Vol.2021-HPC-179 No.3
2021/5/14

インタークラウド環境構築システムVCPによるHPCクラスタ環境構築支援

竹房 あつ子[1,2,a)]　佐賀 一繁[1]　横山 重俊[1]　丹生 智也[1]　合田 憲人[1,2]

概要：複雑なライブラリインストールの簡易化やDockerエコシステムの活用のため，HPC分野でもコンテナが活用されるようになってきた．また，クラウドやオンプレ環境で小規模なHPCクラスタを構築して利用する需要もあるが，HPCクラスタ向け各種ライブラリのインストールやGPUノードの利用設定，Singularityのような HPC用コンテナの設定など，研究者による HPCクラスタの構築や管理の負担は非常に大きい．本研究では，インタークラウド環境構築システム VCPを用いて HPCクラスタを構築，管理するための手順書である HPCテンプレートを開発した．本テンプレートは複数の Jupyter Notebookで構成されており，OpenHPCライブラリ群を用いた HPCクラスタの構築し，Singularityや GPUノードの設定やサンプルプログラムの実行にも対応している．予備実験により，HPCテンプレートで構築した HPCクラスタの有用性を示す．また，テンプレートの品質維持と開発効率向上のための開発自動化の仕組みも開発した．

Support for an HPC Cluster Construction by Using the VCP Inter-cloud Application Environment Construction System

1. はじめに

OSレベルの仮想化技術であるコンテナの普及により，高性能計算（HPCと呼ぶ）分野においてもコンテナが活用されるようになってきた [1]．HPC分野では，複数ユーザの並列ジョブを効率よく処理するため，大規模な並列計算機システム（HPCクラスタと呼ぶ）を構築してバッチジョブスケジューラで資源管理して利用するのが一般的である．各ユーザは，OSの ACL（Access Control List）で保護されているのみで同じ計算機を複数のユーザで共有し，HPCクラスタ管理者により整備された計算ライブラリを用いて，必要なプログラムをコンパイル，実行する．よって，予め用意されていないライブラリの利用では，複雑なライブラリインストールの手間が発生し，インストール済のライブラリと競合する場合は利用ができないこともある．主要なコンテナランタイムである Dockerは，予め整備されたコンテナイメージを用いることでインストールの手間が軽減でき，ライブラリ競合の問題も解決できるため，これらの課題を解決する技術として注目されている．HPCシステムにおいて Dockerコンテナを利用する際の課題として，root権限がなければ実行できない，仮想化のオーバヘッドが許容できるか明らかでないなどが挙げられるが，これらについても Shifter[2]，Singurality[3]，Charliecloud[4] のような HPC用コンテナや，Rootlessのような root権限を必要としない汎用コンテナの開発が進んでいる．これらは Dockerイメージとの互換性を維持しているため，HPC分野でも Dockerのエコシステムの活用が可能になる．また，機械学習分野では GPUを活用したコンテナの利用も期待されている．

学術機関で提供される大規模 HPCクラスタに対して，クラウドまたはオンプレの計算機上で小規模な HPCクラスタ環境を構築して利用する需要も依然として高い．研究活動において急遽大量の計算資源を利用する必要がある場合，学術大規模 HPCクラスタでは利用手続きに時間を要したり，繁忙期や年度末には利用開始できないこともある．また，事前に小規模な実験を手元の環境で繰り返して大規模な実験を大規模 HPCクラスタで実施する使い方も

[1] 国立情報学研究所
National Institute of Informatics
[2] 総合研究大学院大学
The Graduate University for Advanced Studies (SOKENDAI)
[a)] takefusa@nii.ac.jp

一般的であり，定常的に利用する計算機は購入して運用した方が総じてコストもかからない．しかしながら，研究者による HPC クラスタの構築，管理の負担が課題となる．特に，GPU ノードで構成されるような HPC クラスタに大規模 HPC クラスタと同等のライブラリや Singularity 等の HPC 向けコンテナ環境をライブラリの依存関係を考慮して整備するのは非常に困難である．

国立情報学研究所（NII）では，Docker コンテナを利用してインタークラウド環境での資源管理を行う基盤ソフトウェア VCP（Virtual Cloud Provider）[5], [6] を開発し，VCP を用いてアプリケーション環境構築を支援する「学認クラウドオンデマンド構築サービス」（OCS）[7] を学術研究機関に提供している．VCP は，様々なクラウドの IaaS（Infrastructure-as-a-Service）の制御 API を抽象化するインタフェースを提供し，異種クラウド環境を一つの仮想的な計算環境として容易に操作，管理することを可能にする．一方，OCS では学術情報ネットワーク SINET5[8], [9], [10] とクラウドの利活用を促進することを目的として，（1）SINET クラウド接続サービス [11] を用いた VPN 構築等の初期導入支援，（2）VCP によるオンデマンド構築機能の提供，（3）アプリケーション環境構築テンプレート等の情報共有を行っている．特に（3）では，研究・教育目的のアプリケーションの環境構築，管理のための手順書を Jupyter Notebook[12] 形式で公開している [13]．しかしながら，各アプリケーションに関連するソフトウェアライブラリや VCP そのもののアップデートへの対応もあり，各テンプレートの開発・保守のコストが非常に高いという課題もある．

本稿では，VCP と Jupyter Notebook を用いてクラウドまたはオンプレ環境に容易に HPC クラスタ環境を構築することを支援する HPC テンプレートを開発する．本テンプレートでは，OpenHPC 2.x[14] で提供している HPC 向けライブラリ群を配備した HPC クラスタの構築を容易にする．また，構築する HPC クラスタでは GPU および Singularity の利用も可能にする．実験から，構築した HPC クラスタ上で NVIDIA 社が公開している NGC カタログ [15] のコンテナイメージを Singularity コンテナ化して GPU ノードで利用できるようにするとともに，予備評価により有用性を示す．NGC カタログでは，深層学習，機械学習，HPC 向けコンテナイメージが公開されている．また，HPC テンプレートおよびその他の OCS アプリケーションテンプレートの品質維持と開発効率向上のための開発自動化の仕組みも開発した．

