OCTOPUS 上における既存サービスと共存した Docker による資源提供環境の構築

渡場 康弘, 李 天鎬, 伊達 進

大阪大学サイバーメディアセンター

{watashiba-y, leech, date}@cmc.osaka-u.ac.jp

Construction of Resource Provisioning Environment by Docker Coexisting Traditional Service on OCTOPUS

Yasuhiro Watashiba, Chonho Lee, Susumu Date

Cybermedia Center, Osaka University

概要

近年、さまざまな研究分野において高性能な計算機資源を必要としている。特に、ビッグデータ解析や AI を活用する分野では新たなソフトウェアやライブラリの研究開発が活発に行われている。これらのソフトウェアでは、必要とするドライバやライブラリのバージョンでの依存関係が生じることがあるため、利用したいソフトウェアに合わせた計算環境を用意する必要がある。それゆえ、任意の環境構成を柔軟に構成でき再利用可能なコンテナ技術が着目されている。しかし、本センターのクラスタシステムである OCTOPUS では、計算機上でプログラムを実行する従来の資源提供しか行っていない。そこで、OCTPUS 上での既存サービスと共存した Docker による資源提供環境の構築を行う。本稿では、構築した Docker による資源提供環境について報告する。

1 はじめに

今日、さまざまな研究分野において、研究者はシミュレーションを高速に処理するための高性能な計算資源を必要としている。特にビッグデータ解析や AI を活用する研究などでは、データ駆動型の研究へとアプローチが変わりつつあり、高性能計算機の果たす役割は大きくなっている。しかし計算処理を行うプログラムやライブラリ、新たなソフトウェアの開発が活発に行われ、またこれらソフトウェアの機能追加等による更新も頻繁に行われている現状がある。その結果、これらのソフトウェアを利用する際、デバイスドライバやライブラリの特定のバージョンを必要とする場合があり、研究者は利用するソフトウェアの要件を満たす環境を構築する必要がある。また、ユーザが自身の PC でテスト用に構築した環境を、高性能計算機で実行する際に同じ環境を再度構築する（合わせる）必要がある。そこで、実行環境を柔軟に構築して再利用することが可能なコンテナ技術が着目されている。コンテナ技術を利用することで、使用するソフトウェアに適した環境を容易に再構築し使用することができる。

スーパーコンピュータに代表される高性能なクラスタシステムの計算資源をユーザに提供する計算機センターでは、一般的に従来の計算機上で直接ソフトウェアを実行する形態で資源提供が行われている。本形態での資源提供では一般的にサービスの安定性を重視しているため、環境構成には使用実績のある比較的古い、いわゆる枯れた状態となったバージョンのデバイスドライバやライブラリが採用されている。その結果、前述のような最先端の実行環境を必要とするソフトウェアを使用するユーザは、計算機センターが提供する計算資源をすばやく、かつ、容易に活用することができない。

そこで本センターでは、現在運用中であるクラスタシステム OCTOPUS 上で、既存サービスと共存する形でコンテナ技術の 1 つである Docker による柔軟な資源提供を可能とする環境の構築を行った。本稿では、構築した Docker による資源提供環境について報告する。

