データ駆動型研究支援のためのコンテナプラットフォーム

小谷 大祐 [1)], 廣中 詩織 [1)], 首藤 一幸 [1)], 當山 達也 [2)], 疋田 淳一 [2)], 戸田 庸介 [2)], 島袋 友里 [2)]

1) 京都大学学術情報メディアセンター
2) 京都大学情報部

kotani@media.kyoto-u.ac.jp

A Container Platform for Supporting Data Driven Research

Daisuke Kotani[1)], Shiori Hironaka[1)], Kazuyuki Shudo[1)], Tatsuya Tohyama[2)], Junichi Hikita[2)], Yosuke Toda[2)], Yuri Shimabukuro[2)]

1) Academic Center for Computing and Media Studies, Kyoto University
2) Information Management Department, Kyoto University

概要

データ駆動型研究の重要性が高まる中、研究者による大量データの収集・保存・管理・解析を支える環境が求められている。従来のスーパーコンピュータは、データの収集や管理に求められるような常時実行する必要がある処理や外部からの接続を受け付ける必要がある処理には対応が難しい。プロジェクトごとに分離可能な IaaS 型のクラウドサービスの利用についても、研究者が自らそれぞれのクラウドサービス毎にゼロからシステム構築・運用を行う必要があり、高い工数と移植性の課題がある。また、研究分野特化型のプラットフォームの整備も、学際的な研究の促進等を考慮すると必ずしも将来的に有望な選択肢とは限らない。本稿は、これらの課題を解決するため、軽量な仮想化技術と実行環境の移植性を実現するコンテナ技術に着目して京都大学学術情報メディアセンターに導入したデータ駆動型研究支援のためのコンテナプラットフォームについて報告する。本システムはコンテナオーケストレーションツールとして Kubernetes を採用し、namespace 機能を利用したマルチテナントのコンテナプラットフォームを実現する。さらに、テナントごとに Worker ノードを分離することでセキュリティ上の課題に対応するとともに、テナントで占有できる EgressGateway や LoadBalancer を提供することで研究室ネットワークとの接続をサポートし、実験機器とのデータ連携を支援する。また、本プラットフォームの利用例を示し、データ駆動型研究をどのように支援できる可能性があるかを示す。

1 はじめに

近年、研究におけるデータの重要性が増しており、データ駆動型研究が盛んになってきている。データ駆動型研究では、大量のデータを収集・保存・管理し、そのデータを解析するための計算環境を、各研究分野におけるデータの取り扱いに関する要求を満たせるように利用できるようにすることが重要である。学術研究用途では従来より大規模計算のためにスーパーコンピュータが利用されており、データの解析にも利用できると想定される。一方で、データの収集・保存・管理の一連の流れにおいては、常時稼働させるべき処理や定期的に実行すべき処理等、スーパーコンピュータが必ずしも適していないような処理も存在する。また、システムの構築・運用には多くのエフォートを要するため、研究者によるデータの収集から解析までのステップをシームレスに接続できるように支援することで、研究者はより研究に専念できると考えられる。

このようなデータ駆動型研究を支援する計算機システムについて様々な試みがなされてきた。例えば、計算資源の提供の観点では、仮想化技術を用いてプロジェクト毎に分離されたネットワークとそれに接続される仮想マシンを提供する IaaS 型のクラウドサービスにより、常時実行が必要な処理にも対応できるような大規模なデータ処理基盤を提供する試みがされてきた [1, 2, 3, 4]。しかし、IaaS 型のクラウドサービスでは研究者が自らソフトウェア構成の検討やインストール・設定等のシステムの構築・運用作業を実施する必要があり、また、利用する計算機システムが変わると最初から構築し直す必要があるなど他のサービスへの移植性が低く、研究者の負担が大きい。いくつかの研究分野ではそれぞれの分野に特化したデータプラットフォーム構築されている [5, 6, 7] が、学際的な研究においては研究者は複数のデータプラットフォームを利