2. 基盤ソフトウェア VCP とアプリケーションテンプレートの概要

学認クラウドオンデマンド構築サービス（OCS）で利用している基盤ソフトウェア VCP の概要と，VCP を用いてアプリケーション環境構築を支援するアプリケーションテンプレートの概要について説明する．

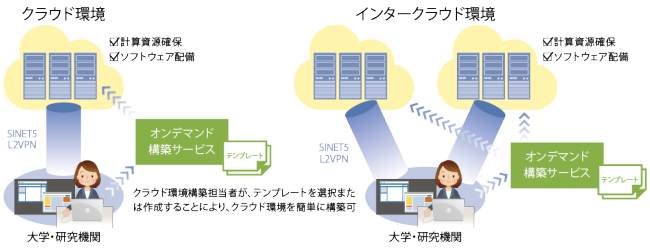

図 1　基盤ソフトウェア VCP の利用イメージ．

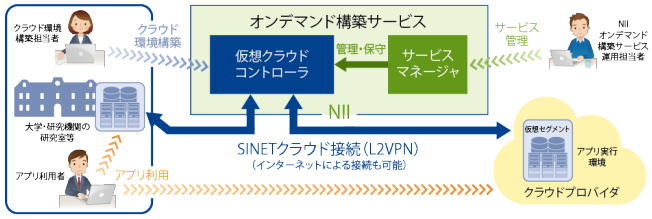

図 2　VCP の構成．

2.1 基盤ソフトウェア VCP

VCP は，図 1 で示すように SINET5 のような VPN（Virtual Private Network）に接続された 1 つ以上のクラウドを 1 つの仮想クラウドとみなし，仮想クラウド上の物理計算機（BM）や仮想計算機（VM）上でのアプリケーション環境構築を支援する．また，VCP では Docker コンテナを用いて低性能オーバヘッドで容易にアプリケーション環境を配備できる，異なるクラウド API の差異を吸収するサービスインタフェースを提供する，という特徴を持つ．

図 2 に，OCS での VCP の配備構成を示す．VCP は，主に VCP サービスマネージャ，仮想クラウドコントローラ（VC コントローラ），仮想ルータで構成される．VCP マネージャは，仮想クラウドを管理する VCP の利用者／グループに対して VC コントローラを配備するとともに，各利用者に対して VC コントローラへのアクセス制御に利用するトークン配布を行う [16]．VCP の利用者はそれぞれ異なる仮想プライベートネットワーク（VPN）を利用するため，VC コントローラは利用者毎に配備する．各利用者は，割り当てられた VC コントローラに対して VCP のサービスインタフェースを介して仮想クラウド内の計算資源の制御を行う．また，仮想ルータは大学等学術研究機関と複数商用クラウドデータセンター間のネットワークのハブ的な役割を担うとともに，必要に応じて BGP による経路制御機能も提供する．クラウド間ネットワークは SINET5 の L2VPN を前提としているが，L2VPN 環境がない，または高性能ネットワークが必要ないアプリケーションを利用する場合は，IPSec トンネルを用いたクラウド間 VPN を構築して利用することができる．

VCP では，様々なクラウド上の計算機を統一的に管理で

きるようにすることと，かつ様々なアプリケーションの既存 Docker イメージを再利用できるようにすることが求められる．よって，VCP による計算ノード起動ではクラウドで確保した VM ／ BM 上にベースコンテナとアプリケーションコンテナを二階層で配備して利用することを前提としている．このような利用形態は Docker-in-Docker と呼ばれており，多少の性能劣化が懸念されるものの柔軟なアプリケーションライブラリの配備を可能にする．ベースコンテナ層ではネットワーク設定や VCP で起動したノード（VC ノード）の死活監視，メトリクス収集を行う．アプリケーションコンテナ層では，目的のアプリケーションで必要とされるコンテナを配備する．ベースコンテナは各 VM ／ BM に 1 つ配備されるものであり，アプリケーションコンテナはベースコンテナ上に複数配備することができる．

VCP では，利用者に対して統一的にクラウドを制御するためのサービスインタフェースとして VCP REST API と Python ベースの開発キット VCP SDK を提供している．サービスインタフェースの詳細は文献 [6] に記している．また，VC コントローラ内部で Terraform[17] を用いて異なるクラウドプロバイダの API の差異を吸収している．Terraform では，各クラウドにおける VM やネットワークスイッチ，コンテナ等の資源を制御するモジュールを Provider と呼び，主要な Provider が予め提供されている．また，VCP のクラウドプロバイダ対応モジュールもプラグイン化されており，拡張が容易になっている [16]．現状では，AWS（Amazon Web Services），Microsoft Azure，さくらのクラウド，Oracle Cloud Infrastructure，北海道大学インタークラウドシステム，VMware vSphere（オンプレ環境）をサポートしている．

2.2 アプリケーションテンプレート

アプリケーションテンプレートは，研究・教育目的で広く利用されているアプリケーションの構築・運用手順をテンプレート化したものを Jupyter Notebook 形式で提供している．各テンプレートは，VCP を用いてクラウドでのアプリケーション構築を可能にするため，VCP SDK を用いて開発されている．これにより，クラウドでのアプリケーション構築に不慣れな利用者に対してクラウドの利活用を支援するとともに，各アプリケーションコミュニティで適宜テンプレートを拡張して情報共有されることを期待している．

現時点では，以下のアプリケーションテンプレートを公開している [13]．

LMS テンプレート　moodle[18] を用いた学習管理システムを構築，運用することができる．短いダウンタイムで更新作業を行うことを目的とし，BlueGreenDeployment[19] という手法に基づく更新手順を記している [20]．

LMS テンプレート簡易構成版　LMS テンプレートより短期的な利用を想定した簡易構成版も用意した．また，簡易構成版では，VCP を利用せずに AWS または Azure に直接 Moodle 環境を構築する手順も合わせて公開している．

HPC テンプレート　OpenHPC[14] v1.x で提供されている科学技術計算のための計算ライブラリ群およびバッチジョブスケジューラからなるクラスタ環境を構築と，構築したクラスタでの基本的な性能評価が行える．

講義演習環境テンプレート　Jupyter Notebook を用いた講義演習環境を構築するもので，講義演習環境の基盤ソフトウェアには JupyterHub を講義演習用に NII で拡張した CoursewareHub[21] を用いている．CoursewareHub では，教材配布，課題の回答収集，操作履歴の収集等の機能が提供されている．

