Kubernetesにおける異なるディストリビューションへのリソースの複製によるエラーの原因の分類

近藤 悠斗[1]　平尾 真斗[2]　串田 高幸[1]

概要：Kubernetes は複数のディストリビューションを持つコンテナオーケストレーションフレームワークである．RKE2 に対して WordPress をデプロイするとエラーが発生する．これは Persistent Volume を使用しており，StorageClass が存在しないためエラーになる．エラーが発生した場合，どの Kubernetes リソースがエラーの原因か絞り込む必要がある．課題は，クラスタに Kubernetes リソースをデプロイし，Pod にエラーが発生した際に，リソース同士の依存関係を知らない場合，クラスタに存在するどの Kubernetes リソースに問題があるのか絞り込めないことである．提案では，複数のディストリビューションに Kubernetes リソースを複製し，起動したディストリビューションと起動しなかったディストリビューションで作成されている全 Kubernetes リソースが存在するか存在しないか比べることで，原因となる Kubernetes リソースを分類する手法を提案する．評価実験では，4 種類のディストリビューションにアプリケーションをデプロイし，提案ソフトウェアを使用することによって何種類の Kubernetes リソースに分類できるか実験した．対象は WordPress と NGINX Ingress Controller とし，クラスタは K3s，k0s，MicroK8s，RKE2 を使用した．WordPress を対象とした実験では，103 種類の Kubernetes リソースから原因となる Kubernetes リソースを含む 6 種類を原因として分類した．これによってエラーの原因となるリソースを特定しやすくなった．

1. はじめに

背景

Kubernetes は最も使用されているコンテナオーケストレーションフレームワークである [1–4]．Kubernetes をサーバに展開するとクラスタを構築できる．クラスタにはマスターノードとワーカーノードが含まれる．Kubectl コマンドを使用することでクラスタに Kubernetes リソースを作成できる．Pod は Kubernetes で管理できるコンテナを作成する Kubernetes リソースである．

Kubernetes には複数のディストリビューションがある [5, 6]．ディストリビューションとは Kubernetes をインストールする異なる方法や実装を提供するものである．ディストリビューションによって，インストールの容易さや軽量性，特定の用途への最適化の特徴が異なる．例えばディストリビューションとして K3s，k0s，MicroK8s，RKE2 がある．

[1] 東京工科大学コンピュータサイエンス学部
クラウド・分散システム研究室
〒 192–0982 東京都八王子市片倉町 1404-1
[2] 東京工科大学大学院バイオ・情報メディア研究科コンピュータサイエンス専攻
〒 192–0982 東京都八王子市片倉町 1404-1

K3s は軽量であることと，使いやすいことを特徴とする Kubernetes ディストリビューションであり，小規模な環境はエッジコンピューティングに適している [7, 8]．K3s にはコンテナを実行するために containerd，ネットワークのための Flannel，サービス検出と負荷分散のための Traefik と CoreDNS を含む複数のコアコンポーネントを統合している [9–11]．

k0s は Kubernetes クラスタの展開と管理を簡素化するために設計されたディストリビューションである [12]．k0s は Kubernetes リソースの名前解決のため CoreDNS，クラスターメトリックのための Metrics Server，水平ポッド自動スケーリングを備えている．

MicroK8s も同様に軽量な Kubernetes ディストリビューションであり，特に開発環境や IoT デバイスを含む Kubernetes リソースの限られた環境での使用に適している [13]．MicroK8s はコンテナランタイムとして containerd，DNS として CoreDNS，ネットワーキングとして Calico が含まれている [14]．

RKE2 は Rancher が K3s をベースに開発した軽量な Kubernetes ディストリビューションである [15]．RKE2 にはコンテナランタイムとして containerd，ネットワーキングとして canal，メトリクスサーバとして Metrics Server が

インストールされている．

Kubernetes クラスタに Kubernetes リソースをデプロイする方法として Helm を使用する方法がある [16, 17]．Helm は Kubernetes でのアプリケーションのパッケージマネージャーであり，チャートを使用して Kubernetes にソフトウェアをインストールできる．Helm を使用してインストールできるソフトウェアとして，WordPress や NGINX Ingress Controller がある [18, 19]．

例えば RKE2 クラスタに WordPress をデプロイする．RKE2 クラスタは公式の Quick Start[*1] のページを参照してインストールし，WordPress のインストールには Helm を使用する．この時エラーが発生して WordPress の Pod が Pending のままになってしまう．Pod が起動しなかったとき，管理者は何が原因で Pod が起動しないのか調査する必要がある．

課題

課題は，クラスタに Kubernetes リソースをデプロイし，Pod にエラーが発生した際に，クラスタに存在するどの Kubernetes リソースに問題があるのか絞り込めないことである．例えば RKE2 ではインストール後の状態で 80 種類の Kubernetes リソースが定義されている．この中から原因となる Kubernetes リソースを絞り込む必要がある．