用する必要があり、必要なデータやデータ分析環境にシームレスにアクセスするのは難しい。

近年のコンテナ技術の発展により、軽量な仮想化技術を用いてアプリケーションの実行環境を分離しつつ、ソフトウェアの実行環境をコンテナイメージとしてまとめることで、移植性のあるソフトウェア実行環境が実現できるようになった。これを利用し、Development Containers や GitHub Codespaces のように開発環境をコンテナで提供する試みや、JupyterHub のようなデータ分析環境をコンテナで提供する試みがなされている。また、DockerHub のように、コンテナイメージを共有するための仕組みも整備されてきている。これらを利用することで、研究者のシステム構築・運用の負担を軽減しつつ、かつ移植性が高いデータ駆動型研究を実施するためのシステムを構築・運用できる可能性がある。

本稿では、研究者によるコンテナを利用したデータ駆動型研究のためのシステムを支援するために京都大学学術情報メディアセンターに導入した、データ駆動型研究支援のためのコンテナプラットフォームについて報告する。本システムでは、コンテナオーケストレーション基盤として Kubernetes[8] を採用し、namespace 機能を利用したマルチテナント環境を実現している。また、テナント毎にコンテナを実行する Worker ノードを分離することにより、軽量な仮想化技術に由来するセキュリティ上の課題に対応する。加えて、各テナント内で実行されるコンテナと研究室のネットワークに接続されている機器との通信を可能にし、研究室の実験機器等とのデータのやり取りを支援するため、テナントが占有できる EgressGateway および LoadBalancer を提供する。

本稿の構成は以下の通りである。第 2 章では関連研究について述べる。第 3 章では本システムにおけるマルチテナントの考え方とそれを満たすシステム設計について述べる。第 4 章では想定している利用例と現在のシステムの制約について述べる。第 5 章ではまとめと今後の検討課題について述べる。

2 関連研究

2.1 データ駆動型研究支援のためのシステム

データ駆動型研究においては、大規模なデータの利用および処理を、データの取り扱いに対するセキュリティ上の要求を満たした環境で行うことが求められる。

これを支援する方向性の一つが研究データのための大規模なストレージの提供である。例えば、Open Storage Network[9] は研究用の分散ストレージを国レベルで構築する試みである。また、従来より大規模なデータセットを利用する研究分野においては、分野独自のデータプラットフォームを運用している。例えば、DIAS[5] は地球環境に関する幅広いデータを長年蓄積し、研究者にデータを提供している。ESGF[10] は気候変動に関するデータを共有するソフトウェアを国際的に開発・運用する取り組みであり、これを用いて大規模な気候学に関するデータが共有されている。INSDC[7] は国際的な塩基配列データベースを連携して整備する取り組みである。ARIM-mdx[6] は材料科学分野を対象に実験機器からデータ分析環境への自動的なデータのアップロードを実現する、多くの研究者で共用可能なシステムである。

近年はデータが大規模化しており研究者の手元の計算機では処理が難しくなっていることから、データの保存に加えてデータの処理も支援できる大規模なストレージと計算資源を両方備えたシステムがある。mdx[1, 2] は、オブジェクトストレージのサービスとプロジェクト単位での分離に対応した仮想ネットワークに接続された仮想マシンを実行できる IaaS 型のクラウドサービスを提供し、大規模なオブジェクトストレージとそのデータを処理する計算資源を高速に接続した環境を提供している。Jetstream2[3] は、オブジェクトストレージのサービスと IaaS 型のクラウドサービスに加え、大規模言語モデルを用いた推論サービスの提供も始めている。また、European Open Science Cloud - EU Node[11] はストレージと IaaS 型のクラウドサービスに加え、コンテナプラットフォームや大規模なデータの転送、Web ブラウザによるデータ分析環境など、様々な機能を包括的に提供している。スーパーコンピュータとの連携を意識して設計されたデータ集約基盤である ONION[12] やクラウド基盤 [4] もある。従来のスーパーコンピュータでは取り扱いに厳しい制約のあるデータを扱うことが難しく、かつ研究者がインタラクティブにデータの分析を行うことが難しい、仮想マシンは研究者にサーバ構築・管理の負担を強いる、等の課題があり、データ駆動型研究を支援するための大規模な計算機システムの模索は続いている。

