Vol.2020-IOT-51 No.4
Vol.2020-SPT-39 No.4
2020/9/4

情報工学科演習用のコンテナ技術を用いた新規サービスの設計・実装

宮平 賢[2,a)]　河野 真治[2,b)]

概要：情報技術の普及に伴い情報系の学生が課題や演習を行う学習環境が必要である。この学習環境では, 課題や演習によっては並列処理により, CPU より GPU が必要となる場合がある。このような学習環境を複数の学生に提供する方法として, VM や コンテナがある。しかし, 琉球大学工学部で運用している VM 貸出サービスでは, GPU を共有に対応していない。そこで, コンテナを利用することができる Docker と Singularity を用いて, オンプレミス環境でコンテナ貸出サービスを提供する。また, PC 上からコンテナへの操作を可能にするために Kubernetes でのコンテナ作成にも対応する。コンテナ貸出サービスは LDAP で管理された学生のアカウントを使用することで, 適切にコンテナの管理を行う。本稿ではサービスを実装する上で必要な技術概要を延べ, サービスの設計・実装を行う。

1. はじめに

情報通信技術の普及に伴い学生が学ぶ学習環境が必要となる。その学習環境として VM や コンテナにより, 手軽に開発し試せる技術が普及している。だが, 手元の PC 上で VM や コンテナを立ち上げ, 開発を行うことはできるが, VM や コンテナの使用には高性能 PC や 有料のクラウドサービスが必要になる場合がある。この大きな負担を学生に負わせない仕組みが必要である。

琉球大学工学部工学科知能情報コースでは新たに人工知能やシステム開発などの先端技術を身につける講義や実験が設けられた。講義の演習や実験は学生の PC で環境を構築し実行する。本コースでは希望する学生に学科のブレードサーバから仮想環境を貸出すサービスを行なっている。貸出をする VM のデフォルトのスペックは CPU 1 コア, メモリ 1GB, ストレージ 10GB である。デフォルトのスペックでは不足する場合, 要望に応じてスペックの変更を行なっている。だが, 貸出 VM でも課題によってはスペックが足りなく, 処理に時間がかかることがあった。例として, 人工知能の講義において課される課題においては CPU より GPU を用いることで処理時間を早くすることができる。しかし, 現在の VM 貸出サービスでは GPU の共有に対応していない。VM 上で GPU を共有するには PCI パススルーを利用することで可能だが, PCI パススルーでは複数の VM に共有することができない。GPU が搭載されている PC は研究室によっては用意されているが, 研究室に所属していない学生は利用することができない。そのため, 本コースの学生が高性能な環境を利用できる新たな仕組みが必要である。

学科のブレードサーバに搭載される GPU は VM の貸出サービスでは利用することができない。そこでコンテナ技術を利用する。コンテナ管理ソフトウェアである Docker[1] では NVIDIA Container Toolkit である nvidia-docker[2] を利用することで, 複数のコンテナで GPU を共有することができる。Docker は基本的に root 権限で動作する。また一般ユーザが docker コマンドを使用するには docker グループに追加する必要がある。そのため Docker をマルチユーザ環境で使用すると, 他ユーザのコンテナを操作できるなどセキュリティ上の問題がある。

そこで, 本論文では, Docker とマルチユーザ環境で利用しやすいコンテナプラットフォームである Singularity[3] を利用したコンテナ貸出サービスを設計・実装する。このコンテナ貸出サービスでは, Web コンソールからコンテナの管理をすることで他ユーザのコンテナへの操作をさせない。

2. 技術概要

本研究で使用した仮想化技術, コンテナ技術, また本コースで利用しているサービスについての概要を説明する。

[1] 琉球大学大学院理工学研究科情報工学専攻
[2] 琉球大学工学部工学科知能情報コース
[a)] mk@cr.ie.u-ryukyu.ac.jp
[b)] kono@ie.u-ryukyu.ac.jp

2.1 Docker

Docker とは OS レベルの仮想化技術を利用して, ソフトウェアをコンテナと呼ばれるパッケージで提供する。またコンテナの実行だけでなく, コンテナの実行に用いるイメージの作成, イメージを共有する仕組みを持つコンテナ管理ソフトウェアである。コンテナの実行には Docker 社が提供している Docker Hub[4] に登録されているイメージ, Dockerfile を用いて作成したイメージを利用することができる。Dockerfile を用いることで, 必要なソフトウェアや各種設定を含んだイメージを作成できる。

