Vol.2016-IOT-32 No.24
2016/3/4

仮想化基盤上でのシステム再構築の一手法

石坂徹[†1]　桑田喜隆[†1]　横山重俊[†2]　合田憲人[†2]

概要：ネットワークサービス，計算環境，事務処理などで用いられるシステムは，利用が終了した場合にはデータ等を保存してシステムを破棄する場合が多いと思われるが，システムを再度稼動させる必要がある場合にこれを再構築するには，データの復元場所，経年による環境変化など，システムに関する知識やスキルが必要である．一方，仮想化技術の発展により，多くのシステムが仮想化基盤上で動作しており,システムをイメージとして保存しておくことも可能となっている.しかし，イメージの保存には多くの記憶容量が必要であり，その中にはインターネット上のリポジトリなどから簡単に再取得できるものが含まれているなど無駄が多い.これを解消するには予め再構築を見込んで必要な分だけバックアップを行うことが必要である．本研究では，近年注目されているコンテナ技術による仮想化基盤上で，容易にシステムを再構築するための手法を提案する．

キーワード：システム再構築，仮想化基盤，コンテナ

A Method for System Reconstruction on Virtualization Infrastructure

Tohru Ishizaka[†1], Yoshitaka Kuwata[†1], Shigetoshi Yokoyama[†2], Kento Aida[†2]

Abstract: In this paper, we describe an available method for system reconstruction on a virtualization infrastructure. In this method, the backup-data is separated Mutable data and Immutable ones at the end of system operation. Accordingly, when it is necessary to boot the system once again, it can be easily reconstructed.

Keywords: system reconstruction, virtualization infrastructure, container

1. はじめに

現在，世界中で様々な目的のために情報システムが稼動している．さらに近年の仮想化基盤の発展により，クラウド上で，仮想マシンを用いたネットワークサービスが数多く展開されている中，これらの運用管理の重要性が増している．ネットワークサービスでは，サービスの品質向上，規模拡大などのため，仮想マシンのデプロイを適宜行う必要があり，その自動化や構成管理を行うために，Vagrant[1]や Chef[2]，Puppet[3]などのツールが用いられている．しかし，仮想マシンを中断して，新たな環境の下で再度稼動させる場合，これらのツールを使うだけでは再現させることは出来ず，中断時と同じ環境を整備するための作業が発生することが多い．また，研究結果の検証[4]など元のシステムと再現するシステムの運用者が異なる場合も考えられる．この場合は再現する側が元のシステムの情報を，必ずしも知っているわけではないことが想定され，そのため，再現が困難となることが考えられる．

本稿では，システムの再開を想定したデータアーカイブと簡易に再開する手法を示し，その実証結果を報告する．さらに，実証結果に基づいて，汎化のためのフレームワークを提案する．

本稿の構成は以下の通りである．2 章では本稿で採用した仮想化基盤 Docker について述べ，3 章では本稿で提案する手法について述べる．4 章では提案手法に基づいた実装方法を示し，5 章で実装結果を報告し，6 章で考察をする．最後に 7 章で本稿をまとめる．

2. 仮想化基盤

仮想化基盤としては，VMware, Microsoft Azure, Linux Xen, KVM など商用ベース，オープンソースなど様々な形態のものがある．その中で本研究では仮想化基盤として Docker[5]を利用した．

Docker は Linux のコンテナ技術による仮想化を行うオープンソースソフトウェアである．VMware などのハイパーバイザ型の仮想化基盤に比べて，仮想マシンの構築，停止などの処理が手軽であることから，軽量な仮想化基盤として近年注目されている．

Docker の基本的な考え方として”Immutable Infrastructure”がある．この考え方は Immutable（不変）部分は「使い捨てる」というものである．筆者らはこれに着目し，本研究で想定している情報システムの再構築を行うための手法と親和性が良いと考えた．次のようなケースを考えてみる．仮

†1 室蘭工業大学
Muroran Institute of Technology.
†2 国立情報学研究所
National Institute of Infomatics