2.2 コンテナオーケストレーションツール

コンテナオーケストレーションツールは、複数のノードにまたがってコンテナを管理・運用するためのツールである。代表的なものに、Kubernetes[8]、

Docker Swarm、Apache Mesos[13] 等がある。これらのツールは、コンテナのデプロイ、スケーリング、負荷分散、障害復旧などの機能を提供し、大規模なコンテナ環境の管理を容易にする。

本システムでは、Kubernetes を採用した。Kubernetes は、Google が開発した Borg[14] を元にするオープンソースのコンテナオーケストレーションツールであり、広く利用されている。また、マルチテナント環境の構築に必要となる様々なアクセス制御機能を備えている。

Kubernetes は、大きく分けて、クラスタ全体を制御するための Control Plane と、実際にコンテナを実行する Worker ノードから構成される。Control Plane は、クラスタに関する操作の要求を受け付ける API サーバ、コンテナを実行するノードを決定するスケジューラ、クラスタの状態を監視し、望ましい状態に保つためのコントローラ等から構成される。コンテナの実行環境の構成の望ましい状態は「オブジェクト」という概念で宣言的に記述され、Control Plane は望ましい状態と実際の状態を比較し、必要に応じて実際の状態を望ましい状態に近づけるように動作する。同じ種類のオブジェクトの集合を「リソース」という。また、オブジェクトにはラベルやアノテーションを付与することができ、これらを利用してオブジェクトをグループ化したり、特定のオブジェクトに対して操作を行ったりすることができる。さらに、Role-Based Access Control (RBAC) によるアクセス認可機構を持ち、利用者毎に個々のリソースに対する権限を細かく設定することができる。Worker ノードは Control Plane のオブジェクトの状態に従い、コンテナを実行する。

Kubernetes における代表的なオブジェクトには以下がある。

Pod　複数のコンテナをまとめて実行するためのオブジェクト。Pod 内のコンテナは同じネットワーク名前空間を共有し、同じホスト上で実行される。

Service　Pod の集合に対する通信を提供するためのオブジェクト。Pod の IP アドレスが変わっても一定の IP アドレスや DNS 名でアクセスできるようにする。

Deployment　Pod の望ましい状態を定義するためのオブジェクト。指定された数の Pod が常に実行されるように管理し、Pod の更新やスケーリングを容易にする。

CronJob　定期的に実行されるジョブを定義するためのオブジェクト。指定されたスケジュールに従ってジョブを実行し、完了したジョブの履歴を管理する。

Namespace　クラスタ内のリソースを論理的に分離するためのオブジェクト。これを利用することで、複数のチームやプロジェクトが同じクラスタを共有しつつ、リソースを分離して管理することができる。

NetworkPolicy　Pod 間の通信を制御するためのオブジェクト。これを利用することで、特定の Pod から他の Pod への通信を許可または拒否することができる。

Node　Worker ノードの状態を示すオブジェクト。

3　システム構成

3.1　本システムにおけるマルチテナント対応

複数人が共同で利用するシステムを Kubernetes を用いて構築する場合、マルチテナント対応が必要になる。これを実現する方向性として 2 つの方法が考えられる。

1 つは、Kubernetes のクラスタをテナント毎に分離する方法である。この方法では、テナント毎に独立した Control Plane と Worker ノードを用意するため、テナント間の分離が強固であり、また、テナント毎に異なる Kubernetes のバージョンや設定を利用できるという利点がある。一方で、Control Plane をテナント毎に用意する必要があるため運用コストが高く、また、Worker ノードの計算資源量が少ない場合に Control Plane で利用される計算資源によるオーバーヘッドが大きくなる。

もう 1 つは、Kubernetes のクラスタを共有しつつ、Kubernetes の namespace 機能を利用してテナント毎にリソースを分離する方法である。この方法では、1 つの Control Plane で複数のテナントを管理できるため、Worker ノードの計算資源量に比べて Control Plane で利用される計算資源によるオーバーヘッドを小さくすることができるという利点がある。一方で、Kubernetes の namespace 機能は Control Plane 内の論理的な分離であるため、テナント間の分離が弱く、また、テナント毎に異なる Kubernetes のバージョンや設定を利用できないという課題がある。さらに、マルチテナントのための設定が複雑になるという課題もある。

