Docker コンテナを用いたマルチクラウド上の動的アーキテクチャの提案

2012SE009 浅井 利文 2012SE164 森 雅達 2012SE284 山中 翔太
指導教員 青山 幹雄

1. 研究背景

近年，サーバ仮想化を実現する新たな方法として，コンテナ型仮想化が注目を集めている．その中でも Docker の利用が急速に増えている．Docker は，1 つのコンテナで 1 つのプロセスを起動することを原則とするため，コンテナ間の連携が必要である．しかし，異なるクラウド上のコンテナ同士では Link 機能を用いた通信ができないため，マルチクラウド上のコンテナ間の連携が困難である．

2. 研究課題

CoreOS 社の提案するマルチクラウドにおけるコンテナ間通信[5]では，ホスト OS として CoreOS，または etcd という分散 KVS(Key-Value-Store)が動作する OS を用いる必要がある．そこで本稿では，環境依存しないコンテナ間の連携を目的に，Docker コンテナを用いたマルチクラウド上の動的アーキテクチャを提案する．

(1) コンテナ間の連携をホスト OS から独立

Docker コンテナのみを用いてコンテナ間通信を行い，コンテナ間の連携をホスト OS から独立させる．

(2) コンテナのクラウド間の動的移動の実現

動的なプロキシ用のコンテナを用いることで，リンク先コンテナを異なるクラウドに移動させた場合にも，設定を変更することなくコンテナ間の連携が行われる．

(3) コンテナの移動における連携の連続性の保証

リンク先コンテナの移動時に，コンテナ間の連携を無停止で行う．

3. 関連研究

3.1. コンテナ型仮想化

コンテナ型仮想化[6]とは，OS 上に隔離された環境(コンテナ)を構築する技術である．コンテナには OS を導入せず，ホスト OS とカーネル部分を共有する．

3.2. Docker

(1) 概要

Docker[4]とは，Docker 社が提供するオープンソースのコンテナ型仮想化ソフトウェアである．

(2) Link 機能

Link 機能[2]とは，単一のクラウドにおけるコンテナ間通信機能である．本稿では Docker Link と呼ぶ．

(3) Dynamic Ambassador パターン

本稿では，CoreOS 社の提案するマルチクラウドにおけるコンテナ間通信[5]を Dynamic Ambassador パターンと呼ぶ．リンク先コンテナの接続情報を etcd に書き込み続けるコンテナと，etcd に登録された接続情報を読み出し続ける動的なプロキシを用いてマルチクラウドにおけるコンテナ間通信を行う．

4. アプローチ

Docker コンテナの「Docker が稼動する環境さえあれば，ホスト OS の環境に依存しない」という特徴に着目した．Dynamic Ambassador パターンを基に，ホスト OS 上で動作させていた etcd をコンテナ内で動作させる．

このようにして，Docker コンテナのみを用いたマルチクラウドにおけるコンテナ間通信を実現することで，コンテナ間の連携をホスト OS から独立させる．図 1 の提案アーキテクチャの階層構造に示すように，ホスト OS では Docker が稼動する環境さえ用意すればよい．

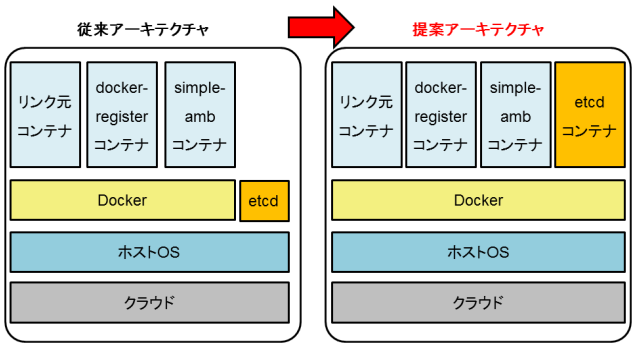

図 1 アプローチ図(階層構造)