* 本稿で使用するシステム名，製品名は，それぞれ各社の商標，または登録商標です．

想化基盤上である情報システムが動作しており，それが運用を終了したとする．その後，何らかの理由で再度稼動させる必要性が生じたが，最新の OS，アプリケーションの環境の下で，システムを再構築することとなる，というものである．図 1 に示す一般的な方法では，システムの運用終了時に必要なデータをバックアップデータとして取得し，運用再開時に OS，アプリケーションのアップデートを行い，さらに運用終了時と同じ状態で稼動するように設定を行う．これらの作業はパッケージ管理システムにより数ステップのコマンド操作と編集作業等で出来る場合が多いが，実際にはアップデートの時間がかかる，設定作業にスキル・知識が必要など，簡単に運用再開が可能な状態にすることが出来ない場合も考えられる．

また，OS を含むすべてのシステムイメージを保存し，それをそのまま稼動させるケースも想定できるが，このような運用方法は，本研究の対象外とする．

図 1　一般的なシステム再構築手法
Figure 1　Common Method of System Reconstruction

3. 本手法の概要

3.1 Mutable データの分離

本研究におけるシステム再構築手法では，システムが運用中断あるいは終了した時点で，システム運用上必要なデータの内，個々のシステムに依存する部分を各ユーザが作成したデータ（ユーザデータ）とともに Mutable データとして抽出・保存する．再構築時には再取得可能な Immutable データは外部リポジトリ等から取得し，保存されている Mutable データを取り込むことによりシステムを構築する (図 2).

この Mutable データの抽出のためには，Mutable/Immutable データの分離のための基準が必要である．表 1 に本手法における基準を示す．

一般的なデータバックアップにおいても，アーカイブすべきデータの分離は行われ，それはアプリケーションの利用で発生したユーザデータやアプリケーションの設定ファイルである．本稿では，アーカイブすべき Mutable データとして扱うのは，ユーザデータ，設定ファイルだけでなく，再開時に必要となる全てのファイルである．

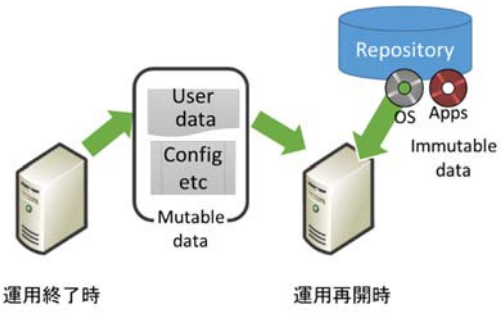

図 2　提案手法
Figure 2　Proposed Method

表 1　Mutable データ分離の基準
Table 1　Criteria for Separation of Mutable Data

種別	概要	例
Mutable	再取得不可能・困難なもの	ユーザデータ アプリケーションの設定ファイル環境
Immutable	不変なもの 再取得可能なもの	OS ミドルウェア(データベースなど) アプリケーションなど

3.2 運用終了時の動作

運用終了時には表 1 に示した基準に従って，Mutable データの検出を行う．検出方法はいくつか考えられるが，最も単純な手法は，運用開始時以降に変更されたファイルを検出することである．これは運用開始時のタイムスタンプを記録するファイルを用意し，そのファイルより新しいものを Mutable データとしてアーカイブすることで実現する．

しかし，ファイルにはプロセス情報(/proc)ファイルなどアプリケーションの動作に直接関与しないファイルも，システムの運用中は逐次更新される．また，動作情報(/var/run)は OS のプロセス状態に依存するため，ファイルを移行すると動作しない・誤った動作をするなどが推測される．従って，これらのファイルは予め Mutable データとして検出しないよう配慮する必要がある．

また，Mutable データの中には OS，アプリケーションのログなども含まれる．これらについては，運用再開時の必要性に応じて Mutable/Immutable の判断を行う．

3.3 運用再開時の動作

運用再開時には，まず Immutable データを取得し運用再開に最低限必要なデータを取得する．ここで取得されるデータとしてはベースとなる OS のイメージ，アプリケーション実体である．その後，Mutable データが展開され，運用再開に必要な環境が整えられる．これらは環境整備のためのプログラムで実行することによって自動化する．そのためには，運用終了時に Mutable データを展開する位置，権限などをプログラム内に適切に指定する必要がある．また，このプログラム自体も Mutable データとして予めアーカイブされている必要がある．

