共用計算サーバへの活用を目的としたコンテナ利用ワークフロー

A new container system constructed on JAIST facilities

宮下　夏苗 *, 間藤　真人 *, 本郷　研太 *, 井口　寧 *,
Kanae MIYASHITA*, and Masato MATO*, and Kenta HONGO*, and Yasushi INOGUCHI*,

北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology

学内共用計算サーバ上に管理者権限を持たない利用者が個々の計算実行環境を構築する作業は、サーバ環境や利用者の技量に大きく依存する。また、構築後も機器ごとにライブラリやコンパイラのバージョンアップ、または機器更新などに伴う再構築が随時必要となり、安定した環境を維持するためにもコストが要求される。本学ではこれらの問題を解決する方策としてコンテナ技術を利用し、全学向けのコンテナ利用ワークフローを構成した。ワークフローに基づいて、利用者は個々の計算実行環境を管理者権限のもとで構築、環境の保存、共有および計算実行時の呼び出しが可能となった。
keywords コンテナ ; Docker ; 計算サーバ

We, RCACI at JAIST, have recently served our users with workflow based on virtual container technology that provides an easier way to deploy and manage hardware and software/apps. In general, construction of their own computing environments on shared machines requires general users without administrative privilege to have sufficient skills and knowledge on the systems. Furthermore, upgrades/updates of software and hardware frequently compel users to reconstruct their own environments. Although users have to become familiar with our containered workflow, by using the workflow, they will be free from such annoying reconstruction and then save their time for research.
keywords Container; Docker; Computing Server

1　はじめに

学内の共用計算サーバにおいて，一般の利用者は管理者権限を与えられない．このため必要なライブラリやアプリケーションを含む計算実行環境は，個人ごとに割り当てられた作業用ディレクトリ内に利用者が個別に構築し計算の実行に利用する．

しかし，通常このような計算実行環境を準備する場合は管理者権限で yum, apt のようなパッケージ管理ツールを利用する方法が一般的であり，ドキュメントも多い．管理者権限をもたない場合は計算サーバの環境を踏まえ

* 情報社会基盤研究センター
923-1292 石川県能美市旭台 1 − 1
Research Center for Advanced Computing Infrastructure
923-1292 1-1 Asahidai, Nomi, Ishikawa

てソースからのコンパイル，コンパイル済のライブラリへのリンクといった作業を手動で繰り返す作業がメインとなり，作業者には相応の技量と労力が要求される．

さらに，このように苦労して構築した環境を利用できるのは基本的にその時構築した計算機環境上のみに限られ，ほかの計算機を使う場合，更新済のライブラリを組み込みたい場合など，毎回再構築が要求される．

本学ではこのような課題の解決に向けてコンテナの導入を試みた．利用者はコンテナを利用して必要なライブラリ，ソフトウェアをパッケージ管理ツールも活用しながらインストールし, 作成したコンテナイメージを共用計算サーバから呼び出して計算実行時に利用できる．一度作成したコンテナイメージは大学プライベートリポジトリ (以下 JAIST リポジトリ) 上のイメージとして，あるいはファイルとして保存，共有することができ，再利用，改修も容易となる．

さらに，コンテナイメージは共用計算サーバの CPU リソース，実行時間等を管理するジョブスケジューラにおけるリソースの一種として扱い，コンテナを用いない従来型の計算との同時実行（同じ計算機上で同時に稼働すること）も可能である．

参考 [1] にて，UNIX 環境に慣れていない利用者がコンテナの概念や利用法についての知識，下準備を要求されることなく, 可能な限り直感的に利用できる利用ワークフローを利用者によるコンテナ作成環境に重点を置いて説明した．本稿では JAIST リポジトリとコンテナの特性を活用した，より一般的な利用ワークフローおよびこれを実現するシステムの構成と課題について述べる．

2　関連研究

本学ではコンテナを共用計算サーバのジョブスケジューラから実行可能とし，利用者視点ではジョブスケジューラのリソースの一つとして利用させることで, 一般の利用者がサーバの管理権限を与えられず，計算環境を整えるために多大な労力を要する，構築した計算環境を容易に共有できないなど従来抱えていた問題の解決を試み，利用者が実際にコンテナを利用するワークフローを本稿に示した．