本システムでは、計算資源を小規模に利用するテナントも想定し、Namespace 機能を利用したマルチテナント対応を採用した。

3.1.1 マルチテナントに求められる要件と実現方針

本システムにおけるテナントは研究グループ程度の粒度の単位での利用を想定している。また、研究室やフィールドに存在する機器からデータを自動的に収集する用途や、データ分析環境としての利用、研究成果の公開のための Web サービスの公開等、様々な用途での利用を想定している。そのため、利用者が自由にコンテナを持ち込めることを前提に要件を検討することにした。以下にマルチテナントに関する要件とそれを満たすための実現方針を示す。

テナント間のコンテナ実行環境の分離 コンテナは OS の名前空間の分離機能を利用して構成された環境の中で実行される。そのため、悪意のあるコンテナによりノードの OS が侵害されると他のコンテナに影響が及ぶ可能性がある。この影響を低減するため、テナントごとにコンテナを実行する Worker ノードを分離し、コンテナによる Worker ノードの侵害が発生した時の影響範囲を単一のテナントに限定できるようにする。この要件は、サーバ仮想化基盤を導入しテナント毎の Worker ノードを仮想マシンとして用意するとともに、Kubernetes のスケジューリング機能を利用してテナント毎にコンテナを実行する Worker ノードを限定することで実現する。また、テナント間の通信をデフォルトで遮断することにより、異なるテナントのコンテナが明示的に許可されない限り通信できないようにする。

テナント毎に異なる複数の外部ネットワークと接続できること 各テナントのコンテナの通信先は、実験機器やフィールドに存在する機器が接続されている研究室のプライベートネットワークや、研究成果の公開のためのインターネット等、テナント毎に異なることが想定される。プライベートネットワークはテナント毎に異なるため、他のテナントのプライベートネットワークには接続できないようにする制御も必要である。また、一つのテナントで、研究室等のプライベートネットワークとインターネット接続の両方を利用したい場合も想定される。さらに、利用されるプロトコルは様々であり、任意の TCP/UDP のポートを利用できる必要がある。この要件は、Kubernetes において外部から接続を受け付けるための LoadBalancer、および、コンテナ毎に外部に接続するためのゲートウェイを強制できる EgressGateway を利用して実現する。外部ネットワークを VLAN で分離して引き込み、VLAN 毎に LoadBalancer および EgressGateway を用意するとともに、Kubernetes の機能を利用してテナント毎に利用できる LoadBalancer および EgressGateway を限定することで実現する。

なお、永続ストレージは、既に研究データ用の大規模なオブジェクトストレージが複数利用可能な状況であり、本学においてもエッジコンピューティング基盤に隣接する大規模な研究データ用のストレージが導入予定であったことから、それらを利用することを想定し、本システムでは提供しないこととした。

3.2 導入システム

3.2.1 ハードウェアおよびソフトウェア

図 1 にハードウェア構成を示す。演算ノードとして、AMD EPYC 9734 (2.2GHz, 112 コア) を 2 基、メモリを 512GiB 搭載したサーバを 14 台、ストレージシステムとして、15.36TB の SSD を 18 台搭載した実効容量 205TB のストレージアプライアンスを 1 台導入した。ネットワークは 100Gbit Ethernet を採用しており、各演算ノードは 100Gbps で、ストレージシステムは 200Gbps で接続している。なお、上流回線となる京都大学学術情報ネットワークシステム (KUINS) には 200Gbps で接続している。

