YAMLファイルを比較して生成したルールによるKubernetes Podのエラー原因の提示

田中 美帆[1]　小山 智之[2]　串田 高幸[1]

概要：Kubernetes 上でアプリケーションを動作させるには，YAML 形式の設定ファイルの記述が必要である．YAML 形式の設定ファイルの記述に誤りがあると，エラーが発生して Pod の作成に失敗する．課題は Pod の起動時にエラーが発生した場合に，エラーの原因箇所の特定ができないことである．提案方式では，YAML 形式の設定ファイルにあるエラーの原因箇所を，ルールファイル内に書き込まれているルールを使用し特定する．ルールには kubectl get pod コマンドに含まれるステータス，kubectl describe コマンドに含まれるイベント，起動に成功した場合と起動に失敗した場合の YAML 形式の設定ファイルの差分が含まれる．エラーの原因箇所の特定では，YAML 形式の設定ファイルとルールを比較する．比較した結果に該当するルールがある場合，そのルールをエラーの原因箇所と判定する．比較した結果に該当するルールがない場合，kubectl describe コマンドの結果の Events に含まれる単語を最も多く含むルールをルールファイルから求め，部分一致した結果として出力する．実験にはプロジェクト実習 II[IT・3] で集めた YAML 形式の設定ファイルのうち，kind が Pod の 27 個のファイルを使用した．27 個のファイルを 20 個のルール用 YAML ファイル (約 75%) と，7 個の評価用 YAML ファイル (約 25%) に分けた．評価では，ルール用 YAML ファイルからルールを作成し，そのルールを使い評価用 YAML ファイルのエラーの原因箇所を特定し検知率を算出する．結果として，実験を行った 7 個の評価用 YAML ファイルのうち，3 個のファイルでルールからエラー原因箇所が特定された．残りの 4 個のファイルではルールからエラー原因箇所が特定されなかった．4 個のファイルを対象とした原因の部分一致では，合計で 16 箇所の原因箇所を検知された．このうち正しく検知された箇所は 4 箇所で、誤検知は 12 箇所であった．

1. はじめに

背景

Kubernetes を使用してアプリケーションをコンテナで起動したいときには，Pod の起動が必要である．Kubernetes とは，オープンソースのコンテナオーケストレーションシステムであり，コンテナ化されたアプリケーションの管理と拡張を自動化できる [1–5]．Pod とは，Kubernetes でコンテナを管理するための最小単位であり，アプリケーションのインスタンスを表すものである [6–10]．Pod の作成には YAML 形式の設定ファイルが必要である．Pod が起動していないとアプリケーションを使うことができない．アプリケーションを作成するために，Pod を作成することはできるが起動しないことがある．Pod が起動しているときは Pod のステータスが Running になる．ステータスが Running 以外のときは Pod が起動していない．Pod が起動していない原因を特定するために kubectl コマンドを使用し，作成することができた Pod の状態を確認する作業がある．kubectl コマンドは，Deployment や Pod をはじめとする Kubernetes のリソースを変更するために使用されるコマンドであり，Kubernetes で情報を取得したり構成を変更したりするために使われる [11–15]．kubectl コマンドで Pod の状態を確認した後に，Pod の情報を書き込んでいる YAML 形式の設定ファイルを書き直す作業がある．エラーが起きてから解決するまでの作業を図 1 に示す．はじめに，YAML 形式の設定ファイルを kubectl apply コマンドで適用し Pod を作成するが，Pod のステータスが Pending であり起動していない．次に，Pod が起動していない原因を特定するためにコマンドを実行する．Pod の詳細情報を確認するコマンドとして kubectl describe コマンドがある．図 2 に kubectl describe コマンドの実行結果の例を示す．kubectl describe コマンドを実行し表示された Pod の詳細情報の例として，Pod の名前として pod-pvc-test，Pod が作成されている namespace として test，優先度の Priority

[1] 東京工科大学コンピュータサイエンス学部
〒 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

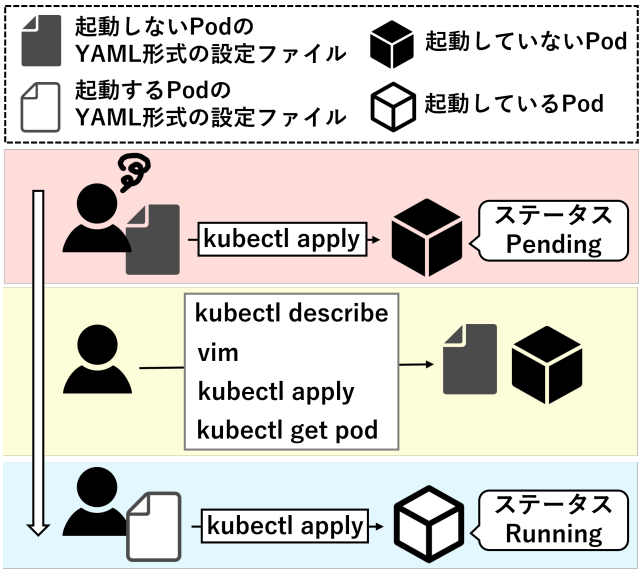

図 1 エラーが起きてから解決するまでの作業

として 0，Pod がスケジュールされているノードがないため <none>，Pod に含まれるコンテナの情報が Containers 以下のインデントが異なる部分，Pod のイベントログの Events 以下の部分がある．Pod のイベントログが Events で表示されている内容である．YAML 形式の設定ファイルを編集するコマンドとして vim コマンドがある．修正した YAML 形式の設定ファイルを Pod に適用するためのコマンドとして kubectl apply コマンドがある．Pod のステータスを確認するコマンドとして kubectl get pod コマンドがある．kubectl get pod コマンドを実行したとき，Pod のステータスが Running になっていると Pod が起動している．Pod のステータスが Running になるまで行う．

kubectl describe コマンドの実行結果には Events とよばれる項目がある．Pod に代表されるリソースの作成でエラーが発生した場合には，Events にエラーメッセージとその原因が出力される．図 3 に Events にエラー内容が表示されている例を示す．この Events では”c0a21151”という名前の persistentvolumeclaim が見つからないというエラー内容であるため，エラーの原因箇所は persistentvolumeclaim を指定している値である．エラー内容が分かる場合は，YAML 形式の設定ファイルで指定している persistentvolumeclaim を確認し直せばよいことが分かる．

課題

課題は Pod の作成は成功するが，アプリケーションの起動に失敗しエラーが発生した場合に，エラー原因の特定がしにくいことである．特定できない原因は，kubectl describe コマンドの結果に含まれる Events に原因を示すメッセージが出力されないためである．特に Kubernetes の初学者は，kubectl describe コマンドの出力内容にどのような情報が書かれており，その内容のどこにエラー原因に関する情報

```
Name:             pod-pvc-test
Namespace:        test
Priority:         0
Service Account:  default
Node:             <none>
Labels:           <none>
Annotations:      <none>
Status:           Pending
IP:
IPs:              <none>
Containers:
  local-path-test:
    Image:        nginx-:stable-alpine
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:
      /data from local-path-pvc (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from
kube-api-access-5r4jh (ro)
Conditions:
  Type          Status
  PodScheduled  False
Volumes:
  local-path-pvc:
    Type:       PersistentVolumeClaim (a reference to a
PersistentVolumeClaim in the same namespace)
    ClaimName:  c0a220106d-data1
    ReadOnly:   false
  kube-api-access-5r4jh:
    Type:                    Projected (a volume that contains
injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-
ready:NoExecute op=Exists for 300s

node.kubernetes.io/unreachable:NoExecute op=Exists for
300s
Events:
  Type     Reason            Age               From
Message
  ----     ------            ----              ----
-------
  Warning  FailedScheduling  21m (x241 over 20h)  default-
scheduler  0/1 nodes are available: persistentvolumeclaim
"c0a220106d-data1" not found. preemption: 0/1 nodes are
available: 1 Preemption is not helpful for scheduling.
```

