人工知能学会
第15回知識流通ネットワーク研究会
SIG-KSN-015-10

コンテナ技術を使ったディザスタ･リカバリ方法に関する考察

〇桑田喜隆(*1)
(*1) 室蘭工業大学

A Study on Disaster Recovery with Container

Yoshitaka Kuwata(*1)
(*1) Muroran Institute of Technology, Japan

概要

災害時等にシステムを被害から守り，回復するための技術はディザスタ・リカバリと呼ばれる．本稿では，クラウドサービス上でコンテナ技術を活用し簡易なディザスタ・リカバリを実現する方法を提案する．システムイメージを効率よく保存し，システム障害時に仮のシステムとして復旧する方式に関して考察する．

Abstract

The term, “Disaster Recovery”, is used for the protection of IT system from disaster, and for the recovery from the damages. In this paper, an implementation of disaster recovery system is proposed, which is based on cloud services. In order to manage system images to preserve on clouds and execute the images on clouds, the method makes use of container technology on cloud services.

1. はじめに

近年，震災をきっかけに災害発生時等にも事業を継続するための事業継続マネジメント(Business Continuity Management, 以下，BCM[1],[2])が，企業や政府機関，自治体等から注目されている. BCM は業務全体の棚卸しを行い，リスク分析を実施したり，各業務プロセスの依存関係や重要性を洗い出す作業を含む．業務の代替手段の準備や，その訓練までも実施することが必要になってくる. BCM は業務全体を含めたマネジメントが求められるが，近年は業務の中で情報システムが占める割合も大きいことから，情報システムを守り，災害時にどのように復旧されるかといった，IT の事業継続計画の立案が求められる．

残念ながら，既存情報システムでは BCM まで考慮して設計された物は少ない．これは設計時に BCM が注目されていなかったこともあるが，一方で可用性の高いシステムを設計構築運用するためには，費用がかかるためである．例えば，広域災害を想定して地域の離れた複数のセンターに分散してシステムを配置し，災害時に切り替えるような仕組みを考えた場合，分散配置しないシステムに比べ数倍以上の費用が必要になる．このため，従来は金融系やインフラ系などミッションクリティカルな業務を中心に分散配置構成が取られてきた．

他方，2000 年代の後半からのクラウドコンピューティング技術の進展を背景に，各種のいわゆるクラウドサービスが比較的安価に提供されるようになってきている．クラウドコンピューティングでは必要な時に必要なサービスをオンデマンドで利用することが可能である．また，サービスを提供するデータセンタも複数の地域に分散配置されていることから，最初からクラウドサービス上にシステムを設計することで，災害に強いシステム構築が期待できる．一方で，クラウドサービス上にシステムを設計するため，多かれ少なかれシステムのアーキテクチャを変更する必要が有る．また，クラウドのセキュリティやサービスレベル保証など，導入には未だ障壁がある．

本稿では, オンプレミスに設置された既存システムのアーキテクチャを大きく変更することなく，クラウドサービスと連携することで災害時のシステム復旧までの仮運用からディザスタ・リカバリが可能な方法を提案する．

1 Yoshitaka Kuwata
室蘭工業大学
北海道室蘭市水元町２７−１
kuwata@mmm.muroran-it.ac.jp

2. 想定するディザスタ･リカバリの要件

2.1 システム構成

図１に本検討で前提とするシステム構成を示す.

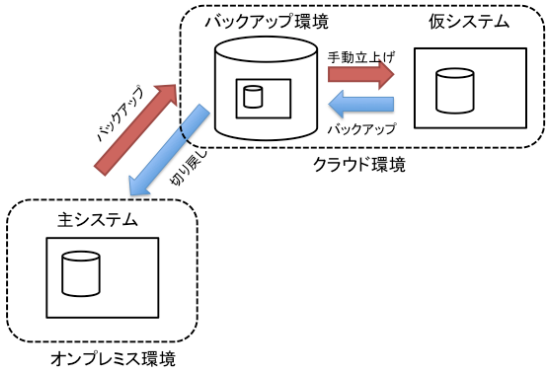

図１ 想定するシステム構成

システム構成として, オンプレミス環境の主システム, クラウド環境のバップアップ, 仮システムから構成されることを前提としている.

2.2 前提とする運用条件

以下に, 本検討の前提とする条件について述べる.