例えば RKE2 に WordPress をデプロイする状況を図 1 に示す．

図 1: RKE2 に WordPress をデプロイする状況

RKE2 クラスタは公式の Quick Start のページを参照してインストールし，WordPress のインストールに Helm を使用すると，エラーが発生して WordPress の Pod が Pending のままになる．これは StorageClass が定義されていないため，WordPress の Pod がが永続的な保存領域である PersistentVolume を確保するための PersistentVolumeClaim(以下 PVC) を作成し紐づけることができなかったため Pending となっている．Pod が起動しない場合には，管理者は解決のため複数の Kubernetes リソースを確認する必要がある．RKE2 に WordPress をインストールして Pending になった時の原因の絞り込み手順として，kubectl describe を使用して WordPress の Pod を確認した出力を図 2 に示す．

```
swipe@c0a21147-swipe:~$ kubectl descri
be pod my-release-wordpress-649bd68c5-
sm88g
Name:             my-release-wordpress
-649bd68c5-sm88g
                  ⋮
reachable:NoExecute op=Exists for 300s
Events:
  Type     Reason            Age
                From                Mess
age
  ----     ------            ----
                ----                ----
---
  Warning  FailedScheduling  3m57s (x9
57 over 3d7h)  default-scheduler  0/1
nodes are available: pod has unbound i
mmediate PersistentVolumeClaims. preem
ption: 0/1 nodes are available: 1 Pree
mption is not helpful for scheduling.
swipe@c0a21147-swipe:~$
```

図 2: kubectl describe を使用して WordPress の Pod を確認した出力

図 2 は 0/1 nodes are available: pod has unbound immediate PersistentVolumeClaims. preemption: 0/1 nodes are available: 1 Preemption is not helpful for scheduling. という部分から Pod が永続的な保存領域である PersistentVolume を確保するための PVC を作成し紐づけることができなかったことを示す．出力からクラスタ内の PVC を取得したところ WordPress の PVC が Pending となっていた．

kubectl describe を使用して WordPress の PVC を確認した出力を図 3 に示す．図 3 は no persistent volumes available for this claim and no storage class is set という部分からこの PVC に StorageClass が設定されていないため，PVC が作成できないことを示している．出力からクラスタ内の StorageClass を取得したところ，何も Kubernetes リソースが作成されていなかった．そのためこの問題を解決するためには StorageClass を設定する必要がある．以上のように，Pod が起動しない場合には，複数の Kubernetes リソースを調査して原因を絞り込む必要がある．この時，管理者は describe の出力からどこにエラーについての記述があるのかを読み，その記述からエラーの原因となる

[*1] https://docs.rke2.io/install/quickstart

```
swipe@c0a21147-swipe:~$ kubectl descri
be pvc my-release-wordpress
Name:          my-release-wordpress
Namespace:     default
StorageClass:
Status:        Pending
Volume:
Labels:        app.kubernetes.io/insta
nce=my-release
               app.kubernetes.io/manag
ed-by=Helm
               app.kubernetes.io/name=
wordpress
               app.kubernetes.io/versi
on=6.7.1
               helm.sh/chart=wordpress
-24.0.8
Annotations:   meta.helm.sh/release-na
me: my-release
               meta.helm.sh/release-na
mespace: default
Finalizers:    [kubernetes.io/pvc-prot
ection]
Capacity:
Access Modes:
VolumeMode:    Filesystem
Used By:       my-release-wordpress-64
9bd68c5-sm88g
Events:
  Type     Reason         Age
             From
    Message
  ----     ------         ----
             ----
    -------
  Normal   FailedBinding  4m57s (x19147
 over 3d7h)  persistentvolume-controll
er  no persistent volumes available fo
r this claim and no storage class is s
et
swipe@c0a21147-swipe:~$
```

図 3: kubectl describe を使用して WordPress の PVC を確認した出力

Kubernetes リソース名を特定し，その Kubernetes リソースに対しても describe を行う．この作業を根本原因となる Kubernetes リソースを特定するまで繰り返す必要がある．そのため管理者は RKE2 に定義された 80 種類の Kubernetes リソースから原因となる StorageClass リソースを絞り込めない．

各章の概要

第 2 章の関連研究では，関連する既存研究を述べる．第 3 章の提案では，課題を解決する提案方式，ユースケース・シナリオの説明をする．第 4 章の実装では，実装方法の説明をする．第 5 章の評価実験では，実験環境，実験結果と分析の説明をする．第 6 章の議論では，提案方式の議論をする．第 7 章のおわりにでは，全体をまとめる．

2. 関連研究

複数の機械学習を使用して Kubernetes の障害の原因を特定する研究がある [20]．この研究では Kubernetes のクラスタから生成されるログをもとにして原因の分類を行っている．一方で実際にはディストリビューションによってインストールされるソフトウェアが異なるため，複数のディストリビューションについて検証する必要がある．

通信のトレースを可視化することによって，Kubernetes の障害の根本原因を支援するツールを提案した研究がある [21]．ツールによって，通信エラーが発生している箇所が示され，どの Pod でエラーが発生しているかが分かる．一方で Kubernetes リソースが作成されていない場合には通信自体が発生しないため，改善の余地がある．

Helm チャートへデジタル署名を行うことによってチャートの整合性を保証する研究がある [16]．この研究によって Helm チャート内にエラーが無いことを保証できる．一方でディストリビューションによってインストールされているソフトウェアが異なるため，正常な Helm チャートをデプロイしてもディストリビューションによってエラーが発生することがある．

複数の Kubernetes ディストリビューションを比較した研究がある [7]．この研究ではクラスタで使用しているソフトウェアや CPU，メモリ性能について比較を行っている．一方で Kubernetes リソースを作成する時の動作の違いについては言及していない．

3. 提案

本稿では Pod が起動しない原因がどの Kubernetes リソースにあるのかを分類することで，原因箇所を特定できるようにすることを目的とする．提案として，インストールに使用した Helm チャートを複数のクラスタに複製し，その結果とクラスタに配置されている Kubernetes リソースから Pod が起動しない原因となる Kubernetes リソースを分類する．

前提としてすべてのクラスタはマスターノード 1 台のみの構成とする．提案の概要を図 4 に示す．

提案は Kubernetes リソースを複数のクラスタに複製する段階と Kubernetes リソースの有無を比べる段階の 2 段階で動作する．Kubernetes リソースを複数のクラスタに複製する段階では，エラーが発生した Kubernetes リソース