図 2 kubectl describe コマンドの実行結果の例

```
  Warning  FailedScheduling  9m5s (x8 ov
er 44m)  default-scheduler  0/2 nodes ar
e available: persistentvolumeclaim "c0a2
1151" not found. preemption: 0/2 nodes a
re available: 2 Preemption is not helpfu
l for scheduling.
```

図 3 Events にエラー内容が表示されている例

があるのか分からない．そのためエラー原因を特定するのに苦労する．図 4 に Events にエラー内容が表示されない例を示す．この Events の 2 行目の「Warning BackOff 86s (x20 over 5m57s) kubelet Back-off restarting failed con-

tainer cassandra in pod cassandra-0_c0a21151(535d487d-514f-423e-8a1f-78abba8bb9d7)」では，Pod 内のコンテナが失敗して再起動を試みる際に，一定時間待機することの Back-off が適用されるというエラー内容である．この結果に Pod の起動に失敗する理由が含まれない．そのため，YAML 形式の設定ファイルのどこにエラーの原因箇所があるのか分からない．この YAML 形式の設定ファイルのエラーの原因箇所をソースコード 1 に示す．ソースコード 1 の 7 行目にエラー原因がある．エラーの原因箇所は，YAML 形式の設定ファイルの環境変数を設定している env セクションの Cassandra クラスタのシードノードを設定している値にある．7 行目の値が”cassandra0.cassandra.c0a22099b5.svc.cluster.local”ではなく，”cassandra-0.cassandra.c0a22099b5.svc.cluster.local”だとエラーにならない．この 2 つの値の異なる部分は，1 番始めの部分のハイフンの有無である．図 4 の Events にはエラー内容が含まれず，どこを直すのか分からない．

各章の概要

第 2 章以降の概要は以下の通りとなる．第 2 章では本稿の関連研究について述べる．第 3 章では本稿の課題を解決するための提案方式について述べる．第 4 章では提案方式をもとに作成したソフトウェアの実装について述べる．第 5 章では実験と提案方式についての評価を述べる．第 6 章では提案方式についての議論を述べる．第 7 章では本稿のまとめを述べる．

```
1  Normal   Pulled     4m29s (x2 over 5m14s)  kubelet            Successfully pulled image "gcr.io/google-samples/cassandra:v13" in 326ms (326ms including waiting). Image size: 106662458 bytes.
2  Warning  BackOff    86s (x20 over 5m57s)   kubelet            Back-off restarting failed container cassandra in pod cassandra-0_c0a21151(535d487b-514f-423e-8a1f-78abba8bb9d7)
```

図 4　Events にエラー内容が表示されない例

ソースコード 1　YAML 形式の設定ファイルのエラーの原因箇所

```
1  env:
2    - name: MAX_HEAP_SIZE
3      value: 512M
4    - name: HEAP_NEWSIZE
5      value: 100M
6    - name: CASSANDRA_SEEDS
7      value: "cassandra0.cassandra.
       c0a22099b5.svc.cluster.local"
```

2. 関連研究

アセンブリ言語で書かれたプログラムのエラーを検出するツール作成の論文がある [16]．この論文で作成されたツールのエラー検出対象はアセンブリ言語であり YAML 形式は対象ではない．セマンティックベースの言語でのコードクローンによるバグの検出についての論文がある [17]．この論文では C，Java，Python の 3 つの言語を検証対象にしており YAML 形式は検証対象に入っていない．N-gram 言語モデルを使用して Java の構文エラーを特定する論文がある [18]．この論文で構文エラーを特定できる対象は Java だけであり YAML 形式は対象ではない．

3. 提案方式

提案として，YAML 形式の設定ファイルにあるエラーの原因箇所を，ルールファイル内に書き込まれているルールを使用し特定する．ルールは次の 3 つの要素から作成され，ルールファイルに書き込まれる．

- Pod の作成に失敗する YAML 形式の設定ファイルの kubectl get pod コマンドの出力結果に含まれるステータス
- Pod の作成に失敗する YAML 形式の設定ファイルの kubectl describe コマンドの出力結果に含まれる Events
- Pod の作成に失敗する YAML 形式の設定ファイルと Pod の成功する YAML 形式の設定ファイルの差分

YAML 形式の設定ファイルが kubectl apply コマンドで適用された後，作成された Pod が Running にならないときに，ルールファイルを使用しエラーの原因箇所を特定しターミナルに表示する．提案方式の流れを図 5 に示す．

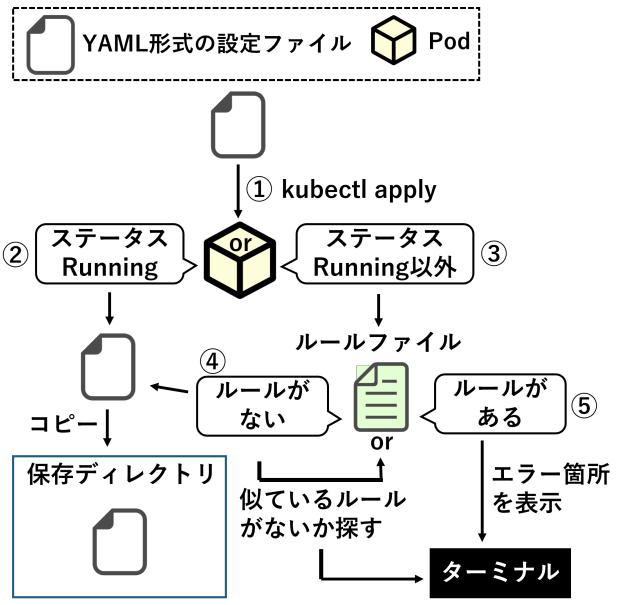

図 5　提案方式の流れ

表 1　ルールファイルの内容例

ステータス	Events	エラーの原因箇所
Pending	Warning FailedScheduling …	claimName
Error	Warning Failed …	value
CrashLoopBackOff	Warning BackOff …	args

① 開発者が YAML 形式の設定ファイルを kubectl apply コマンドで適用する．Pod が作成されるまで 5 秒待つ．
② 作成された Pod のステータスが Running のときは既に Pod は起動しているため，YAML 形式の設定ファイルを保存ディレクトリにコピーする．
③ 作成された Pod のステータスが Running 以外のとき，Pod のステータスとルールファイルのルールに含まれるステータスを比較する．また，Pod に関して kubectl describe コマンドを実行した際の Events と，ルールファイルのルールに含まれる Events を比較する．ステータスと Events がどちらも一致する場合に，ルールがあると判定する．
④ ルールがない場合は，ルールファイルに似ているルールがないか探し，検索結果の中から Pod が起動していない原因であるエラーの原因箇所を部分一致としターミナルに表示する．また，YAML 形式の設定ファイルを保存ディレクトリにコピーする．
⑤ ルールがある場合は，Pod が起動していない原因であるエラーの原因箇所をターミナルに表示する．