- オンプレミスで主システムが動作する.
- データバックアップおよび仮サービス提供用にクラウドサービスを利用する.
- 主システムから定期的にクラウドサービス上にバックアップを実施する.
- 主システム被災時には, 手動でクラウドサービス上の仮サービスを立ち上げる.
- 主システム復旧時には, 手動で仮サービスを停止し切り戻しを行う.

2.3 要件

上記の前提を置いた場合, 要件として以下があげられる.

(1) 主システム被災時に, 遠隔地にデータが保全させること
(2) 主システム被災時に, 遠隔地で仮サービスが継続できること
(3)サービスの切り換え,切り戻しが手動で行えること.

また, 上記以外に必須ではないが望ましい要件として以下があげられる.

(4) 主システムアーキテクチャの変更が少ないこと.

3. これまでの取り組み

１章で述べた通り, 従来は複数拠点に設置されたデータセンタにシステムを冗長化して互いに連携して動作する仕組みを取ることで, 災害時にもサービスを継続可能なシステムを構築する方法が取られていた.

近年では, より容易に耐障害性を向上する方法として, クラウドサービスを使う方法が提案されてきている. ここでは一例として, パブリッククラウドサービスである Amazon Web Services (以下, AWS[5])を使った方法について説明する.

3.1 クラウドデザインパターン

AWS を使った設計の方法は, クラウドデザインパターン協議会によって「AWS クラウドデザインパターン[3] (以下, AWS-CDP) 」として収集され整理が進められている. 以下に AWS-CDP に記載されている可用性向上に関連するパターンとその概要を示す.

表１ AWS-CDP の可用性向上に関連するパターン

番号	パターン名称	方式
1	Multi- Data-center	複数のデータセンタにまたがってサーバを分散させることで, データセンタレベルの障害を避ける
2	Sorry Page (※)	メインサイトの障害時にバックアップサーバ (Sorry サーバ) に切り替える
3	Cross-Region Replication (※) [4]	地域 (リージョン) をまたがってバックアップデータをコピーする

(※CDP2.0 候補)

関連するパターンとして３つあげたが, 特に今回検討した要件に近いものとして, Cross-Region Replication パターン[4]がある. 次節でその内容について論じることとする.

3.2 Cross-Region Replication パターン

図２に Cross-Region Replication パターンの構成図を示す.

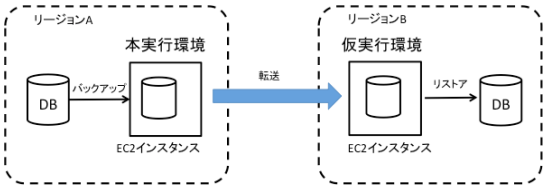

図２ AWS-CDP の Cross-Region Replication パターン（[2]より引用、一部追記）

実際の操作は以下の順序で行われる．

(1) リージョン A で実行している DB を，同リージョン内の EC2 インスタンスにバックアップする．

(2) データを内包する EC2 インスタンスのイメージをリージョン B に転送する．

(3) リージョン B で EC2 インスタンスを起動後，リージョン B で動作している DB にデータをリストアする

このパターンではデータを EC2 インスタンスイメージに一旦格納して遠隔コピーする点がポイントであると考えられる．AWS 内では EC2 インスタンスやイメージは扱いやすいため，良い方法である．

本パターンを基にオンプレミスと連携するように拡張することも可能である．例えば，リージョン A を AWS ではなくオンプレミスとした場合，直接 EC2 イメージを扱えないため，転送時にイメージ変換が必要になる．このため，よりポータビリティの良い別の方法が望ましいと考える．

そこで，本稿ではよりポータビリティの高いコンテナ技術を利用した方法を提案する．

4. コンテナ技術を使ったバックアップ方式に関する検討

4.1 コンテナ技術(Docker)

Docker は 2013 年から注目を集め始めた技術で，特にソフトウェア開発やシステムのデプロイの分野で期待されている．サーバの自動構築による継続的なインテグレーション(Continuous Integration, CI)の側面や，ソフトウェア試験環境の構築，運用中のシステムの逐次アップグレードなどの応用が可能である．