2.3 新しい HPC テンプレートの必要性とテンプレート開発の課題

VCP の HPC 用途での利用の課題と，テンプレート開発の課題について以下に述べる．VCP では，VM または BM 上にベースコンテナを配備し，アプリケーションコンテナをベースコンテナ上に起動して Docker-in-Docker（D-in-D）構成で利用することを前提としている．D-in-D のオーバヘッドはある程度許容できるものの [6]，HPC 環境ではオーバヘッドをできる限り小さくすることが求められる．また，新たなアプリケーションコンテナを配備する場合，三階層の仮想化が必要になるケースが考えられ，そのオーバヘッドが許容できるか明らかでない．また，2020 年 10 月時点で CentOS8 または OpenSUSE Leap 15 を前提とした OpenHPC v2.x がリリースされるなど新しいライブラリへの対応が必要であるとともに，ベースコンテナ上での一般ユーザによる root 権限の利用制限や，GPU ノードおよび Singularity への対応など機械学習処理の需要を満たすことも必要である．

上記のように新しいライブラリのリリースへの対応だけでなく，各アプリケーションに関連するソフトウェアライブラリや VCP そのもののアップデートへの対応もあり，各テンプレートの開発・保守のコストが非常に高いという課題もある．アプリケーションテンプレートでは，誰が実行してもテンプレートに従って実行すれば想定しているアプリケーション環境の構築ができることを目指している．しかしながら，気づかないうちに特定のライブラリが更新されてアプリケーションのインストール作業に失敗したり，公開されていたコンテナイメージやデータファイルが削除されて手順どおりに実行できなくなることもある．よって，開発・保守のコストを軽減するため，ソフトウェア開発で行われている CI/CD（継続的インテグレーション／継続的デリバリー）のような開発自動化の仕組みが，

表 1　HPC テンプレート v1 と v2 の比較.

HPC テンプレート	v1	v2
OpenHPC ver.	1.3.x	2.x
ベースコンテナ OS	CentOS 7.7	CentOS 8
対応アーキテクチャ	x86_64	x86_64
対応スケジューラ	Slurm	Slurm
Singularity サポート	✓	✓
GPU サポート	-	✓

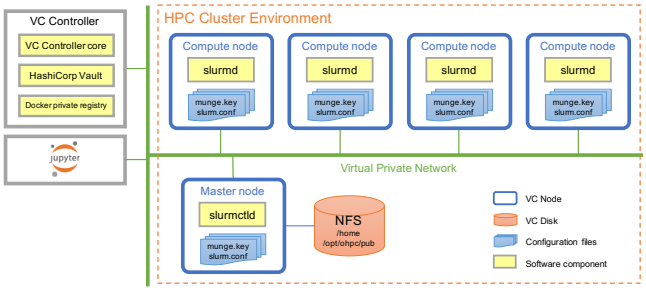

図 3　HPC テンプレートで構築する HPC クラスタの概要.

Jupyter Notebook で開発されているアプリケーションテンプレートにも不可欠である.

よって，本稿では OpenHPC v2.x および GPU ノードに対応した HPC クラスタの構築・管理を可能にする HPC テンプレートを開発する．また，開発自動化の仕組みとして Jupyter Notebook ベースのアプリケーションテンプレートの CI/CD 環境の構築について述べる.

3. HPC クラスタ構築テンプレートの開発

本稿では，OpenHPC v2.x をベースとした HPC クラスタを構築する HPC テンプレート v2 の開発について紹介する．OpenHPC は Linux Foundation プロジェクトの一つであり，オープンソース HPC ソフトウェアコンポーネントとベストプラクティスのリファレンスコレクションを提供している．HPC テンプレートでは，通常の VCP アプリケーション環境と異なり，必要なライブラリを予めベースコンテナイメージにインストールしておき，VC ノードを起動して簡単な設定のみで利用できるようにする．表 1 に，公開中の HPC テンプレート v1 と本稿で開発する v2 の違いをまとめた．v1，v2 とも，バッチジョブスケジューラは Slurm，CPU アーキテクチャは x86_64 を対象としており，Singularity の利用も可能である．GPU ノードのサポートは，VCP としては対応可能であるが，v1 テンプレートでは未対応となっている.

図 3 に VCP と開発する HPC テンプレートで構築される HPC クラスタの概要を示す．本テンプレートを用いることで，NFS サーバを兼ねた Slurm マスターノード（Master node）と数台の計算ノード（Compute node）で構成される HPC クラスタの構築，管理が容易に行えるようになる．以降では，HPC テンプレート v2 で開発した HPC 用ベースコンテナイメージの概要と，Jupyter Notebook で開発された HPC テンプレートの詳細について述べる.

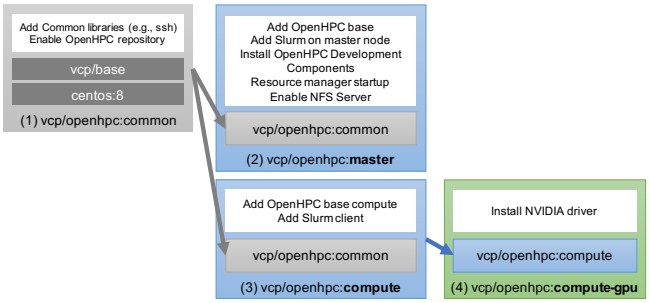

図 4　HPC テンプレート v2 用ベースコンテナイメージ.

3.1 HPC テンプレート v2 のベースコンテナイメージ

OpenHPC の GitHub サイト [22] で公開されているドキュメント「Install Guide (with Warewulf + Slurm)」を元に開発した．インストールされるライブラリ群の詳細は，本ドキュメントを参考されたい.

図 4 に HPC テンプレート v2 用ベースコンテナ用の Docker イメージの概要を示す．イメージファイルは，以下の 4 つで構成される.