すでに実社会の様々なフェーズでコンテナ技術が活用されており，国内の大学でも参考 [3] に示されるように，学生の開発環境をコンテナで提供し，実行環境のスケールイン・スケールアウトやパブリッククラウドへのオフロードなどコンテナの特性を活用した興味深い事例が報告されている．また参考 [4] のように，High Performance Computing においてコンテナを使用した場合の性能に関する先行研究，参考 [5] のようなコンテナの実行に焦点を当てたバッチスケジューラの開発など興味深い事例が多い．計算機分野におけるコンテナの活用は今後より広まるものと期待している．

3　本学環境

本ワークフローにおいてコンテナイメージの格納場所となる JAIST リポジトリは Intel(R) Xeon(R) Silver 4116 2.10GHz, 384GB メモリを搭載したオンプレミスの VMWare ハイパーバイザ上で動作する仮想マシン上に作成し，現時点で 4vCPU, メモリ 8GB を割り当てている．仮想マシンであるため，今後の利用状況に応じたマシンスペックの増設は必要に応じて柔軟に対応できる．

最終的にコンテナイメージを用いた計算実行を想定する共用計算サーバは，現時点で Intel Xeon Gold 6130 2.1GHz 2CPU/32Core, 64GB メモリを搭載したオンプレミスの計算サーバ 48 台，同 CPU に 128GB メモリと NVIDIA Tesla P100 を搭載した GPU 付き計算サーバ 8 台の計 56 台である．

各計算サーバはジョブスケジューラである OpenPBS によって制御されている．スケジューラは登録された個々の計算ジョブが指定するリソースサイズに応じて計算ジョブを

計算サーバに割り付け，実行させる．計算ジョブのリソースサイズの要求に合致すれば，一つの計算サーバ内に 2 から 4 程度，場合によってはより多くの計算ジョブが実行されることもあり，異なるコンテナを利用する計算ジョブが一つの計算サーバ内で実行されることもあれば，コンテナを利用する計算ジョブと利用しない計算ジョブが同時に実行されることもある．

4　ワークフローの概要

4.1　全体

利用者が実際に共用計算サーバでの計算実行にコンテナを用いるには以下のフローに沿って作業する.(各項番は図 1 中の番号を参照)

① コンテナイメージ作成
② JAIST リポジトリへのアップロード
③ 共用計算サーバでのジョブ実行時におけるコンテナリソース呼び出し

コンテナイメージの作成，改修は事前準備された専用の仮想環境のほか，利用者が個々にセットアップした個人用，研究用端末などどこからでも作業できる．作成したコンテナイメージをもとに個人用，研究用端末から大規模計算機まで同一の計算実行環境を再現可能となり，手元の個人用端末，研究用端末でテストした実行環境を大規模計算機に展開する，プロジェクト，研究室などのメンバーで実行環境を共有するなど様々な活用法が考えられる．

利用ワークフロー全体の構成を図 1 に示す．

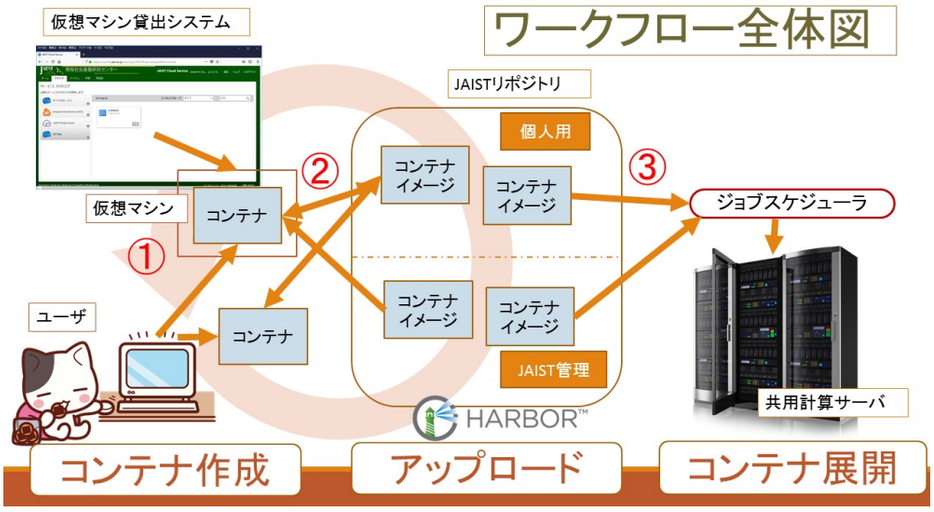

図 1　全体構成．(図中の番号 1，2，3 は本文中のワークフロー番号に対応．)