Docker は以下の２つのコンポーネントから構成される．

・ Docker Engine
可搬性が高く軽量な実行環境および実行環境を含むイメージを自動構築するためのツール

・ Docker Hub
アプリケーションやワークフロー等を利用者に配布し，利用者間で共有するためのクラウドサービス

本稿で対象としているディザスタ・リカバリに関しては，Docker の以下の特徴が活かせると考える．

(1)ファイルシステムの差分管理
Docker Engine は AUFS(統合ファイルシステム)を利用してファイルの差分を管理しており，任意の時点のスナップショットの取得および再開が可能である．また，イメージ間の依存関係が保持されており，例えばあるプログラムを導入したイメージから複数のインスタンスイメージを作成した場合にも，元のイメージは共有され，差分のみがマシン台数分保存される．

(2) Docker イメージのポータビリティ
Docker Engine 上でイメージのポータビリティが確保されており，Docker を導入した別のマシン環境でも動作させることができる．

(3) レポジトリとの連携
Docker Engine はイメージの管理のために，レポジトリとの連携機能を持つ．パブリックなレポジトリとして Docker Hub が提供されており，必要なイメージを Pull して利用可能である．ローカルにリポジトリを構築し，連携先としてそちらを設定することも可能である．ローカルにリポジトリにバックアップ用にイメージを Push しておく等の方法も考えられる．

(4) LXC による軽量な仮想化
Docker は LXC(Linux コンテナ)によってマシンをパーティショニング（仮想化）しておりは，一台の Linux マシン上で仮想的に複数の異なる機能を持つ Linux サーバを動作させることが可能である．通常の仮想化に比べ LXC は軽量であるため，少ないリソースでより多くの仮想サーバが稼働する．

(5) 最適化された実行環境
Docker では必要最低限のプログラムのみ

を用いて構築したサーバのイメージが使われる．従来の方法では必要なソフトウェアを手動で導入する必要があり，手間がかかるが，Docker ではイメージ自動構築機能を活用することで，最適化した実行環境が容易に得られる．

4.2 Docker を使ったディザスタ・リカバリ(DR)方式

図３に Docker を使った最も簡単な DR のシステム構成を示す．

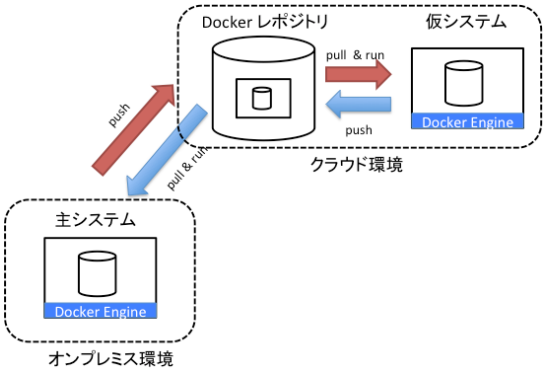

図３ Docker を使った最も簡単なディザスタ・リカバリ方式

- オンプレミスの本実行環境として Docker を利用し，本システムは Docker 上に構築する
- クラウドサービス上のプライベートな Docker リポジトリを構築する
- 定期的に本番環境の Docker イメージを Docker レポジトリにバックアップとして push する
- 仮実行環境に実行を切り替える場合には，クラウドサービス上の Docker から，Docker レポジトリを参照して最新のイメージを pull して実行する．
- 切り戻しを行う場合には，仮実行環境の Docker イメージを Docker レポジトリに push する．仮システムを停止後，オンプレミスの Docker から pull する．

なお，ここでは起動停止などの操作やアクセス先の切り換えのための DNS の書き換え等は手動で実施することとしている．主システムから Docker レポジトリに push した最新イメージがリカバリーに使われが，主システムの push 後の更新データ等は反映されない．また，切り換え中のサービス停止などは考慮していない．