5. 提案アーキテクチャ

5.1. アーキテクチャの構成

提案アーキテクチャの構成を図 2 に示す．リンク元コンテナからリンク先コンテナへの経路を実線矢印，etcd への経路を点線矢印で表す．

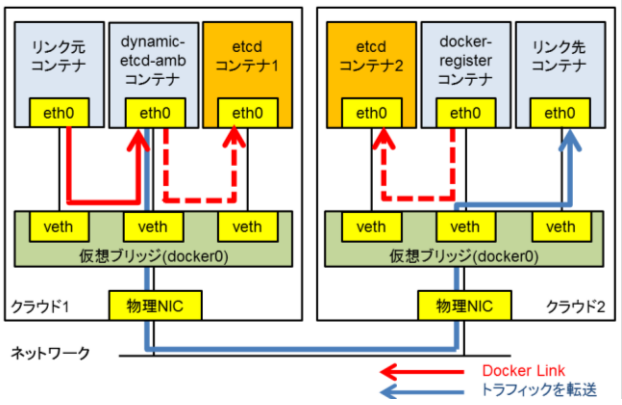

図 2 提案アーキテクチャの構成

従来の Dynamic Ambassador パターンではホスト OS 上で動作させる etcd を，コンテナ内で動作させる．これにより，Docker コンテナのみを用いたマルチクラウドにおけるコンテナ間通信が可能になる．各クラウドの etcd コンテナ間ではクラスタを形成し，クラウド間でリンク先コンテナの接続情報を共有する．

また，etcdをコンテナ化したことによって，Docker Linkを用いて etcd へ接続できる．そのため，従来の Dynamic Ambassador パターンで用いる simple-amb コンテナのような，etcd へトラフィックを転送するだけのコンテナを起動する必要がない．

各コンテナの役割を以下に示す．

(1) docker-register コンテナ

Docker API を用いて取得したリンク先コンテナの接続情報を etcd に書き込み続けるコンテナ．

(2) dynamic-etcd-amb コンテナ

etcd に登録されたリンク先コンテナの接続情報を読み出し続けるコンテナ．また，自らの特定のポートへのトラフィックを，etcd から読み出した接続情報へ転送するプロキシの役割がある．

(3) etcd コンテナ

etcd が動作し，各クラウドの etcd コンテナ間でクラスタを形成するコンテナ．リンク先コンテナの接続情報を各クラウドで共有するために用いる．

5.2. コンテナ間通信システムのユースケース図

コンテナ間通信システム(docker-register コンテナ，dynamic-etcd-amb コンテナが構成するシステム)のユースケース図を図 3 に示す．

