Docker を用いたアプリケーション長期保存の試み
ALMA データの Calibrated MeasurementSet 生成システムの開発を通じて

◯林洋平、 中里剛、森田英輔、小杉城治
（国立天文台・アルマプロジェクト）

概要(Abstract)

開発から時間が経過したアプリケーションは依存するライブラリや OS を準備するのに手間や困難をきたすことがある。我々は ALMA データの Calibrated MesurementSet 生成システムでこの課題に直面し、仮想化技術である Docker を用いて課題解決を行った。従来の仮想化技術との違いや将来の起きうる課題について述べる。

1. MS 生成システム開発の背景

アルマ望遠鏡はチリのアタカマ砂漠に設置された電波望遠鏡である[1]。アルマプロジェクトでは、観測時のキャリブレーションで用いられた同一バージョンの CASA というソフトウェアを使用し Calibrated MeasurementSet（Calibrated MS）という形式のデータを生成しなければならないユースケースが存在する。電波望遠鏡の場合、ユーザの個別の科学目的に応じて Calibrated MS から FITS 形式のデータを生成するのが一般的である。Calibrated MS は ASA (Alma Science Archive)[2] では提供していない。ユーザは ASA から取得した Raw Data とキャリブレーションスクリプトを用い、みずからの計算機環境で CASA を用いて Calibrated MS を生成する必要がある。一方、一部のバージョンの CASA は RedHat6 上でのみ動作するが、RedHat6 は 2020 年 11 月にサポート終了(EoL)を迎えた[1]。RedHat6 でのみ動作する CASA が必要な事例が存在し、課題となっている。

2. Calibrated MS 生成システム

この課題に対してアルマプロジェクトの日本ノードでは、昨年からユーザの代理で Calibrated MS を生成・提供するサービスを開始した。Calibrated MS 生成システム[2]は、アルマの担当者による Calibrated MS 生成を支援する目的で我々が開発を行った。

Calibrated MS 生成システムでは、RedHat の EoL に対処するため、仮想化技術の一種である Docker を用い解決を図った。Docker を用いれば、サポート期間内である RedHat7 上で RedHat6 のアプリケーションを動作させることが可能である。今回、我々は Docker Hub で提供されている RedHat と互換性がある CentOS 6 のイメージ（KVM など従来の仮想化技術の仮想マシン（VM）に相当）をベースに、CASA 動作に必要な Python、ImageMagick などのライブラリ、CASA をインストールすることにより CASA の実行環境であるイメージを構築した。

本システムを用い Calibrated MS を生成する場合、担当者は本システムの入ったサーバに ssh でログインし、引数としてデータ ID を本システムに与えるだけで、あとは自動で処理が進む。まず本システ

---

[1] https://researchers.alma-telescope.jp/j/
[2] https://almascience.org/asax/

表１．実行性・再現性、保守性、利活用性からみたソフトウェアの長期保存

長期保存の観点		物理マシン	Docker	ハイパーバイザー型仮想化	ホスト型仮想化
1. 実行性・再現性	アプリケーションの動作維持、解析結果の再現	△	◎	○	○
2. 保守性	実行環境の維持	困難	◎	○	○
	マイグレーション	面倒・困難	容易	容易	容易
3. 利活用性	アプリケーション実行までの時間	◎	◎	△	△
	アプリケーション実行のパフォーマンス	◎	◎	◎	△
	大量処理（拡張性）	ノード増加に難あり	◎	◎	◎

ムが処理に必要な Raw Data やキャリブレーションスクリプトを ASA から取得する。その後システムは、キャリブレーションスクリプトの動作に必要な CASA の入ったイメージを起動し、キャリブレーションスクリプトを実行する。最終的に Calibrated MS が生成されるので、担当者はこれをユーザに提供することになる。

3.アプリケーションの長期保存

実行性・再現性、保守性、利活用性の３つの観点からアプリケーションの長期保存について表１にまとめた。アプリケーションの長期保存を実現する方法として物理マシン、Docker、ハイパーバイザー型仮想化、ホスト型仮想化の４環境の比較をおこなう。後者２つは Docker 以前の従来の仮想化技術である。４環境のアーキテクチャを図１に示す。今回の検証では、長期保存対象のアプリケーションは複数のバージョンが存在するという仮定に立つ。またアーキテクチャは、物理マシンの場合、直接、１物理マシンに OS やライブラリ、複数のバージョンのアプリケーションを同居させることとする。アプリケーションのバージョンごとに物理マシンを用意するのは非現実的なためである。一方の仮想化技術を用

