演習環境の自動構成システムの実現と運用評価

広川 優也[1,a)]　萩原 威志[1,b)]

概要：運用管理者を複雑な管理業務から解放するために情報システムを仮想化する取り組みは様々な組織で行われてきた．しかし，仮想化基盤の導入は管理対象を物理レイヤから仮想レイヤに置き換えるものに過ぎず，時として運用をより複雑にする．新潟大学工学部で独自に運用管理するコンピュータルームでは 2016 年から演習環境をコンテナ化して授業ごとに構成管理する取り組みを行ってきたが，教員による演習環境の構成のサポートや Docker Swarm のトラブル対応など，運用管理者にとっては以前よりも面倒なシステムになっていた．本稿では演習環境の構成を自動化するためのアーキテクチャを OpenShift 上で実装し，2019 年からの運用状況を報告する．また，新たな仮想化基盤の導入が運用管理者の負担の要因にならないようなシステムを実現したことを示す．

キーワード：デスクトップ仮想化，コンテナ，仮想化基盤，オンプレミス

Realization and Operational evaluation of Automatic configration system for Practice environment

Yuya Hirokawa[1,a)]　Takeshi Hagiwara[1,b)]

Abstract: Various organizations have made efforts to virtualize information systems in order to free operation managers from complicated management tasks. However, the introduction of the virtualization infrastructure is merely a replacement of the management target from the physical layer to the virtual layer, and sometimes makes the operation more complicated. In the computer room that is independently operated and managed by the Faculty of Engineering at Niigata University, we have been making efforts to containerize the practice environment and manage the configuration for each class since 2016, it was a more troublesome system for operational managers than before. In this paper, an architecture for automating the configuration of the practice environment is implemented on OpenShift, and the operational status from 2019 is reported. We also show that a system that does not cause the burden on the operation manager is realized by introducing a new virtualization infrastructure.

Keywords: Desktop virtualization, Container, Virtualization infrastructure, On-premises

1. はじめに

情報通信技術の普及に伴って技術者の育成やプログラミング教育の重要性は高まっている．実践的な技術を習得するには実際に手を動かしながら学習する過程が欠かせないため，多くの教育機関では専用のコンピュータルームを活用した実践形式の授業を行っている．このような情報システムを運用する場合，日々の管理業務のために学内の人員を割かなければならない．近年ではモバイル端末の普及もあって，PC 必携制度を導入して管理コスト削減を実現する大学もある [1]．しかし，初学者に対してより手軽な学習の切っ掛けを与えることのできるコンピュータルームは未だに必要とされている．

新潟大学工学部では情報系学生を対象としたプログラミングなどの演習授業を実施しており，目まぐるしく変化する情報技術に対応した柔軟な演習を行うために独自のコンピュータルームを運用管理している．これまでに演習環

[1] 新潟大学
Niigata Uniersity
[a)] y-hirokawa@cs.ie.niigata-u.ac.jp
[b)] hagiwara@ie.niigata-u.ac.jp

境のコンテナ化によって，担当教員が個別の演習環境を構成するための取り組みを行ってきた [2]．しかし，ソフトウェア固有のコンテナ構成手順を各教員に習得してもらうことは難しく，積極的な導入の妨げになっており，実際には殆どの授業を一つの演習環境で賄う状況が続いていた．また，以前は Docker Swarm[3] の管理業務を学内の運用管理者に一任していたが，ドキュメントは Docker に慣れた技術者向けで体系的に運用技術を習得することは難しく，些細なトラブルが発生する度にシステム構築担当者に対応を依頼する状況であった．そのため，クラスタ管理の外部委託を前提としたシステムの構築によって改善する必要があった．本稿では，演習環境の構成にかかる手順を自動化するためのアーキテクチャをコンテナ基盤の OpenShift[4] 上で実装し，2019 年 4 月から運用開始しているシステムの利用状況を紹介する．その際に，新たなコンテナ基盤の導入によって学内の運用管理者の負担が増大することなく実現できることを示す．

本稿の構成は以下の通りである．2 章では，コンピュータルームのシステム構成の概要を説明する．3 章では，演習環境の構成を自動化するシステムについて説明する．4 章では，コンテナ基盤の導入による運用の変化について説明する．5 章では，本稿のシステムの運用によって新たに得た知見を紹介する．6 章では，本稿のシステムを利用者と運用管理者の双方の観点から評価する．7 章で本稿の結論を述べる．