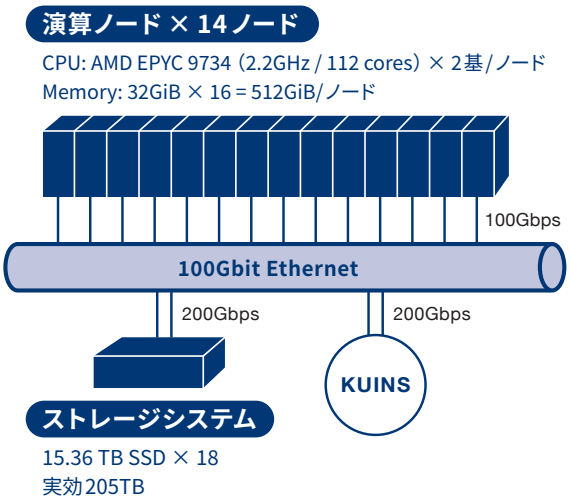

図 1　ハードウェア構成

図 2 にシステムのソフトウェア構成を示す。サーバ仮想化基盤として Proxmox VE[15] を導入し、その上に Kubernetes の Worker ノードが仮想マシンとして動作している。また、Kubernetes の Control Plane を構成する制御用ノードは本学情報環境機構が提供する VM ホスティングサービスを利用した仮想マシン

上に構築されている。コンテナ基盤は Rancher[16] により管理される Kubernetes[8] クラスタであり、アドオンとして、Namespace ベースのマルチテナントを実現する Capsule[17] およびオブジェクトに対する細かなポリシーの制御を行う Kyverno[18]、ネットワークの設定および制御を行う Cilium[19]、外部への接続に際し経由するノードを強制する EgressGateway 機能を提供する egressgateway[20]、および外部からの接続を受け付けるための LoadBalancer 機能を提供する MetalLB[21] が導入されている。2025 年 9 月 26 日現在のコンテナ基盤を構成する各ソフトウェアのバージョンを表 1 に示す。

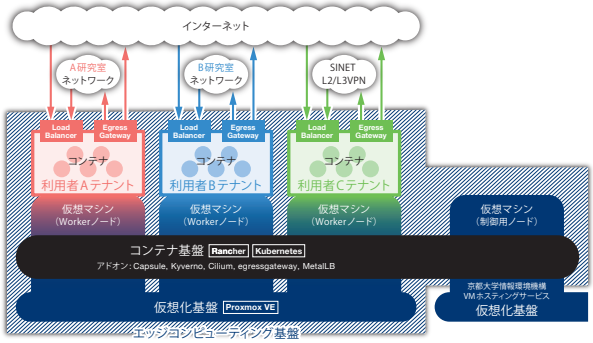

図 2　ソフトウェア構成

ソフトウェア名	バージョン
Rancher	v2.10.2
Kubernetes	v1.31.5
Capsule	v0.7.0
Kyverno	3.3.7
Cilium	1.16
egressgateway	0.6.0
MetalLB	0.14.9

表 1　コンテナ基盤を構成するソフトウェアのバージョン

利用者によるテナントの管理のため、利用者ポータルを提供する。利用者ポータルは、利用者が Kubernetes API Server にアクセスするため認証情報等を含む kubeconfig のダウンロード機能、API へのアクセスの閲覧機能、Pod の動作状況を閲覧する機能、テナントの管理者によるテナントの利用者の追加・削除等の管理機能等を有している。また、利用者ポータルの認証は本学スーパーコンピュータシステムの認証基盤と SAML を用いて連携しており、利用者ポータルにアクセスする全ての利用者は本学スーパーコンピュータシステムの利用者番号を有することを前提としている。

3.3　マルチテナントを構成するための設定

本節で述べる設定は、今後ソフトウェアのアップデートおよび構成変更に伴い変更される可能性があることに注意されたい。

3.3.1　コンテナを制御するためのラベルおよびアノテーション

本システムの要件を満たすマルチテナント構成を実現するため、オブジェクトにラベルを付与し、ラベルを用いて制御している。表 2 に本システムで独自に利用しているラベルを示す。

また、LoadBalancer において外部からの接続を受け付ける IP アドレスを指定するために Service リソースにアノテーションとして付与する `metallb.io/loadBalancerIPs` も利用している。