2.2 Kubernetes

Kubernetes[5] とは, アプリケーションのデプロイ, スケーリング, 及び管理を用意にするためのコンテナを動的管理するコンテナオーケストレーションである。Kubernetes ではオブジェクトによりクラスターの状態を表現する。オブジェクトはコンテナだけでなく, ネットワークやストレージ, 接続ポリシーの望ましい状態を記述できる。本研究では以下のオブジェクトを主に利用する。

- Pod
  - Kubernetes で作成, 管理できる最小単位。Pod 内に 1 つ以上のコンテナを起動できる。
- ReplicaSet
  - 安定した Pod の維持を行い, クラスタで必要な Pod 数を管理する。Pod のセルフヒーリングを行う。
- Deployment
  - ReplicaSet のロールアウトを図るなど, 管理を行う。
- Service
  - Pod にアクセスするための IP アドレスやポートを割り振る。
- Ingress
  - 外部からのアクセスを管理する。負荷分散, SSL 終端, 名前ベースの仮想ホスティングの機能を提供する。
- Namespace
  - 仮想クラスタとしてグループ化して取り扱える。
- Role
  - Kubernetes API の利用権限 Namespace ごとに定義する。
- RoleBinding
  - ユーザやグループに Role を関連付ける。

2.3 Singularity

Singularity とは, HCP クラスタ上で複雑なアプリケーションを実行するために開発されたコンテナプラットフォームである。Singularity は マルチユーザに対応しており, コンテナ内での権限は実行ユーザの権限を引き継ぐため, ユーザに特別な権限の設定が必要ない。またデフォルトで, $HOME, /tmp, /proc, /sys, /dev がコンテナにマウントされ, サーバ上の GPU を簡単に利用できる。Singularity のコンテナイメージは Docker Hub に登録されているイメージ, または Dockerfile から作成したイメージを変換することで利用することができる。

2.4 GitLab

GitLab[6] とは バージョン管理システムである Git のリポジトリマネージャである。GitLab は GitHub と違い, オンプレミス環境に構築することができるため, 本コースでは GitLab を使用している。本研究では GitLab の統合機能の GitLab CI/CD[7], また GitLab CI/CD と組み合わせて使用する GitLab Runner[8] を利用する。

GitLab CI/CD は 継続的インテグレーション (CI)・継続的デリバリー (CD) を GitLab から利用することができる。CI では GitLab のコードを定期的または自動的にビルド・テストを行う。CD は CI を拡張した機能であり, ビルドやテストだけでなくリリースの準備も行う。

GitLab Runner とは, ビルドのためのアプリケーションであり, GitLab CI と連携することで別の場所でビルドを動かすことができる。

2.5 Kernel-based Virtual Machine (KVM)

KVM[9] とは, Linux 自体を VM の実行基盤として機能させるソフトウェアである。CPU の仮想化支援機能を必要とし, 完全仮想化により OS の仮想化環境を提供する。

2.6 ie-virsh

ie-virsh[10] とは, 本コースの Operating System という講義に向けに libvirt の CLI である virsh をラップし複数のユーザで利用することができる VM 管理ツールである。学生が手元の PC で作成した VM をブレードサーバ上にデプロイすることができる。

2.7 ie-docker

ie-docker とは Docker をラップし複数のユーザで利用することのできるコンテナ管理ツールである。利用する学生は ssh でブレードサーバへ接続し, ie-docker を使用してコンテナを操作することができる。

2.8 digdog

digdog とは Kubernetes を利用し Web コンソールからコンテナを作成することができるコンテナ貸出サービスである。学生は学科アカウントを使用して Web サービスへログインし, 登録されている Docker イメージでコンテナを作成することができる。

3. サービスの設計

サービスでは本コースの学生や教員がにコンテナ貸出を

行う。このコンテナ貸出の構成を図 1 に示し, 概要を以下で説明する。

3.1 コンテナの作成

学生または教員は学科アカウントで Web コンソールへログインする。Web コンソールでは ユーザのコンテナ一覧や Docker イメージ一覧を確認することができる。コンテナ作成は Docker コンテナと Kubernetes コンテナの 2 つから選択することができる。コンテナ作成を選択するとコンテナを作成するために必要な情報を入力する。入力する内容は表 1 である。コンテナ名にはユーザのアカウント名が補完されるため, 他のユーザと被ることはない。Docker イメージには Docker Hub に登録されているイメージや, ユーザが作成したイメージを入力することができる。環境変数とゲストポートはスペース区切りで複数入力することができる。また, コンテナ内で GPU を使用するにはチェックボックスにチェックを入れる。ホストポートは, エフェメラルポートの範囲から設定される。ユーザは設定されたホストポートを使用してコンテナのサービスへアクセスする。また, ユーザは Docker コンテナに対しては Web コンソールから, Kubernetes コンテナには手元の PC から操作することができる。必要なくなったコンテナは Web コンソールのコンテナ一覧から削除することができる。