2. コンピュータルームのシステム構成

本章では 2019 年 4 月から運用開始した時点でのコンピュータルームのシステム構成を説明する．

2.1 概要

本システムは情報系学生を対象にしたプログラミングなどの授業を行うための演習環境を提供するものである．演習環境には Linux デスクトップ環境を採用し，XDMCP や VNC，SSH のいずれかの方法で接続する．コンピュータルームには 100 台程度の X 端末が設置され，演習環境を利用することができる．また，室外からであっても学内ネットワークであれば利用が可能であるほか，学外からの VPN を用いた利用も想定する．授業時間中は受講者の利用が優先されるが，授業に支障のない範囲で自主学習のための利用も認める．

演習環境はコンテナ基盤上に構築され，Linux サーバに直接ログインする代わりに学生ごとにコンテナを割り当てる．以前はコンテナ基盤として Docker Swarm を採用していた [2] が，本システムでは新たに OpenShift を採用する．

2.2 構成

本システムの初期構成を**図 1** に示す．前提として，全ての

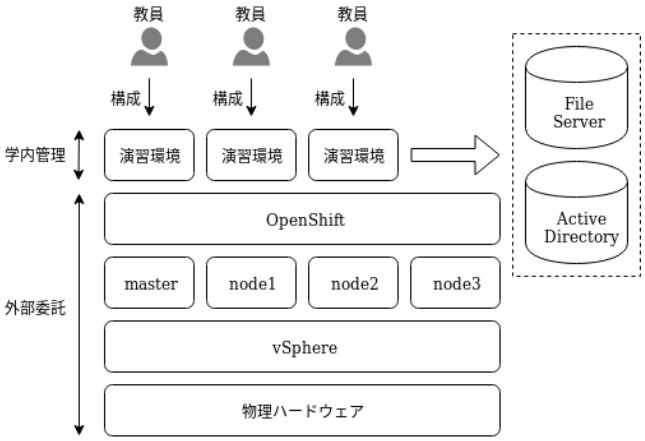

図 1 システム構成
Fig. 1 System architecture

表 1 VM ホストの性能
Table 1 Specification of VM host

OS	VMware ESXi 6.7
CPU（合計）	Intel(R) Xeon(R) Gold 6154 (x72)
メモリ（合計）	384GB

表 2 クラスタの性能
Table 2 Specification of cluster

	master	node (x3)
OS	RHEL 7.6	RHEL 7.6
vCPU	3.00GHz (x4)	3.00GHz (x24)
メモリ	16GB	32GB
システム領域	100GB	40GB
データ領域	30GB	20GB

システムは仮想化基盤の vSphere 上に構築する．vSphere のホストマシンの性能を**表 1** に示す．仮想マシン上にコンテナ基盤の OpenShift を 1 マスター 3 ノードの最小構成で導入する．ディスクはシステム領域とデータ領域に分けられ，それぞれファイルサーバの NFS 上に保存される．OpenShift のクラスタに割り当てた仮想マシンの性能を**表 2** に示す．OpenShift の導入までを外部委託して，学内の運用管理者による管理対象外とする．一方，演習環境の構築は OpenShift 上のコンテナアプリケーションとして行い，学内の運用管理者による管理対象とする．演習環境を構成・変更は，授業の方針に臨機応変に対応可能にするため，管理運用者に依頼せず各授業担当の教員に委任する方針で設計する．演習環境は揮発性のコンテナであるため，ホームディレクトリやユーザ管理は別途サーバを用意して，各コンテナから参照する．

3. 演習環境の自動構成システム

授業担当の教員が自ら演習環境の構成を行うために，運用管理者への依頼を不要にするシステムが求められる．本章では OpenShift 上に実装した演習環境の構成を自動化す

```
model: Image
name: example-image
owner: ce-admin
email: ce-admin@example.com
uuid: e7a1f44d-616f-41a1-a467-dc750444f4f4
description: 演習環境の定義例
packages:
- sbcl
- slime
overrides:
- path: /etc/emacs/site-start.d/slime.el
  mode: 644
  data: —
    (require 'slime)
    (setq inferior-lisp-program "sbcl")
    (slime-setup '(slime-repl slime-fancy slime-banner))
```

図 2 演習環境を定義するマニフェスト
Fig. 2 Manifest for Image configuration

```
model: Subject
name: example-subject
owner: ce-admin
email: ce-admin@example.com
uuid: 404750b9-0b48-4452-868a-2cf70f368459
description: 授業の定義例
image: example-image
students:
- test101
- test102
- test103
schedules:
- term: term-1
  day: Mon
  period: 1
- term: term-1
  day: Mon
  period: 2
```