ルールファイルとルール

次にルールファイルの説明をする．Pod が起動していないエラーの原因箇所を特定するためにルールファイルを作成する．ルールファイルには作成されたルールが書き込まれ，書き込まれているルールを使用しエラーの原因箇所の特定をする．ルールは，ステータスと Events と設定ファイルの差分から取り出したエラーの原因箇所の 3 つの要素で構成されている．ステータスを含める理由は，ステータスによって Pod の現在の状態が分けられているため，状態からどこでエラーの種類を絞るためである．Events を含める理由は，エラーの原因箇所を特定するために Kuberneetes の知識のある人使われており，Pod のログが書かれているため何が起こったのか分かるからである．設定ファイルの差分を含める理由は，ステータスと Events が同一であれば，YAML 形式の設定ファイルの間違えた箇所が同じになると想定したからである．

ルールファイルの内容の例を表 1 に示す．ステータスと Events とエラーの原因箇所で 1 つのルールとなる．表 1 の 1 行目のルールはステータスが Pending であり，Events が Warning FailedScheduling …のとき，エラーの原因箇所は claimName になるルールである．

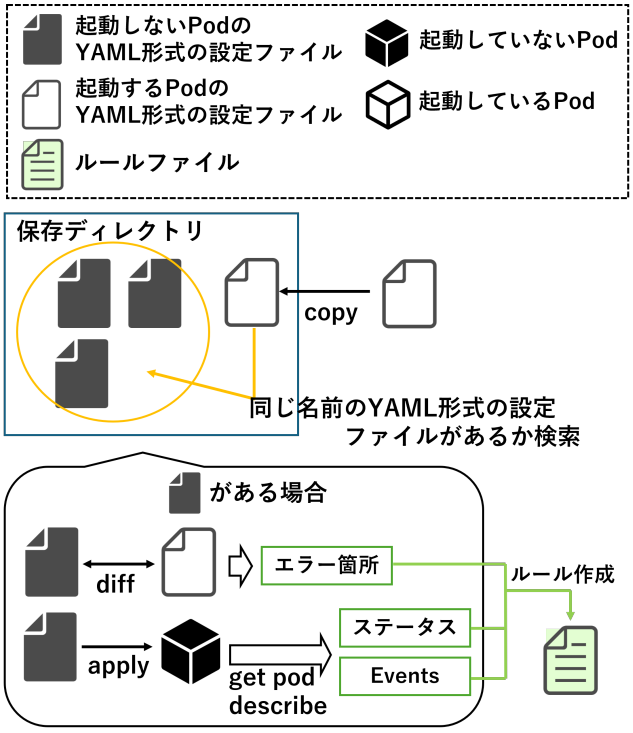

図 6　ルールとルールファイル作成方法

ルールファイルの作成方法

ルールとルールファイルの作成方法を図 6 に示す．保存ディレクトリとは kubectl apply コマンドが実行されるたびに，実行された YAML 形式の設定ファイルがコピーされ蓄積されていくディレクトリである．Pod が Running になる YAML 形式の設定ファイルと Pod が Running にならない YAML 形式の設定ファイルのどちらが kubectl apply コマンドで実行された場合でもコピーされる．保存ディレクトリには YAML 形式の設定ファイルのみ保存される．保存ディレクトリに Pod が Running になる YAML 形式の設定ファイルがコピーされたときに同じ名前の YAML 形式の設定ファイルがあるか検索する．起動しない Pod の YAML 形式の設定ファイルがある場合は，ルールが作成される．

ルールを作成するために 2 つの過程がある．1 つ目の過程は，Pod が Running にならない YAML 形式の設定ファイルと起動する Pod の YAML 形式の設定ファイルを diff コマンドで比較し差分を出す．差分からキーの部分のみを取り出し，ルールのエラーの原因箇所とする．図 7 は diff コマンドの実行結果である．YAML 形式の設定ファイルをプロジェクト実習 II[IT・3] で収集し，2 つの Nginx を作成する YAML 形式の設定ファイルを比較した．2 つのファイルは，正しいファイルと間違えているファイルである．図 7 の 1 行目と 4 行目に比較したそれぞれのファイルに書かれている差分が出力されている．図 7 では行の差分に含まれるキーは「ClaimName」であり，これをルールに記録する．この diff コマンドの実行結果からエラーの

```
1 <        claimName: c0a21151-data1
2 <
3 ---
4 >        claimName: c0a21151
```

図 7 diff コマンドの実行結果

```
NAME           READY   STATUS    RESTARTS   AGE
pod-pvc-test   0/1     Pending   0          20s
```

図 8 kubectl get pod コマンドの実行例

```
Events:
  Type     Reason            Age                From
Message
  ----     ------            ----               ----
  -------
  Warning  FailedScheduling  21m (x241 over 20h)  default-
scheduler  0/1 nodes are available: persistentvolumeclaim
"c0a220106d-data1" not found. preemption: 0/1 nodes are
available: 1 Preemption is not helpful for scheduling.
```

図 9 図 2 の Events の部分

原因箇所を求める方法を述べる．正しいファイルと間違えているファイルを比較した結果から，エラーの原因は行の差分に含まれるものとする．この差分をルールに記録すると，YAML 形式のキーと値の両方がルールに含まれる．値は YAML 形式の設定ファイルごとに異なる値であるため，ルールに含めると一致する YAML 形式の設定ファイルの件数が減る．そのため，行の差分から値を除いたキーの部分のみルールに含める．

2 つ目の過程は，起動していない Pod の YAML 形式の設定ファイルを kubectl apply コマンドで Pod を作成することである．kubectl get pod コマンドを使用し作成された Pod のステータスを取得し，kubectl describe コマンドを使用し Events を取得する．kubectl get pod コマンドの実行例を図 8 に示す．出力内容の SUTATUS の「Pending」の部分が Pod のステータスである．kubectl describe コマンドの実行例は図 2 に示されている．図 2 の Events の部分を図 9 に示す．「Events」以下に書かれている部分が Events の内容であり，「Message」の内容が Pod の実行内容である．この「Message」の内容をルールの Events とする．取り出したステータスと Events とエラーの原因箇所でルールを作成しルールファイルに書き込む．ルールファイルは csv 形式であるため，ステータスと Events とエラー原因箇所の順に,（カンマ）で区切って書き込んでいる．保存ディレクトリに起動する Pod の YAML 形式の設定ファイルがコピーされたとき同じ名前の YAML 形式の設定ファイルが無い場合はルールが作られない．