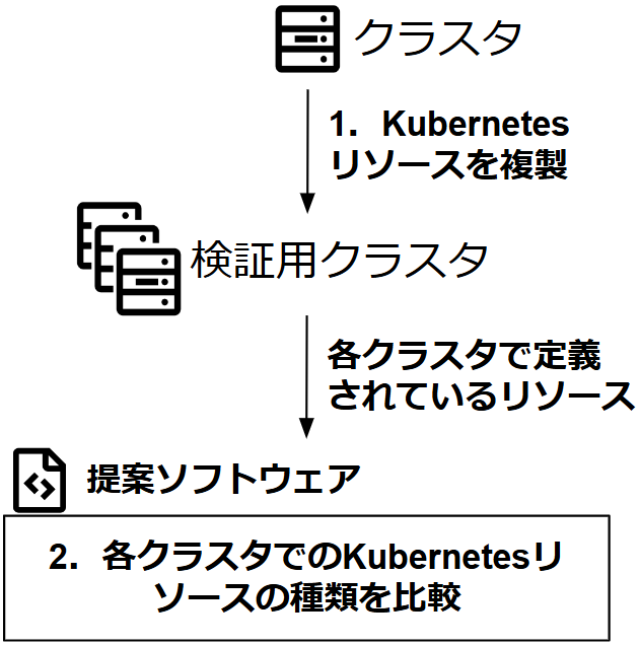

図 4: 提案の概要

を複数のクラスタに複製する．この時，いずれかのクラスタのみで Pod がエラーになった場合は Kubernetes リソースの有無を比べる段階に移行する．Kubernetes リソースの有無を比べる段階では複数のクラスタから定義されているすべての Kubernetes リソースを取得しリソースが存在するか存在しないか比べる．

提案方式

Kubernetes リソースを複数のクラスタに複製する段階を図 5 に示す．図 5 のクラスタは管理者がデプロイのために使用しているクラスタを指す．管理者は Helm チャートでエラーが発生したときに，コマンドラインで提案ソフトウェアを呼び出す．呼び出された提案ソフトウェアはコマンド引数から対象の Kubernetes リソースを検証用クラスタに作成する．ここでは例として K3s，k0s，MicroK8s，RKE2 の 4 つのクラスタを使用している．図 5 では K3s，k0s，MicroK8s，RKE2 がそれぞれのディストリビューションで作成されたクラスタを指す．その後，提案ソフトウェアは Status が Running になるかを確認する．

全ての検証用クラスタで Pod が Running になった場合の出力を出力 1 に示す．全ての検証用クラスタで Pod が Running になった場合，アプリケーションには問題がないと判断し，標準出力でサーバに問題があることを管理者に通知する．

全ての検証用クラスタで Pod が Running にならなかった場合の出力を出力 2 に示す．全ての検証用クラスタで Pod が Running にならなかった場合，アプリケーションに問題があると判断し，標準出力でアプリケーションに問題があることを管理者に通知する．

いずれかの検証用クラスタで Pod が Running になった場合は Kubernetes リソースの有無を比べる段階に移行する．図 5 では WordPress が各クラスタに作成され K3s でのみ起動している．Kubernetes リソースの有無を比べる段階ではそれぞれのクラスタで作成されている Kubernetes リソースをすべて取得し，Kubernetes リソースの種類ごとにまとめる．その後は Kubernetes リソースの種類ごとにリソースが存在するか存在しないか比べる．Kubernetes リソースを種類ごとの数で比べた例を表 1 に示す．

表 1: Kubernetes リソースを種類ごとの数で比べた例

リソース名	**K3s**	k0s	MicroK8s	RKE2
cronjobs.batch	**0**	0	0	0
configmaps	**12**	13	10	19
ingressroutes .traefik.io	**1**	0	0	0
events	**0**	1	1	1

表 1 のリソース名はクラスタに定義された Kubernetes リソースを示す．表では例として 4 つの Kubernetes リソースのみを示している．まず Pod が起動したクラスタのグループと起動しなかったクラスタのグループに分ける．図