図 3 演習環境を使用する授業を定義するマニフェスト
Fig. 3 Manifest for Subject configuration

るシステムの設計を説明する．

3.1 構成手順の自動化

コンテナ型仮想化基盤として知られる Docker は，イメージの構築を Dockerfile と呼ばれるソースファイルで構成するため再利用やカスタマイズが可能である．Dockerfile の各行はレイヤと呼ばれ，イメージ間で共通のレイヤを使用することでディスク消費量やイメージ再構築の時間を節約できる．演習環境を Docker によってコンテナ化する場合，演習環境の構成は Dockerfile の記述によって行われるが，予め最低限必要な設定を施したイメージを作成しベースイメージとして共有することで教員が記述すべき Dockerfile の内容を簡単にすることができる．しかし，上手く Dockerfile を記述できたとしても Docker Registry にログインして構築したイメージを送信する必要があるなど，ソフトウェア固有の操作は多い．実際のところ教員の多くはこのようなツールの使用に不慣れであり，運用管理者に作業を代行してもらう場合が多く，面倒を嫌った運用管理者は一般的なソフトウェアを追加した巨大な演習環境を作成するという結果になっていた．

これらの手順を自動化するために，継続的インテグレーションツールの Jenkins を導入する．あらかじめ構築手順を Jenkinsfile として記述して Git リポジトリに配置し，アップロードすれば自動的に構築が実行される．Dockerfile や Jenkinsfile は演習環境ごとに共通部分が多いため，Git リポジトリのブランチを作成することで差分管理する．この方法は教員による作業を簡単にするが，代わりに運用管理者に Jenkins の管理をしてもらう必要がある．幸い OpenShift には Jenkins と Docker Registry が内包されており，本システムでは特に負担にならない．

本稿では，演習環境を構築するために必要な Dockerfile と Jenkinsfile を雛型から生成するための独自のマニフェストを定義した．**図 2** にこのマニフェストの概要を示す．

3.2 演習環境の一覧

授業ごとに演習環境の構成を行うと，学生は授業の際にどの演習環境を利用するか選択する必要が出てくる．存在する全ての演習環境が表示されるとすると，数が増えるほど正しいものを選択することが難しくなる．実際に，選択する演習環境を間違えたために授業に支障の出る学生もいた．そこで，すべての演習環境のうち学生が必要とするものだけを抽出することでトラブルを回避するために，授業に使用する演習環境と受講者リスト，開講時間を記述する独自のマニフェストを定義した．**図 3** にこのマニフェストの概要を示す．この場合，test101, test102，test103 という学生だけがこの演習環境を使用可能となる．

本来，該当する授業期間にだけ演習環境を表示することが最も選択ミスの発生しない方法であるが，期間外に自習などで使用したいという要請もある．そのため，その日授業に使用される演習環境が上位にソートされるような一覧を生成することで対処する．

3.3 WebUI の提供

本稿で定義する独自形式のマニフェストを記述しやすくするための WebUI を提供する．ここでは，Debian が提供するパッケージを検索・追加したり，テキスト形式の設定ファイルを追加したりすることができる．**図 4** に示すように WebUI に入力することで，図 2 や図 3 のようなマニフェストが記述される．その内容に応じて Dockerfile や Jenkinsfile を雛形から生成し，Git リポジトリを更新するまでを行う．現時点では，WebUI を用いて作成される演習環境は一つの授業だけで使用される．これは，授業ごとに個別の演習環境を作成するという本稿の目的から言えば当然の制約である．

図 4　WebUI の入力画面の一部
Fig. 4　Part of editing on WebUI

3.4　システムの設計

教員が演習環境を構成し，学生が演習環境を利用する流れを図 5 に示し，概要を以下で説明する．

教員による演習環境の構成

教員は WebUI を利用して演習環境の構成を行う．入力内容に基づくマニフェストに従って Dockerfile や Jenkinsfile を生成し，Git リポジトリに追加されることを契機に演習環境が自動的に構築される．マニフェストファイルも別途 Git リポジトリに追加され，演習環境の一覧を表示する際に使用される．構築された演習環境は Docker Registry に保存される．