エラーの原因箇所の部分一致

エラーの原因箇所の部分一致は，ルールファイルに書かれているルールの中から作成された Pod のステータスと

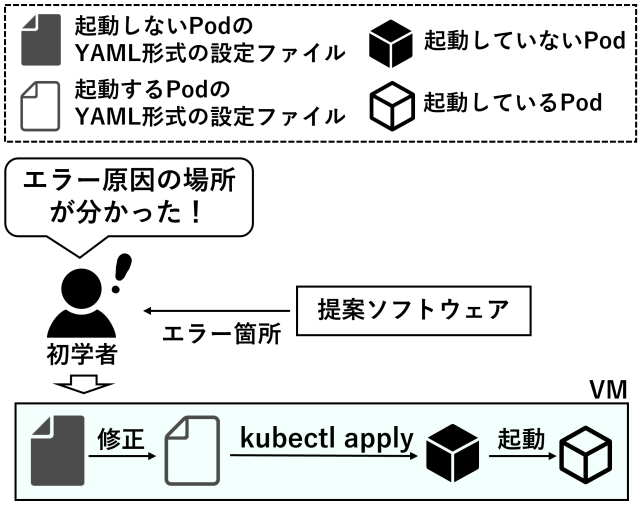

図 10 提案ソフトウェアを使用した例

ルールのステータスが同じものを検索し，作成された Pod の Events に含まれている単語がルールの Events にあるか比較し，含まれている単語数が 1 番多いルールから行っている．含まれている単語数が 1 番多いルールのエラーの原因箇所が部分一致したエラーの原因箇所となる．

ユースケース・シナリオ

Kubernetes を使用してアプリケーションを構築しようとしている初学者が YAML を勉強している状況を想定する．初めて Kubernetes を使用するため Kubernetes の知識や経験がない．初学者は Pod を作成することはできたが起動しない．Pod が起動していない原因を特定するために kubectl get pod コマンドや kubectl describe コマンドを実行する．しかし，実行結果のどこにエラーの原因箇所が書かれているのか分からない．提案ソフトウェアを使用した例を図 10 に示す．提案ソフトウェアを使用するとエラーの原因箇所が分かる．エラーの原因箇所が分かるため，起動しない Pod の YAML 形式の設定ファイルの間違えている部分をすぐに修正することができる．修正した起動する Pod の YAML 形式の設定ファイルを kubectl apply コマンドで再適用することで，起動していない Pod を起動させることができる．

4. 実装

ターミナルにエラーの原因箇所を表示するソフトウェアを作成する．この提案ソフトウェアは Python3 で作成する．ソフトウェアの設計を図 11 に示す．提案ソフトウェアである create-rule.py は保存ディレクトリである keep-file を監視している．保存ディレクトリに YAML 形式の設定ファイルをコピーするソフトウェアである copy-file.py を使用し追加する．保存ディレクトリに YAML 形式の設定ファイルが追加されると，追加されたことを提案ソフトウェアに伝える．提案ソフトウェアの処理内容の中にルー

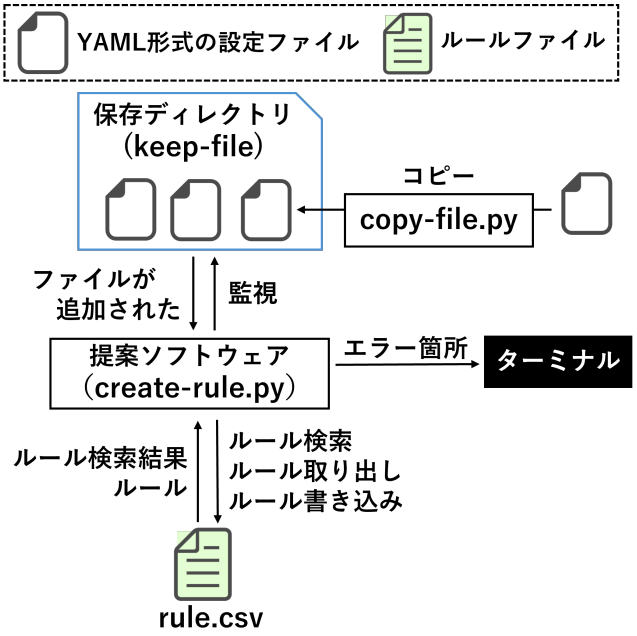

図 11 ソフトウェアの設計

ルファイルである rule.csv を使用する処理がある．rule.csv はあらかじめ空の状態で作成しておく．提案ソフトウェアはルールファイルにあるルールの検索やルールの取り出しを要求し，ルールファイルは検索結果やルールを返す．また，提案ソフトウェアは作成したルールをルールファイルに書き込む．提案ソフトウェアは特定したエラーの原因箇所をターミナルに表示する．

エラーの原因箇所のターミナル表示

ターミナルの表示はルールファイルにルールがあった場合と，ルールがなく部分一致した場合の 2 種類がある．ルールファイルにルールがあった場合のターミナルの表示の例をソースコード 2 に示す．1 行目に，「Status:」の文字の後ろにルールにあるステータスを追加し表示する．2 行目に，「Error:」の文字の後ろにルールにある Events を追加し表示する．3 行目に，「YAML ファイルの key が”」の文字の後ろにルールにあるエラー箇所を追加し，その後ろに「”の部分を確認してください」という文字を追加し表示する．

ルールファイルにルールがなく部分一致した場合のターミナルの表示の例をソースコード 3 に示す．1 行目に，「エラーがある箇所を部分一致で探します」という文字を表示する．2 行目に，「YAML ファイルの key が”」の文字の後ろに部分一致したエラー箇所を追加し，その後ろに「”の部分を確認してください」という文字を追加し表示する．

5. 評価実験

YAML 形式の Pod を作成するファイル内にあるエラーの原因箇所をターミナルで指摘できたかの精度を評価する．収集した YAML 形式の設定ファイルの 75%を使用しあらかじめルールを作成する．その後収集した YAML 形式の設定ファイルの 25%を使用しエラーの原因箇所を特定する実験を行う．このデータの分割にはホールドアウト (Hold-out cross-validation) を使用した [19,20]．ホールドアウトとは，効率性と簡便さで広く使われているクロスバリデーション手法である．データセットを訓練用とテスト用に分け性能を評価する．YAML 形式の設定ファイル内にあるエラーの原因箇所を提案ソフトウェアはターミナルに表示によって指摘できたのかを比較し，検知率を計算する．

ソースコード 2 ルールがあった場合のターミナルの表示例

```
1 Status: Pending
2 Error: Warning  InspectFailed  5s (x2
  over 5s)  kubelet Failed to apply
  default image tag "nginx;1.12":
  couldn't parse image name
  "nginx;1.12": invalid reference format
3 YAMLファイルのkeyが”image:”の部分を確
  認してください
```

ソースコード 3 ルールがなく部分一致した場合のターミナルの表示例

```
1 エラーがある箇所を部分一致で探します
2 YAMLファイルのkeyが”image:”の部分を確
  認してください
```

YAML 形式の設定ファイルの収集

評価実験にはプロジェクト実習 II[IT・3] で集めた YAML 形式の設定ファイルを使用した．プロジェクト実習 II[IT・3] は，少人数のグループによる実習を行なうことで，サイト・リライアビリティ・エンジニアリング (Site Reliability Engineering, 以下 SRE とする) と関連する知識とスキルの習得を目的としている．SRE は，Google が最初に提唱したシステム管理とサービス運用の方法論である．SRE は，システムの信頼性に焦点をあてて，システム運用を正しく設計することでシステム管理を効率化することによって，システムの信頼性を高めることを目的としている．プロジェクト実習 II[IT・3] は，SRE の実習を通して実学にもとづく専門能力，論理的な思考力と問題解決力を身に着ける授業である．また，実習におけるグループワークと成果発表を通してコミュニケーション能力を強化し実習結果のレポート作成と報告を通して分析・評価能力も身につける授業である．