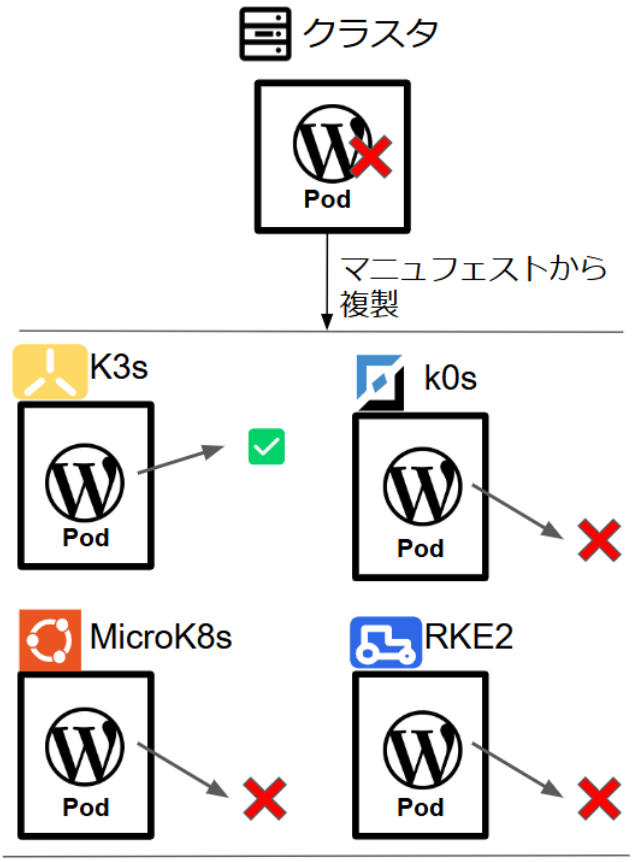

図 5: Kubernetes リソースを複数のクラスタに複製する段階

出力 1: 全ての検証用クラスタで Pod が Running になった場合の出力

```
1 サーバやネットワークに問題があります.
```

出力 2: 全ての検証用クラスタで Pod が Running にならなかった場合の出力

```
1 applyしたアプリケーションに問題があります.
```

5 の場合，K3s のみ Pod が起動したグループに入る．次に，ある Kubernetes リソースについて Pod が起動したクラスタのグループに Kubernetes リソースが 1 つ以上あった場合，起動しなかったグループの同じ Kubernetes リソースを確認し，その Kubernetes リソースを 1 つもない場合，特徴としてその Kubernetes リソースの種類を記録する．表 1 の場合 ingressroutes.traefik.io の Kubernetes リソースが K3s に 1 つあり，K3s 以外の 3 つのクラスタでは 0 個である．そのため ingressroutes.traefik.io を記録しておく．

逆にある Kubernetes リソースについて Pod が起動したクラスタのグループに Kubernetes リソースが 1 つも無い場合，起動しなかったグループの同じ Kubernetes リソースを確認し，その Kubernetes リソースが 1 つ以上あった場合，特徴としてその Kubernetes リソースの種類を記録する．表では events の Kubernetes リソースが K3s にはなく，K3s 以外の 3 つのクラスタではそれぞれ 1 つ以上ある．そのため events を記録しておく．

残り 2 つの cronjobs.batch と configmaps は上記の 2 つの条件に満たないため記録しない．最後に記録した Kubernetes リソースの種類を原因範囲として管理者に通知する．表の例では ingressroutes.traefik.io と events が管理者に通知されることになる．提案ソフトウェアからの出力例を図 6 に示す．ingressroutes.traefik.io の部分は ingressroutes.traefik.io に差分があったことを示す．起動したクラスタ以下の部分は起動したクラスタで ingressroutes.traefik.io の Kubernetes リソースを取得した結果を表す．使用中のクラスタ以下の部分は管理者が使用しているクラスタでの ingressroutes.traefik.io の Kubernetes リソースを取得した結果を示す．events 以降は events に対して同様の形式で出力をしている．

```
Podが起動しない原因の可能性があるリソース：
ingressroutes.traefik.io
| 起動したクラスタ
| NAMESPACE     NAME                AGE
| kube-system   traefik-dashboard   26d
|
| 使用中のクラスタ
| error: the server doesn't have a resource type "ingressroutes"

Podが起動しない原因の可能性があるリソース：
events
| 起動したクラスタ
| No resources found
|
| 使用中のクラスタ
| LAST SEEN   TYPE      REASON
    OBJECT
                MESSAGE
| 4m53s       Normal    FailedBinding
    persistentvolumeclaim/data-my-release-mariadb-0   no persistent volumes available for this claim and no storage class is set
| 38s         Normal    FailedBinding
    persistentvolumeclaim/data-wordpress-mariadb-0    no persistent volumes available for this claim and no storage class is set
```

図 6: 提案ソフトウェアからの出力例

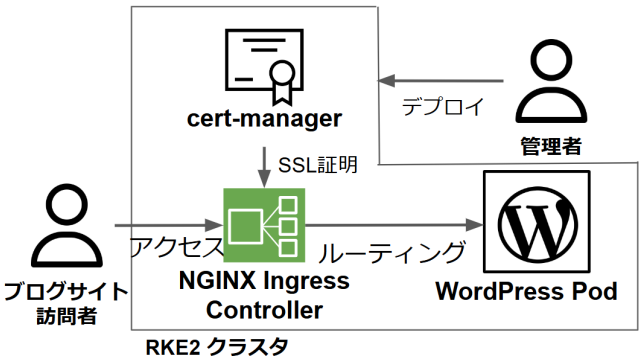

図 7: Kubernetes クラスタに WordPress をインストールする状況

ユースケース・シナリオ

ユースケースとして管理者が Kubernetes クラスタに WordPress をインストールする状況を想定する．その状況を図 7 に示す．管理者は自身のブログサイトを WordPress を使用して作成した．個人的なブログとしての使用を考えており，想定するアクセス数は 1 日 2 件ほどだが，アクセス数が増えた際にスケーリングできるように Kubernetes 上にデプロイする．使用するディストリビューションは RKE2 を想定する．サイトは Google 検索に結果として表示されるように構成するため SSL 化する．そのために管理

者は cert-manager と NGINX Ingress Controller をクラスタ上にデプロイする．

ここで，WordPress を RKE2 にデプロイするタイミングでエラーが発生した．提案ソフトウェアを実行することで，RKE2 が原因でエラーが発生している状況が分かる．また原因範囲として提案ソフトウェアが Kubernetes リソースを切り分けるため，特定の種類の Kubernetes リソースのみを調査して修正を行うことができる．