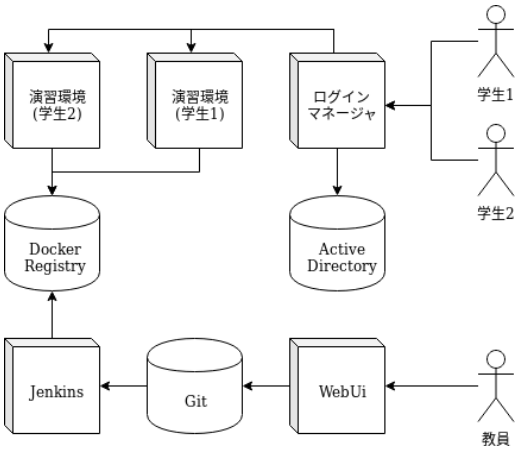

図 5　利用の流れ
Fig. 5　Usege flow

学生による演習環境の利用

学生が XDMCP や VNC，SSH のいずれかの方法でログインすると，Git リポジトリに追加されたマニフェストファイル群から必要なだけの演習環境の一覧が表示される．学生が演習環境を選択すると，選択された演習環境が Docker Registry から配信され，デプロイされる．演習環境はコンテナとして学生ごとに作成され，ログアウトと同時に削除される．

4.　コンテナ基盤による運用の変化

本章では 2016 年から 2018 年の Docker Swarm による運用 [2] で明らかになった以下の課題について説明し，それらが OpenShift に移行した本システムでは問題にならないことを示す．

4.1　クラスタの障害対応

Docker Swarm は学内で構築から運用まで行っていたため，クラスタ障害時の対応は学内の運用管理者だけで行う必要があった．Docker Swarm にはクラスタの自動復旧機能などは提供されておらず，障害が発生した際の原因究明や復旧作業は手作業であった．実際に，Docker Swarm の運用中は障害が発生したことが明らかな状況であっても，運用管理者だけで対応できない状態が続いていた．仮想化基盤ごとに異なる障害対応の手順を習得することは運用管理者にとって負担であり，今後のシステム見直しの度に同様の問題が起きる可能性がある．

そこで，コンテナ基盤のクラスタ管理までを外部委託して，その上で構築される演習環境の運用だけを学内で行うことを検討し，OpenShift を導入した．しかし，クラスタ障害のような急を要するトラブル対応は，委託先の技術者に依頼を出しても間に合わず，学内で対応する必要がある．OpenShift には各ノードの健康状態に応じたコンテナの再配置や自動復旧の機能がある．大抵の場合，障害が発生し

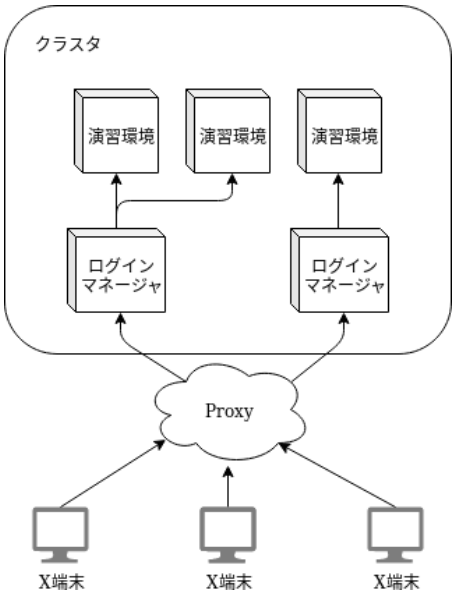

図 6　負荷分散
Fig. 6　Load balancing

たノードは自動で再起動して復旧するため，運用管理者が特別に行うことは何もない．制限として，本システムはシングルマスター構成であるため，マスターに障害が発生すると自動復旧の機能が働かなくなる．この問題はマルチマスター構成にすれば解決される．

4.2　ログイン処理における単一障害点の解消

ログインマネージャは，学生が接続・認証した後，演習環境のデスクトップ画面を X 端末に転送する．ログインマネージャが単一のインスタンスで実行されている場合，計算機演習室を利用中の全ての学生からアクセスが集中することになり，障害が発生した際に全ての学生の利用を中断してしまう問題がある．そのため，ログインマネージャは負荷分散と高可用性のために複数のインスタンスで実行することが望ましい．しかし，Docker Swarm ではインスタンスの増減やそれに伴うネットワーク経路の制御が簡単ではなく，実現していなかった．