3.3.2　マルチテナントを実現するための制御

テナント間のコンテナ実行環境の分離　各テナントは Capsule の Tenant リソースを用いて定義される。Tenant リソースの `.spec.nodeSelector` において、`edge.kudpc.kyoto-u.ac.jp/node_`**テナント名**ラベルを指定することにより、一致するラベルを持つ Node オブジェクトに対応する Worker ノードでのみコンテナが実行されるように制御している。加えて、Control Plane において PodNodeSelector の Admission Controller を有効にすることにより、コンテナを実行するノードをラベルを用いて限定する機能を有効にしている。また、Tenant リソースの `.spec.namespaceOptions` を利用して、当該テナント内で実行される全ての Pod に対し `pod-security.kubernetes.io/enforce: restricted` ラベルを付与することにより、コンテナが Worker ノード自体に不適切な操作を行うことを禁止している。Tenant リソースにおける Tenant に属する利用者の追加や削除は、利用者ポータルにおけるテナントの利用者の管理機能と連動している。

テナント間の通信の遮断は、CiliumClusterwideNetworkPolicy を利用して実現している。

テナント毎に異なる複数の外部ネットワークと接続できること　EgressGateway については、egressgateway の EgressGateway リソースを利用し、`.spec.nodeSelector` を用いて EgressGateway として利用するノードを `edge.kudpc.kyoto-u.ac.jp/eg_`**テナント名**ラベルが付与されているノードに限定するとともに、外部との通信に利用する IP アドレスプールを指定している。

ラベル名	付与対象のリソース	用途
edge.kudpc.kyoto-u.ac.jp/lb_テナント名	Node	「テナント名」のテナント用の LoadBalancer を有するノードを示す
edge.kudpc.kyoto-u.ac.jp/eg_テナント名	Node	「テナント名」のテナント用の EgressGateway を有するノードを示す
edge.kudpc.kyoto-u.ac.jp/node_テナント名	Node	「テナント名」のテナントのコンテナを実行するノードを示す
edge.kudpc.kyoto-u.ac.jp/egress-policy	Pod	外向きの通信の際に特定の EgressGateway の利用を強制するポリシーを適用することを示す

表 2　制御に利用するために本システムで独自に付与するラベル

また、egressgateway の EgressClusterPolicy リソースを利用し、`.spec.appliedTo.podSelector` に `edge.kudpc.kyoto-u.ac.jp/egress-policy` ラベルを指定し、かつ`.spec.appliedTo.namespaceSelector` に Capsule が namespace に付与するテナントを示すラベルを指定することで、各テナント内で動作し指定されたラベルを持つ Pod が外部に接続する際の IP アドレスの強制を実現している。

また、LoadBalancer については、MetalLB の L2Advertisement リソースにおいて`.spec.nodeSelectors` に `edge.kudpc.kyoto-u.ac.jp/lb_テナント名` ラベルを指定することで外部からの接続を受け付けるノードを限定し、IPAddressPool を利用し`.spec.serviceAllocation.namespaceSelectors` に Capsule が namespace に付与するテナントを示すラベルを指定することで、各テナントでのみ利用可能な LoadBalancer を実現している。

なお、EgressGateway および LoadBalancer は L2 モードで動作させ、かつ、LoadBalancer および EgressGateway として動作するノードのみ外部との通信に利用する VLAN と接続している。また、コンテナ環境は IP アドレスを大量に利用することが想定され、既存の研究室等のプライベートネットワークで利用されている IP アドレス空間への影響を抑えることが求められると想定されること、および、Kubernetes が有するコンテナのオートスケーリング等の機能を利用しやすくするため、全ての外部との接続は一旦 LoadBalancer および EgressGateway の役割を持つノードで終端する設計となっている。

3.4　利用者の利用方法

利用者は以下の手順により本システムを利用することを想定している。

1. クラスタの管理者またはテナントの管理者が、利用者に割り当てられている本学のスーパーコンピュータシステムの利用者番号をテナントに追加する。
2. 利用者は利用者ポータルから Kubernetes API Server にアクセスするための設定ファイル (kubeconfig ファイル) をダウンロードする。
3. 利用者は kubeconfig ファイルを利用して Kubernetes API Server にアクセスし、Pod オブジェクト等を操作する。これには、広く利用されている kubectl コマンド等が利用できる。
4. 本システムの Kubernetes が、Kubernetes の Control Plane の実装に従い、コンテナの実行等の処理を行う。