4. 実装

提案ソフトウェアの実装を図 8 に示す．

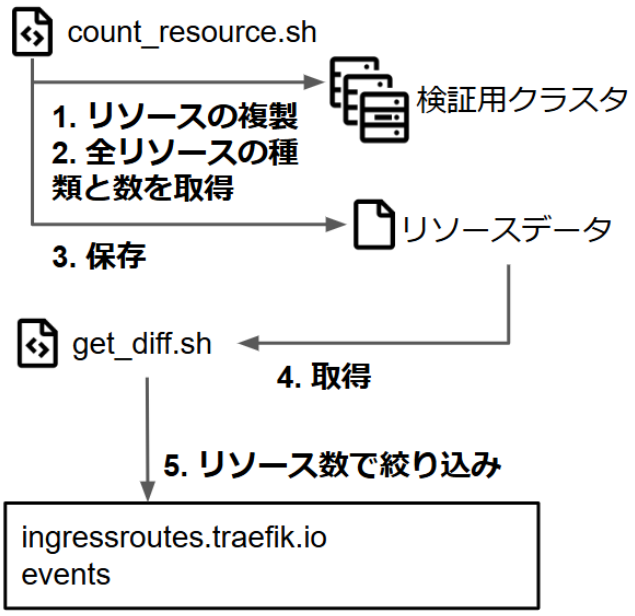

図 8: 提案ソフトウェアの実装

提案ソフトウェアはシェルスクリプトで実装されており，複製ソフトウェアと比較ソフトウェアの 2 つで構成される．count_resource.sh は複製ソフトウェアでコマンドライン入力で呼び出されると，helm install を使用して Kubernetes リソースの複製を行い，どのクラスタで起動したかを確認する．次に kubectl api-resources を使用してそのクラスタに定義されている Kubernetes リソースの種類をすべて取得する．そして kubectl get コマンドを使用してすべての Kubernetes リソースの種類で，Kubernetes リソースがいくつ作成されているかを調べる．これをすべてのクラスタに対して行い，その結果をリソースデータファイルにテキスト形式で記録する．記録後は get_diff.sh を呼び出す．

get_diff.sh は比較ソフトウェアで，count_resource.sh で収集した Kubernetes リソースの種類と数をもとにクラスタごとにリソースが存在するか存在しないか比べる．get_diff.sh ははじめにリソースデータファイルからどのクラスタで Pod が起動したかの情報と各 Kubernetes リソースの数を取得する．次に取得した Kubernetes リソースを種類ごとに分け，1 つの種類の Kubernetes リソースに対して各クラスタでいくつ存在するか比べる．起動したクラスタと起動しなかったクラスタで Kubernetes リソースが 1 つ以上存在するか 1 つも存在しないかで違いがでた Kubernetes リソースの種類を記録する．これをすべての Kubernetes リソースに対して行う．最後に記録した Kubernetes リソースの種類を標準出力で管理者に返答する．

5. 評価実験

評価実験は VM に構築した提案ソフトウェアと 4 種類のディストリビューションで構築したクラスタを使用する．提案ソフトウェアを使用して Pod が起動しない原因となる Kubernetes リソースの種類の分類を行う．エラーの原因として分類した Kubernetes リソースの件数と，分類した中に原因となる Kubernetes リソースの種類が含まれるかを評価とする．

実験環境

実験環境を図 9 に示す．

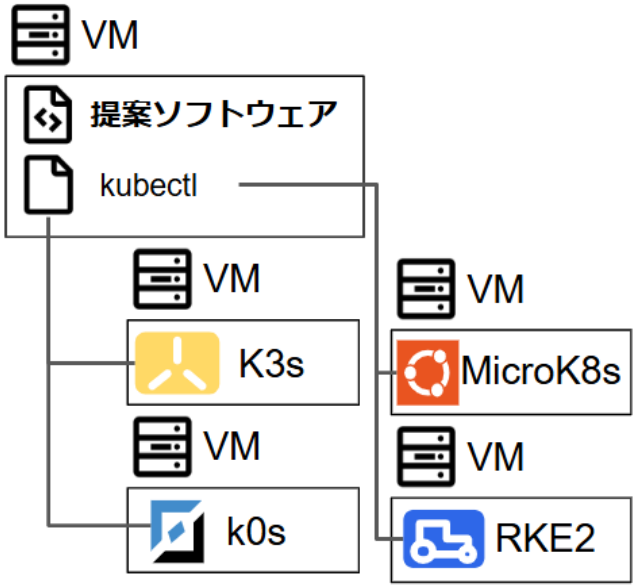

図 9: 実験環境

実験環境として 5 台の VM を使用する．4 台はそれぞれ Kubernetes クラスタを構築する．CPU は 2 コア，メモリは 4GB，OS は Ubuntu22.04 を使用する．それぞれのクラスタには 1 つずつクラスタがインストールされる．それぞれのバージョンは K3s は v1.30.6+k3s1，k0s は v1.31.2+k0s，MicroK8s は v1.31.2，RKE2 は v1.30.6+rke2r1 である．

1 台の VM には提案ソフトウェアをインストールする．CPU は 2 コア，メモリは 4GB，OS は Ubuntu22.04 である．この VM には kubectl コマンドと，提案ソフトウェアを導入する．kubectl のバージョンは v1.30.5+k3s1 を使用する．

実験結果と分析

1 つ目の実験はアプリケーションとして WordPress をインストールする．この場合原因となる Kubernetes リソースは StorageClass が定義されていないことが原因であり，提案ソフトウェアによってエラーの原因として分類した結果の件数と，分類した中に StorageClass が含まれることが評価となる．WordPress を対象とした評価実験の結果を図 10 に示す．