プロジェクト実習 II[IT・3] は，受講している学生が 40 人おり，4 人 1 グループで作業するため 10 グループに

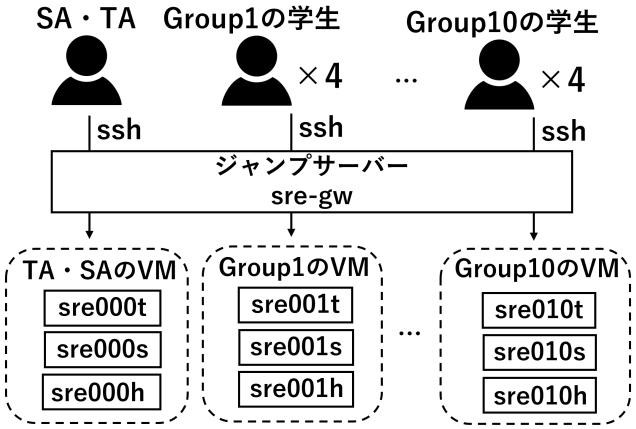

図 12　作業環境の構成

分かれる．また，スチューデントアシスタント（Student Assistant，以下 SA とする）の学部 4 年生が 6 人とティーチングアシスタント（Teaching Assistant，以下 TA とする）の大学院生 2 人と教授 1 人で行っている．プロジェクト実習 II[IT・3] では，SA，TA と学生のグループごとに作業環境があり，作業環境にはバーチャルマシン（Virtual Machine，以下 VM とする）が 3 つずつ用意されている．プロジェクト実習 II[IT・3] の作業環境の構成を図 12 に示す．3 つの VM の名前は，sre と 3 桁のグループ番号と h，t，s のどれかで決められている．VM の名前の h，t，s は VM の役割で決められており，kubectl コマンドを実行する VM を h とし，マスターノードとして使用する VM を t とし，Kubernetes クラスタのワーカーノードとして使用する VM を s としている．各 VM にはジャンプサーバーに SSH で接続し，そこから各 VM に SSH で接続することで VM を使用することができる．ジャンプサーバーとは，外部からアクセスされたとき直接，目的の VM にアクセスすることはセキュリティ面から危険であるため，中継サーバーとして使用されるサーバーである．SSH とは，Secure Shell の略であり，ネットワークを通じて別のコンピュータを安全に遠隔操作するためのプロトコルである．

YAML 形式の設定ファイルの収集方法を図 13 に示す．グループ 1 からグループ 10 の学生が kubectl コマンドを実行する sre001h から sre010h に my-kube.py を配置する．my-kube.py は kubectl apply コマンドが実行されると kubectl apply コマンドで適用された YAML 形式の設定ファイルを同 VM 内にある collect ディレクトリに cp コマンドを使用してコピーする．SA，TA が使用できる h の VM である sre000h で異なる VM から指定したファイルをコピーする scp コマンドを実行し，グループ 1 からグループ 10 の h の VM である sre001h から sre010h にある collect ディレクトリに格納されているファイルを sre000h にある collect ディレクトリにコピーする．sre000h にある collect ディレクトリにはグループ 1 からグループ 10 の学

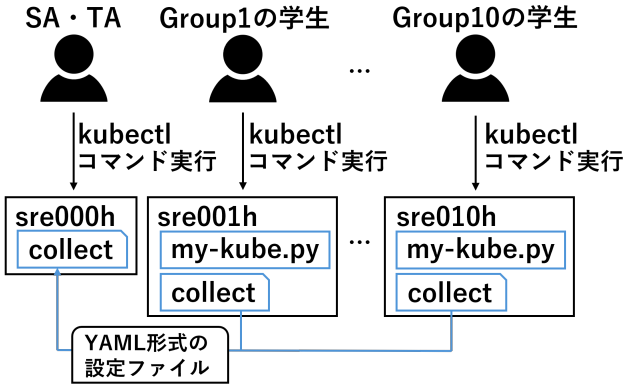

図 13　YAML 形式の設定ファイルの収集方法

生が kubectl apply コマンドで適用した YAML 形式の設定ファイルが格納され収集できる．collect ディレクトリに収集した YAML 形式の設定ファイルの例を図 14 に示す．14 回の実習のうち 7 回の実習で 1798 個の YAML ファイルを集めることができた．集めた YAML 形式の設定ファイルを格納したディレクトリで ls コマンドを実行したときの表示はソースコード 4 の通りとなった．ls コマンドとは，YAML 形式の設定ファイルを収集する際に，YAML 形式の設定ファイル名を学生が付けた名前に_学籍番号_日時を追加し収集した．学籍番号は学生の個人情報であるため下 5 桁を x にしている．

収集した YAML 形式の設定ファイルの kind をソース

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: c0a2201673-ingress
  annotations:
    ingress.kubernetes.io/ssl-redirect: "f
alse"
    ingress.kubernetes.io/rewrite-target:
"/"
spec:
  rules:
  - http:
    paths:
    - path: /c0a2201673/app-service-1
      pathType: Prefix
      backend:
        service:
          name: app-svc-1
          port:
            number: 4000
  - http:
    paths:
    - path: /c0a2201673/app-service-2
      pathType: Prefix
      backend:
        service:
          name: app-svc-2
          port:
            number: 4000
```

図 14　YAML 形式の設定ファイルの例

ソースコード 4 YAML 形式の設定ファイルの一覧表示の一部

```
1  ...
2  ingress-app-all_c0a22xxxxx_14-11-51.yaml
3  ingress-app-all_c0a22xxxxx_14-13-43.yaml
4  ingress-app-all_c0a22xxxxx_21-32-46.yaml
5  ingress-app-all_c0a22xxxxx_09-57-43.yaml
6  ingress-app-all_c0a22xxxxx_10-02-21.yaml
7  nginx-content_c0a21xxxxx_17-10-28.yaml
8  nginx-content_c0a21xxxxx_17-11-19.yaml
9  nginx-content_c0a21xxxxx_17-30-43.yaml
10 nginx-content_c0a21xxxxx_17-31-11.yaml
11 nginx-content_c0a21xxxxx_17-37-43.yaml
12 nginx-content_c0a21xxxxx_17-55-54.yaml
13 rbac_c0a21xxxxx_2024-11-09-15-33-49.yaml
14 rbac_c0a21xxxxx_2024-11-09-16-42-56.yaml
15 rbac_c0a21xxxxx_2024-11-09-14-17-18.yaml
16 rbac_c0a21xxxxx_2024-11-09-16-56-02.yaml
17 rbac_c0a22xxxxx_2024-11-09-14-18-46.yaml
18 rbac_c0a22xxxxx_2024-11-08-14-35-36.yaml
19 rbac_c0a22xxxxx_2024-11-08-14-35-37.yaml
20 ...
```

コード 5 に示す．kind では，オブジェクトの種類の設定を書く．各 YAML 形式の設定ファイルの kind を一覧にするために grep コマンドと awk コマンドと sort コマンドと uniq コマンドを使用した．grep コマンドは，指定された文字列が含まれているファイルを表示する．オプションの-r を付けると，サブディレクトリまで指定された文字列が含まれているファイルがあるか確認する．awk は，プログラミング言語であるが Linux コマンドとして使える．$NF は最後のフィールドを指定する．ここでは，grep コマンドで出力されるファイル内の最後のフィールドを出力している．sort コマンドは，ファイルを並べ替える．uniq コマンドは，重複しているものを削除し 1 つにする．実行したコマンドは 1 行目の$の後ろに表示されている．各コマンドは |（パイプ）で繋げられ，コマンド実行内容を次のコマンドへ橋渡しされている．2 行目以降は，収集した YAML 形式の設定ファイルにあった kind の設定値とその数である．kind の設定値が Middleware が 741 個，kind の設定値が Ingress が 1 個，kind の設定値が ServiceAccount が 403 個，kind の設定値が ConfigMap が 284 個，kind の設定値が Service が 262 個，kind の設定値が ClusterRole が 239 個，kind の設定値が Deployment が 234 個，kind の設定値が Role が 166 個，kind の設定値が ClusterRoleBinding が 121 個，kind の設定値が RoleBinding が 83 個，kind の設定値が Pod が 29 個，kind の設定値が Pod に設定ファイルを渡すことができるリソースが 14 個，kind の設定値が Rule が 3 個，kind の設定値が Ingress-redirect が 2 個，kind の

ソースコード 5 収集した YAML 形式の設定ファイルの kind