表 1: コンテナ作成時の入力内容

ContainerName	コンテナ名
Image	Docker イメージ
Environments	コンテナ作成時の環境変数
GuestPort	コンテナが使用するポート番号
GPU	GPU を使用するか

3.2 イメージの作成

Docker イメージの作成は学科で使用している GitLab の CI/CD の CI 機能を利用する。ユーザは学科 GitLab から CI トークンを取得し, Web コンソールに取得したトークンをセットする。この時 Docker 側に GitLab Runner の立ち上げを依頼する。トークンの設定後, Web コンソールから CI 用の YAML ファイルをダウンロードし Dockerfile と一緒に学科 GitLab のリポジトリにプッシュする。Docker イメージのビルドが成功すると Web コンソールのイメージ一覧で確認ができる。作成した Docker イメージは編集からイメージの使い方の記述や他の学生に共有するか設定を行える。必要なくなったイメージは Web コンソールのイメージ一覧から削除することができる。

3.3 Singularity の利用

コンテナに大量のデータを送信する必要がある場合や, データを永続化させたい場合に Singularity を利用する。Singularity は Docker イメージを変換し使用できるが, イメージの変換には sudo 権限が必要となる。Docker イメージの変換を申請性にすると, 管理者の仕事が増え, またユーザも利用しづらい。Singularity はユーザ権限で動作することから, 学生が ssh でブレードサーバへ接続し利用する方が適している。そこで, Web コンソールから Singularity 用のイメージをダウンロードできる仕様とする。

ユーザは利用したいイメージをダウンロードし, ブレードサーバへ送信して Singularity を使用する。Singularity を利用する流れを図 2 に示す。

4. サービスの実装

本コースでは学科システムを教員の指導の下, 学生主体でシステム管理チームと呼ばれる組織によって構築・運用・管理が行われている。学科システムはブレードサーバを 4 台, SAN 用ストレージと汎用ストレージをそれぞれ 2 台ずつ導入している。本コースの基幹サービスはこのブレードサーバの仮想環境上で VM として動作している。新たにサービスを実装するとなると, システム管理チームが運用・管理を行いやすい実装にする必要がある。

Web コンソールや Docker の操作を 1 つにまとめると, Docker コンテナの作成が 1 台のブレードサーバのみになってしまう。そこで, コンテナ貸出システムは, 機能ごとに以下の 3 つにサービスに分ける。Docker や Kubernetes の操作を HTTP API で提供することにより, リクエスト先を変更することで複数のブレードサーバにコンテナを分散することができる。だが, 現時点では未実装である。

実装には Docker や Kubernetes の実装言語であり, 操作するためのライブラリが揃っている Go 言語を使用する。

- Web コンソール
- Docker の操作
- Kubernetes の操作

4.1 Web コンソール

Web コンソールは本コースの学生や教員が利用するため, 学科アカウントでログインできる必要がある。学科の LDAP サーバを利用して学科アカウントで LDAP 認証を実装する。

Docker の操作や Kubernetes の操作を行う HTTP API はセッション管理を行わないため, Web コンソールで管理する必要がある。そのため, ユーザのコンテナやイメージの情報をデータベースに格納して管理する。ユーザが作成する Docker イメージの情報を取得しユーザのアカウント ID と紐付けを行う。また, 作成した Docker イメージは共有することができ, 共有されたイメージはユーザのイメージ一覧とは別の一覧で確認することができる。ユーザはコンテナ作成時にイメージを入力することができる。この時,