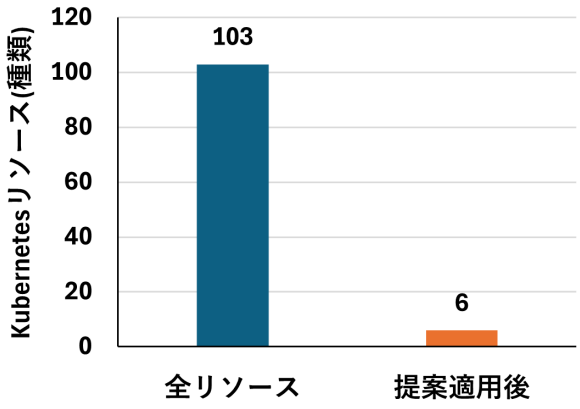

図 10: WordPress を対象とした評価実験の結果

WordPress に対して提案を適用した結果，全 103 種類 Kubernetes リソースのうち 6 種類の Kubernetes リソースをエラーの原因として分類した．

エラーの原因として分類された 6 種類の Kubernetes リソースの詳細を表 3 に示す．WordPress をインストールした結果，K3s のみ起動したため，K3s だけが異なっている Kubernetes リソースのみが出力されている．WordPress が K3s だけで起動する理由は K3s だけクラスタ構築時に自動的 Local Path の StorageClass が設定されているためである．エラーとして分類された Kubernetes リソースには storageclasses.storage.k8s.io が含まれているため分類は成功していた．

提案ソフトウェアからの出力を図 11 に示す．出力には StorageClass のリソースについて起動したクラスタと使用中のクラスタでリソースを取得した結果が示されており，出力の使用中のクラスタの部分には No resources found と表示されているため，使用中のクラスタには StrageClass が存在しないことが分かる．

2 つ目の実験はアプリケーションとして NGINX Ingress Controller をインストールする．この場合原因となる Kubernetes リソースは IngressClass にすでに NGINX が定義されていることが原因であり，提案ソフトウェアによってエラーの原因として分類した結果の件数と，分類した中に IngressClass が含まれることが評価となる．次に NGINX Ingress Controller を対象とした評価実験の結果を図 12 に示す．NGINX Ingress Controller のエラーに対して提案を

表 2: エラーの原因として分類された 6 種類の Kubernetes リソースの詳細

リソース名	K3s	k0s	MicroK8s	RKE2
events	0	1	1	1
events.events.k8s.io	0	1	1	1
ingressroutes .traefik.io	1	0	0	0
persistentvolumeclaims	0	1	1	1
runtimeclasses.node.k8s.io	10	0	0	0
storageclasses .storage.k8s.io	1	0	0	0

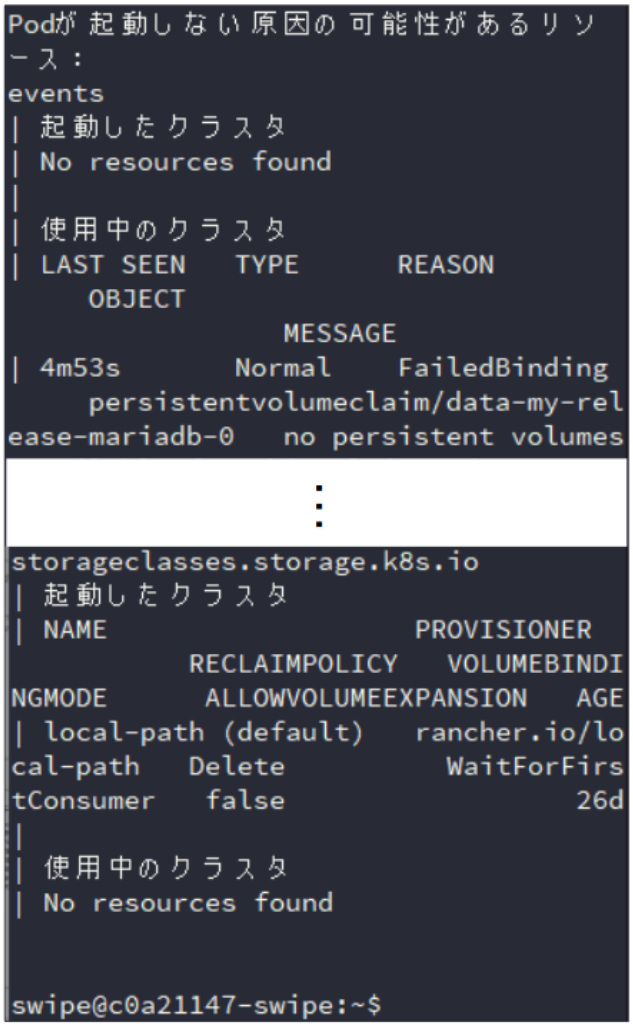

図 11: 提案ソフトウェアからの出力

適用した結果，全 103 種類 Kubernetes リソースのうち 2 種類の Kubernetes リソースをエラーとして分類した．

エラーとして分類された 2 件の Kubernetes リソースの詳細を表 3 に示す．NGINX Ingress Controller をインストールした結果，RKE2 のみ起動しなかったため，RKE2 だけが異なっている Kubernetes リソースのみが出力されている．NGINX Ingress Controller が RKE2 だけで起動する理由は，RKE2 だけクラスタ構築時に NGINX Ingress