```
1  tanaka@sre000h:~/collect-2/tecrepo$
   grep -r "kind: " collect* | awk
   '{print $NF}' | sort | uniq -c | sort
   -rn
2      741 Middleware
3      610 Ingress
4      403 ServiceAccount
5      284 ConfigMap
6      262 Service
7      239 ClusterRole
8      234 Deployment
9      166 Role
10     121 ClusterRoleBinding
11      83 RoleBinding
12      29 Pod
13      14 Podに設定ファイルを渡すことが
        できるリソース
14       3 Rule
15       2 Ingress-redirect
16       2 Configmap
17       2 configmap
18       1 PersistentVolumeClaim
19       1 PersistentVolume
20       1 IngressRoute
21 tanaka@sre000h:~/collect-2/tecrepo$
```

設定値が Configmap が 2 個，kind の設定値が configmap が 2 個，kind の設定値が PersistentVolumeClaim が 1 個，kind の設定値が PersistentVolume が 1 個，kind の設定値が IngressRoute が 1 個である．kind の各設定値の加算数は 1798 より大きくなるが，これは 1 つの YAML 形式の設定ファイルで複数のリソースを作成しているファイルがあるからである．

本稿の実験で使用する YAML 形式の設定ファイルは kind が Pod のものである．12 行目から，kind が Pod である YAML 形式の設定ファイルは 29 個であることが分かる．29 個のファイルの中には，ファイル名である学生が付けた名前_学籍番号_日時の日時のみ異なるものがある．ファイル名の日時のみ異なる複数のファイルは，1 個の Running になるファイルとその他の Running にならなずエラーが起きるファイルである．ファイル名の日時のみ異なる複数のファイルは 1 つのファイルのまとまりとする．実験用 YAML ファイルの 29 個の内 2 個のファイルは，エラーが無く Running になるファイルであり，ファイル名の日時のみ異なる他のファイルがないためファイルのまとまりがない．ファイルのまとまりがない場合，ルールを作成できないためルール用 YAML ファイルとして使用できず，

エラーが無いため評価用 YAML ファイルとしても使用できない．2 個のファイルは使用できないため実験用 YAML ファイルから取り除き，実験用 YAML ファイルを 27 個とする．ルール用 YAML ファイルは実験用 YAML ファイルの 75%とし，評価用 YAML ファイルは実験用 YAML ファイルの 25%とする．ルール用 YAML ファイルは 20 個となり評価用 YAML ファイルは 7 個となる．

実験手順

ソースコード 5 から Middleware が最も使われるように見える．しかし本稿では，Kubernetes でアプリケーションを動かすときに使用するリソースは Pod であるため，kind が Pod である YAML 形式の設定ファイルで実験を行った．収集した YAML 形式の設定ファイルのうち，kind が Pod のものを取り出し使用する．この取り出した YAML 形式の設定ファイルを実験用 YAML ファイルとする．実験の手順を図 15 に示す．

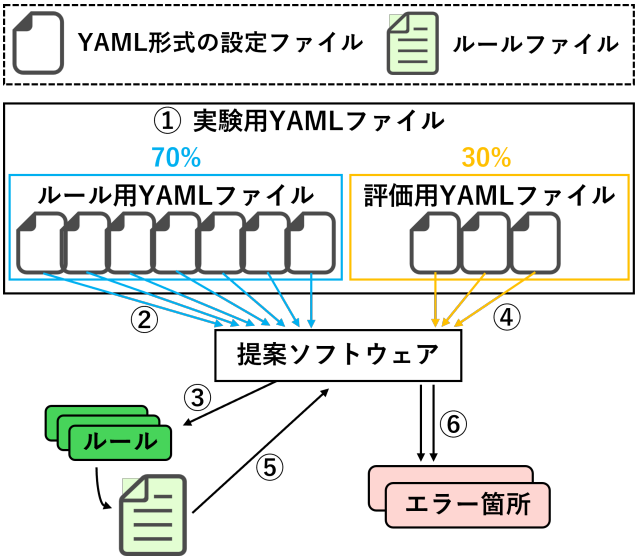

図 15 実験の手順

① 実験用 YAML ファイルを，ルールを作成するもの（以下，ルール用 YAML ファイル）と，エラーの原因箇所を特定する実験を行うもの（以下，評価用 YAML ファイル）に分ける．このとき，ルール用 YAML ファイルは実験用 YAML ファイルの 75%とし，評価用 YAML ファイルは実験用 YAML ファイルの 25%とする．
② ルールを作成するため，ルール用 YAML ファイルごとに提案ソフトウェアを実行する．
③ ルール用 YAML ファイルから提案ソフトウェアで作成されるルールをルールファイルに書き込む．
④ エラーの原因箇所を特定するため，評価用 YAML ファイルごとに提案ソフトウェアを実行する．
⑤ 提案ソフトウェアはエラーの原因箇所を特定するためにルールファイルに書き込まれているルールを使用する．
⑥ 提案ソフトウェアで評価用 YAML ファイルにあるエラーの原因箇所を特定する．

評価用 YAML ファイルのうち，何個の YAML 形式の設定ファイルのエラーの原因箇所を特定することができたか，特定されたエラーの原因箇所と YAML 形式の設定ファイルにあるエラーの原因箇所から検知率を計算する．

表 2 作成されたルール

ステータス	Events	エラーの原因箇所
Pending	Warning Failed 3s kubelet Failed to pull image ””nginxlatest””: failed to pull and unpack image ””docker.io/library/nginxlatest: latest””: failed to resolve reference ””docker.io/library/nginxlatest:latest””: pull access denied	labels:
Pending	Warning Failed 3s kubelet Error: ErrImagePull	app:
Pending	Warning Failed 3s kubelet Error: ImagePullBackOff	containerPort:
Pending	Warning FailedMount 2s (x4 over 5s) kubelet MountVolume. SetUp failed for volume ””html-volume”” : configmap ””nginx-helm-configmap”” not found	name:
Pending	Warning FailedMount 2s (x4 over 5s) kubelet MountVolume. SetUp failed for volume ””html-volume”” : configmap ””nginx-helm-configmap”” not found	kind:

実験環境

プロジェクト実習 [IT・3] で使用している環境である sre000h の VM で行う．sre000h の構成は，vCPU が 2Core，メモリが 6GB，ディスクが 50GB である．sre000h に Python3 の仮想環境を作成し，提案ソフトウェアを実行する．仮想環境にインストールしたライブラリは watchdog である．

実験結果と分析

ルール用 YAML ファイルである 20 個の YAML 形式の設定ファイルを時系列順に実行するために昇順に並べ，提案ソフトウェアでルールを作成したところ，5 つのルールが作成され，ルールファイルに書き込まれた．作成されたルールを表 2 に示す．

次に，7 個の評価用 YAML ファイルで実験を行う．評価

表 3　実験結果

	ルールがあり特定したYAML ファイル	ルールがなく部分一致したYAML ファイル
評価用YAML ファイル数（個）	3	4
割合（%）	43	57

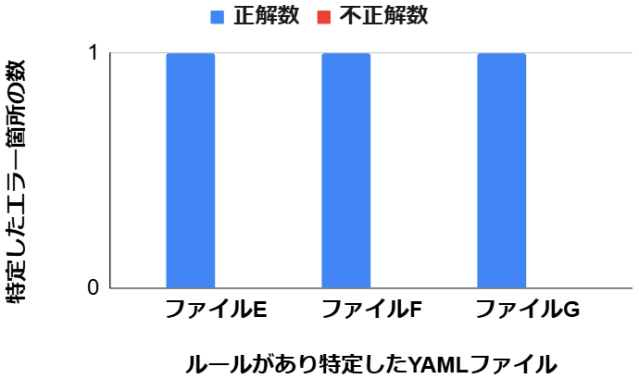

図 16　ルールがあり特定した YAML ファイルの内訳

に使用する YAML 形式の設定ファイルはエラーの原因箇所を特定できる必要があるため，作成される Pod が Running にならないものである必要がある．実験結果を表 3 に示す．ルールがありエラーの原因箇所を特定できた YAML 形式の設定ファイルは 7 個のうち 3 個で約 43%，ルールがなく部分一致した YAML 形式の設定ファイルは 7 個のうち 4 個で約 57%となった．ルールがありエラーの原因箇所を特定できたものは全てエラー原因がある場所だった．しかし，ルールがなく部分一致したものは部分一致したエラーの原因箇所が 4 個あり，エラーの原因箇所を正解しているものと不正解のものがあった．ルールがあり特定したものの内訳を図 16 に示す．3 個のファイルすべてで特定したエラーの原因箇所は 1 個だった．エラーの原因箇所の特定は複数のルールを特定することもあるが，1 個しか検出されなかった．特定されたエラーの原因箇所は 1 個だが，YAML 形式の設定ファイルには特定されたエラーの原因箇所のキーが複数行にあった．同じキーが 1 つの YAML 形式の設定ファイルにある例をソースコード 6 に示す．ソースコード 6 ではキーが name の箇所が，4 行目の metadata の name と 8 行目の containers の name と 12 行目の volumeMounts の name と 17 行目の volumes の name の 4 か所ある．YAML 形式の設定ファイルでは階層が異なる場合，同じキーを使用することができる．階層は，木構造で作られているインデントごとに分かれている．そのため YAML 形式の設定ファイルを修正するときに複数個所を確認する必要があり，1 箇所に特定しきることができていなかった．ルールがなく部分一致したものの内訳を図 17 に示す．4 個のファイルすべてで部分一致したエラーの原因箇所は 4 個ずつであり，合計で 16 箇所の原因箇所

ソースコード 6　収集した YAML 形式の設定ファイルの kind

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: pod-pvc-test
5    namespace: c0a21151-test
6  spec:
7    containers:
8    - name: local-path-test
9      image: nginx-:stable-alpine
10     imagePullPolicy: IfNotPresent
11     volumeMounts:
12     - name: local-path-pvc
13       mountPath: /data
14     ports:
15     - containerPort: 80
16   volumes:
17   - name: local-path-pvc
18     persistentVolumeClaim:
19       claimName: c0a21151-data1
```

