデル

6.3. Chef の構成管理モデル

Chef の構成要素を連携させた管理モデルを示す(図 5).

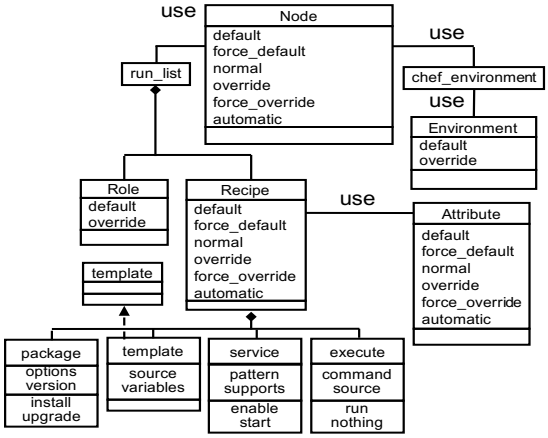

図 5 Chef の構成管理モデル

6.4. 構成管理モデルの対応づけ

作成した Chef と Kubernetes の構成管理モデルから双方の対応づけを行い, 提案するコンテナ管理の構成管理モデルを示す. これによって Chef の構成ファイルに Kubernetes が設定ファイルのどのリソースを管理するか明確になる. Chef に対応する Kubernetes の管理範囲の説明を示す(表 1).

表 1 Chef に対応する Kubernetes の管理範囲

Chef			Kubernetesフィールドとの連携	
優先順位	構成要素	Chefにおける用途	対応フィールド	提案するコンテナ管理における用途
1	Node	ノードを定義	apiVersion,kind	サービスの定義(Kubernetes APIのバージョン, サービスの種類)
2	Role	ノードの役割を定義	metadata,spec	Podの名前, ラベル設定, Docker imageなどの役割のリソースを定義
3	Environment	ノードの環境を定義	metadata,spec	コンテナ, ポート, ボリューム, レプリカ数などの実行環境のリソースを定義
4	Recipe	コードで書いたノード設定を手順通りに実行	-	Attribute系のkey-valueを集templateに代入し, Kubernetesに適用
5	Attribute	Recipeを実行すると必ず実行するAttribute	status	サービスのステータスを確認するリソースを定義

Chef と Kubernetes の管理対象の対応づけを Pod, Replication Controller, Service の 3 種類のサービスごとに定義する. 例として Pod サービスの Chef と Kubernetes の管理対象の対応づけを示す(図 6). また, それぞれの Kubernetes のフィールドごとに Chef のどの構成要素で管理するか 4 つの構成要素について説明する.

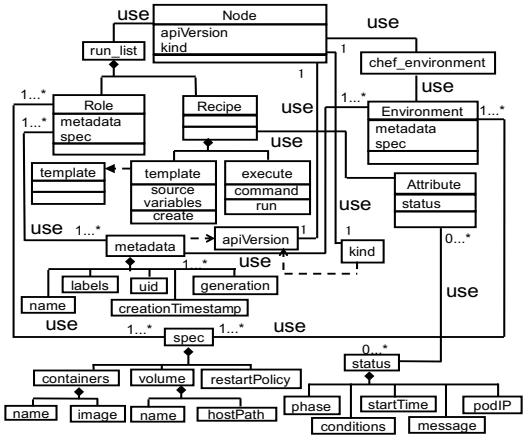

図 6　Pod の Chef と Kubernetes の管理対象の対応づけ

(1)　Node

Node は apiVersion, kind のフィールドを管理する. これらのフィールドがないと他の 4 つのフィールドが利用できない. また, Kubernetes の提供する Pod などのサービス自体を提供するフィールドであるため, Kubernetes のサービスに必要なリソースを提供する. よって, 構成要素の優先順位が最高でサーバの定義をする Node の要素が相当する.

