KubernetesのPodステータスに基づくルールファイルを用いたエラー原因箇所の特定による実行コマンド数の削減

田中 美帆[1]　小山 智之[2]　串田 高幸[1]

概要：東京工科大学では，Kubernetes を使用する実習を 3 年生に向けて行っている．Kubernetes を使用し Pod を作成するとき，Pod が起動せずエラーが起こることがある．学生は Pod を起動するために kubectl describe コマンドや kubectl apply コマンドを使用し直す作業を Pod が起動するまで行う．課題は，Kubernetes を使用して Pod を起動させる際に起きるエラーの原因を特定し Pod が起動するために kubectl コマンドを何度も打つ作業や，YAML ファイルをテキストエディタで修正する作業が手間になることである．提案として，エラーが起きた YAML ファイルからエラー箇所を特定するためのルールファイルを作成する．kubectl describe コマンドの Events にエラー原因がある箇所がないときはルールファイルを使用しエラー箇所を特定する．基礎実験として，14 回のプロジェクト実習のうち 6 回で 18 個の YAML ファイルを集め，エラーが起きた Pod のステータスとその原因について集計した．結果として，ステータスは CrashLoopBackOff と Error と Pending が約 89%を占めた．そのため，CrashLoopBackOff と Error と Pending のエラー原因を解決すれば，エラーが発生したとき約 89%を解決できる．これらエラーの解決に 14 個のルールが必要である．評価は，エラー解決までに kubectl apply コマンドを実行した回数と提案プログラムで正しいエラー箇所を特定できているかの精度を計測する．

1. はじめに

背景

東京工科大学のコンピュータサイエンス学部では，Kubernetes を使用する Site Reliability Engineering の実習であるプロジェクト実習 [IT・3] を 3 年生に向けて行っている．実習は 1 クラス約 40 人の受講学生に Student Assistant(以下:SA) の学部 4 年生が 6 人と Teaching Assistant(以下:TA) の院生 2 人と教授 1 人で行っている．1 回の実習は 5 時間あり前期の実習では 14 回行われる．はじめに TA が資料の説明を行い，その後学生は課題を進める．学生は課題の中で不明な点を SA・TA に質問しながら課題を行う．終わらなかった課題は宿題となる．学生は 1 グループ 3，4 人で 10 グループに分かれてグループごとに同じ Virtual Machine(以下:VM) を使用する．学生は踏み台 VM に SSH し学生ごとにソフトウェアを VM 上に配置する．踏み台 VM は，通信の制御をしたりリバースプロキシで外部の Port と内部の Port を繋ぐことで外部ネットワークと内部ネットワークのアクセスを制御している．

[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

プロジェクト実習内の使用環境を**図 1** に示す．

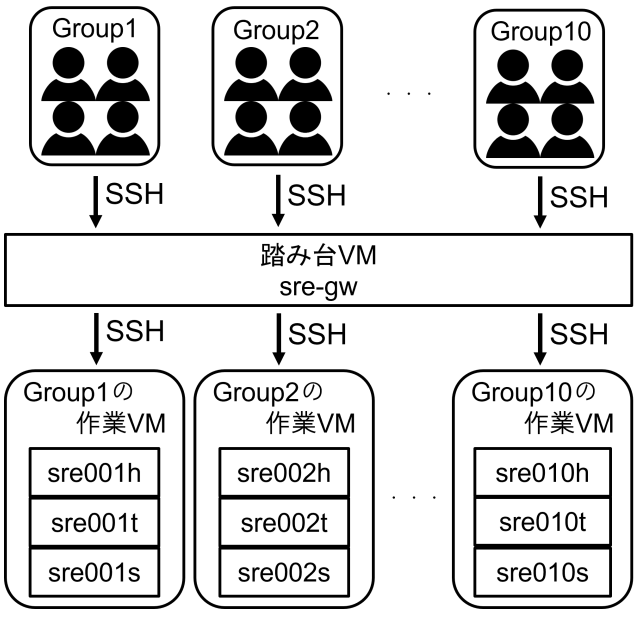

図 1　プロジェクト実習の使用環境

1 グループの学生は 4 人で 10 グループあり，学生は踏み台 VM である sre-gw へ SSH する．sre-gw から自分が所属するグループの VM へ SSH する．各グループには作業 VM が 3 つずつ用意されている．用意された 3 つの VM は

sre にグループ番号と h，t，s のいずれか 1 つの組み合わせで決められている．例として，グループ 1 だと sre001h と sre001t と sre001s の 3 つとなる．h が付く VM は，学生が kubectl コマンドを実行する VM である．t が付く VM は，マスターノードとして使用されている VM である．s が付く VM は，Kubernetes クラスタのワーカーノードとして使用されている VM である．