5　コンテナイメージ作成

5.1　作成手順

利用者は自分の利用したいライブラリ，ツールをインストールしたコンテナイメージを作成することで，作成イメージの共用計算サーバへの展開はもとより計算環境の共有，保存など柔軟な利用が可能になる．イメージは Docker ファイルに記述して作成することも，コンテナを起動してインタラクティブに環境を構成することもできる．Docker ファイルを利用すれば実際にイメージの構成手順，内容ごと保存しておくことができ，更新やデバック，他の利用者との共有にあたっても便利に扱えるが，本稿の主な目的である共用計算機の計算環境作成という観点においては，記法を学ぶ必要のある Docker ファイルよりも起動した

コンテナ上でインタラクティブにイメージ作成する方が直感的でわかりやすい場合もある．

以下に例 (Sample 1/2/3 を参照) として，事前に xxd ライブラリを apt パッケージでインストールした上で，plumed-2.4.3 を手動コンパイルしたイメージを作成し，アップロードする手順を示す．

手順：

(1) gcc:8 の公式イメージを Docker Hub よりダウンロード
(2) 不足のパッケージを apt-get で追加
(3) ダウンロード済の tgz ファイルをコンテナ上にマウントして展開
(4) コンパイル
(5) 作成したコンテナを my-plumed というイメージに変換，バージョン 1 のタグ付け
(6) アップロード

Sample 1　イメージ作成例：インタラクティブ作成．括弧中の数字は本文中の作成手順に対応．

```
(1,3) host$ docker run -it -d -v /home/test:/root/test --name container gcc:8.2
    /bin/bash
      host$ docker attach container

(2)   root@container:/# apt-get update
(2)   root@container:/# apt-get install -y xxd
(3)   root@container:/# tar zxvf /root/test/plumed-2.4.3.tgz
      root@container:/# cd /root/test/plumd-2.4.3
(4)   root@container:/# ./configure
(4)   root@container:/# make
(4)   root@container:/# make install
      root@container:/# exit

(5)   host$ docker commit container moby.jaist.ac.jp/test/my-plumed:1
```

Sample 2　イメージ作成例：Docker ファイル．括弧中の数字は本文中の作成手順に対応．

```
      ---Dockerfile:
(1)   FROM gcc:8
      Maintainer sentan <sentan@jaist.ac.jp>

(3)   ADD plumed-2.4.3.tgz /tmp
(2)   RUN apt-get update && apt-get install -y xxd \
      && apt-get clean \
      && rm -rf /var/lib/apt/lists/*
(4)   RUN cd /tmp/plumed-2.4.3 \
      && ./configure \
      && make \
      && make install
      --------------

(5)   host$ docker build -t moby.jaist.ac.jp/test/my-plumed:1 .
```

Sample 3　アップロード．括弧中の数は本文中の作成手順に対応．

```
      host$ docker login moby.jaist.ac.jp
      Username:
      Password:
(6)   host$ docker push moby.jaist.ac.jp/test/my-plumed:1
```

5.2　仮想コンテナホスト貸出

利用者がコンテナイメージを作成するための環境を簡単に手に入れられるよう，ひとり 1 台までのコンテナホスト用仮想マシン貸出を可能とした．

事前に管理者がカスタマイズした仮想マシンのテンプレートを Web ポータルのアイコンから選択することで，バックエンドで仮想マシンのクローンが作成され，JAIST リポジトリ用の証明書の準備や Docker の起動などの下準備がすべて完了した仮想マシンを手軽に入手できる．