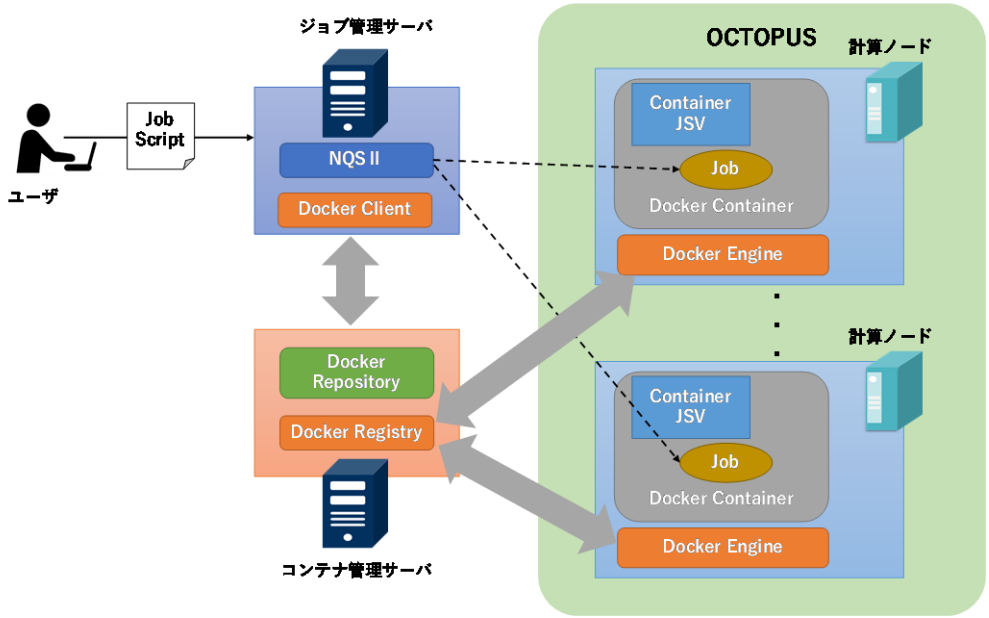

図 1：Docker による資源提供を可能とするシステムの構成

2　OCTOPUS における既存の資源提供

クラスタシステム OCTOPUS における資源提供は、高性能計算環境における資源提供で広く採用されているジョブ管理システムを採用している。ジョブ管理システムでは、ユーザは必要とする資源量および実行内容を記載したジョブスクリプトを作成し、そのファイルをフロントエンドノードからジョブ管理サーバへ、コマンドを用いて投入することで計算資源の要求を行う。投入された計算要求はジョブとして管理され、ジョブ管理システムが要求に応じて適切な計算ノードを選択して計算を実行する。

コンテナ技術を用いた資源提供を構築するにあたり、われわれは従来のジョブ投入と同様の方式での提供を検討してきた。理由の 1 つは利用方法を統一することでユーザの利便性を保持することである。もう 1 つは運用効率の観点から、クラスタシステムの計算ノードを分割して環境を構築するのではなく、従来通りまたはコンテナを利用した、どちらの利用であってもすべての計算ノードを割り当て可能とするためである。

OCTPUS で採用しているジョブ管理システムである NEC 製 NQSII は、コンテナ技術の 1 つである Docker コンテナと連携した資源割当を行う機能を有している。そこで、本機能を活用した資源提供環境を構築すべくクラスタシステムの拡張を行った。

3 Docker による資源提供環境

3.1 システム構成

NQSII による Docker コンテナを用いた資源提供環境は、使用する Docker イメージをテンプレートとして登録して提供する。そこで、OCTPUS に適用するにあたり、既存のシステム構成に対して下記の拡張が必要である。

(1) ジョブ管理サーバにおける Docker Client の導入
(2) 計算ノードにおける Docker Engine の導入
(3) Docker イメージの取得・再構築、および NQSII のテンプレートに登録するイメージの管理を行う Docker Registry を配備するためのコンテナ管理サーバの構築

これらの拡張を行った OCTOPUS のシステム構成を図 1 に示す。ジョブ管理サーバでは、(1)に示した Docker Client の導入を行うことで、NQSII における Docker 連携機能の設定、および Docker コンテナのテンプレート登録が可能となる。また、既存サービスと共存させるため、Docker を用いた資源利用専用のジョブキューの構築を行った。(2)に関しては計算ノードに Docker Engine を導入後、Docker 環境の設定および計算ノードが有する GPU を活用するため NVIDIA Docker に関する設定を行った。新規に構築を行ったコンテナ管理サーバについて、今回は仮想環境上でサーバを構築し、(3)に示した環境構成を行った。また、ユーザが個