例えば，学生がグループの VM で行う実習内容の 1 つに Cassandra ソフトウェアの構築がある．Cassandra とは，NoSQL の一種であり分散ストレージシステムである [1,2]．NoSQL とは，リレーショナルデータベースとは異なるデータ構造を持ち，大量のデータを柔軟に扱うことや高速なデータの読み書きが可能である [3]．Cassandra を作成するためには VM 内で Kubernetes を使用し YAML 形式で書かれた YAML ファイルでコンテナを作成し Pod を起動する必要がある．コンテナや Pod を作成し管理するために幅広く使用されているものが Kubernetes である [4,5]．Kubernetes とは，コンテナ化されたアプリケーションを実行管理するためのオープンソースのプラットフォームである [6,7]．Kubernetes がコンテナを管理するために kubectl というコマンドラインインターフェースがある [8–10]．kubectl を使用して Kubernetes コンテナのリソースやログの確認という管理を行うことができる [11]．リソースとは，コンピュータのソフトウェアやハードウェアを動作させるために必要なハードディスク容量や CPU の処理速度やメモリ容量のことである．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 [12,13]．コンテナとは，アプリケーションの構成要素をパッケージ化するものであり，Pod の一部である [14,15]．YAML 形式とは，データのシリアライゼーション (直列化) 形式であり，人間にとって読みやすいだけでなく，コンピュータにとっても解析しやすいテキスト形式のデータ表現である [16,17]．

Pod が起動しないとき，ソフトウェアを使用することができない．Pod が起動しない原因は YAML 形式で書かれた YAML ファイルの書き間違いやアプリケーションを構築している環境である Kubernetes クラスタが考えられる．Pod には状態を表すステータスがある．Pod が起動しているときステータスが Running になる．エラーが起きた時のステータスは例として Error や Pending や CrashLoopBackOff や CreateContainerConfigError がある [18]．起動しない Pod の原因を特定するために Pod の詳細情報をターミナルに表示する kubectl describe コマンドで Pod の詳細情報をみる．詳細情報に含まれる Events の欄には Pod で行われ終了した出来事が記載されている．出来事にはエラーが起きず通常終了したものや，エラーが起きて異常終了したものがある．Pod が起動しない原因は異常終了した出来事に書かれていることがある．学生はこの部分から Pod が起動しない原因を特定し，Pod を起動するために Pod のステータスが Running になるように直し Pod を動かす．

課題

課題は，Kubernetes を使用して Pod を起動させる際に起きるエラー原因の特定に手間がかかることである．手間とは，Kubernetes を使用して Pod を起動させる際に起きるエラー原因を特定し Pod が起動するように kubectl コマンドを何度も打つ作業と，YAML ファイルをテキストエディタで修正する作業のことである．Pod が起動しない原因の例として，YAML ファイルの設定に文字が足りないという問題が含まれている YAML ファイルを kubectl apply コマンドを実行し適用すると，kubectl apply でエラーは起きずコンテナが作成される．

YAML ファイルの設定に文字が足りないという問題が含まれている YAML ファイルの例として，ハイフンが足りていない YAML ファイルのソースコードをソースコード 1 に示す．ソースコード 1 は Cassandra の設定の YAML ファイルである．ソースコード 1 の 2 行目に書かれている cassandra0 が正しくは cassandra-0 である．エラー原因を特定するために kubectl describe コマンドを実行し，Pod の詳細情報に含まれる Events を取得する．