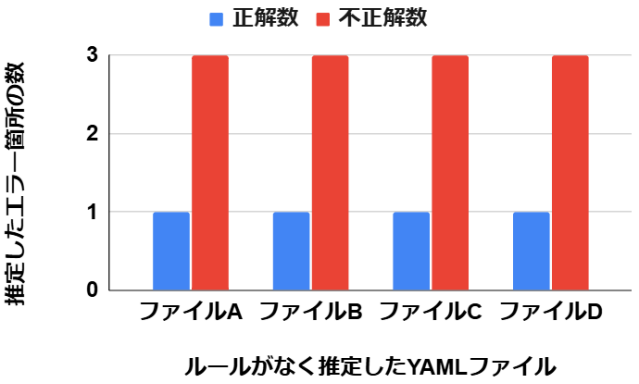

図 17　ルールがなく部分一致した YAML ファイルの内訳

を検知した．部分一致した各 YAML ファイルに実際のエラー原因が 1 箇所あり，提案ソフトウェアが部分一致したエラーの原因箇所は 1 個が正解であり，3 個が不正解だった．検知した 16 箇所の原因箇所の内，正しく検知された箇所は 4 箇所であり誤検知は 12 箇所であった．

評価として，実験を行った 7 個の評価用 YAML ファイルのうち，ルールがありエラーの原因箇所を特定できたものの検知率は 4/4 から 100%となり，ルールがなく部分一致したものの検知率は $(1 \times 4)/(4 \times 4)$ から 25%で，誤検知率は $(3 \times 4)/(4 \times 4)$ から 75%となった．

6. 議論

エラーの原因箇所を識別するために提案方式では YAML 形式の設定ファイルのキーの部分をルールに含めた．識別に使用する情報がキーの部分だけでは，1 個の YAML

形式の設定ファイル内に検出されたキーが複数個あると，エラーの原因箇所を 1 箇所に特定できなかった．例えば，name の部分が重複しているためエラーの原因箇所を特定するとき，ルールのエラーの原因箇所を階層まで含めることで，特定するエラーの原因箇所を 1 箇所にする．階層は，木構造で作られているインデントごとに分かれており，同じ階層内では同じキーは使えないため 1 箇所に特定できる．

提案手法では，kubectl describe コマンド内にある Events でエラーの原因箇所が特定できない場合に，ルールファイルを使用しエラーの原因箇所を特定している．一方で，kubectl describe コマンド内にある Events でエラーの原因箇所が特定できる場合にも，ルールファイルを使用しエラーの原因箇所を特定している．そのため，図 3 をはじめとした本来は必要のない処理がよばれている．そのため，Events にエラーの原因箇所が含まれている場合は Events のエラーメッセージを使用し，Events にエラーの原因箇所が含まれない場合はルールファイルを使用する．

ルールファイルにルールがない場合に，ルールファイルに書かれているルールからエラーの原因箇所を部分一致する．部分一致の方法は，ルールファイルに書かれているルールの中から作成された Pod のステータスとルールのステータスが同じものを検索し，作成された Pod の Events に含まれている単語がルールの Events にあるか比較する．その後，含まれている単語数が最も多いルールのエラーの原因箇所を，エラーの原因箇所と判定する．仮に含まれている単語数が最も多いルールが 2 つ以上ある場合，それら全てを部分一致した原因箇所と判定する．解決策はエラーの原因箇所が 2 個以上ある場合に，ルールファイルに書かれているルールの中で作成されたのが最も新しいルールのエラーの原因箇所を部分一致した原因箇所とする．そのためにルールを作成する要素に作成したときの日時を追加する．部分一致するエラーの原因箇所を 1 個に絞り，確実にエラーがある場所を検出する．