ログインマネージャをコンテナ化して OpenShift 上で複数インスタンスで実行し，インスタンス当たりの学生の接続数を削減することは容易である．例えば，インスタンスを 3 つにしたければ，管理ツールから replicas の値を 3 にすれば良い．これにより，万が一障害が発生した場合にも影響範囲を一部の学生に限定することができるようになる．**図 6** にログインマネージャの複製による接続例を示す．この際，負荷分散のために接続先を時間経過で変更する機能によってセッションが切断されるのを防ぐために，sticky session の設定を行う．**図 7** にそのための設定例を示す．Docker Swarm と異なり OpenShift ではネットワーク経路の制御が簡単で，導入の手間はない．

```
apiVersion: v1
kind: Service
spec:
  sessionAffinity: ClientIP
  sessionAffinityConfig:
    clientIP:
      timeoutSeconds: 86400
  ...
```

図 7　sticky session の設定
Fig. 7　Configuration for sticky session

5.　運用開始の経過報告

本章では，2019 年 4 月から運用を開始した本システムで得た知見を紹介する．

5.1　トラブル発生のタイムライン

4 月 9 日 一年生の初回ログインに大幅な遅延が生じる
一年生の最初の授業において 100 人規模の初回ログインが同時に行われ，その際に大幅な遅延が発生した．他学年の授業では再現されなかった．また，vSphere や OpenShift の監視モニタでは異常な負荷などは見られなかった．後日の調査により，ファイルサーバとの通信によるリングバッファの溢れが確認されたため，ファイルサーバの IO 性能不足が原因であると結論した．未だ根本的な対策は取られていないため，今後も再現する可能性がある．

4 月 10 日 適切な演習環境の選択がされない
二年生のプログラミング授業において，どの演習環境を選択すれば良いか分からないという学生の声があった．また，間違った環境にログインしたことに気付かずに授業を受け，コンパイルが正常にできないという形で報告を受けたケースもあった．その場で案内を行いログインを促すことで対応したが，演習環境の一覧が直感的に判断できるように改良する必要があった．対策として，各演習環境に説明文を付与した上で，学生が必要とするものだけを抽出し，授業が近いものにはマークをつけるといった改良を行った．

4 月 16 日 一年生の授業中にノードがダウンする
一年生の二度目の授業において 100 人規模のログインが同時に行われ，その際にノードダウンが発生した．10 分後，ダウンしたノードは自動的に再起動した．一年生の授業は必修かつ出席率が良いため他学年の授業よりも利用者が多かった．収束後，ノードのログを調査した結果データ領域の溢れが確認された．ログイン時のコンテナ作成にかかるデータ領域の消費が想定よりも多かったことが原因であると結論した．データ領域はイメージやコンテナレイヤの保存域として使用されるが，各ノードで 20GB と少なかったため倍の

40GB に拡張する対策を取ったところ，同様の障害はこれ以降発生していない．

4 月 17 日 一部の学生で演習環境の一覧が表示されない

一部の学生で，ログイン時に演習環境の一覧が表示されない不具合が発生した．調査により，前日のノードダウンによる再起動後もコンテナネットワークが復旧していないことが確認された．演習環境の一覧を生成するための HTTP サーバを含むコンテナを再起動することで回復した．また，当該コンテナの liveness probe パラメータにより，一定時間ごとの疎通確認と障害検知時の再起動を設定することで再発は防がれている．

5 月以降 一部の学生で IME 等のプロセスが停止する

一部の学生で，ログイン後に演習環境で動作する IME 等のプロセスが突発的に停止する不具合が発生した．コンテナはホストの計算リソースを共有しており，ノード上のコンテナが増えるとメモリや CPU が不足することが原因として知られていた．対策としてコンテナのリソース上限を設定することができるため，一時的にメモリの上限値を 4GB に設定した．頻度は落ちたものの，依然として同様の不具合が発生している．これに関しては，10 月以降の運用において異なる原因が示唆されている．

9 月 4 日 演習環境の構成に失敗する

ある教員が演習環境の構成を行ったところ，構築に失敗した．演習環境には複数のプログラミング環境と，texlive 環境が含まれていた．調査の結果，Docker の Base Device Size の値が 10GB なのに対して，追加されたパッケージが 10GB を越えたことが原因と結論した．あまり大きなパッケージを追加しないように案内しつつ，OpenShift 全体への影響が無ければ値を引き上げることも検討している．また，この件から巨大なイメージ数が増加することでデータ領域が逼迫する可能性を感じたため，イメージ保存域であるデータ領域を 40GB から 100GB に拡張した．