人のホーム領域のデータにアクセスできるよう、ホスト OS 上でマウントしている領域を Docker コンテナ内にマッピングして利用可能とした。

3.2 サービス提供の形態

構築した Docker による資源提供には、まず初めに管理者による Docker イメージのテンプレート登録が必要となる。管理者は図 1 に示すコンテナ管理サーバで登録する Docker イメージを取得する。Docker イメージに対して編集が必要な場合はこの段階で作業を行い、その後コンテナ管理サーバ内の Docker Repository に Docker コマンドを用いて登録する。次にジョブ管理サーバ上で NQSII のコマンドを用いて Docker イメージをテンプレートとして登録する。テンプレート名は任意に設定可能である。また、この際にテンプレートを使用する際の CPU、メモリ、GPU などの資源量、および実行上限時間などについて既定値の設定を行う。図 2 に上述の Docker イメージの登録とコンテナの利用例を示す。

ユーザの利用方法については、基本的には通常のジョブ投入と同様であり、ジョブスクリプトに投入するジョブキュー名、実行時間、環境変数などを指定する。投入先のジョブキューについては、今回の構築では専用のジョブキューを 1 つだけ作成しているため固定となる。従来の利用方法との大きな違いとしては、指定する資源の中に利用する Docker イメージに該当するテンプレート名を専用オプションで指定することである。この形式で投入されたジョブは、図 1 に示すように指定したコンテナ内でジョブスクリプトの実行内容が実行されるようになる。逆に、テンプレート指定オプションを設定しなければ、従来どおり計算ノード上で実行される。現在、本センターの運用では各計算ノードには 1 つのジョブしか割り当てられないため、クラスタシステムの計算ノードをサービスごとに分割することなく運用が可能となる。

4 おわりに

本稿では大阪大学サイバーメディアセンターが運用する高性能クラスタシステム OCTOPUS において、従来の資源提供サービスを変更せず、新たな Docker コンテナによる資源提供を共存した形で提供する環境構成について報告した。この対応により、これまで利用が困難となっていたビッグデータ解析、AI 活用などの利用を希望するユーザに対しても資源提供が可能となると考える。

今後の課題としては、提供する Docker イメージをどのように扱うかが考えられる。現在はテスト運用の段階のため、多くの利用者が使用していると考えられる主要な Docker イメージをテンプレートとして登録している。しかし、ユーザの立場では自身の PC で使用している Docker 環境を使用して計算を行いたいとの要望が出てくることが考えられるため、より柔軟な Docker イメージの作成方法や実践的なサービスを提供するための本システム検証を行っていく必要があると考える。

@コンテナ管理サーバ
[Dockerイメージ構築＆確認]

```
$ docker image ls
REPOSITORY                    TAG    IMAGE ID
127.0.0.1:15000/oct-keras     py36   xxxxx
```

[Dockerイメージの内部レジストリへの登録]

```
$ docker push 127.0.0.1:15000/oct-keras:py36
```

@ジョブ管理サーバ
[コンテナtemplate登録]

```
$ qmgr –Pm
$ create container_template=keras-py36 ¥
        image=oct-keras:py36 ¥
        cpu=24 gpu=4 memsize=190GB ¥
        comment="tf 1.12, keras 2.2, py 3.6"
```

@フロントノード（ユーザ）
[利用可能コンテナ確認]

```
$ qstat –template
=====================================
Template      Image       CPU  Memory GPU  Comment
----------    ----------  ---- --------- ---- ---------------
keras-py27    oct-keras:py27    24  190.0G  4  tf 1.7, keras 2.0, py 2.7
keras-py36    oct-keras:py36    24  190.0G  4  tf 1.12, keras 2.2, py 3.6
chainer-py35  oct-chainer:py35  24  190.0G  4  chainer 1.4 py 3.5
pytorch-py36  oct-pytorch:py36  24  190.0G  4  pytorch 1.0 py 3.6
```

[ジョブスクリプト投入]

```
$ cat job.sh
 #!/bin/sh
 #PBS -q OCTOPUS
 #PBS -l elapstim_req=0:10:00
 #PBS --template=keras-py36
 #PBS -v TF_CPP_MIN_LOG_LEVEL=2
 cd ${PBS_O_WORKDIR}
 python sample_train.py

$ qsub job.sh
Request 601072.oct submitted to queue: OCTOPUS
```

図 2 : Docker イメージの登録とコンテナ利用例