本稿の提案では，ルールファイルにルールがなくエラーの原因箇所を部分一致する場合，ルールファイルに書かれているルールの中からエラーの原因箇所を単語の出現回数を元に部分一致しており，部分一致したエラーの原因箇所が複数個でも全て出力する．部分一致したエラーの原因箇所の 16 個のうち 12 個が誤りであり誤検知率が 75%と高い．そのため，エラーの原因箇所を特定したい YAML 形式の設定ファイルとルールの Events の単語の一致率が 50%より低い場合はエラーの原因箇所が見つからなかったと判断する．単語の一致率が低い場合はエラーの原因箇所を提示しないことで誤検知率を下げる．

評価実験では実験用 YAML ファイルをルール用 YAML ファイルと評価用 YAML ファイルを 75%と 25%に分けている．本稿の実験ではプロジェクト実習 II[IT・3] で集めた 27 個の YAML 形式の設定ファイルを使用しており，これを 75%と 25%に分割するとデータ件数が少なく実験結果の信頼性が十分とはいえない．実験と評価を行ったファイル数が 27 個であることは少ないため，収集した全ての YAML 形式の設定ファイルを使い実験と評価を行う．

本稿では評価はプロジェクト実習 II[IT・3] で集めた YAML 形式の設定ファイルの一部を，ルール用 YAML ファイルと評価用 YAML ファイルの 2 種類に分け，エラーの原因箇所が特定できるかを評価した．仮にエラーの原因が特定ができたとしても，YAML 形式の設定ファイルのエラーの原因を正確に修正できなければ，Pod は起動いない．そのため，Pod が起動するかを含めた評価実験が必要である．この問題の解決策は，評価実験で実際に提案ソフトウェアを学生に使用してもらい，エラーの原因解決に役立ったかアンケートで回答してもらうことである．

7. おわりに

課題は，Pod の起動時にエラーが発生した場合にエラー原因の特定ができないことである．提案として，YAML 形式の設定ファイルからルールファイルを作成しエラーの原因箇所を特定する．評価実験では，YAML 形式の設定ファイル内にあるエラーの原因箇所を特定できたかとして検知率を評価する．結果として，実験を行った 7 個の評価用 YAML ファイルのうち，ルールがありエラーの原因箇所を特定できたものの検知率は 100%となり，ルールがなく部分一致したものの検知率は 25%で，誤検知率は 75%となった．

参考文献

[1] Cheng, X., Fu, E., Ling, C. and Lv, L.: Research on Kubernetes Scheduler Optimization Based on real Load, *2023 3rd International Conference on Electronic Information Engineering and Computer Science (EIECS)*, pp. 711–716 (online), DOI: 10.1109/EIECS59936.2023.10435549 (2023).

[2] 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).

[3] Park, J., Choi, U., Kum, S., Moon, J. and Lee, K.: Accelerator-Aware Kubernetes Scheduler for DNN Tasks on Edge Computing Environment, *2021 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 438–440 (online), DOI: 10.1145/3453142.3491411 (2021).

[4] 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).

[5] Bhavsar, S., Agrawal, A., Ropalkar, T., Kamdi, P., Hajare, A., Deshpande, S., Rathi, R. and Garg, D.: Kubernetes Cluster Disaster Recovery Using AWS, *2023 7th International Conference On Computing, Communication, Control And Automation (ICCUBEA)*, pp. 1–6 (online), DOI: 10.1109/ICCUBEA58933.2023.10391973

(2023).

[6] Kakade, S., Abbigeri, G., Prabhu, O., Dalwayi, A., G, N., Patil, S. P. and Sunag, B.: Proactive Horizontal Pod Autoscaling in Kubernetes using Bi-LSTM, *2023 IEEE International Conference on Contemporary Computing and Communications (InC4)*, Vol. 1, pp. 1–5 (online), DOI: 10.1109/InC457730.2023.10263031 (2023).

[7] Zhu, M., Kang, R., He, F. and Oki, E.: Implementation of Backup Resource Management Controller for Reliable Function Allocation in Kubernetes, *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, pp. 360–362 (online), DOI: 10.1109/NetSoft51509.2021.9492724 (2021).

[8] Junior, P. S., Miorandi, D. and Pierre, G.: Good Shepherds Care For Their Cattle: Seamless Pod Migration in Geo-Distributed Kubernetes, *2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC)*, pp. 26–33 (online), DOI: 10.1109/ICFEC54809.2022.00011 (2022).

[9] Vasireddy, I., Wankar, R. and Chillarige, R. R.: Recreation of a Sub-pod for a Killed Pod with Optimized Containers in Kubernetes, *International Conference on Expert Clouds and Applications*, Springer, pp. 619–628 (2022).

[10] Nguyen, T.-T., Yeom, Y.-J., Kim, T., Park, D.-H. and Kim, S.: Horizontal pod autoscaling in kubernetes for elastic container orchestration, *Sensors*, Vol. 20, No. 16, p. 4621 (2020).

[11] Rör, A.: Finding the Sweet Spot: Optimizing Kubernetes for Scalability and Resilience: A Comprehensive Study on Improving Resource Utilization and Performance in Containerized Environments. (2023).

[12] Ahuja, R. A.: Securing the End Points of Microservices using Gitlab Client-Based Authentication, PhD Thesis, Dublin, National College of Ireland (2023).

[13] Untersander, J. and Nebai Kidane, T.: LTB Operator, PhD Thesis, OST Ostschweizer Fachhochschule (2023).

[14] Khatami, A. A., Purwanto, Y. and Ruriawan, M. F.: High Availability Storage Server with Kubernetes, *2020 International Conference on Information Technology Systems and Innovation (ICITSI)*, pp. 74–78 (online), DOI: 10.1109/ICITSI50517.2020.9264928 (2020).

[15] Buchanan, S., Rangama, J., Bellavance, N., Buchanan, S., Rangama, J. and Bellavance, N.: kubectl Overview, *Introducing Azure Kubernetes Service: A Practical Guide to Container Orchestration*, pp. 51–62 (2020).

[16] Johnson, L. and Pheanis, D. C.: Automated Error-Prevention and Error-Detection Tools for Assembly Language in the Educational Environment, *Proceedings. Frontiers in Education. 36th Annual Conference*, pp. 19–23 (online), DOI: 10.1109/FIE.2006.322560 (2006).

[17] Chen, Z.: Semantic based Cross-Language Clone Related Bug Detection, *2021 2nd International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT)*, pp. 494–499 (online), DOI: 10.1109/AINIT54228.2021.00101 (2021).

[18] Campbell, J. C., Hindle, A. and Amaral, J. N.: Syntax errors just aren't natural: improving error reporting with language models, *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, New York, NY, USA, Association for Computing Machinery, p. 252–261 (online), DOI: 10.1145/2597073.2597102 (2014).

[19] Reitermanova, Z. et al.: Data splitting, *WDS*, Vol. 10, Matfyzpress Prague, pp. 31–36 (2010).

[20] Nurhayati, Soekarno, I., Hadihardaja, I. K. and Cahyono, M.: A study of hold-out and k-fold cross validation for accuracy of groundwater modeling in tidal lowland reclamation using extreme learning machine, *2014 2nd International Conference on Technology, Informatics, Management, Engineering Environment*, pp. 228–233 (online), DOI: 10.1109/TIME-E.2014.7011623 (2014).