10 月 1 日 夏季休業明けのログインに大幅な遅延が生じる

夏季休業明け最初に行われた二年生の授業において 100 人規模のログインが行われ，大幅な遅延が発生した．4 月 9 日と同様の IO 性能に起因する事象であるが，遅延が大きくホームディレクトリのマウントに失敗するなど，より深刻であった．夏季休業中には複数のイメージが新たに作成されており，各ノードにキャッシュされていない状態であったことから，サイズの大きなイメージ数が増加したことによる転送時間の増大が事態を悪化させたと考えられる．対策として，イメージの構築に成功した時点で各ノードにイメージを転送しておくことにした．また，コンテナ起動のタイミングをずらし，6 人以上のログイン処理が同時に行われないようにした．授業中のイメージ構築によるディスク負荷の影響を避けるため，授業に使用しないノードで構築を行うようにした．

10 月中旬 IME プロセスの起動に失敗する

一部の学生で，ログイン時に IME プロセスの起動に失敗する不具合が発生した．5 月の状況とは異なり，メモリや CPU の使用率が低い状況であった．同時期に IO 性能の低下によるログイン障害があったため，同様の原因の可能性が考えられた．すなわち，ネットワーク上のホームディレクトリに存在するファイルへのアクセスに失敗し，デーモンが正常に起動しなかったことが原因と推測される．これを根本的に解決するためにはファイルサーバの IO 性能を向上させるしかないが，実現できていない．

5.2 トラブル発生から得た知見

5.2.1 IO 性能

本システムではホームディレクトリや Docker 用のデータ領域に NFS ボリュームを使用している．4 月や 10 月のトラブルから，イメージキャッシュの無い状態で多くのログイン処理を同時に行うと，IO 性能が不足することが分かった．予めイメージを各ノードに転送しておく対策をとった後でも同時ログイン数が多ければ IO 性能が低下し，マウントに失敗したりプロセスの起動に影響を及ぼしたりする．これらのことから，本システムを運用する上では，データ領域をネットワークストレージに配置することを避け，高速な SSD 等を採用するべきであったといえる．

5.2.2 障害時の自動復旧

学生がログインすると演習環境はコンテナとして起動されるが，この際にノードのデータ領域が一時的に消費される．受講者数が多い場合データ領域が溢れてしまい，ノードダウンが発生した．しかし，10 分程度でノードが自動的に再起動され，運用管理者が行った対応は学生へのアナウンスとデータ領域の増強依頼を出すことだけであった．また，ノードダウンによってコンテナのネットワーク経路が遮断され，復旧後も正常な状態に戻らないことがある．OpenShift では，コンテナの死活監視によって障害を検知して自動的に再起動する liveness probe を設定することで手動による再起動は必要なくなった．

5.2.3 演習環境のリソース配置

コンテナはメモリや CPU をホストと共有しており，リソース消費量がノードの上限に達するとコンテナ内のプロセスが停止される．この場合は学生に再ログインを促せば良いが，根本的な解決にはならない．演習環境は異なる複数のデスクトップアプリケーションを含み，授業ごとにどのように使用されるか予測できないため，リソースクォータやノード配置を適切に行うことが難しい．コンテナはクラスタ内の各ノードに分散して配置されるが，学生がログ

表 3　各演習環境の概略
Table 3　Summary of environment images

授業	サイズ	主なパッケージ
(ベース)	2.15GB	xfce4, firefox, emacs
データ工学	2.67GB	gcc, python, ruby, r-base
知能情報システム実験 III,IV	2.81GB	python, scikit-learn
プログラミング基礎 I,II	3.16GB	gcc, imagemagick, libreoffice

表 4　ノード構成の変化
Table 4　Changing node configrations

	2019 年 4 月	2019 年 10 月
ノード数	3 台	5 台
vCPU	16 コア	24 コア
メモリ	32GB	48GB
データ領域	20GB	100GB

インしたあとで負荷率の変動に合わせて動的に配置を変更することはできない．コンテナライブマイグレーションの実現を期待される CRIU を利用する研究 [5] もあるが，未だ実用段階とはいえない．そこで，授業用に予約されたノードを用意して利用者の優先度を区別することで授業中の利用を保証する対策をとった．