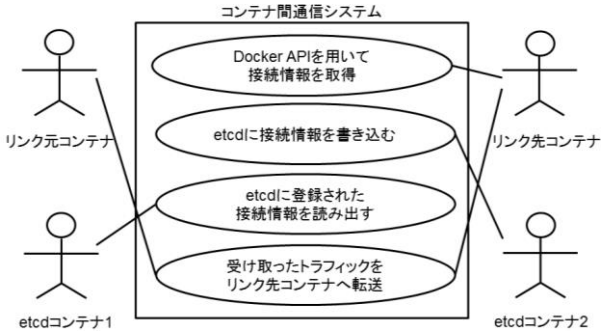

図 3 コンテナ間通信システムのユースケース図

5.3. アーキテクチャの振る舞い

2 つのユースケース図について説明する．

(1) etcd への書き込み/読み出しのシーケンス図

etcd への書き込みと読み出しを表すシーケンス図を図 4 に示す．

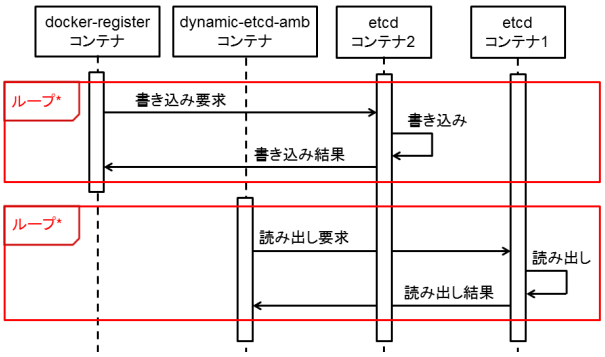

図 4 etcd への書き込み/読み出しのシーケンス図

まず，docker-register コンテナが etcd コンテナ 2 に対してリンク先コンテナの接続情報を書き込む．そして，dynamic-etcd-amb コンテナは etcd コンテナ 1 に登録された接続情報を読み出す．etcd コンテナ 1，2 はクラスタリングされているため，登録された情報を共有できる．

この書き込みと読み出しが繰り返し行われることによって，dynamic-etcd-amb コンテナのトラフィック転送先は動的に変更される．

(2) リンク先コンテナへの接続のシーケンス図

リンク元コンテナからリンク先コンテナへの接続のシーケンス図を図 5 に示す．

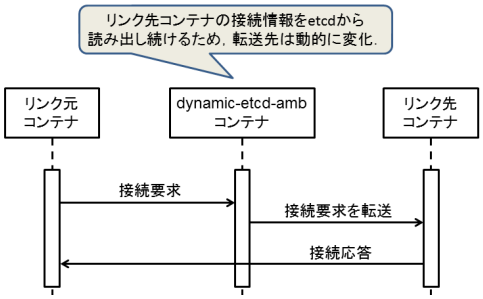

図 5 リンク先コンテナへの接続のシーケンス図

まず，リンク元コンテナは Docker Link を用いて変数で dynamic-etcd-amb コンテナの特定のポートに接続要求を送る．次に，dynamic-etcd-amb コンテナは etcd から読み出した接続情報をもとに，接続要求をリンク先コンテナへ転送する．

5.4. 従来の Dynamic Ambassador との比較

従来の Dynamic Ambassador パターンとの比較をまとめて表 1 に示す．

表 1 従来と提案の比較

比較項目	従来 アーキテクチャ	提案 アーキテクチャ
(1) etcdの動作場所	ホスト上で動作	コンテナ内で動作
(2) etcdへの接続方法	simple-ambコンテナを介して接続	etcdコンテナに対し，Docker Linkを用いて変数で接続
(3) simple-ambコンテナの有無	あり	なし
(4) ホストOSの指定	CoreOS，もしくはetcdが動作するOS	なし

(1) 従来は etcd をホスト OS 上で動作させる．提案アーキテクチャでは，Docker コンテナのみを用いたコンテナ間通信を実現するため，etcd をコンテナ内で動作させる．

(2) 従来は simple-amb コンテナを介して etcd に書き込みと読み出しを行っていた．また，simple-amb コンテナ起動時に手動で etcd への接続情報を入力する必要があった．しかし，提案アーキテクチャでは docker-register コンテナなどから etcd コンテナへ直接 Docker Link を用いる．そのため，etcd への接続情報が自動登録される．

(3) simple-amb コンテナは docker-register コンテナなどに etcd への接続情報を教えるだけのコンテナである．しかし(2)で述べたように，提案アーキテクチャでは etcd への接続情報が Docker Link で自動登録されるため，simple-amb コンテナを用いる必要がない．

(4) 提案アーキテクチャは Docker コンテナのみを用いるため，ホスト OS の環境に依存しない．よって，OS の種類の指定は無い．

6. プロトタイプの実装

6.1 実装環境

プロトタイプの実装環境を表 2 に示す. ホスト 1 をリンク元コンテナが動作するホスト, ホスト 2 をリンク先コンテナが動作するホストとする. ホスト 3, 4 は 7 章でリンク先コンテナを異なるクラウドへ移動する場合の移動先のホストとする.