4.3 複数のシステムの DR を行う場合の処理

4.2 で述べた方式で複数の異なる主システムの DR が可能である．図４にシステム A，システム B の DR を同時に行う場合を図示している．

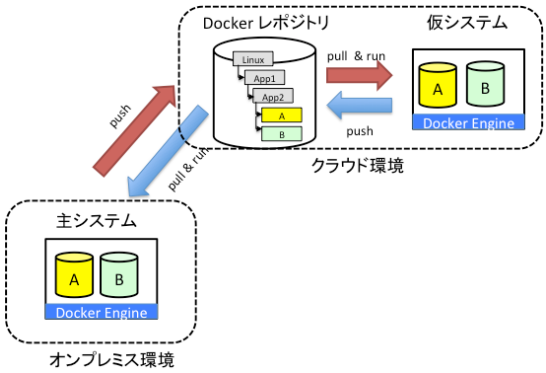

図 4 同一イメージを基にした複数のシステムを構築した場合の動作例

Docker のイメージ管理は AUFS によって差分管理されているため，システム A およびシステム B で共通に使われる OS やアプリケーションのイメージは共有され，A,B 固有の部分のみ個別に転送される．この機能によって，Docker レポジトリとのデータ転送および Docker レポジトリへのデータ格納は最適化される．

4.4 バックエンドにストレージサービスを使う構成

図 3 で提案した方式では，イメージの格納領域として，Docker レポジトリプログラムを実行するサーバ内のストレージを利用した．長期間にわたり多くのシステムのバックアップを取得し続けると，レポジトリ内のデータが大きくなる可能性がある．

これに対して，Docker レポジトリのデータ格納にストレージサービスを利用することで，効率よく安全にイメージを管理することが可能であると．

図５に Docker レポジトリのバックエンドにストレージサービスを利用した構成を示す．データの入出力がバルクで行われるこ

とから，Amazon S3 や OpenStack Swift 等のオブジェクトストレージを利用することが最適であると考えられる．

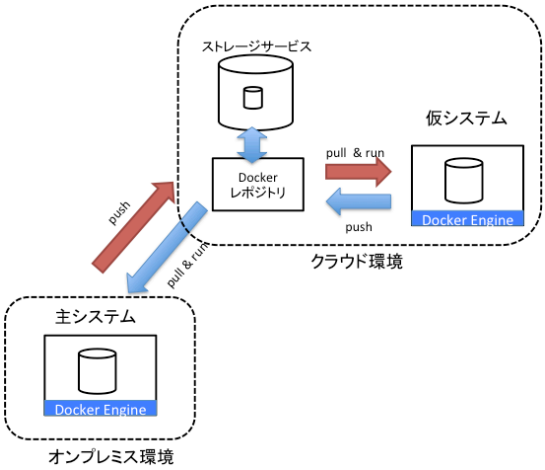

図５ Docker レポジトリのバックエンドにストレージサービスを利用する

なお，Docker レポジトリをオンプレミス環境に設置し，データの格納にクラウドのストレージサービスだけを利用する構成も可能である．しかし本件では DR を想定し，オンプレミス環境が使えない前提のため，Docker レポジトリはクラウドサービス上に構築することが望ましいと考える．

5. 評価

5.1 評価目的および条件

上記の検討結果の検証のため，実際のアプリケーションを取り上げて，実機評価した．本稿では，Docker を使った基礎的な評価結果のみを示す．

評価用のアプリケーションとして，最も利用実績の多い学習管理システムである Moodle[5]を採用した．なるべく実際の利用形態に近いように，半期のコースを作成し，各回にダミーの講義資料を登録した．１つのシステムで複数のコースをサポートすることが必要なため，コースを順次追加してそのイメージサイズを測定した．またローカルマシン上に構築した Docker レポジトリへの登録(push)時間を調べた．

表２に評価条件，評価環境を示す．

表２ 評価環境

項目	スペック
アプリケーション	Moodle 2.7[7] Docker-moodle[8]を使って導入，日本語 Language Pack
コース数	０（未登録）～20
授業回数	１５
レポジトリ	docker-registry[8]を利用 ローカルマシンの Docker 上で動作させる
CPU	Intel Core2 Quad Q8400 2.66GHz
メモリ	4GB
ディスク	SATA 128GB SSD
NIC	1000Base-T (1Gbps)
OS	Ubuntu 14.04.1LTS Server
Docker	V1.2.0, build fa7b24f

5.2 評価結果

評価中に Docker によって保存されたイメージのサイズを図６に示す．

Moodle を導入した段階でイメージサイズが 1GByte 程度であった．コースを追加するに従いコース数に比例してイメージサイズが大きくなり，２ コースめ以降はコースあたり約 33MB 増加した．