4. Docker による実装

4.1 想定するシナリオ

ここでは，2 節で述べたケースを具体化したシナリオに基づいて，本手法の実装及び検証を行う．想定シナリオは以下の通りである．

想定シナリオ

Ubuntu[6] *10.04 でアプリケーション Samba*[7] *を利用して、共有ファイルサービスを行っていた.*

しかし、5 年後，サービスを再開する必要性が生じたが，このサーバの OS はリリースから 5 年以上経過し，サポートも終了してしまっているため，Ubuntu 14.04 以上のサーバにアップデートして同一のサービスを提供することを考える.

この環境が docker コンテナ内のサービスとして提供されている場合に，本稿で提案する手法により再構築を試みた．

4.2 運用開始状態の構築

はじめに，Docker によって運用初期状態のシステムを構築する．Docker で動作するシステムイメージは，Dockerfile と呼ばれるテキストファイルに記述する．上記のシナリオに基づいて初期状態の Samba 環境を構築するための Dockerfile を図 3 に示す．

```
FROM ubuntu:trusty
RUN apt-get update -y && apt-get dist-upgrade -fy
RUN apt-get install -fy samba
EXPOSE 137 138 139 445
VOLUME ["/share"]
VOLUME ["/archive"]
RUN touch /tmp/ts0
```

図 3　Samba 用の Dockerfile
Figure 3　Dockerfile for Samba

Docker ではベースとなるイメージを指定することで，OS をインストールすることなく利用することができる．ここでは 1 行目でシナリオの初期状態である Ubuntu 10.04(trusty)を指定している．指定されたイメージは Docker のリポジトリである DockerHub から自動的にダウンロードされる．さらに，2 行目，3 行目でパッケージデータベースの更新と Samba のインストールを行い，4 行目で通信ポートの開放，5，6 行目で共有サービスのためのディレクトリと Mutable データ格納のためのディレクトリをボリュームとして作成している．Docker ではホストから直接，仮想マシンのディレクトリにアクセスできないため，ボリュームを作成して実行時に共有指定する必要がある．最終行では，touch コマンドによって Immutable データ検出に用いるタイムスタンプファイルを作成している．

build コマンドを実行することにより，Samba がインストールされた ubuntu の OS イメージが構築される．さらに run コマンドの実行によりインスタンス(Docker コンテナ)が作成される．サービス運用開始のため，Docker コンテナ内でユーザ登録と Samba の設定を行い，最後に Samba を動作させる．

4.3 運用終了

4.2 で構築したシステムが運用を終了すると同時に，Mutable データの抽出を行う．ここでは，OS 標準の検索コマンドを用いて，タイムスタンプファイルよりも新しいファイルをルートディレクトリから検出し，それをリスト化する．さらに日付が新しくても Immutable データとして扱うべき除外ファイルリスト予め作成しておき，この二つのリストから Mutable データとしてアーカイブを作成する．アーカイブファイルは運用再開時に利用できるように，ホスト側からもアクセスできる/archive ディレクトリに保存される．

実際の Mutable データとして検出されたリストを図 4 に示す．なお，検索に要した時間は仮想ディスク容量約 284MB に対して約 0.06 秒(time コマンド real)であった．

```
etc
etc/resolv.conf
etc/samba
etc/hostname
etc/mtab
etc/hosts
etc/passwd
etc/shadow
etc/group
share
var/lib/samba
var/lib/samba/registry.tdb
var/lib/samba/ntforms.tdb
var/lib/samba/passdb.tdb
var/lib/samba/ntdrivers.tdb
var/lib/samba/perfmon
var/lib/samba/ntprinters.tdb
```

図 4　Mutable データとして検出されたファイル
Figure 4　Detected Files as Mutable Data

図 4 の青字で示すように OS のユーザ情報(etc/passwd 等)と Samba のユーザ情報(var/lib/samba/passdb.tdb)が検出されていることがわかる．また，resolv.conf や hosts など，個々のインスタンスに依存し，Mutable データとして扱うべきでないファイルは除外リストに加える．なお，Docker ではここにあげた赤字のファイルはイメージ構築時に自動生成されるため，Mutable データとして検出されている．