(1) vcp/openhpc:common：共通ベースイメージ
(2) vcp/openhpc:master：マスターノードイメージ
(3) vcp/openhpc:compute：計算ノードイメージ
(4) vcp/openhpc:compute-gpu：GPU 計算ノードイメージ

(1) 共通ベースイメージは，HPC テンプレートで利用する VC ノードすべてに共通するソフトウェアのインストールの設定を行うものであり，VCP と CentOS8 のイメージをベースとして ssh のような基本的なソフトウェアのインストールや OpenHPC リポジトリをローカル環境で利用可能にする．(2) マスターノードイメージは，HPC テンプレートのマスターノード用のベースコンテナイメージであり，(1) 共通イメージをベースとして，マスターノードに必要なソフトウェアを設定する．具体的には，OpenHPC のベースライブラリ，Slurm マスター用ライブラリ，OpenHPC 開発用ライブラリのインストールや，NFS サーバ設定などを行う．(3) 計算ノードイメージは計算ノード用のイメージであり，(1) をベースとして OpenHPC の計算ノード用ライブラリおよび Slurm クライアント用ライブラリをインストールする．また，GPU ノードを計算ノートとして利用する場合は (4) GPU 計算ノードイメージを用いる．(4) では，(3) をベースとして NVIDIA GPU 用ドライバ，ランタイム，および NVIDIA CUDA Toolkit をインストールする．VCP v21.04 では，Docker の GPU 連携機能である `docker run --gpus` オプションに対応したため，ベースコンテナでベンダ固有のドライバのインストールが不要になると期待される．しかしながら，我々が想定しているユースケースの 1 つである NVIDIA 社が提供する機械学習

用アプリケーションコンテナ（NGC）の実行では CUDA がベースコンテナにインストールされている必要があり，CUDA とドライバのインストールが一体化されているため，GPU ノード用のイメージを別途用意した．

3.2 HPC テンプレート v2 の構成

開発したベースコンテナイメージを利用して HPC クラスタ構築・管理を行う手順を，手順ごとに複数の Jupyter Notebook で構成された HPC テンプレートとして開発した．図 5 に HPC テンプレート v2 の構成を示す．HPC テンプレートは以下の Notebook で構成されており，Notebook に従って HPC クラスタ環境構築，ジョブの実行，設定変更，HPC クラスタの停止／再開，破棄を行うことができる．ジョブの実行では，標準的なベンチマークプログラムである Linpack の実行や，Singurality による Docker イメージの実行，NGC カタログのコンテナ実行の手順も示している．

010: パラメータ設定　構築する HPC クラスタを構成するノードに関するパラメータを設定する．具体的には VCP および VC ノードに関するパラメータ，ノード構成や VM サイズなど HPC クラスタに関するパラメータ，クラスタの認証サービス MUNGE や Slurm のパラメータ設定を行う．

020: OpenHPC の起動　図 3 の HPC クラスタを構成する VC ノード，VC ディスクを VCP で作成する．VC ディスクは，AWS の EBS のようなブロックストレージサービスを VCP から利用するものであり，マスターノードの NFS ボリュームとして利用する．ノード群の起動後，Slurm の動作確認を行うとともに，ノード群の操作を容易にするために Ansible の設定もここで行う．VCP で起動した VC ノードは，通常の計算ノードと同様に扱うことができる．

030: 設定ファイルの編集　設定ファイルを編集し，編集後のファイルを各計算ノードに配布する．

031: 設定ファイルの編集–slurm.conf　設定ファイル `slurm.conf` を編集し，編集後のファイルを各計算ノードに配布して，変更を反映させる．

032: 設定ファイルの編集–GPU(GRES) の登録　GPU などを Slurm の Generic Resource（GRES）として追加するため，`slurm.conf` を編集して編集後のファイルを各計算ノードに配布し，変更を反映させる．計算ノードに GPU ノードを利用する場合は，この Notebook を実行する．

051: ユーザの追加　HPC クラスタにユーザを登録する．

052: ユーザの削除　HPC クラスタからユーザを削除する．

061: Singularity のロード　Singularity をデフォルトで利用できるようにするため，`/etc/profile.d/`の設定ファイルを変更する．

062: Singularity のアンロード　Singularity をデフォルトではロードされないようにするため，`/etc/profile.d/`の設定ファイルを変更する．

121: Linpack ベンチマーク–HPL　HPL[23] を実行するための準備作業を行った後，HPL を実行する．

140: Singularity の利用　Singularity で Docker コンテナイメージが実行できることを確認する．

141: NGC Catalog のコンテナを実行する–PyTorch　NGC カタログから PyTorch の Docker コンテナイメージを取得して Singurality イメージに変換し，PyTorch コンテナで MNIST を実行する．コンテナから GPU が利用できていることも確認する．

142: NGC Catalog のコンテナを実行する–TensorFlow　141 と同様に，NGC カタログの TensorFlow コンテナイメージを取得して MNIST を実行する．コンテナから GPU が利用できていることも確認する．

911: ノードの停止　VCP のノード停止機能を用いて HPC クラスタを構成する VC ノード群を停止する．

912: ノードの再開　VCP のノード再開機能を用いて HPC クラスタを構成する VC ノード群を再稼働させる．再稼働後，Slurm クラスタのノードの状態を確認する．

920: OpenHPC 環境の削除　構築した HPC クラスタを削除し，削除した環境に対応する Ansible の設定も削除する．

HPC クラスタの管理者は，通常のノードで HPC クラスタを構築し，Linpack の実行まで確認する場合は 010→020→031→051→121，GPU ノードで HPC クラスタを構築し，NGC カタログのコンテナ実行を試す場合は 010→020→032→061→051→140→141 または 142 のように Notebook を実行していく．

911 と 912 の Notebook では，VCP のノード停止（*power_off*），再開（*power_on*）機能を用いて，設定済みの HPC クラスタ全体を停止させ，必要なときに再開することができる．クラウドのオンデマンドサービスでは稼働時間に応じて課金されるため，利用していないときは停止しておくことが望ましい．しかしながら，HPC クラスタのように複数台のノードで構成されて設定が複雑なものの場合，環境を壊して作り直すには管理者の負担が大きすぎる．VCP の内部で用いている Terraform ではノードの停止，再開のための共通 API がないため，VCP v20.04 でノード停止，再開機能を独自にサポートし，複雑な構成のシステムをまるごと停止，再開することを可能にした．