貸出用のインタフェースとして, 本学で導入している VMWare vRealize Automation(以下 VRA)(図 2) を利用した．

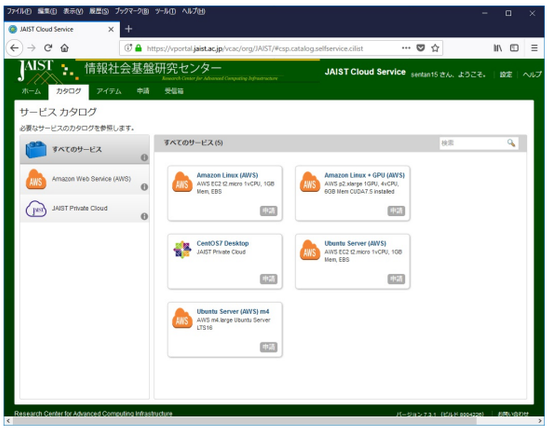

図 2　仮想マシン作成ポータル (VRA)

5.3　個人用端末の利用

JAIST リポジトリは前項のコンテナホストに留まらず学内のどこからでもアクセスできる．利用したい計算実行環境をテストするためのテストデータが手元の個人用端末からアクセスしやすい場所に格納されている場合も，電子証明書をダウンロードして個人用端末に設定すれば，docker をセットアップした個人用端末でイメージの作成と計算テストまで完了し，アップロードすることができる．

6　JAIST リポジトリ

利用者は作成したコンテナイメージを大学内に構築された JAIST リポジトリにアップロードし，イメージを利用，共有，展開する際の起点にできる．JAIST リポジトリには Web インタフェースからアクセスでき，利用者が各々の権限でアクセス可能な範囲のコンテナイメージとそれらのステータスを視認することができる．

6.1　プロジェクト

コンテナイメージは JAIST リポジトリ内に,「プロジェクト」単位に分けて格納する．各プロジェクトにはプロジェクト所有者, 開発者, ゲストの三通りの権限を付与できる．

利用者は自分が所有者となる「プロジェクト」を作成し，プロジェクト内に作成したイメージをアップロードして利用することとなる．プロジェクト作成時は利用者が別サイトの CGI フォームにアクセスし，REST API 経由で作成するものとした．Harbor のデフォルトの機能として，利用者がリポジトリの Web インタフェースから直接プロジェクトを作成することも可能であるが，本学ではこの機能は無効化している．理由は 7.2.1 に後述する．

一旦作成されたプロジェクト内には自由に自身の作成したイメージをアップロードし，イメージごとにバージョン，版数などを示すタグを付けて管理することができる．図 3 に前項で作成した my-plumed イメージを test プロジェクトにアップロードした画面を示す．

利用者所有のプロジェクトのほかに管理者が提供する基本的なイメージを格納する jaist プロジェクトを作成し，当センターで動作確認済のコンテナイメージを格納することとした．jaist プロジェクトに対して一般利用者はゲスト権限のみ付与され, イメージの読み込みと利用のみ可能となる．

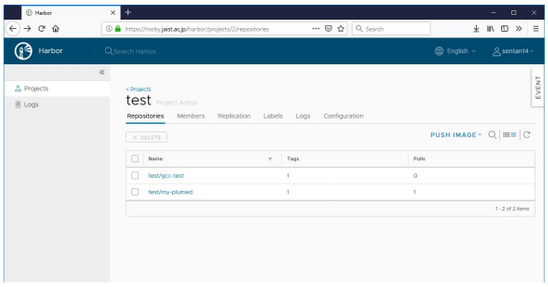

図 3　イメージとタグ

6.1.1　権限と共有

利用者が作成したプロジェクトについては作成者によるイメージの書き込み, 読み出しのほか, プロジェクトへのメンバー追加も可能である. 研究室などでコンテナイメージを共有し，イメージの更新，利用など複数メンバーで作業できる. 所有者が意図しない他人はイメージを利用することはもちろん, 視認することもできない.

6.2　共用計算サーバでのコンテナ実行

共用計算サーバではジョブスケジューラとしてオープンソースの OpenPBS を用いている. 利用者が実行したい計算内容と必要なリソースをスクリプトにまとめてジョブとして登録しておくことで, リソースが空き次第順番にジョブを実行する.