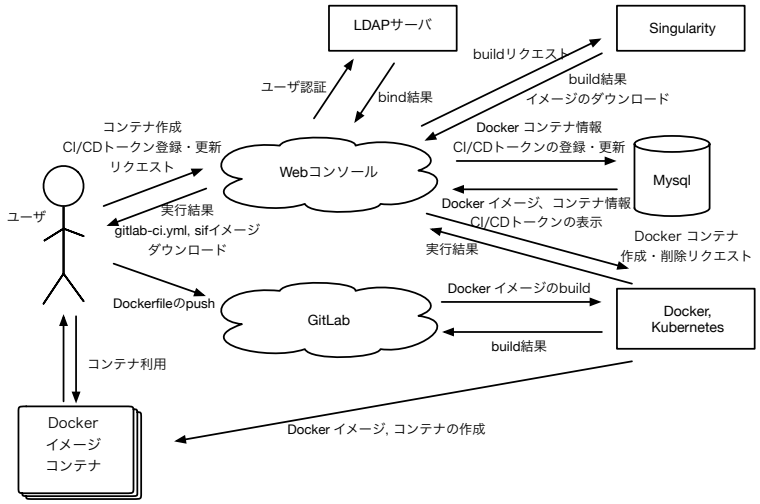

図 1: システム構成

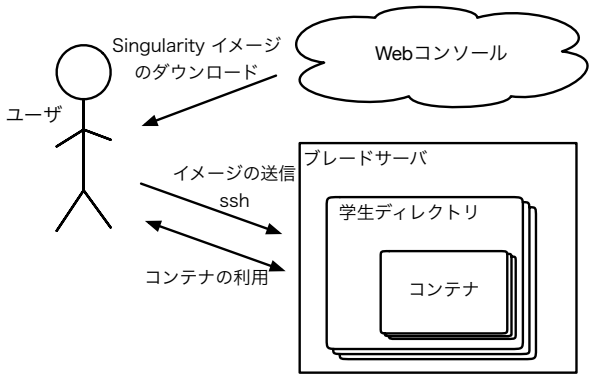

図 2: Singularity の利用

他のユーザが作成したイメージの場合, そのイメージが共有されたイメージなのか確認を行うことで, 非共有に設定されたイメージではコンテナの作成はできない。コンテナの操作を行う時, コンテナに紐づけられたアカウント ID との確認が行われることで, 他のユーザのコンテナを操作することはできない。同様にイメージの削除を行う時にもアカウント ID の確認が行われる。

4.2 Docker の操作

Docker は Docker Engine API を提供している。Docker デーモンは指定した IP アドレスと ポート を リッスンする。 IP アドレスと ポートの指定を行うことで外部から Docker の操作が可能になる。だが, Dockr デーモンが稼働しているホスト上の root アクセスを得られるため, 推奨されてない。また, 本研究で実装するサービスでは Docker のすべての操作を必要としない。そこで, Docker の操作を行うための SDK [11] を使用し, 必要な機能のみを実装する。

サービスを提供する上で Docker の必要となる操作は以下である。

- コンテナの作成
- コンテナの削除
- コンテナでのコマンド実行
- コンテナへファイル送信
- イメージ一覧の取得
- イメージの削除

コンテナは, 表 1 で入力した情報を下に作成を行う。コンテナ名は Web コンソールからリクエストを送るタイミングで補完される。また, コンテナが属するネットワーク名も補完される。リクエストは JSON 形式でやり取りを行う。リクエストからコンテナを作成後, 作成したコンテナ ID やネットワーク ID, コンテナのステータスを返却する。返却したコンテナ ID やネットワーク ID を下にコンテナ削除やコマンドの実行, ファイルの送信を処理する。ファイルの送信では JSON 形式ではなく multipart/form-data 形式でリクエストを受ける。ユーザが作成するコンテナとは別に, GitLab CI/CD で Docker イメージをビルドするための GitLab Runner を立てる必要がある。立ち上げはユーザが Web コンソールで CD トークンの設定時に行われる。GitLab Runner をユーザごとに立ち上げることで, 複数のユーザが同時にビルドを行うことができる。

Docker イメージは GitLab CI/CD を利用して作成するが, ビルドが成功したかを判断することはできない。そのため, Web コンソール側から 5 分に一度イメージの更新リクエストを受け, Docker イメージの一覧をリストにまとめ返却を行う。

4.3 Kubernetes の操作

Docker と同様に Kubernetes のすべての操作を必要としないため, Kubernetes と対話するためのライブラリである client-go [12] を使用し, 必要な機能のみを実装する。サービスを提供する上で Kubernetes の必要となる操作は以下である。

- コンテナの作成
- コンテナの削除

- 認証トークンの取得

Kubernetes でのコンテナ作成は Pod を作成することである。 Kubernetes でのコンテナ作成は Namespace, Deployment, Service, Ingress の流れでオブジェクトを作成する。コンテナの作成は Docker と同様に表 1 の情報を下に作成する。作成するそれぞれのオブジェクト名は Web コンソールでコンテナ名とアカウント ID で補完される。また, Namespace はアカウント ID となる。コンテナの削除にはそれぞれのオブジェクト名と Namespace を用いる。