表 2 実装環境

	ホスト1 (リンク元コンテナの動作するホスト)	ホスト2 (リンク先コンテナの動作するホスト)	ホスト3	ホスト4
OS	CentOS7.1	CentOS7.1	CentOS7.1	Ubuntu15.04
CPU	2.66GHz	2.66GHz	2.66GHz	2.66GHz
メモリ	1874192kB	1874192kB	2038492kB	2038492kB
Docker	1.6.2	1.6.2	1.6.2	1.6.2
IPアドレス	10.48.134.220	10.48.134.223	10.48.134.204	10.48.134.204

6.2 プロトタイプの構成

プロトタイプの構成を図 6 に示す. リンク元コンテナからリンク先コンテナへの経路を実線矢印, etcd への経路を点線矢印で表す.

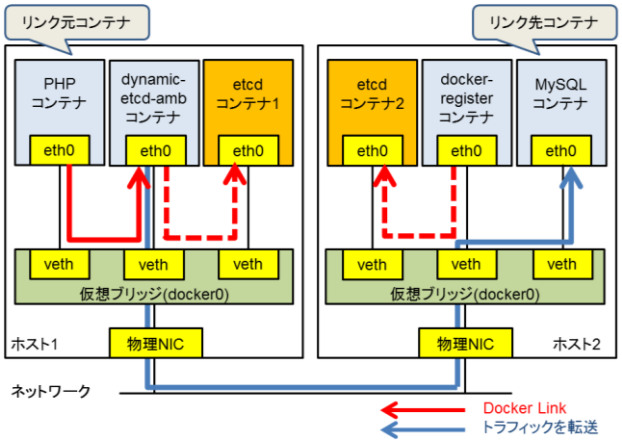

図 6 プロトタイプの構成

各コンテナの起動に用いる Docker イメージを表 3 に示す. これらの Docker イメージは DockerHub もしくは Quay.io で公開されている.

表 3 コンテナ起動に用いる Docker イメージ(1)

docker-registerコンテナ	polvi/docker-register
dynamic-etcd-ambコンテナ	polvi/dynamic-etcd-amb
etcdコンテナ	quay.io/coreos/etcd:v0.4.6

また, 7 章の例題で用いるリンク元コンテナとリンク先コンテナの起動に用いる Docker イメージを表 4 に示す.

表 4 コンテナ起動に用いる Docker イメージ(2)

PHPコンテナ	phptest(自作のDockerfileから構築したDockerイメージ, Apache:2.4.10, PHP:5.6.16)
MySQLコンテナ	mysql:5.7.9

7. 例題への適用

7.1 例題

データベースに接続する Web アプリケーションを実行する. そこで, MySQL データベース用のコンテナ(以下, MySQL コンテナ)と, PHP を用いた Web アプリケーション用のコンテナ(以下, PHP コンテナ)とに分離して連携させる. また, この 2 つのコンテナは異なるホスト上に配置するものとする.

7.2 実行シナリオ

ホスト 1, 2 で各コンテナを起動し, コンテナ間通信を行う. ブラウザで PHP コンテナに接続して「データベース接続成功」と表示されれば, コンテナ間通信が成功したことが確認できる.

次に, MySQL コンテナ(リンク先コンテナ)を他のホストに移動した場合にも「データベース接続成功」と表示されれば, dynamic-etcd-amb コンテナのトラフィック転送先が動的に変更されることが確認できる.

また, 移動先のホストは以下の 2 つの場合に分けて検証を行う.

(1) 同一ホスト OS を用いた異なるホスト(ホスト 3)へ移動
(2) 異なるホスト OS を用いた異なるホスト(ホスト 4)へ移動

移動の手順としては, まず移動先のホストで docker-register コンテナとリンク先コンテナをあらかじめ起動しておく. 次に, 移動元の docker-register コンテナとリンク先コンテナを停止する. その結果, etcd に書き込みを行うコンテナは移動先の docker-register コンテナのみになり, 移動後のリンク先コンテナとの連携に切り替えられる. この手順で移動させることによって, コンテナ同士の連携を停止することなくリンク先コンテナを異なるホストへ移動できる.