4.4 運用再開

運用再開時には開始時と同様に Dockerfile を用いて，コンテナが作成されるが，ベースとなるイメージは Ubuntu の最新版を指定している．（図５）．

また，再構築時には最新の Samba のバージョンが新たにインストールされる．最後の[ENTRYPOINT]では，Mutable データをイメージ内に展開するプログラム(restore.sh)を動作させている．

```
FROM ubuntu:latest
RUN apt-get update -y && apt-get dist-upgrade -fy
RUN apt-get install -fy samba
  (略)
RUN
VOLUME [“/archive”]
ENTRYPOINT /bin/sh -c /archive/restore.sh >/log
2>/log && tail -f /dev/null
```

図５　再開時の Dockerfile
Figure 5　Dockerfile at Reconstruction

5. 実装結果

ここでは, 図 1 に示した一般的な再構築方法である OS, アプリケーションのアップデートによる手法（アップデート）と，本稿で述べた手法（本手法）とを比較する．

(１)運用終了時

アップデートによる手法において，運用終了時に行う作業は，Docker コンテナを保存するだけである．従って，一つのコマンドで済ませることが出来る．ただし，OS，アプリケーション全てを含んだイメージとして保存するため，保存データは大きくなる．今回のイメージでは，前述の通り，ホストマシン内のリポジトリに約 284MB の保存領域を必要とする．

本手法では，運用終了時に Mutable データの抽出及びアーカイブを行うため，アップデートによる再構築に比べ手間がかかるが，実際には，作業を自動化したプログラムを実行するだけであり，手間としては多くない．

（２）運用再開時

アップデートによる運用再開をシナリオに従って行ったところ, OS のアップデートで失敗した．一般的に 2 世代以上のメジャーバージョンアップはベンダーによってサポートされないことが多い．今回用いた Ubuntu でも，10.04 から 14.04 への直接アップグレードはサポートされていない．さらに，Ubuntu10.04 は 2014 年でメンテナンスが終了しているため, 10.04 の次のメジャーバージョンアップである 12.04 へのアップグレードも不可能となっている．

参考として，アップグレードが可能だったと仮定して，12.04 から 14.04 へのアップグレードを行ったところ, 計 24 分 10 秒の時間がかかっている．従って，2 段階のアップグレードを行う場合，10.04 の OS のディスク使用量が少ないことを考慮しても，40 分近くの時間がかかることが予想される．

さらに Samba アプリケーションを動作させるための作業がいくつか発生する．

(１)Samba パッケージのインストール
(２)OS 上へのユーザ登録
(３)設定ファイルの編集
(４)Samba サービスの起動

これらの作業は一般的に手作業で行われるため，OS 操作，Samba に関する知識などが必要となり，スキルによって運用再開までの時間は異なる．

一方，本手法では運用再開時用の Dockerfile にはリストアプログラムの動作が記述されているため，運用再開時に行う作業は，コンテナを作成するだけである．コンテナの作成時には，最新バージョンのベース OS のイメージダウンロードとイメージ構築に要した時間は, 計 2 分であった．インターネットアクセス環境は, 室蘭工業大学-SINET 間の接続が 100Mbps であり，DockerHub 及び Ubuntu リポジトリへのアクセスが発生する．

イメージ構築では，リストアプログラムが動作し，上記（２），（３）が行われた状態のファイルが展開される．また，プログラム内にサービスの起動コマンドを記述することで（４）の作業も発生しない．

しかし, Dockerfile 及び Mutable データのリストアを 4 で述べたシナリオで実行した結果，ファイル階層をそのままリストアしただけでは，運用終了時と同じサービスを提供できなかった．原因を調査したところ，運用終了時の Ubuntu10.04 で提供されている Samba のバージョンは 3.4 系なのに対して，運用再開時に利用される Ubuntu:latest(10.04)で提供されているバージョンは 4.1 であることがわかった．このバージョンの違いはファイルの階層構造の変更に現れていた．この差異を吸収するため，リストアプログラムに対象のファイルをバージョン 4.1 での格納箇所へコピーする動作を記述した．再度，運用再開時の操作を行ったところ，運用終了時と同じサービスを提供するシステムとして動作した．