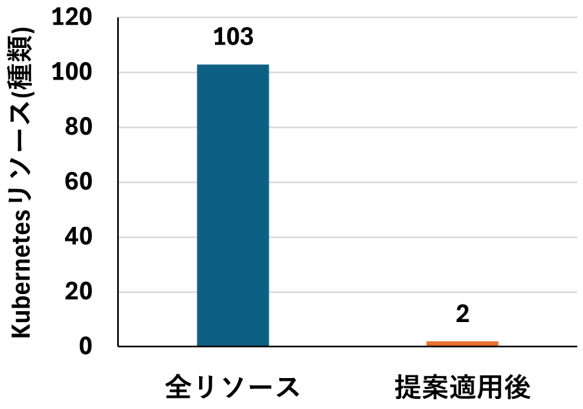

図 12: NGINX Ingress Controller を対象とした評価実験の結果

Controller がインストールされる設定になっており IngressClass がすでに設定されているため重複でエラーが出ているからである．エラーとして分類された Kubernetes リソースには IngressClass に関する Kubernetes リソースが含まれないため，分類は失敗していた．

表 3: エラーとして分類された 2 件の Kubernetes リソースの詳細

リソース名	K3s	k0s	MicroK8s	RKE2
etcdsnapshotfiles .k3s.cattle.io	0	0	0	5
validatingwebhook configurations .admissionregistration.k8s.io	0	0	0	2

原因は，K3s にも Traefik の IngressClass が作成されているにもかかわらず K3s では NGINX Ingress Controller の Pod が Running になるためである．しかし実際には K3s では NGINX Ingress の Ingress リソースを作成した際に Traefik と重複するため正常には動作しない．

6. 議論

本稿では評価実験で WordPress に対して 6 種類まで Kubernetes リソースを分類した．しかし問題となる Kubernetes リソースを明確にするためには 1 種類まで分類を行い絞り込む必要がある．これに対して，Helm チャートの内容から作成される Kubernetes リソースを取得し，その Kubernetes リソースの ownerReferences の項目から Kubernetes リソースが依存している Kubernetes リソースを取得する．これによって作成した Kubernetes リソースに関連する Kubernetes リソースだけが分類の対象になり，分類の結果の種類を 6 種類よりも減らすことができる．

本稿では検証用クラスタとして，4 種類のディストリビューションを使用した．しかし実際には K3s，k0s，MicroK8s，RKE2 以外のディストリビューションもある．これに対して，ディストリビューションの種類増やすことでさらに Kubernetes リソースを分類することができる．

本稿ではすべてのディストリビューションでマスターノード 1 台の構成とした．しかしノードの数によって Pod が作成できない問題が発生することもある．例えばノード間で通信が発生する場合である．これに対して，各ディストリビューションで使用するノードの数を変動させ複数回検証を行う方法がある．これによってノード数による問題にも適用することができる．

本稿では評価実験で NGINX Ingress Controller に対しては提案が適用できなかった．原因は，K3s にも Traefik の IngressClass が作成されているにもかかわらず K3s では NGINX Ingress Controller の Pod が Running になるためである．しかし実際には K3s では NGINX Ingress の Ingress リソースを作成した際に正常に動作しない．本稿では Pod が Running になった時を動いた時と定義したが，この問題に対して Kubernetes リソースを作成できるだけではなく，Ingress リソースを作成することで使用できるかまでを検証し，Ingress リソースが動作した時を Pod が動いた時とすることで K3s でも正常に起動していないと判定することができる．

本稿では Pod にエラーが発生したときにエラーの原因となる Kubernetes リソースを絞り込むことができないことを課題とした．しかし，実際にはデバッグする方法は人によって異なる．例えば describe コマンドの出力結果を参照して調べる方法がある．この方法の場合，出力内で提示された Kubernetes リソースから調査を行うため，評価で示した全 103 件の Kubernetes リソースを調査することにはならない．そのため提案がエラーの解決につながるか，追加実験を行う必要がある．具体的には，提案ソフトウェアを使用しない場合と，使用した場合でエラーの原因箇所の特定する実験を行い，エラーの特定にかかる時間とコマンドの入力回数を評価する必要がある．

7. おわりに

Kubernetes は複数のディストリビューションを持つコンテナオーケストレーションフレームワークである．RKE2 に対して WordPress をデプロイするとエラーが発生する．これは Persistent Volume を使用しており，StorageClass が存在しないためエラーになる．エラーが発生した場合，どの Kubernetes リソースがエラーの原因か絞り込む必要がある．課題は，クラスタに Kubernetes リソースをデプロイし，Pod にエラーが発生した際に，リソース同士の依存関係を知らない場合，クラスタに存在するどの Kubernetes リソースに問題があるのか絞り込めないことである．提案では，複数のディストリビューション間で Kubernetes リ

ソースを複製し，起動したディストリビューションと起動しなかったディストリビューションで作成されている全 Kubernetes リソースが存在するか存在しないか比べることで，原因となる Kubernetes リソースを分類する手法を提案する．評価実験では，4 種類のディストリビューションにアプリケーションをデプロイし，提案ソフトウェアを使用することによって何種類の Kubernetes リソースに分類できるか実験した．対象は WordPress と NGINX Ingress Controller とし，クラスタは K3s，k0s，MicroK8s，RKE2 を使用した．WordPress を対象とした実験では，103 種類の Kubernetes リソースから原因となる Kubernetes リソースを含む 6 種類を原因として分類した．これによってエラーの原因となるリソースを特定しやすくなった．

参考文献

[1] Balla, D., Simon, C. and Maliosz, M.: Adaptive scaling of Kubernetes pods, *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–5 (online), DOI: 10.1109/NOMS47738.2020.9110428 (2020).

[2] Pahl, C., Brogi, A., Soldani, J. and Jamshidi, P.: Cloud Container Technologies: A State-of-the-Art Review, *IEEE Transactions on Cloud Computing*, Vol. 7, No. 3, pp. 677–692 (online), DOI: 10.1109/TCC.2017.2702586 (2019).