7.3 コンテナの起動

各ホストで必要なコンテナを起動する. 実行シナリオ(1)の場合の, リンク先コンテナ移動前の各ホストの階層構造を図 7 に示す.

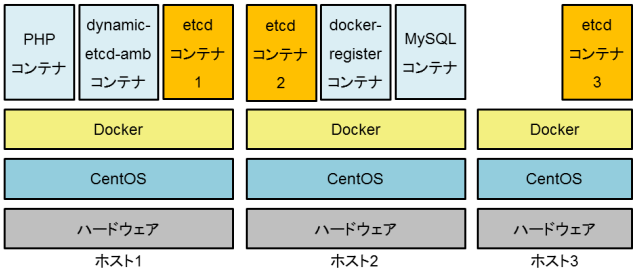

図 7 例題の階層構造

ホスト 1, ホスト 2, ホスト 3(実行シナリオ(2)の場合はホスト 4)上の etcd コンテナは各 etcd コンテナ間でクラスタを組むよう設定して起動する.

PHP コンテナは, ブラウザから接続して MySQL コンテナとの通信が成功していることを確認するため, 80 番ポートを開放する. また, コンテナの 80 番ポートをホストの 8080 番ポートとポートフォワーディング設定して起動する.

7.4 結果

dynamic-etcd-amb コンテナでリンク先コンテナの接続情報(10.48.134.223:32769)を読み出し続けていることが確認できた(図 8).

また, ブラウザから PHP コンテナに接続すると「コンテナ間通信成功」と表示され, マルチホストにおけるコンテナ間通信が成功したことを確認できた.

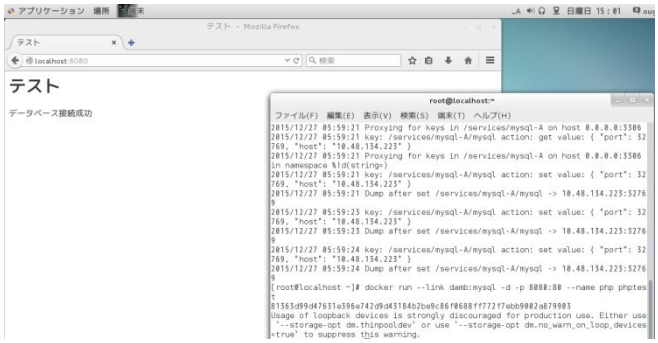

図 8 コンテナ間通信の確認(リンク先コンテナ移動後)

次に, リンク先コンテナと docker-register コンテナを異なるホストに移動して dynamic-etcd-amb コンテナにアタッチすると, 移動後のリンク先コンテナの接続情報(10.48.134.204:32769)を読み出していることが確認できた(図 9).

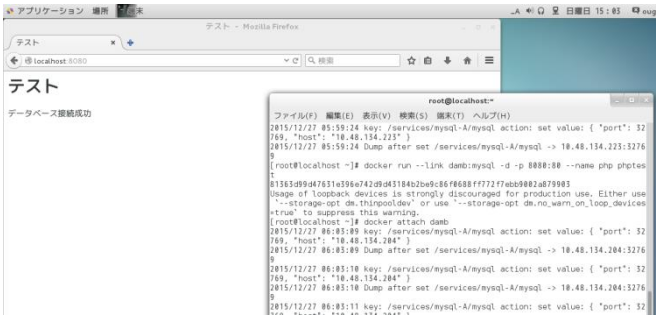

図 9 コンテナ間通信の確認(リンク先コンテナ移動)

また, リンク先コンテナ移動前と同様にブラウザから PHP コンテナに接続すると,「コンテナ間通信成功」と表示された.