現状では，HPC テンプレートは AWS，Microsoft Azure，Oracle Cloud Infrastructure で動作を確認している．また，VCP のノード停止，再開機能は AWS と Azure で利用可能である．VCP に依存しない部分については，他のクラウドやオンプレ環境でも活用できると考えられる．

図 5　HPC テンプレート v2 の構成.

```
!cat {batch_file}
!scp {ssh_opts} {str(batch_file)} {target}:{work_dir}

#!/bin/bash

#SBATCH -J pytorch-mnist         # create a short name for your job
#SBATCH -o pytorch-mnist.%j.out  # Name of stdout output file (%j expands to jobId)
#SBATCH -N 1                     # Total number of nodes requested
#SBATCH -n 1                     # Total number of across all nodes
#SBATCH --gres=gpu:1             # number of gpus per node
#SBATCH -t 00:10:00              # Run time (hh:mm:ss)

cd $HOME/pytorch
singularity exec --nv $HOME/pytorch_23.12-py3.sif python3 mnist_classify.py --epochs=3
pytorch_mnist.job                              100%  527     1.5MB/s   00:00

ジョブを実行します。

!ssh {ssh_opts} {target} bash -l -c \
  "'cd {work_dir} && sbatch pytorch_mnist.job'"

Submitted batch job 7

ジョブの実行状況を確認します。

!ssh {ssh_opts} {target} squeue

             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
                 7    normal pytorch-    user0  R       0:03      1 c1

ノードのGPU利用状況を確認します。

!ssh {ssh_opts} {target} sinfo --Node --Format=NodeHost,Gres,GresUsed

HOSTNAMES           GRES                GRES_USED
c1                  gpu:1               gpu:1
```

```
!ssh {ssh_opts} {target} bash -l -c \
  "'cd {work_dir} && tail pytorch-mnist*.out'"

==> pytorch-mnist.6.out <==
  File "/opt/conda/lib/python3.8/urllib/request.py", line 640, in http_response
    response = self.parent.error(
  File "/opt/conda/lib/python3.8/urllib/request.py", line 569, in error
    return self._call_chain(*args)
  File "/opt/conda/lib/python3.8/urllib/request.py", line 502, in _call_chain
    result = func(*args)
  File "/opt/conda/lib/python3.8/urllib/request.py", line 649, in http_error_default
    raise HTTPError(req.full_url, code, msg, hdrs, fp)
urllib.error.HTTPError: HTTP Error 403: Forbidden
0it [00:00, ?it/s]
==> pytorch-mnist.7.out <==
Train Epoch: 3 [55680/60000 (93%)]      Loss: 0.063726
Train Epoch: 3 [56320/60000 (94%)]      Loss: 0.111317
Train Epoch: 3 [56960/60000 (95%)]      Loss: 0.005355
Train Epoch: 3 [57600/60000 (96%)]      Loss: 0.004527
Train Epoch: 3 [58240/60000 (97%)]      Loss: 0.040675
Train Epoch: 3 [58880/60000 (98%)]      Loss: 0.003392
Train Epoch: 3 [59520/60000 (99%)]      Loss: 0.002788

Test set: Average loss: 0.0321, Accuracy: 9904/10000 (99%)
```

図 6　NGC カタログの PyTorch コンテナを Singularity で実行した結果のスナップショット．上図は実行時のスナップショット，下図は出力結果のスナップショットを表す．

3.3　HPC テンプレート v2 の実行確認

図 6 は，NGC カタログの PyTorch コンテナを Singularity で実行した際のスナップショットを示す．図 6 上図では，バッチスクリプトを用いた Slurm への Singularity ジョブの投入を行った後，GPU ノードが利用されていることが確認できる．図 6 下図は，ジョブ処理後の出力結果を確認し，MNIST の実行が正常に行われたことが分かる．よって，本テンプレートにより HPC クラスタ環境の構築から Singularity および GPU ノードを用いたジョブ実行までが正常に行われていることが確認できる．また，HPC クラスタの構築経験のないエンジニアがほぼ 1 日の作業でクラウドでの HPC クラスタ構築および GPU ノードを用いた動作確認まで行えることが確認できた．

4.　アプリケーションテンプレートの開発自動化の取り組み

アプリケーションテンプレートの開発効率をあげるため，CI/CD 環境を構築してテンプレートの自動実行の仕組みを構築した．コードの開発プラットフォームとしてオープンソースソフトウェアの GitLab[24] を用いており，自動実

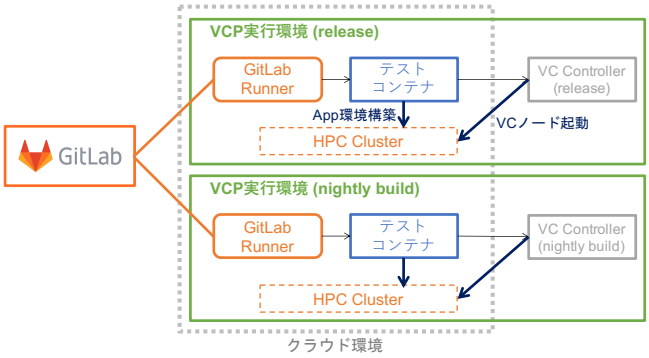

図 7　アプリケーションテンプレートの CI/CD 環境．

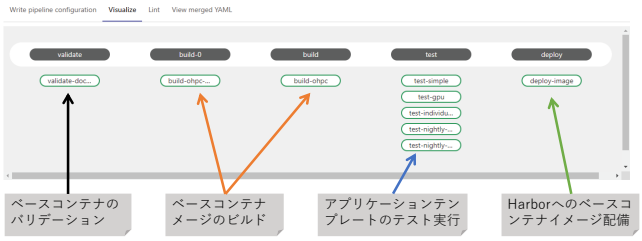

図 8　CI/CD パイプラインの可視化結果．

行では GitLab の CI/CD パイプライン機能でアプリケーションテンプレートの自動実行を行った．

図 7 にアプリケーションテンプレートの CI/CD 環境の概要を示す．GitLab はオンプレ環境，VC コントローラは NII VCP サイトに配備されており，VCP 実行環境はクラウド上に構築した．VCP 実行環境は，リリース版と nightly build 版の VC コントローラを接続した環境を構築した．リリース版はユーザに提供されている安定版であり，nightly build 版はリリース前の版となっている．アプリケーションテンプレートは VCP 本体の開発に追随してできるだけ短いタイムラグで公開する必要があるため，nightly build 版での動作検証も並行して行っている．