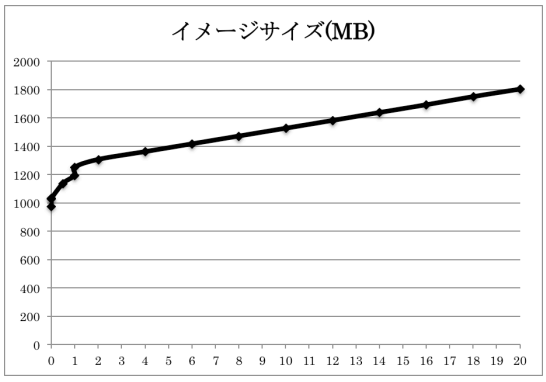

図６Moodle にコースを追加した場合のイメージサイズの変化

次に，上記イメージを Docker レポジトリに登録するのに要した時間を図７に示す．

転送時間はイメージサイズの差分の大きさに比例している．また，ローカルアクセスであることもあり，全体としては５分程度以内に処理を終えることが可能であることが分かった．

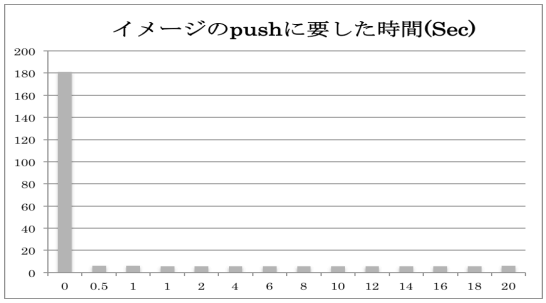

図 7 Moodle にコースを追加した場合の Push に要する時間の変化

6. 考察

6.1 複数システムの DR の場合

同じイメージを基にした複数のサーバを DR する場合には，大幅にイメージの転送時間を節約することが可能である．

例えば，5 章で示した Moodle サーバを 2 台構築した場合, それぞれに 20 コースの教材を格納した状態で，イメージサイズは 1.8GByte 程度になる. 差分イメージ分だけを転送することが必要になることから，転送の必要なイメージの総量は 2.6GByte となり，共通の Moodle 導入イメージの 1GByte 分の転送を省くことができる．

6.2 パブリッククラウドでの実行

本評価では, ローカルマシン上にレポジトリを構築したため，ネットワークの遅延等の影響がなく，理想的な処理が可能であった．ネットワークに遅延がある場合や，エラーの発生した場合等の考慮が必要となる．

また, パブリッククラウドにデータを配置することになるため，経路の暗号化やイメージのレポジトリ内での暗号化などを考慮することが必要になる．

7. まとめと今後の課題

本稿ではコンテナを使った簡易な DR の方法を提案した．コンテナ上に構築したシステムをクラウドサービス上にバックアップし，必要に応じてサービスの起動ができることを示した．また，同一イメージから構築した複数のサイトの場合には，データ転送量が押さえられることが分かった．

本稿では Docker レポジトリもローカルマシン上に構築したが，実際にクラウドサービス上を使って評価を実施することが今後の課題である．

A. 参考文献

[1] Elliott, Dominic, Ethné Swartz, and Brahim Herbane. Business continuity management: A crisis management approach. Routledge, 2010.

[2] Stoneburner, Gary, Alice Goguen, and Alexis Feringa. "Risk management guide for information technology systems." Nist special publication 800.30 (2002): 800-30.

[3] クラウドデザインパターン協議会, AWS Cloud Design Pattern, http://aws.clouddesignpattern.org/ (2014/9/10 アクセス)

[4] クラウドデザインパターン協議会, CDP:Cross-Region Replication パターン, http://aws.clouddesignpattern.org/index.php/CDP:Cross-Region_Replication%E3%83%91%E3%82%BF%E3%83%BC%E3%83%B3 (2014/9/10 アクセス)

[5] Amazon Web Services Inc., Amazon Web Services, http://aws.amazon.com/jp (2014/9/10 アクセス)

[6] Docker Inc., Docker, https://www.docker.com/ (2014/9/10 アクセス)

[7] Moodle project, Moodle, https://moodle.org/ (2014/9/10 アクセス)

[8] Sergio Gómez, Docker-moodle, https://github.com/sergiogomez/docker-moodle (2014/9/10 アクセス)

[9] Docker Inc., Docker-registory, https://github.com/docker/docker-registry (2014/9/10 アクセス)