ホスト間の移動に関する実行シナリオ(1), (2), どちらの場合においても「コンテナ間通信成功」と表示された.

8. 評価

(1) コンテナ間の連携をホスト OS から独立

Docker コンテナのみを用いてマルチホストにおけるコンテナ間通信が可能となった. そのため, Docker が稼動する環境さえあればホスト OS の環境に依存しない. 本稿では CentOS, Ubuntu を用いて, これを確認した.

(2) コンテナのクラウド間の動的移動の実現

リンク先コンテナを他のホストに移動させた場合に, dynamic-etcd-amb コンテナの設定を変更せず, 移動後のリンク先コンテナの接続情報を読み出すことが可能である. しかし, リンク先コンテナと共に docker-register コンテナも移動する必要がある.

(3) コンテナの移動における連携の連続性の保証

実行シナリオの手順でリンク先コンテナを移動させることで, コンテナ間通信を無停止で行うことができるため, コンテナ間の連続した連携が可能となった.

9. 考察

本稿の提案アーキテクチャの最大の特徴は Docker コンテナのみを用いているため, Docker が稼動する環境さえあればホスト OS の環境に依存しない点である. 従来のアーキテクチャでは, CoreOS, もしくは etcd の動作する OS をホスト OS として用いなければならなかった. クラウドサービスでは利用できる OS の種類が限られていることもあるため, 環境依存しない点は大きなメリットであると考えられる.

従来のアーキテクチャと同様に, リンク先コンテナを他のクラウドに移動しても, dynamic-etcd-amb コンテナは移動後のリンク先コンテナの接続情報を読み出すことができる. また, リンク先コンテナを異なるクラウドへ移動させる場合, 例題のように手順を工夫することで, 移動時にも連続したコンテナ間の連携が可能である.

このように, 従来の Dynamic Ambassador パターンのメリットを持ち, かつホスト OS から独立してコンテナ同士が連携できる.

10. 今後の課題

リンク先コンテナを他のクラウドに移動する場合, それに伴って docker-register コンテナもリンク先コンテナと同じクラウドに移動する必要がある.

そのため, リンク先コンテナがどのクラウド上で動作しているかに関わらず, docker-register コンテナがリンク先コンテナを認識できるようにすることが今後の課題である.

11. まとめ

本稿では, 環境依存しないコンテナ間の連携を目的に, Docker コンテナを用いたマルチクラウド上の動的アーキテクチャを提案した.

Dynamic Ambassador パターンにおいてはホスト OS 上で動作させる etcd を, コンテナ内で動作させる. それによって Docker コンテナのみを用いたコンテナ間の連携が可能になるため, ホスト OS の環境に依存しなくなる.

また, 動的なプロキシコンテナを用いることでリンク先コンテナの動的移動に対応できる. そのとき, 移動の手順を工夫することで, 連続したコンテナ間の連携が可能である.

そして, 提案アーキテクチャのプロトタイプを実装して例題に適用することで, その妥当性を確認した.

参考文献

[1] 阿佐 志保, プラグラマのための Docker 教科書, 翔泳社, 2015.

[2] Docker, Linking Container Together, https://docs.docker.com/userguide/dockerlinks/ ..

[3] 松井 暢之, マルチホスト Docker ネットワーキング(1), 2015 年 4 月, http://tech-sketch.jp/2015/04/multi-host-docker-1.html.

[4] 中井 悦司 ほか, Docker のしくみ徹底解説 Software Design, 技術評論社, 2014 年 12 月, pp. 17-57.

[5] A. Polvi , Dynamic Docker Links with an Ambassador Powered by etcd, CoreOS, Feb. 2014, https://coreos.com/blog/docker-dynamic-ambassador-powered-by-etcd/.

[6] 佐藤 司, 冨永 善視, 森永 敏雄, Docker コンテナ実践検証, インプレス, 2015.