各 VCP 実行環境は，予め GitLab Runner とテストコンテナを配備しておき，GitLab へのコードの push などを契機に CI/CD パイプラインのジョブが GitLab Runner により開始される．CI/CD パイプラインは，YAML 形式の設定ファイル `.gitlab-ci.yml` 内に実行するジョブを記述することができる．基本的なパイプライン処理の流れは以下のようになる．

(1) `.gitlab-ci.yml` に記述されている Docker コンテナ（テストコンテナ）を GitLab Runner が VCP 実行環境内に起動する．
(2) テストコンテナ内に GitLab プロジェクトのソースを展開 (git clone) する．
(3) テストコンテナ内で.gitlab-ci.yml に記述されているジョブを順次実行する．

図 8 は，GitLab で CI/CD パイプラインを可視化した結果をキャプチャしたものである．1 つめのジョブでベー

表 2　予備実験環境.

計算機	AWS g4dn.xlarge（4 スレッド，15GB メモリ）
CPU	Intel Xeon Platinum 8259CL CPU @ 2.50GHz
GPU	NVIDIA Tesla T4, v. 455.45.01 driver
カーネル	Linux 4.18.0-240.15.1.el8_3.x86_64
Singularity	v. 3.4.2-5.1.ohpc.2.0
Docker	v. 20.10.5
コンテナ OS	CentOS 8
ホスト OS	Ubuntu 16.04.4

表 3　実験の組み合わせ.

		VCP 利用	Singularity 利用
D	Docker	-	-
DinD	Docker-in-Docker	有	-
S	Singularity	-	有
SinD	Singularity-in-Docker	有	有

スコンテナイメージのバリデーション，2 つめと 3 つめのジョブでイメージのビルドを行い，4 つめのジョブでアプリケーションテンプレートをテスト実行して，最後に NII が管理する Docker イメージリポジトリである Harbor[25] にベースコンテナイメージを配備する．各ジョブは，テストシナリオを表す Feature ファイルと，Feature ファイルで記述した「振る舞い（Behavior）」の定義を実装した Step ファイルで表される．Feature ファイルはテスト記述言語の Gherkin で記述し，Step ファイルは Python で実装して，pytest-bdd[26] ツールを用いてテストシナリオを実行していく．アプリケーションテンプレートの Notebook の実行では，Papermill[27] を用いた．Papermill では，Notebook で入力するパラメータ設定を実行時に与えることで，Notebook を自動的に実行してその出力結果を含む Notebook を得ることができる．出力された Notebook は一定期間保存されるため，ジョブの実行に失敗した場合も Notebook の出力結果を適宜確認することができるようになっている．

アプリケーションテンプレートの CI/CD による自動実行は，現状では HPC テンプレート，LMS テンプレート，講義演習環境テンプレートに対して AWS と Azure 環境で実施している．

5. 予備評価

予備評価として，VCP のベースコンテナ利用の有無と，アプリケーションコンテナとして Docker または Singularity を利用した際の MNIST の学習時間を比較する．MNIST は，28x28 画素の手書き数字からなるデータセットであり，0 から 9 までの数字の分類を学習する．実験では，HPC テンプレート v2 を用いて HPC クラスタを構築し，Slurm で NGC カタログの PyTorch と TensorFlow のコンテナを用いて MNIST を 1 台の計算ノード上で実行する．表 2 に，使用した計算ノードの実験環境を示す．実験は AWS で行い，計

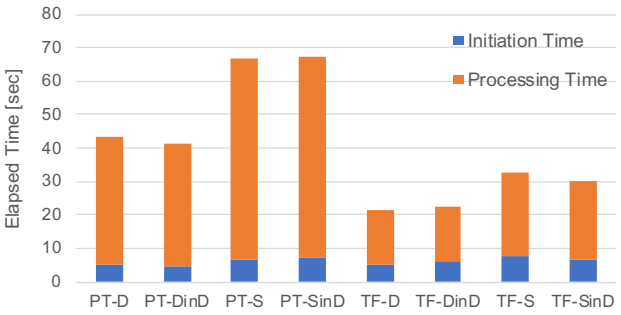

図 9　VCP 利用の有無, D-in-D または Singurality 利用時の MNIST 実行時間の比較.

算ノードには g4dn.xlarge インスタンスを用い，MNIST の実行では NVIDIA Tesla T4 GPU を利用する．また，実験の組み合わせを表 3 に示す．D は Docker，S は Singurality，DinD は Docker-in-Docker，SinD は Singurality-in-Docker を表す．VCP 利用時にはベースコンテナの上でアプリケーションコンテナを実行するため，入れ子型仮想化となっている．PyTorch では PyTorch Examples GitHub サイト [28] の `mnist/main.py` を用いて 3 エポック分の学習時間を，TensorFlow では「Quickstart for beginners」[29] で公開されているものと同等の MNIST スクリプトを用いて 5 エポック分の学習時間を測定した．本研究では，仮想化のオーバヘッドの比較が目的であり，PyTorch と TensorFlow の実行時間を比較するものではない．

図 9 に MNIST の実行時間の比較結果を示す．縦軸は総実行時間を [sec] で，横軸は実験の組み合わせを示しており，先頭の PT と TF はそれぞれ PyTorch または TensorFlow を用いた結果を表している．グラフ中のオレンジ色は MNIST の処理時間であり，青色はコンテナ起動などの起動時間を表す．ただし，起動時間には Slurm のオーバヘッドは含まれない．PyTorch と TensorFlow いずれの場合においても，VCP のベースコンテナの利用の有無を問わず Singularity を用いた方が総実行時間が長くなることがわかる．実行時間の内訳では，起動時間は Singularity を用いるほうがわずかに長くなっていたが，処理時間の差の影響が大きかった．一方で，VCP の利用の有無で比較すると，D と DinD，S と SinD いずれの場合もその性能差は僅かであり，入れ子型仮想化のオーバヘッドは無視できることがわかる．よって，VCP によるオーバヘッドは僅かであり，現時点では性能を重視する場合にはアプリケーションコンテナは Docker で起動する方がよく，ユーザが root 権限の利用が許されないケースでは Singularity を利用する方が望ましいことが示唆された．