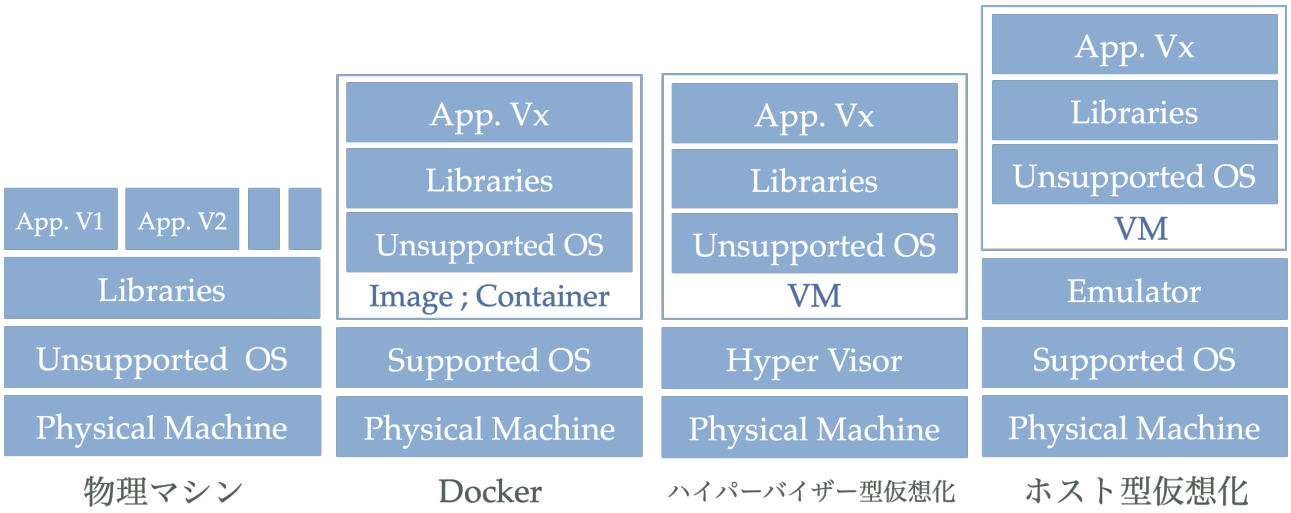

図１．ソフトウェア長期保存を実現する環境のアーキテクチャ

いる場合、アプリケーションのバージョン単位でカプセル化することとする。

3.1 実行性・再現性：アプリケーションの動作維持・実行結果の再現

アプリケーションの動作維持は将来にわたりアプリケーションが実行できるという観点から必須条件である。また、結果の再現性も CASA のように計算目的のアプリケーションでは必須条件である。これの実現にはバージョン単位でアプリケーションおよび依存ライブラリを同一にする必要がある。

物理マシンの場合、複数バージョンのソフトウェアを同居させる構成であり、アプリケーションのバージョンごとに使用するライブラリの制御が必要なことがあり実行環境維持は困難を伴う。

一方、仮想化の場合、長期保存対象のソフトウェアのバージョンごとにライブラリがカプセル化されるため、実行環境の維持が容易である。特に Docker はイメージの実行により変更された内容は破棄されるというイミュータブルな性質があり、仮に環境を大幅に変更しても、次回にアプリケーションを実行するときには変更内容は破棄される。残り２つの従来の仮想化技術の VM は常に変化するため、ライブラリに変更を加えられる問題がある。VM の利用時に、VM のコピーで環境維持が可能だが、Docker に比べ注意を要する上、システムの作り込みが必要である。

3.2 保守性

保守性は環境維持の容易さやマイグレーションの容易さであり、将来に渡りアプリケーションの動作を担保するには必須条件である。

3.2.1 実行環境の維持

仮想化技術の各評価およびその理由は実行性・再現性と同じだが、物理マシンに大きな違いがある。物理マシン上に直接構築した環境は、何らかの操作でライブラリのリンクが変更されたり、実行に必要なファイルが壊れたりすることにより変化しやすい。実行環境維持には手間が多く、困難なことが多い。