Kubernetes で作成したコンテナは Web コンソールからコマンドの実行やファイルの送信などの操作を行わず, Role と RoleBinding を用いた Role-based access control (RBAC) を利用することで手元の PC より操作を行うこととする。RBAC で使用する認証トークンはユーザごとに作成された Namespace に設定されるトークンを返すことで, 他のユーザが認証することはできない。またアクセス制御は Namespace ごとに設定されることで, 他のユーザのコンテナを操作することはできない。RBAC で許可するリソースの操作は表 2 である。

表 2: kubectl のコマンド

get	Pod, Deployment, Service, Ingress の一覧を表示する
log	Pod の Log を表示する
exec	Pod にアクセスする
cp	Pod にファイルを送信する

5. 他のサービスとの比較

今回作成した Web サービスは主に学生の学習環境をコンテナ技術を利用して提供する。そこで, これまで本コースで使用されてきたサービス, また近年普及しているクラウドサービスと比較する。

5.1 ie-virsh

ie-virsh は手元の PC で作成した VM を学科のブレードサーバにデプロイできるサービスである。ie-virsh は ユーザの UID 及び GID 情報を取得することで, 他のユーザの VM を操作させない。表 3 は ユーザが利用できる ie-virsh の機能である。ie-virsh では VM の実行環境を提供するため, ユーザが好みの VM を構築できるなど自由度が高い。

本研究で実装したサービスでは, コンテナ作成に使用する Docker イメージで構築されたアプリケーションに限定される。また, ユーザが欲しい環境は Docker イメージを作成しなければいけないため, Docker について学習する必要がある。だが, コンテナは VM と違い開発環境やテスト環境の構築の容易さや, 他のユーザにイメージを共有することで同じ環境を利用することができるなどの良さがある。

表 3: ie-virsh のコマンド

define	XML の template を下に domain を作成
undefine	define で作成した domain を削除
list	define で作成した domain の一覧表示
start	指定した domain 名の VM を起動
destroy	指定した domain 名の VM を停止
dumpxml	domain の XML を参照

5.2 ie-docker

ie-docker は Docker をラップしたツールであり, ユーザは学科のブレードサーバへ ssh で接続を行い CUI から利用することができる。ie-docker は ユーザの UID 及び GID 情報を取得することで, 他のユーザのコンテナを操作させない。表 4 は ie-docker で利用できる機能である。この機能を使用しコンテナ作成などの操作を行うことができる。だが, ie-docker ではユーザがコンテナで使用するイメージを管理者が用意する必要がある。そのため, コンテナで使用したいイメージが用意されていないなどの問題があった。

本研究で実装したサービスでは, コンテナで使用するイメージは Docker Hub に登録されているイメージ, または作成したイメージを利用することができる。ユーザが Docker イメージを作成できることで管理者の負担が少なくなると考える。

表 4: ie-docker のコマンド

ps	起動中のコンテナの一覧を表示する
run	コンテナを作成する
start	コンテナを起動する
stop	コンテナを停止する
attach	起動しているコンテナに attach する
cp	コンテナにファイルを送信する
rm	コンテナを削除する

5.3 digdog

digdog は Kubernetes を利用したコンテナ貸出サービスである。学生は Dockerfile を GitLab CI/CD を利用してビルドし, 学科の GitLab Container Registry に Docker イメージを登録する。Web コンソールからコンテナ作成で使用するイメージは, 登録したイメージから選択することができる。Kubernetes 上にコンテナを作成するため, 学生の入力を下に Deployment, Service, Ingress の作成を行う。また, Namespace はユーザのアカウント ID で作成され, Namespace ごとに RBAC の設定がされている。そのため, 学生は手元の PC から Web コンソールから作成したコンテナを操作することができる。表 5 は RBAC で許可されているリソースの操作である。

本研究のサービスは digdog を参考に実装されている。そのため, Kubernetes でのコンテナ作成は digdog と同じ

流れで作成を行う。だが, digdog では Docker Hub に登録されているイメージを選択することができない。また, Kubernetes でのコンテナ貸出のため, Kubernetes クラスター上の Master が停止するとサービスを提供できないなどの課題があった。本研究のサービスでは, Kubernetes でのコンテナの作成だけでなく, Docker でのコンテナの作成にも対応することで, Docker のみでサービスの提供ができるように改良を行った。また, コンテナ作成時のイメージを選択ではなく入力にすることで, Docker Hub に登録されているイメージを利用できるように改良を行った。