このジョブの実行前処理として, リソースとしてコンテナイメージが指定された場合に JAIST リポジトリから指定されたコンテナイメージを読み込み, コンテナ上のシェルを起動してジョブを実行する処理を加えている.

7　システムの効果と課題，および解決への方策

7.1　プライベートリポジトリの効果

一般的に機関内でプライベートリポジトリを管理することで，以下のような点がメリットとして挙げられる

- 作りこまれたコンテナイメージはデータサイズが大きくなりやすいため，学内ネットワーク内に置くことで読み込み，書き込みの速度を改善できる
- 利用者が学内で用いる ID と認証システムを利用することで，利用者間のイメージ共有，共同作成が容易となる

今回共用計算サーバでのコンテナ実行を想定したワークフローにおいて, 上記のメリットは計算サーバの計算開始までの速度の短縮, 計算機との ID の共用という点で有効であった.

次項では実際にプライベートリポジトリを学内に構築・運用したことで課題と考えられた点と検討状況について述べる.

7.2　JAIST リポジトリの課題

7.2.1　パブリックプロジェクト

Sample 4　API によるプロジェクト作成および権限追加例

```
curl -u 'admin:password' -X POST --header 'Content-Type: application/json' --
    header 'Accept: text/plain' -d 'test-project' 'https://moby.jaist.ac.jp/api/
    projects'
curl -u 'admin:password' -X POST --header 'Content-Type: app
lication/json' --header 'Accept: text/html' -d '{  "role_id": 1,  "member_user":
     {    "user_id": userID,    "username": userNAME  }}' 'https://moby.jaist.ac
    .jp/api/projects/projectID/members'
```

プライベート，パブリックを問わず一般的なリポジトリでは，最初に個人用領域を作成し領域内に個人ごとのコンテナイメージを格納する．ここで，JAIST リポジトリにおいては作成された領域の無認証化 (パブリック化＝アクセス可能なネットワーク内からの無認証ダウンロードの許可) を許容するか否かが問題として挙げられた．

現時点で JAIST リポジトリの構築にはオープンソースの VMWare Harbor を利用しており，基本機能として利用者による個人用領域（プロジェクト）作成の許可/不許可を設定できる．プロジェクト作成を許可すると利用者権限で UI からのプロジェクト作成が可能になり，利用者の利便性が向上する．しかしデフォルトの UI 構成ではプロジェクト作成時のチェックボックスひとつでプロジェクトがパブリック化されるため，利用者の操作ミス等で意図せずプロジェクト内のイメージが学内ネットワーク内でパブリック化され，無認証ダウンロードが可能になることが問題と考えられた，また，必要のないプロジェクトが全利用者に公開されて UI 上も視認性を損なう，管理者側でイメージの利用者を特定できない公開プロジェクトが乱立する可能性があることなどからも，大学管理リポジトリとして望ましくない．

そこで，利用者によるプロジェクト作成は設計上許可せず，別立ての CGI フォームからプロジェクト作成を申請，REST API 経由で作成するワークフローとした．API の実行例を Sample4 に示す．現在，フォームを作成中である．

7.2.2　イメージ管理

JAIST リポジトリ内のイメージについて，標準では自動削除の機構がないため所有者が修了，異動した後にも不要となったイメージが残り続けることとなり，大学内のストレージリソース浪費にもつながることから管理上望ましくない．現時点では以下のような定期削除案を検討している．

案 1．イメージの最終 pull 日付に基づいてイメージを削除

長期履修や研究プロジェクトなど，日付が古くとも利用可能性があるイメージや保管しておきたいイメージも削除される可能性がある．

案 2．所有者が LDAP サーバ上で修了のステータスになった後にプロジェクトを削除

Harbor が利用するアカウント情報は，認証は大学の LDAP サーバを利用するもののアカウントデータベースは JAIST リポジトリ内に，作成順を UserID として自動作成される設計となっている．LDAP サーバと Harbor 上のアカウントデータベースは同期していないため何らかのステータス同期手段を講じる必要がある．

7.2.3　脆弱性診断

Harbor が標準機能としてコンテナイメージの脆弱性静的チェックツールである Clair に対応している．コンテナイメージのひな形の選定，イメージ作成は利用者に任されているため，意図せず脆弱性を含んだライブラリが取り込まれてしまう可能性も想定される．脆弱性診断によりこのようなリスクを視認させ，更新を促す効果が期待される．図 4 に脆弱性診断結果の例を示す．