ソースコード 1 ハイフンが足りていない YAML ファイル

```
1 - name: CASSANDRA_SEEDS
2   value:
    "cassandra0.cassandra.c0a0000000.svc.
    cluster.local"
```

kubectl describe コマンドで取得できる詳細情報に含まれる Events は Pod の処理内容やエラーが起きている原因がある箇所を提示している．Events にはエラーが起きず通常終了した Normal とエラーが起きて異常終了した Warning がある．文のはじめに Normal と Warning と書かれており，その後に処理内容やエラー内容が書かれる．

エラー原因がある箇所を提示するステータスの Pod で kubectl describe コマンドの出力に含まれる Events をソースコード 2 に示す．ソースコード 2 の persistentvolumeclaim ”c0a21151” not found の部分から，”c0a21151”という名前の persistentvolumeclaim が見つからないという原因である．また，persistentvolumeclaim の名前を書いてある場所がエラー箇所である．

しかし，ステータスによってはエラー箇所を提示してくれない．

エラー原因がある箇所を提示しないステータスの Pod で kubectl describe コマンドの出力に含まれる Events をソースコード 3 に示す．ソースコード

ソースコード 2 エラー原因がある箇所を提示するステータスの Events

```
1  Warning  FailedScheduling  32s
   default-scheduler  0/4 nodes are
   available: persistentvolumeclaim
   "c0a21151" not found .  preemption:
   0/4 nodes are available: 4 Preemption
   is not helpful for scheduling .
```

ソースコード 3 エラー原因がある箇所を提示しないステータスの Events

```
1  Normal  Pulled  54m (x1822 over 6d17h)
    kubelet  Container image
   "nginx:alpine" already present on
   machine
2  Warning  BackOff  4m47s (x43020 over
   6d17h)  kubelet  Back-off restarting
   failed container nginx-proxy in pod
   nginx-proxy-deployment-6b85fdbc59-9ck4f
   _c0a21151-test1(fd959b7d-05b5-44bf-a4de-
   a364697cd376)
```

3 の 2 行目を日本語訳すると，”警告 バックオフ 4 分 47 秒 (6 日 17 時間を超える x43020) kubelet Back-off が nginxproxydeployment6b85fdbc599ck4f_c0a21151-test1(fd959b7d05b544bfa4de a364697cd376) 内の失敗したコンテナ nginxproxy を再起動しています”となり，エラー原因があるエラー箇所を提示していない．そのためエラーの原因と箇所を特定するために何度もコマンドを実行する必要がある．

エラー解決の手順

エラー解決の手順を**図 2** に示す．kubectl get pod コマンドを実行し，Pod のステータスが Running になっていないときエラーが起きている．学生は，Pod のステータスが Running になるまで YAML ファイルの修正と kubectl apply コマンドの実行を繰り返し行う．YAML ファイルの修正を行うためにエラーが起きた原因を特定する必要がある．kubectl describe コマンドを使用すると Pod の詳細情報をみることができるため，kubectl describe コマンドを実行しエラーが起きた原因を特定する．YAML ファイルや Pod の構築環境にあるエラーの原因を直す．再度，kubectl apply コマンドを実行しコンテナに YAML ファイルの内容を適用する．kubectl get pod コマンドを実行し Pod のステータスが Running 以外のときエラーが起きているため再度エラーの原因を特定し直す必要がある．kubectl get pod コマンドを実行し Pod のステータスが Running になったときエラーが起きておらず Pod が起動している．