6. 考察

まず, 本手法の定性的な特徴として, アーカイブ･リストアを自動的に行うために用いられる Dockerfile やシェルスクリプトは，可読性に優れたテキストファイルであり，メンテナンス性が高いことが言える．さらに複数世代の OS にまたがった再構築の検証では，終了時と再開時のアプリケーションのバージョンの違いによって，サービスが動作しなかった，という問題が発生したが，結果として差異を吸収するように改良することで解消することが出来た．従って，経年変化に対応したシステム再開が可能であることが言える．

しかし，発生した問題は，今回利用した Samba だけではなく，他のアプリケーションや OS のバージョンの違いによっても発生することは容易に推測できる．これに対応するため，再開時に実行されるリストアプログラムを変更したが, この作業にもスキルや知識が必要であり, 本手法の，簡易に再構築を行う，という利点が損なわれている．そこで本稿では, このような作業を行うことがないよう, 終了，再構築を見越したアーカイブ・リストア手法をフレームワ

ークとして整備することを提案する．

このフレームワークとしては，

(1) 終了時に OS, アプリケーションのバージョンを記録し，Mutable データの一つとしてアーカイブする．

(2) 再開時の OS, アプリケーションのバージョンを検出し，終了時と比較して差異を吸収するためのリストアプログラムを実行する．

というものである．これを実現するには，例えば，アプリケーションのバージョンごとに，差異が発生する可能性のあるファイル配置等を記録するデータベースを構築し，リストアプログラムがそれを参照して，ファイル配置を変更する処理を行うことが考えられる．このデータベースはアプリケーションの開発元が提供するのが理想的ではあるが，OS のディストリビューションによる違いも考えられるため，これも考慮したデータベースとする必要がある．

また，本手法では運用開始状態から運用終了の間に修正・追加されたファイルについては Mutable データとして扱われる．しかし，これらのデータの中でも Immutable データとして扱われるべきものも検出されることが予見される．例えば，運用中にパッケージ管理システムによってアップデートされた OS, アプリケーション等である．アップデートはアプリケーションによって頻度が異なるが，セキュリティパッチは重要性が高い場合が多い．しかし，これらを Immutable データとして扱う除外リストに手動で記入するのは困難である．これ対しては，パッケージ管理システムと連携し，動的に除外リストに加える仕組みを構築することが考えられる．

7. おわりに

本稿では，Linux コンテナ技術を用いた仮想化基盤 Docker の上で，運用を終了したシステムの再構築方法について実証を行い，再構築に必要なデータを分離し，再取得可能なデータを外部から取得することで，少ない保存領域で高速にシステムの再構築が可能であることが確認された．

今後の発展として，6 節で述べた問題点の解消方法の実現がまず挙げられる．そのためには，今回検証に用いた Samba だけでなく，様々なアプリケーションでの実証が必要である．また，多くの実証によって，提案したフレームワーク構築の実現性が高まると考える．

本稿で述べた手法は，Docker に特化したものではなく，他の仮想化基盤や物理マシンにおいても有効な手法であると思われる．また，今回は特別なツールを用いず，検出プログラム，リストアプログラムを自作したが，Vagrant, Packer などのツールを導入した手法も検討の余地がある．

なお，本稿は，国立情報学研究所の平成 27 年度研究企画会合公募型共同研究「インタークラウド活用のためのアーカイブおよびリポジトリ管理技術」として採択された研究の成果である．

参考文献

[1] Vagrant: https://www.vagrantup.com/ (2016.2.1)
[2] Packer: https://www.packer.io/ (2016.2.1)
[3] Puppet: https://www.puppetlabs.com (2016.2.1)
[4] 横山重俊, 政谷好伸, 小笠原理, 大田達郎, 吉岡信和, 合田憲人:Overlay Cloud で構成する論文再現環境，情報処理学会研究報告，2015-IOT-31(1), pp.1-8,2015 年 9 月
[5] Docker: https://www.docker.com (2016.2.1)
[6] Ubuntu: http://www. (2016.2.1)
[7] Samba: https://www.samba.org (2016.2.1)