4　想定される利用例と制約

4.1　データ転送・収集

大規模なデータの移動には長時間の転送時間を要する。例えば、スーパーコンピュータを利用した大規模なデータ解析のために、外部のオブジェクトストレージ等からスーパーコンピュータのストレージにデータを転送することが考えられる。また、外部のオブジェクトストレージ等に保存されている情報の一部に機微なデータを含む場合、スーパーコンピュータのストレージに格納する前に機微なデータを削除することが想定される。

このような用途においては、rsync[*1]や rclone[*2]等のソフトウェアが利用できる。いずれも広く利用されているソフトウェアであり、コンテナイメージが公開されている。また、これらのソフトウェアを組み合わせたスクリプトを実行して転送する場合においても、既存のコンテナイメージを利用して新たにコンテナイメージを作成することにより、コンテナイメージの作成の手間が軽減されると考えられる。加えて、本システムの演算ノードは 100Gbps で接続されているため、研究室等で有する計算機を利用して転送する場合より高速にデータを転送できることが期待される。

また、実験機器等からの push 型のデータ収集のユースケースにおいても、実験機器が自動的にデータをアップロードするサーバを実行するために利用することができる。例えば、一部のデジタル一眼レフカメラやスキャナ機能を有する複合機は FTP サーバへの自動アップロード機能を有する。これに対し、オブジェクトストレージに自動的にデータをアップロードする機能を持つ FTP サーバを実行することで、大規模なオブジェクトストレージへの自動アップロードを実現できる。このような機能を持つソフトウェア (rclone 等) を実行できるコンテナイメージは広く配布されており、利用者は本システムを利用することでサーバをセットアップすることなくデータの効率的な収集を実現できる可能性がある。

定期的な Web のクローリングのような pull 型のデータ収集についても、本システムがサポートする Kubernetes の CronJob オブジェクトを利用したコンテナの定期実行を利用することにより実現できる。

4.2 データ分析環境・開発環境

データ分析やソフトウェアの開発においては、JupyterHub[*3]や Coder[*4]が利用されている。これらは公式のコンテナイメージが配布されており、利用者はこれを指定してコンテナを実行するだけでこれらのソフトウェアを利用することができる。追加のプラグインが必要な際は、公式のコンテナイメージを利用して独自のコンテナイメージを作成すればよい。

本システム上でこれらのソフトウェアを動作させる利点の一つは、高速なネットワークを介した外部のストレージへのアクセスが可能である点である。頻繁に大きなサイズのデータにアクセスする処理については、本システムを利用することにより処理時間の短縮が期待できる。

4.3 制約

本システムは、Kubernetes の namespace 機能を用いたマルチテナントを実現している。そのため、利用者はクラスタ全体に影響があるような操作を行うことができない。この制約に該当する例として、サービスの自動的な運用を実現するオペレーターパターン[*5]がある。オペレーターパターンは、クラスタ全体に新たな種類のリソース (カスタムリソース) とカスタムリソースを制御するコントローラを追加することを要求する。これを許すとクラスタ全体のマルチテナントの設計に問題が生じる可能性があるため、許可することは難しい。

LoadBalancer や EgressGateway で利用する外部接続の IP アドレスは Kubernetes クラスタ全体で一意である必要がある。本学の学術情報ネットワークから割り当てられた IP アドレスが付与されたネットワークであればこれが保証されているが、それ以外のプライベート IP アドレスが付与されたネットワークについては別途対応が必要である。なお、本システムは IPv6 に対応しており、グローバル IPv6 アドレスが付与されたプライベートネットワークについてはこの制約の影響はない。