[3] Bernstein, D.: Containers and Cloud: From LXC to Docker to Kubernetes, *IEEE Cloud Computing*, Vol. 1, No. 3, pp. 81–84 (online), DOI: 10.1109/MCC.2014.51 (2014).

[4] Pahl, C., Brogi, A., Soldani, J. and Jamshidi, P.: Cloud container technologies: a state-of-the-art review, *IEEE Transactions on Cloud Computing*, Vol. 7, No. 3, pp. 677–692 (2017).

[5] Aqasizade, H., Ataie, E. and Bastam, M.: Kubernetes in Action: Exploring the Performance of Kubernetes Distributions in the Cloud, *arXiv preprint arXiv:2403.01429* (2024).

[6] Ascensão, P., Neto, L., Velasquez, K. and Perez Abreu, D.: Assessing Kubernetes Distributions: A Comparative Study, pp. 832–837 (online), DOI: 10.1109/MELECON56669.2024.10608706 (2024).

[7] Koziolek, H. and Eskandani, N.: Lightweight Kubernetes Distributions: A Performance Comparison of MicroK8s, k3s, k0s, and Microshift, *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*, ICPE '23, New York, NY, USA, Association for Computing Machinery, p. 17–29 (online), DOI: 10.1145/3578244.3583737 (2023).

[8] Kjorveziroski, V. and Filiposka, S.: Kubernetes distributions for the edge: serverless performance evaluation, *J. Supercomput.*, Vol. 78, No. 11, p. 13728–13755 (online), DOI: 10.1007/s11227-022-04430-6 (2022).

[9] Yoon, Y., Kim, M., Bae, J.-H. and Lee, J.: Performance Comparison Study of Containerd and CRI-O in Kubernetes Environment, *The Journal of Korean Institute of Information Technology*, (online), available from ⟨https://api.semanticscholar.org/CorpusID:271079244⟩ (2024).

[10] Liu, H., Luo, Y., Chen, B. and Yang, Y.: Performance Evaluation of Container Networking Technology, *2023 IEEE 3rd International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA)*, Vol. 3, pp. 815–818 (online), available from ⟨https://api.semanticscholar.org/CorpusID:259363224⟩ (2023).

[11] Teppan, H., Flå, L. H. and Jaatun, M. G.: A Survey on Infrastructure-as-Code Solutions for Cloud Development, *2022 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 60–65 (online), DOI: 10.1109/CloudCom55334.2022.00019 (2022).

[12] Ascensão, P., Neto, L. F., Velasquez, K. and Abreu, D. P.: Assessing Kubernetes Distributions: A Comparative Study, *2024 IEEE 22nd Mediterranean Electrotechnical Conference (MELECON)*, pp. 832–837 (online), DOI: 10.1109/MELECON56669.2024.10608706 (2024).

[13] Vanichchanunt, P., Yamyuan, I., Sasithong, P., Wuttisittikulkij, L. and Paripurana, S.: Implementation of Edge Servers on an Open 5G Core Network, *2023 International Conference on Information Networking (ICOIN)*, pp. 642–645 (online), DOI: 10.1109/ICOIN56518.2023.10049000 (2023).

[14] Ganguli, M., Ranganath, S., Ravisundar, S., Layek, A., Ilangovan, D. and Verplanke, E.: Challenges and Opportunities in Performance Benchmarking of Service Mesh for the Edge, *2021 IEEE International Conference on Edge Computing (EDGE)*, pp. 78–85 (online), DOI: 10.1109/EDGE53862.2021.00020 (2021).

[15] Bryant, L., Gardner, R. W., Hu, F., Jordan, D. and Taylor, R. P.: Kubernetes Deployment Options for On-Prem Clusters (2024).

[16] Gajananan, K., Kitahara, H., Kudo, R. and Watanabe, Y.: Scalable Runtime Integrity Protection for Helm Based Applications on Kubernetes Cluster, *2021 IEEE International Conference on Big Data (Big Data)*, pp. 2362–2371 (online), DOI: 10.1109/BigData52589.2021.9671944 (2021).

[17] Donca, I.-C. and Miclea, L.-C.: Automated Detection and Management of Deprecated Helm Releases in Kubernetes Clusters, *2023 International Conference on Advanced Scientific Computing (ICASC)*, pp. 1–5 (online), DOI: 10.1109/ICASC58845.2023.10328029 (2023).

[18] Kumar, A., Kumar, A., Hashmi, H. and Khan, S. A.: WordPress: A Multi-Functional Content Management System, *2021 10th International Conference on System Modeling Advancement in Research Trends (SMART)*, pp. 158–161 (online), DOI: 10.1109/SMART52563.2021.9675311 (2021).

[19] Rathi, G., Amin, S., Jain, S., Yachawad, M. and Digholkar, K.: Performance Analysis of Different Ingress Controllers Within the Kubernetes Cluster, *2024 IEEE International Conference on Information Technology, Electronics and Intelligent Communication Systems (ICITEICS)*, pp. 1–6 (online), DOI: 10.1109/ICITEICS61368.2024.10625280 (2024).

[20] Sarika, P. K., Badampudi, D., Josyula, S. P. and Usman, M.: Automating Microservices Test Failure Analysis using Kubernetes Cluster Logs, *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, EASE '23, New York, NY, USA, Association for Computing Machinery, p. 192–195 (online), DOI: 10.1145/3593434.3593472 (2023).

[21] Zhou, X., Peng, X., Xie, T., Sun, J., Ji, C., Li, W. and Ding, D.: Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study, *IEEE Transactions on Software En-*

gineering, Vol. 47, No. 2, pp. 243–260 (online), DOI: 10.1109/TSE.2018.2887384 (2021).