このほかプロジェクトごとに，イメージ更新時にスキャンを行うオプションを付与できるため，API によりプロジェクト作成時にオプション追加しておくことを想定している．

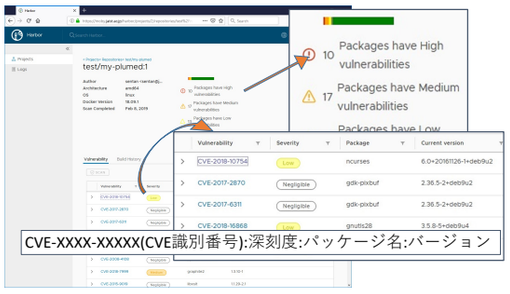

図 4　イメージの脆弱性情報 (Harbor)

7.3 共用計算サーバでのコンテナ実行

7.3.1 サーバ上に残るイメージの削除

共用計算サーバでイメージを読み出して実行した後, サーバ内にイメージが残る．このイメージの削除タイミングの検討が課題として挙げられる．

計算サーバ内に前回のイメージが残っていればイメージダウンロードの時間を削減できる．とくに利用者が作成したイメージはデータサイズが大きくなり，イメージに簡単なライブラリを追加するだけでも簡単に数 GB から数十 GB 程度の容量になる． このため，前回利用したイメージを再利用できれば計算の実行の上では効果的な時間短縮になりうる．

しかし多数の利用者が共用する計算サーバであれば内部に残るイメージのデータサイズが相当の容量になり, 場合によってはサーバのディスク容量を圧迫し障害の原因にもなりうる．また利用されなくなったイメージがサーバ内に残ってしまうことにもなるため，適切なタイミングでのイメージ削除が必要である．

現時点では，週に 1 度のタイミングで，サーバ上でコンテナとして起動されていないイメージの全削除タスクを動作させる方法を検討している．

7.3.2 コンテナ呼び出しにおける権限

共用計算サーバでイメージを呼び出して実行するにあたり，最低でもイメージの格納されたプロジェクトに対するゲスト権限が必要になる．

共用計算サーバ専用のユーザを作成し，これを各利用者のプロジェクトに自動的に追加する方式を検討しているが，ゲスト権限でのサーチによって利用者自身には権限のないプロジェクトが検索でき，イメージの呼び出しも可能となってしまうリスクがある．

これを防ぐためには共通ゲストユーザを用いず計算実行の際に都度本人の権限で JAIST リポジトリにログインする方法が考えられる．しかし，ひとつの計算機を複数の利用者が同時に利用する共用サーバ環境では，JAIST リポジトリへの重複ログインができず先にログインした利用者の権限が引き継がれてしまうなどの問題が起きうる．

現時点では，リスクを説明したうえでの利用を呼びかけるとともに，コンテナを利用した計算の実行開始時にゲスト権限にて JAIST リポジトリにログイン，コンテナダウンロードの完了時にログオフする処理が実現できれば，現実的なリスク低減につながると考えている．複数の計算が同時に実行される環境においてのログイン，ログオフ処理が期待通り実行できるか引き続き検証したい．

7.3.3 パフォーマンス

コンテナ上で計算が実行されることでのパフォーマンス劣化がどの程度のものか，あらかじめ検討しておきたい．

以下に参考として，1CPU, 4GB の仮想マシンを用い，実計算とコンテナ実行のパフォーマンスを積分計算の実行速度測定 (表 1)，姫野ベンチマーク [2] L サイズでの実行性能測定 (表 2) の 2 種類を用いて簡単に比較した結果を示す．ここで,「実計算」とは仮想マシン上でそのまま実行した結果,「コンテナ上計算」とは仮想マシン上に呼び出したコンテナ上のライブラリを用い，コンテナ上でコードをコンパイルして実行した結果である．コンパイラは実計算，コンテナ上計算のいずれも gcc4.8.5 を利用した．