(2)　Role

Role は metadata, spec のフィールドを管理する. name や labels のようなリソースで Pod などのサービスの一意に識別し, 分類するのに用いられるため, Pod サービスの役割と区別を提供していることになる. よって, サーバの役割の設定を記述する Role の要素が相当する.

(3)　Environment

Environment は metadata, spec のフィールドを管理する. Volume や subdomain のようなリソースで Pod などの実行環境を定義, 操作する. よって, サーバの実行環境の設定を記述する Environment の要素が相当する.

(4)　Attribute

Attribute は status のフィールドを管理する. status のフィールドはサービスの状態を通知する役割を持つ. しかし, 他の 4 つのフィールドは必須のリソースだが, status フィールドがなくてもサービスは実行できるため、任意のリソースとなる. 構成要素の Attribute は構成要素の中でも優先度は最下位である. Attribute は status リソースに相当する.

6.5. 設定ファイル作成と適用

クライアントが Chef-Solo を実行し, 生成された設定ファイルを Kubernetes に適用するアクティビティを図 7 に示す.

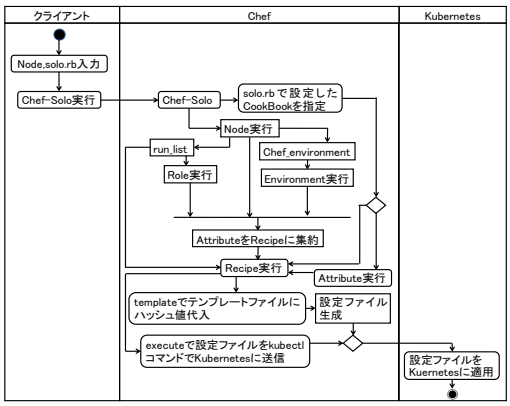

図 7　コンテナ管理方法のアクティビティ図

(1)　クライアント側は chef-solo コマンドで Node の構成ファイルと solo.rb を実行する.

(2)　solo.rb には実行する Cookbook が記述されており, 実行する Cookbook 内の Recipe と Attribute を選択する.

(3)　chef-solo から Node が実行されると Role, Environment, Recipe の要素を run_list と chef_environment で指定し, 実行する.

(4)　実行したすべての Attribute 系の要素にある Attribute を Recipe に集約させる.

(5)　template の Resource によって Recipe のハッシュ値を設定ファイルのテンプレートファイルに代入し, 設定ファイルを生成する.

(6)　生成した設定ファイルを Kubernetes に適用する.

7. 例題への適用

提案構成管理モデルから構成ファイルを作成し, 例題に適用して構成管理モデルの妥当性を示す.

7.1. 実行環境の構成

構成ファイルを作成し, Kubernetes に適用するクラスタの環境と作成した構成ファイルの規模をサービスごとに示す(図 8)(表 2).

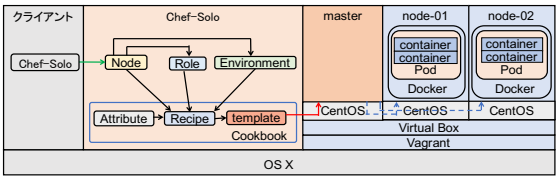

図 8　クラスタ環境の構成

表 2　Chef 構成要素と Pod サービスリソースの対応

	Pod		Replication Controller		Service	
構成ファイル	形式	規模(LOC)	形式	規模(LOC)	形式	規模(LOC)
Node	JSON	11	JSON	11	JSON	11
Role	JSON	21	JSON	37	JSON	22
Environment	JSON	14	JSON	20	JSON	16
Attribute	Ruby	1	–		–	
Recipe	Ruby	22	Ruby	25	Ruby	23
template	eRuby	24	eRuby	19	eRuby	22
規模の合計		93		112		94

クライアントは Chef の構成ファイルの作成と実行を行う．VM として動作する CentOS 上で Kubernetes のクラスタを運用する．クライアントは Chef-Solo コマンドを実行し，Chef の各構成要素を連携させ，設定ファイルを生成する．設定ファイルは VM の master サーバに適用し，コンテナ管理のサービスを実行できる．サービスを実行したらそれぞれの Node に適用する．

7.2. Pod サービスの作成

Pod サービスで nginx を実行するためのコンテナ配置する例を示す．Chef の構成要素に記述する設定ファイルのフィールドとリソースとの対応を示す(表 3).