5.3　演習環境構成ツールの利用状況

表 3 は実際に作成された演習環境の例を示す．実際には，ベースイメージと各演習環境の差分だけのディスクが消費される．第一ターム (4・5 月) は導入の遅れから全ての授業で運用管理者が用意した演習環境を利用してもらっていたが，第二ターム (6・7 月) には一部の演習環境を担当教員に作成してもらった．第三ターム (10・11 月) 以降は全ての演習環境を各授業の担当教員に作成してもらった．演習環境を作成するためのツールを適切に使用してもらえるかどうかが運用管理者の負担削減に大きく寄与するが，現時点では運用管理者による個別対応をとらずに授業に導入できている．

WebUI によって，教員が授業中に足りないパッケージを追加したり，別のフレームワークを試すなどのことが気軽に行えるようになった．従来のすべての演習に対応した単一の環境を用意する方法では，他の演習にトラブルが起きる可能性を考慮して導入を見送る場合があった．また，パッケージの追加を運用管理者に依頼する方法では対応に時間がかかる上，思うように動かなければ再依頼する必要があるなど積極性を阻害する欠点があった．その他，WebUI があっても従来のような全部入りのイメージが作成される懸念もあったが，教員にとっては自身の演習環境が動作すれば充分なため余分なパッケージの追加は却って面倒であり，今のところ起こっていない．

一方で，本システムには限界がある．例えば「知能情報システム実験 III,IV」では機械学習を題材にしており，Python ライブラリの Chainer を使用する予定であった．しかし，Chainer は Debian の公式パッケージに含まれておらず，本システムの WebUI では導入できない．そのため，Python のパッケージ管理システムを使用して各自導入する形で授業を行っている．本システムは，このようにパッケージ管理だけで済むような場合にのみ使用可能なものとなっている．

5.4　演習環境の個別化

一つの授業ごとに一つの演習環境を作成してもらい，実際に演習を実施した．初期段階では誤った演習環境を選択したことによるトラブルが発生していたため，演習環境の増加に伴い学生が適切な演習環境を選択して利用できるかという不安はあったが，各学生に必要な演習環境だけを表示する対策を講じたところさほど目立ったトラブルは発生しなかった．

演習授業は学生にとって多くの時間と努力を必要とする科目であり，年間に受講する授業数はさほど多くならないため，演習環境の選択が困難になるほど多くの授業が表示されることは考えられない．また，授業を行う教員の人数も多くないため，年間で作成されるイメージ数は多くても 10 個程度である．前述の対策と一年ごとの受講者リストの更新を徹底することで，授業ごとに個別の演習環境を作成していくことは現実的に可能である．

5.5　クラスタの増強

クラスタの構成に関して，OpenShift の知見がなかったため最小構成案（2.2 節参照）を採用したが，運用開始後にいくつかの障害発生を受けて再検討を行った．授業時間の利用のためのリソースを確保するためにノードを追加し，ディスク溢れの対策のためにデータ領域を増強した．また，ログイン時の遅延への対策としてメモリや CPU を増強した．2019 年 10 月時点でのノード構成の変化を**表 4** に示す．

この背景には，OpenShift が要求するリソースが Docker Swarm よりも多いことが挙げられる．本システムのハードウェア構成は Docker Swarm の運用経験から判断して，OpenShift のライセンスにかかる費用の問題から CPU は 96 コアから 72 コアに縮小したが，それ以外はほぼ同じ構成をとっている．しかし，実際には安定稼働のために余裕を持ったリソース設計が必要であり，特にディスク性能に関しては予想外に不足した状態になっている．変更しようにも，パブリッククラウドでは数クリックで完了する作業であるがオンプレミスでは手間が大きく，授業開始までに対応が間に合わないという問題もあった．この点は，学内で OpenShift のようなコンテナ基盤を運用することの難点であるといえる．

6. 評価

コンテナ基盤上でアプリケーションを運用する場合，コンテナ基盤自体の管理を必要としないことが望ましい．このような要求から，パブリッククラウドのコンテナ基盤がよく利用されている．オンプレミスでの運用が前提であっても，OpenShift のような企業向けコンテナ基盤の構築・管理を外部委託すれば近い条件で運用できる (2.2 節参照)．しかし，トラブルの発生に対して迅速な対応を求められる状況では，組織内の運用管理者だけで初期対応を行う必要がある．本章では，コンテナ基盤上のアプリケーションの利用者とコンテナ基盤の運用管理者それぞれの観点から本システムの設計を評価し，その有効性を示す．