5 まとめと今後の課題

本稿では、京都大学学術情報メディアセンターに導入した、Kubernetes における namespace 機能を利用したマルチテナントを活用したコンテナプラットフォームの構成を紹介した。本システムは、利用者が持ち込むコンテナにより悪意のある操作がされた際の影響を低減するためのテナント間の Worker ノードの分離と、研究室の実験機器等との通信を可能にするためのテナント固有のプライベートネットワークの利用を主な特徴として有する。これらを、Kubernetes およびアドオンを組み合わせ、Kubernetes のオブジェクトに付与するラベルやアノテーションを用いて制御することにより実現していることを示した。

今後の課題としては、制約への対処や、アドオンの絞り込みによる運用性の向上、広く利用されている Helm Chart 等を利用したより多くのアプリケーションの実証、性能評価等が挙げられる。

[*1] https://rsync.samba.org/
[*2] https://rclone.org/
[*3] https://jupyter.org/hub
[*4] https://coder.com/
[*5] https://kubernetes.io/ja/docs/concepts/extend-kubernetes/operator/

謝辞

本システムの設計・導入にご尽力いただいたデジタルテクノロジー株式会社および株式会社 ScaleWorX の各位、本学のデータ駆動型研究に対応する環境整備と支援体制の観点から多くの助言をいただいた京都大学学術情報メディアセンター 岡部 寿男 教授および京都大学情報環境機構 渥美 紀寿 教授、本システムのハードウェア構成の仕様検討にご協力いただいた総合地球学研究所 深沢 圭一郎 教授 (元 京都大学学術情報メディアセンター准教授) に謝意を表します。

参考文献

[1] Toyotaro Suzumura, et al. mdx: A Cloud Platform for Supporting Data Science and Cross-Disciplinary Research Collaborations. In *2022 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pp. 1–7, 2022.

[2] Keichi Takahashi, et al. Performance analysis of mdx II: A next-generation cloud platform for cross-disciplinary data science research. In *15th International Conference on Cloud Computing and Servicwes Science (CLOSER 2025)*, 2025.

[3] David Y. Hancock, et al. Jetstream2: Accelerating cloud computing via Jetstream. In *Practice and Experience in Advanced Research Computing 2021: Evolution Across All Dimensions*, PEARC '21. ACM, 2021.

[4] 杉木章義, 棟朝雅晴. 北海道大学ハイパフォーマンスインタークラウドの設計, 構築, 運用まで. 日本オペレーションズ・リサーチ学会機関誌「オペレーションズ・リサーチ」, Vol. 64, No. 9, pp. 507–513, 2019.

[5] Eiji Ikoma and Masaru Kitsuregawa. DIAS—Earth Environment Data Integration and Analysis System. *Commun. ACM*, Vol. 66, No. 7, p. 85–86, June 2023.

[6] Masatoshi Hanai, et al. ARIM-mdx Data System: Towards a Nationwide Data Platform for Materials Science. In *Proceedings of 2024 IEEE International Conference on Big Data (BigData)*, pp. 2326–2333, 2024.

[7] International nucleotide sequence database collaboration. https://www.insdc.org/.

[8] Kubernetes. https://kubernetes.io/.

[9] Christine Kirkpatrick R., et al. Open Storage Network Retrospective the Future of Distributed Storage for eInfrastructure. 2021.

[10] Earth system grid federation. https://esgf.github.io/.

[11] European open science cloud - eu node. https://open-science-cloud.ec.europa.eu/.

[12] 伊達進, 寺前勇希, 勝浦裕貴, 木越信一郎, 木戸善之. Onion：大阪大学のデータ集約基盤. 学術情報処理研究, Vol. 26, No. 1, pp. 87–96, 2022.

[13] Apache mesos. https://mesos.apache.org/.

[14] Abhishek Verma, et al. Large-scale cluster management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15. ACM, 2015.

[15] Proxmox ve. https://www.proxmox.com/en/proxmox-ve.

[16] Rancher. https://rancher.com/.

[17] Capsule. https://projectcapsule.dev/.

[18] Kyverno. https://kyverno.io/.

[19] Cilium. https://cilium.io/.

[20] egressgateway. https://https://spidernet-io.github.io/egressgateway/.

[21] Metallb. https://metallb.io/.