6. 関連研究

クラウドでの HPC クラスタ構築商用サービスおよびシ

ステムは複数ある．StarHPC[30], [31] は，並列プログラミング教育を目的としてローカル環境の VMWare や Virtual Box，および AWS クラウド環境に HPC クラスタ環境を構築するものであり，クラウドでの HPC 環境構築ツールの先駆けである．AWS での HPC 用クラスタ構築を支援するため，OpenMPI，Hadoop，Open Grid Scheduler，NFS 等のライブラリが予めインストールされた VM イメージ（AMI）を公開している．Azure Cycle Cloud[32] および AWS ParallelCluster[33] は，簡単な操作で Azure または AWS のクラウド環境で HPC クラスタ環境の構築・管理をサポートするサービスである．これらはいずれも利用可能なクラウドプロバイダに制限があるが，HPC テンプレートでは AWS，Azure，Oracle Cloud Infrastructure での動作を確認している．Frexera Cloud Management Platform（旧 RightScale）[34] では複数のクラウド環境に対応しており，同様に簡易な操作でアプリケーション環境を構築可能にするサービスを提供している．

いずれも HPC クラスタ構築を支援する点で類似性があるが，コンテナベースで Jupyter Notebook を用いたテンプレートを提供する点で異なる．NII では，属人化しがちな計算機システムの構築・運用の手順の再現性や情報共有を目的として，Literate Computing for Reproducible Infrastructure（LC4RI）という方法論に基づいて Jupyter Notebook を所内クラウド基盤の構築，運用に活用している [35], [36], [37]．OCS のアプリケーションテンプレートでもノウハウの共有を目的として Jupyter Notebook を採用しており，利用者による環境設定のカスタマイズやエラー発生時の原因究明がしやすいなどの利点がある．また，コンテナベースであることにより，VM イメージベースで作るよりも可搬性，拡張性が高いと考えられる．

Chameleon Cloud は米国のアカデミッククラウドの 1 つであり，OCS 同様に Jupyter Notebook で計算環境の構築手順を共有している [38]．Chameleon Cloud での利用を前提としている点で異なる．BinderHub[39] は Jupyter Notebook ベースの再現性のある計算環境を K8s 上で提供するものである．必要なライブラリがインストールされた Jupyter Notebook 環境を提供できるが，HPC クラスタのような複雑な構成のアプリケーション環境の構築はサポートしていない．

Kubernetes（K8s）[40] は Google により設計・開発され，現在オープンソースシステムとして開発が進められているコンテナオーケストレーションツールである．大規模環境での大量マイクロサービスの実行・管理を目的として設計されており，複数クラウド環境で K8s ベースのコンテナ実行環境が整備されている．K8s では，YAML 形式で記述されるマニフェストファイルや Helm と呼ばれるパッケージ管理の仕組みを利用することでアプリケーション環境の構築・管理が可能である．K8s はコンテナオーケストレーションツールのデファクトスタンダードと考えられているが，OCS で対象とするような個々の VPN（Virtual Private Network）で隔離された比較的少数の計算機環境の管理にはソフトウェアスタックがやや大きいと考えられる．VCP は現在ポータブル版も開発されており，ポータブル版コンテナを起動すればすぐに利用できるようになっている．ただし，本研究で開発した HPC クラスタのコンテナイメージ自体は K8s 環境でも流用することができる．

Jupyter Notebook の運用手順書の CI を GitLab のパイプラインと Papermill で実現する取り組み [41] もある．本研究では，コンテナイメージの検証から複数 Notebook で構成されるテンプレートの実行，リポジトリへのコンテナイメージの配備までを行っている．

7. まとめと今後の課題

本研究では，インタークラウド資源管理のための基盤ソフトウェア VCP を用いて，クラウドでの HPC クラスタの構築，管理を容易にする HPC テンプレートの開発を行った．仮想化のオーバヘッドを削減するために OpenHPC ライブラリがインストールされた Docker イメージを作成するとともに，HPC クラスタの構築，管理，サンプルプログラムの実行の手順を Jupyter Notebook で記述し，HPC テンプレートとして開発した．HPC テンプレートでは，Singularity コンテナや GPU ノードの利用もサポートしており，GPU ノード上で Singularity を用いて NGC カタログの機械学習用コンテナの利用ができることを確認した．また，アプリケーションテンプレートの開発効率化のための CI/CD の仕組みも開発した．予備評価から，VCP で配備した環境のオーバヘッドは僅かであること，Singularity 自体のオーバヘッドは大きく，D-in-D も選択的に利用できる方が望ましいことが示唆された．HPC テンプレート v2 の公開版では，D-in-D の手順も HPC テンプレートに含める予定である．

今後は，aarch64 アーキテクチャ対応やスケールアウトシナリオの実装など，HPC テンプレートの拡張を行う．また，VCP および各種テンプレートの mdx[42] への適用を検討している．mdx はデータ活用型社会創成プラットフォームであり，SINET5 に接続されたクラウド型の高性能データ活用環境を提供する．開発中の VCP ポータブル版と既存のアプリケーションテンプレートの活用により，mdx でのデータ活用支援にも貢献できると考えている．