6.1 運用管理者の観点からの評価

運用管理者の主な業務の一つはトラブル対応である．特に，授業中に発生したトラブルが長引いて次の授業にまで影響が出ることは避けなければならない．OpenShift では，ノードやコンテナに障害が発生すると一定時間後に対象のノードやコンテナが再起動して自動的に復旧する (5.2.2 節参照)．そのため運用管理者がその場で行うべきことは利用中の学生へのアナウンスのみである．また，演習環境内でプロセスが停止する不具合が生じることが確認されているが (5.2.3 節参照)，その際に運用管理者が行うべきことは学生に再ログインを促して演習環境を再起動させることのみである．このように，大抵のトラブルは再起動すれば解決すると考えると，再起動さえ容易であればトラブル対応は簡単なものになる．本システムはそれを実現しているといえる．

6.2 利用者の観点からの評価

演習環境は授業内容に応じて構成される必要があるが，授業内容は担当教員の裁量で決められる．複数の授業に対応する唯一の演習環境を構成する方針では，授業によって異なるバージョンのソフトウェアが混在するなどの問題がある．これを解決するために，授業ごとに個別の演習環境を構成する方法が考えられるが，教員に仮想マシンのインスタンスを与える単純な方法では手間が大きく積極的に利用されない．本稿では Dockerfile と Jenkinsfile を記述することで演習環境の構成を自動化する方法を採り，各ファイルを生成するための独自のマニフェストを定義した (3.1 節参照)．さらに，WebUI を提供することでマニフェストの記述さえも容易にした (3.3 節参照)．これによって，教員は演習環境へソフトウェアを追加するような変更をいつでも気軽に行うことができ，運用管理者へ連絡する必要はない．新たな授業内容のために実験的な演習環境を構成することが容易になった点は大きな利点といえる．

6.3 構築した技術者の観点からの評価

Docker によるアプリケーション開発は Dockerfile の記述とコンテナの動作テストの繰り返しである．Git によるソースコード管理と Jenkins による継続的インテグレーションを用いると，開発効率は格段に上がる．OpenShift にはこの仕組みが内包されており，演習環境の自動構成 (3.1 節参照) にも利用している．また，OpenShift 独自の S2I 機構により，Dockerfile を書かずにコンテナアプリケーションをデプロイすることもできる．これによって，クラスタ外のサーバで開発していたアプリケーションをコンテナ向けに改造することなく OpenShift 上に移行することができた．あらゆるアプリケーションを OpenShift で一元管理する体制に移行することで他のサーバを必要としなくなったことは，期待以上の利点である．

7. おわりに

本稿では新潟大学工学部のコンピュータルームで 2019 年から運用しているシステムについて述べた．2016 年から 2018 年まで運用した Docker Swarm によるシステムで実現していなかった演習環境の構成の自動化を OpenShift 上で行い，そのために管理運用の負担が増大することを防いだ．2019 年に運用を開始してからいくつかのトラブルを経験したが，基本的には IO 性能やメモリといったリソース不足による問題で，OpenShift のリソース消費に対して余裕を持った設計が必要だった．一方で，トラブル時でもクラスタは自動的に復旧するため運用管理者のするべきことは学生への案内程度であり，対応は容易になった．また，教員による演習環境の構成は既に授業で実際に活用されており，各担当教員に任せることができている．授業期間中であっても運用管理者への問い合わせを必要とせず気軽にソフトウェアの追加が行える点は大きな利点といえる．以上のことから今回のシステム更新は概ね期待通りの成果を挙げたと考えている．

参考文献

[1] 藤村直美，緒方広明：九州大学における学生 PC 必携化（BYOD）の実現と成果について，技術報告 7，九州大学情報統括本部, 九州大学基幹教育院ラーニングアナリティクスセンター (2017).
[2] 佐藤　悠，萩原威志：Docker を用いたコンピュータ演習室向け Linux 端末システムの設計，技術報告 10，新潟大学, 新潟大学 (2017).
[3] Docker: Swarm mode overview, https://docs.docker.com/engine/swarm/. (2019 年 11 月 5 日閲覧).
[4] Red Hat: Red Hat OpenShift, https://www.redhat.com/ja/technologies/cloud-computing/openshift. (2019 年 11 月 5 日閲覧).
[5] 永井陽太，松原克弥：高頻度な再配置を想定したコンテナマイグレーション機構の実現，技術報告 47，公立はこだて未来大学システム情報科学研究科, 公立はこだて未来大学システム情報科学研究科 (2019).