表 5: kubectl のコマンド

get	Pod の一覧を表示する
log	Pod の Log を表示する
exec	Pod にアクセスする

5.4 クラウドサービス

近年様々なクラウドサービスが普及し手元の PC から高性能なクラウド環境を利用することができる。だが, クラウドサービスは無料から有料までであり。無料では時間制限や容量制限など様々な制限がある。また有料だと設定のミスにより高額な請求がくるなど, 気軽に利用しづらい。そのため, 本サービスはオンプレミス環境で実装を行った。オンプレミス環境にすることで利用の制限がなく, サービスで使用するデータを自身で管理できるなどの良さがある。だが, クラウドサービスと違い物理サーバなどのメンテナンスや, サービスの導入にあたって環境構築などの課題がある。しかし, オンプレミスで運用していくことで, 学生の学習に繋がると考える。

6. 今後の課題

本研究で実装したサービスでは学生が学習環境として利用するには, まだ必要な実装が不足している。

本サービスでは, 大量のデータを用いる時に Singularity を使用できる環境を用意している。だが, Web コンソールから作成した Docker や Kubernetes のコンテナではデータの永続化が対応していないため, コンテナの削除時に共に削除されてしまう。学科のブレードサーバでは学生ごとのディレクトリがある。Docker ではそのディレクトリをコンテナ立ち上げ時にマウントすることで, コンテナ内のデータの永続化に対応できる。また, Kubernetes では Peresistent Volumes という永続ボリュームの仕組みがある。この Peresistent Volumes をユーザごとに管理することで, コンテナ内のデータの永続化に対応できる。このような対策でコンテナ内のデータの永続化に対応できるが, コンテナごとにデータの保存場所が違うなどの問題があるため, データを管理する仕組みが必要だと考える。

本サービスでは, 学生が自由に Docker イメージを作成できる。また, Docker イメージを Singularity 用のイメージに変換する。そのため, イメージの容量でブレードサーバのストレージを圧迫してしまう可能性があることから, 定期的にイメージを削除する必要がある。また, 本サービスではユーザごとにリソースの制限を行っていないため, 過剰なリソースの占有を防ぐための対策をする必要がある。GPU などの負荷がかかるプログラムの実行で使用されるリソースにはジョブ管理ソフトウェアなどで対策をとる。

本サービスは Docker Hub に登録されている Dokcer イメージを利用できるが, Docker Hub は誰でもイメージを登録することができる。そのため, Docker Hub に登録されているイメージにマルウェアが仕込まれている可能性がある。イメージの取得時にスキャンを行うなど, セキュリティ対策を考える必要がある。

7. まとめ

本稿では, 本コースで利用する新規サービスの設計と実装, また本コースで利用しているサービスとの比較を行った。学生がコンテナ技術を用いて学習環境を利用できるサービスの設計をし, マルチユーザ環境に対応した実装をすることができた。学生は学科アカウントを持っていれば, Web コンソールからコンテナの作成ができる。また, GitLab CI/CD を利用し Dockerfile から Docker イメージをビルドすることで, イメージの作成を学ぶこともできる。

今後, テスト環境にデプロイを行い, ユーザやシステム管理チームからのフィードバックをもらい, 改善や実証実験を目指す。

参考文献

[1] Docker: *https://www.docker.com/*.
[2] NVIDIA Container Toolkit: *https://github.com/NVIDIA/nvidia-docker*.
[3] Singularity: *https://sylabs.io/singularity/*.
[4] Docker Hub: *https://hub.docker.com/*.
[5] kubernetes: *https://kubernetes.io/*.
[6] GitLab: *https://about.gitlab.com/*.
[7] GitLab CI/CD: *https://docs.gitlab.com/ce/ci/*.
[8] GitLab Runner Docs: *https://docs.gitlab.com/runner/*.
[9] KVM: *https://www.linux-kvm.org/*.
[10] 平良太貴，河野真治：OS 授業向けマルチユーザ VM 環境の構築，研究報告システムソフトウェアとオペレーティング・システム (OS) (2014).
[11] Docker Engine API: *https://docs.docker.com/engine/api/*.
[12] Go clients for talking to a kubernetes cluster: *https://github.com/kubernetes/client-go*.
[13] 秋田海人，高瀬大空，上地悠斗，長田智和，谷口祐治：情報系学科における教育研究情報システムの運用管理並びに新規システムの構築に関する取り組み，インターネットと運用技術シンポジウム (2019).