今回試した程度の小規模計算において，懸念される実行性能の低下は見られず，逆にコンテナ上の実行速度が非常にわずかながら速いという結果になっている．コンパイラは同じバージョンの gcc を用いているが，その他のライブラリ等の環境によるものか，CPU などのリソースが効率的に割り当てられているなどの理由によるものかなど，今後並列実行なども考慮に入れた上で動作検証を行いたい．

表 1　積分計算実行時間 (秒)

	実計算	コンテナ上計算
1 回目	102.26	99.92
2 回目	102.15	99.98
3 回目	102.19	99.99

表 2　姫野ベンチによる性能測定 (MFLOPS)

	実計算	コンテナ上計算
1 回目	251.6	252.5
2 回目	252.3	252.6
3 回目	251.6	253.1

8　まとめ

本学では学内利用者が計算機での実行に活用するためのコンテナ利用ワークフローを構成し，必要となるコンテナリポジトリを含む試験環境を作成した．結果，各利用者にリポジトリの適切な利用権を付与する機構を別途作成するほうが望ましいこと，リポジトリ内にアップロードされたイメージおよび，共用計算サーバ内に蓄積される実行済みイメージの適切な削除方法とタイミングを検討する必要があること，共用計算サーバからコンテナイメージを呼び出す際の権限を適切に設定する必要があることが，システム公開までに解決すべき主な課題であると判った．現在はこれらの課題の解決と JAIST リポジトリの学内公開に向け，準備を進めている．

著者略歴

宮下 夏苗 2003 年 北海道大学大学院工学研究科情報システム工学専攻修了．修士． 同年北陸先端科学技術大学院大学 情報科学センター (現情報社会基盤研究センター) 技官，2018 年同技術専門職員．

間藤 真人 1999 年 富山大学大学院工学研究科博士前期課程修了，修士．同年北陸先端科学技術大学院大学　情報科学センター (現情報社会基盤研究センター) 技官．2016 年同技術専門職員．

本郷 研太 2000 年 東北大・工・金属卒. 2005 年 東北大院・工・材物了, 博士（工学）. 東北大・金研, 北陸先端科学技術大学院大学・情報, ハーバード大・化学 (海外学振研究員), 統数研を経て, 2012 年北陸先端科学技術大学院大学・情報・助教, 2017 年同・情報社会基盤研究センター・准教授. 現在, JST さきがけ研究員, NIMS 特別研究員を兼任. 材料計算科学, マテリアルズ・インフォマティクス研究に従事.

井口 寧 1991 年 東北大学工学部機械工学科卒業 1994 年〜1997 年 日本学術振興会特別研究員．1997 年 北陸先端科学技術大学院大学 情報科学研究科 博士後期課程修了．同大学情報科学センター助手，准教授を経て， 現在，情報社会基盤研究センター 情報環境研究開発部門 教授．また，2002 年から 2006 年まで科学技術振興機構さきがけ研究 21(機能と構成) に参加し研究に従事．2008 年〜2009 年 米国南フロリダ大学上級客員研究員．この間並列システム，ウェーハスタック集積システム，FPGA などを用いた並列処理に関する研究を行なう．IEEE, 日本バーチャルリアリティ学会各会員， 情報処理学会，電子情報通信学会各シニア会員．

参考文献

[1] 宮下夏苗，間藤真人，本郷研太，井口寧，「共用計算サーバにおけるユーザ作成コンテナの実行を目的としたコンテナ作成ワークフローと実行環境の試作」，大学 ICT 推進協議会年次大会 2018.

[2] 姫野ベンチマーク， http://accc.riken.jp/supercom/documents/himenobmt/

[3] 東京工科大の学生が学内システムを Docker で開発、その舞台裏を聞く， https://cloud.watch.impress.

co.jp/docs/case/689058.html

[4] Minh Thanh Chung, Nguyen Quang-Hung, Manh-Thin Nguyen, Nam Thoai, "Using Docker in High Performance Computing Applications" , 2016 IEEE Sixth International Conference on Communications and Electronics (ICCE), 2016, pp.52-57

[5] Olivier Sallou ; Cyril Monjeaud , "GO-Docker: A Batch Scheduling System with Docker Containers" 2015 IEEE International Conference on Cluster Computing, IEEE, 2015, pp.514-515