謝辞　本研究にご協力いただいた数理技研の小泉敦延様に深く感謝いたします．

本研究の一部は，JST CREST「ビッグデータ統合利活用のための次世代基盤技術の創出・体系化」JPMJCR1501 の助成を受けたものである．

参考文献

[1] Younge, A. J., Pedretti, K., Grant, R. E. and Brightwell, R.: A Tale of Two Systems: Using Containers to Deploy HPC Applications on Supercomputers and Clouds, *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 74–81 (online), DOI: 10.1109/CloudCom.2017.40 (2017).
[2] R. S. Canon and D. Jacobsen: Shifter: Containers for HPC, *Proc. Cray Users Group Conference (CUG'16)*, pp. 1–6 (2016).
[3] Kurtzer, G. M., Sochat, V. and Bauer, M. W.: Singularity: Scientific containers for mobility of compute, (online), DOI: 10.1371/journal.pone.0177459 (2017).
[4] Priedhorsky, R. and Randles, T.: Charliecloud: Unprivileged Containers for User-Defined Software Stacks in HPC, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, New York, NY, USA, Association for Computing Machinery, (online), DOI: 10.1145/3126908.3126925 (2017).
[5] S. Yokoyama and Y. Masatani and T. Ohta and O. Ogasawara and N. Yoshioka and K. Liu and K. Aida: Reproducible Scientific Computing Environment with Overlay Cloud Architecture, *Proc. 9th IEEE Cloud*, pp. 774–781 (2016).
[6] Takefusa, A., Yokoyama, S., Masatani, Y., Tanjo, T., Saga, K., Nagaku, M. and Aida, K.: Virtual Cloud Service System for Building Effective Inter-Cloud Applications, *Proc. IEEE CloudCom2017*, pp. 296–303 (2017).
[7] 学認クラウドオンデマンド構築サービス, https://cloud.gakunin.jp/ocs/.
[8] 学術情報ネットワーク SINET5, https://www.sinet.ad.jp/.
[9] S. Urushidani and S. Abe and K. Yamanaka and K. Aida and S. Yokoyama and H. Yamada and M. Nakamura and K. Fukuda and M. Koibuchi and S. Yamada : New Directions for a Japanese Academic Backbone Network, *IEICE Transactions on Information and Systems, E98.D (3)*, pp. 546–556 (2015).
[10] Kurimoto, T., Urushidani, S., Yamada, H., Yamanaka, K., Nakamura, M., Abe, S., Fukuda, K., Koibuchi, M., Ji, Y., Takakura, H., and Yamada, S.: A fully meshed backbone network for data-intensive sciences and SDN services, *Proc. ICUFN2016*, pp. 909–911 (2016).
[11] SINET クラウド接続サービス, https://www.sinet.ad.jp/connect_service/service/cloud_connection.
[12] Jupyter Notebook, http://jupyter.org/.
[13] 学認クラウドオンデマンド構築サービスアプリケーションテンプレート, https://github.com/nii-gakunin-cloud/ocs-templates.
[14] OpenHPC, https://openhpc.community/.
[15] NVIDIA NGC カタログ, https://www.nvidia.com/ja-jp/gpu-cloud/containers/.
[16] 竹房あつ子, 丹生智也, 佐賀一繁, 横山重俊, 合田憲人：インタークラウド環境構築システムによるアプリケーション環境構築支援のための機能拡張, 情報処理学会研究報告 Vol.2018-OS-144(10), pp. 1–10 (2018).
[17] Terraform, https://www.terraform.io/.
[18] Moodle, https://moodle.org/.
[19] BlueGreenDeployment, https://martinfowler.com/bliki/BlueGreenDeployment.html.
[20] 浜元信州, 横山重俊, 竹房あつ子, 合田憲人, 桑田義隆, 石坂徹：Moodle 運用における Jupyter Notebook の活用, *MoodleMoot Japan 2018* (2018).
[21] 長久 勝, 政谷好伸, 合田憲人：Notebook による講義・演習環境の開発, 情報処理学会研究報告 2019-CLE-27(20), pp. 1–4 (2019).
[22] OpenHPC Community building blocks for HPC systems (2.x), https://github.com/openhpc/ohpc/wiki/2.X.
[23] HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers, http://www.netlib.org/benchmark/hpl/.
[24] GitLab, https://about.gitlab.com/.
[25] Harbor, https://goharbor.io/.
[26] pytest-bdd, https://pypi.org/project/pytest-bdd/.
[27] papermill, https://papermill.readthedocs.io/.
[28] PyTorch Examples, https://github.com/pytorch/examples/.
[29] TensorFlow, Quickstart for beginners, https://www.tensorflow.org/tutorials/quickstart/beginner.
[30] Ivica, C., Riley, J. T. and Shubert, C.: StarHPC — Teaching Parallel Programming within Elastic Compute Cloud, *Proceedings of the ITI 2009 31st International Conference on Information Technology Interfaces*, pp. 353–356（オンライン）, DOI: 10.1109/ITI.2009.5196108 (2009).
[31] Software Tools for Academics and Researchers, http://star.mit.edu/.
[32] Azure CycleCloud, https://azure.microsoft.com/features/azure-cyclecloud/.
[33] AWS ParallelCluster, https://docs.aws.amazon.com/parallelcluster/latest/ug/what-is-aws-parallelcluster.html.
[34] Frexera Cloud Management Platform（CMP）, https://www.flexera.com/products/cloud-management-platform.html.
[35] Masatani, Y.: Collaboration and automated operation as literate computing for reproducible infrastructure, https://conferences.oreilly.com/jupyter/jup-ny/public/schedule/detail/59995 (2017).
[36] 政谷好伸, 谷沢智史, 横山重俊, 吉岡信和, 合田憲人：インフラ・コード化の実践における IPython notebook の適用, 信学技報, SC2015-6, pp. 27–32 (2015).
[37] NII Cloud Operation Team, https://github.com/NII-cloud-operation/.
[38] Anderson, J. and Keahey, K.: A Case for Integrating Experimental Containers with Notebooks, *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 151–158 (online), DOI: 10.1109/CloudCom.2019.00032 (2019).
[39] BinderHub, https://binderhub.readthedocs.io/.
[40] Kubernetes, https://kubernetes.io/.
[41] 阿部秀一, 井浦陽一郎, 宇野耕平, 原野昌幸, 山崎航史：MicroOpsCI with LC4RI（ソフトウェア開発実践演習ポスター_30P1.pdf）, https://www.topse.jp/images/.
[42] 合田, 遠藤, 小野, 工藤, 姜, 小林, 下川辺, 菅沼, 杉木, 関谷, 田浦, 竹房, 田中, 谷村, 出口, 中島, 中村覚, 中村宏, 中村遼, 南里, 塙, 深沢, 松島, 水木, 宮嵜, 森, Lee：mdx: データ活用社会創成プラットフォーム, 大学 ICT 推進協議会 2020 年度年次大会 (AXIES2020), pp. 1–4 (2020).