表 3 Chef 構成要素と Pod サービスリソースの対応

構成要素	対応フィールド	Pod サービスリソース	形式	規模(LOC)
Node	apiVersion, kind		JSON	11
Role	metadata	name, labels	JSON	21
	spec	containers - name - image		
Environment	spec	containers - ports -containerPort	JSON	14
Attribute	status	phace	Ruby	1

7.3. Pod サービスの配置確認

Chef の各構成ファイルにリソースを記述し，Chef-Solo で実行した結果を示す(図 9). Chef の環境内で設定ファイルが生成されることを確認した．Chef から生成された設定ファイルが Kubernetes の Master に適用されたことを確認した．Kubernetes の Master から Pod サービスの状態を取得し，Pod が Node に配置されていることを確認できた．これにより，Chef で Kubernetes の Pod を管理できることを確認した．

図 9 Pod サービスの配置確認

7.4. 作成した構成ファイルの規模

3 つのサービスを実行するために作成した構成ファイルの規模を表 4 に示す．

表 4 サービスごとの設定ファイルの規模

サービス	JSON	Ruby	eRuby	規模合計(LOC)
Pod	46	23	24	93
Replication	68	25	19	112
Service	49	23	22	94

8. 評価と考察

8.1. Chef と Kubernetes の構成管理モデルの実現

Chef の構成ファイルで Kubernetes のリソースを管理することを実現した．これによって，Chef の構成要素に Kubernetes のどのフィールドを管理すればいいか明確になる．

8.2. コンテナ管理方法の考察

Kubernetes のコンテナ管理は Docker と Kubernetes の双方のモデルを理解しないと使用が難しい特徴があった．本稿では，Chef の構成要素にある役割範囲の特徴を活かし，構成管理モデルで Kubernetes のリソースを管理することができる．また，Chef は一度書いたコードを利用できる特徴がある．一度作成した構成ファイルの再利用ができ，コンテナ管理の効率化につながる．

9. 今後の課題

(1) Kubernetes のリソース適用範囲の拡張

構成管理モデルは Kubernetes のリソース全ての管理範囲を拡張する必要がある．

(2) Chef の構成ファイルの複雑化への対応

提案するコンテナ管理方法では，Chef の構成ファイルが複雑化する恐れがある．リポジトリを分けるなど詳細化が必要になる．

(3) 構成管理ツールの適用範囲の拡大

Ansible, Puppet のような構成管理ツールでも管理可能か確認する必要がある．

10. まとめ

本稿では，Kubernetes のコンテナ管理の効率化を目的に Chef を応用したコンテナ管理方法を提案した．Chef の構成要素の各構成の役割範囲の特徴を活かし，Kubernetes のフィールド対応範囲を明確にした構成管理モデルを提案した．提案モデルに基づき，Chef の各構成ファイルに Kubernetes のリソースを記述し，その妥当性を示した．これによって，Chef で Kubernetes のコンテナ管理ができることを確認した．

参考文献

[1] Chef, https://www.chef.io/.
[2] Docker, https://www.docker.com/.
[3] Kubernetes, http://kubernetes.io/.
[4] V. Marmol, et al., Networking in Containers and Container Clusters, Proc. of netdev 0.1, Feb. 2015, 4 pages.
[5] 佐藤 司, ほか, Docker コンテナ実践検証, インプレス, 2015.
[6] D. Spinellis, Don't Install Software by Hand, IEEE Software, Vol. 29, No. 4, Jul./Aug. 2012, pp. 86 - 87.
[7] 吉羽 龍太郎, ほか, Chef 実践入門, 技術評論社, 2014.