3.2.2 マイグレーション

物理マシンの場合、新しいハードウェアでのライブラリやアプリケーションのインストール作業が必要であり、開発から時間の経過したライブラリは入手困難なこともあり困難も予想される。仮想化のマイグレーションは、非常に簡単である。VM やイメージはライブラリの入手が容易なときに一度作るだけで良い。Docker の場合、イメージはリポジトリで管理されているため、サーバに Docker をインストールするだけで移行作業は終わる。残り２つの仮想化技術の場合、VM を旧サーバから移行する必要があり、作業が生じる。

3.3 利活用

利活用性は直接的に長期保存に関係するわけではないが、利用されてこそ長期保存する意義が生まれるため重要な観点である。

3.3.1 アプリケーション実行までの時間

仮想化技術の場合、アプリケーションを利用するためにはアプリケーションの起動の前にイメージや VM を起動する必要がある。

従来の仮想化技術の場合、VM の起動には数分の時間が発生する。そのため予め起動しておきアプリケーション利用に備える方法も考えられるが、使っていないときに消費する全バージョンのソフトウェアの VM で使用するメモリや CPU の待機リソースは無視できない。保守性の観点から、イミュータブ

ルな環境実現は必須である。よって、リクエストの度に VM を起動する必要がある。

一方 Docker の場合、起動が数 mm 秒であり無視できるため従来の仮想化技術に比べて優位である。

3.3.2 アプリケーション実行のパフォーマンス

ホスト型仮想化は OS 上で OS を実行する性質上、オーバヘッドが大きく、パフォーマンスへの影響は避けられない。

Docker の場合、一見 OS 上で OS が動いているように見えるが、イメージ内の OS は単なるライブラリに過ぎず、Kernel はホストのものを使用しているので物理マシンでの動作と同等である。

3.3.3 大量処理

様々な実装がありうるが、ここではジョブキューを用い、処理を捌いていく想定とする。スケーラビリティはノードを増やすことにより実現することとする。この実装の場合、すべての方式で大量処理が可能である。ただし、物理マシンの場合、環境構築が面倒なためノード追加が大変である。仮想化技術の場合、仮想化環境を用意するだけである。従来の仮想化技術の場合、キューを読み込み、VM を起動する必要から、コマンドラインで VM を操作する Vagrant (-exec) などを用いる必要がある。

3.4 アプリケーション長期保存の観点

アプリケーションの長期保存にとって実行性・再現性や保守性は必要不可欠である。一方、利活用性はユースケース次第では無意味、あるいは提示した観点以外の観点もありうる。例えばメモリや CPU のリソース消費が少ないアプリケーションの場合、従来の仮想化技術の採用でもよいかもしれない。

3.5 Docker をアプリケーション長期保存に用いる際の注意事項

セキュリティレベルはソフトウェアが開発された時点のものになるためインターネットに直接触れる環境での利用は控えるべきある。また、ハードウェアに依存したソフトウェア、特定のデバイスをコントロールするようなソフトウェアの場合、ハードウェアの保持も考慮する必要がある。最後に Linux Kernel の後方互換性の注意点を指摘する。Docker はホスト OS の Kernel を共用している。Linux Kernel は後方互換性を保持する方針で開発が進められているが、影響範囲が軽微ながら破られることもあった[3]。OS メジャーアップデート時にはイメージの動作確認、場合によっては改修が必要かもしれない。

4 まとめ

前章では Calibrated MS システムの事例を念頭におきつつ、アプリケーションの長期保存の検証を行った。全般的に Docker が良好であり、Docker を用いた実装は Calibrated MS 生成システムにとって最適な選択であったと考えている。

参考文献

[1] Red Hat, Inc. “Red Hat Enterprise Linux Life Cycle” https://access.redhat.com/support/policy/updates/errata

[2] Hayashi Y. et al. 2020, “Automated system to generate calibrated MeasurementSet in East Asian ALMA Regional Center” ADASS XXX.

[3] Linux Foundation 2015. “LSB 5.0 Release Notes”. Wayback Machine, https://web.archive.org/web/20170708050131/https://wiki.linuxfoundation.org/en/ReleaseNotes50