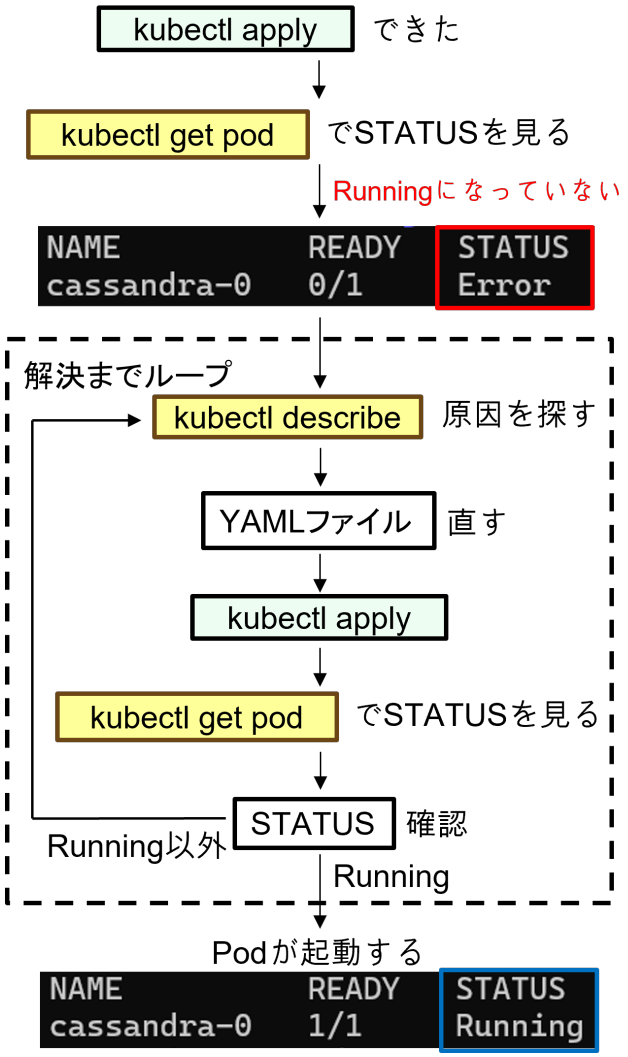

図 2 エラー解決の手順

エラーの原因を特定し，エラーが起きないように直してから，YAML ファイルを適用するという修正の繰り返しが Kubernetes の知識のない学生には負担になる．学生は簡単に Kubernetes でリソースを作れない．学生は YAML ファイルを適用する段階で Running になるのを待たずに，エラーになる箇所が事前にわかって修正することができると，学生が繰り返し YAML ファイルを修正する必要がなくなる．現状だと，kubectl apply コマンドを実行しても Running になるまで待つ必要がある．また，kubectl apply コマンドや kubectl describe コマンドという kubectl コマンドを打つ作業や，YAML ファイルをテキストエディタで修正する作業が手間になる．学生にとってこの待ち時間や修正を繰り返す回数の増加が負担になる．

各章の概要

第 2 章以降の概要は以下の通りである．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 章では基礎実験や，実験環境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．

2. 関連研究

構文エラーを高速に検出することができ，パーサーコンビネータが消費する RAM リソースが少なくなる論文がある [19]．一般的なプログラミング言語 (Java，C，Python) の文法がエラー検出対象であり，YAML 形式は対象ではない．

Kubernetes の YAML ファイルの内容をチェックができるツールがある．既存の Lint ツール (KubeLinter[*1], Kubeval[*2]) のみでは，Persistent Volume に代表される状態をもつリソースの作成に失敗する原因は特定できない．

Java のコードスタイル改善ツールを開発し，より一般的なセマンティックスタイルの問題を検出しリファクタリングする論文がある [20]．リファクタリングする対象は Java であり，YAML 形式ではない．

Java Native Interface(JNI) を介してネイティブ コードで発生したエラーを検査し，報告するための静的解析フレームワークを提案している論文がある [21]．エラーを検査する対象は JNI であり，YAML 形式ではない．

3. 提案方式

本稿の提案では，Kubernetes を使用して作成した Pod にエラーが起きてから，ルールファイルを使用し起動するまでの手間を削減する．Pod が起動しないエラーの原因を特定するために，Pod のステータスと kubectl describe コマンドで取得できる詳細情報に含まれる Events を利用する．Pod のステータスによってエラーが起きる箇所が異なる．そのためエラーごとにエラー原因やエラーが起きている箇所を分類する．ステータスと Events の内容でエラー箇所を分類する．分類内容をルールとし，ルールをまとめたものをルールファイルとする．ルールファイルの内容例を**表 1** に示す．ルールファイルには，1 行にステータス

表 1 ルールファイルの内容例

ステータス	Events	エラー箇所
Pending	Warning FailedScheduling …	claimName
Error	Warning Failed …	value
CrashLoopBackOff	Warning BackOff …	args

と Events の内容とエラー原因があった箇所のキーを書き込む．このルールファイル作成作業を YAML ファイルが Running にならないたびに行う．ルールファイル作成作業を行うたびにルールファイルのルールが増え，エラー原因を特定するときに対応できるエラーが増える．

ルールファイルの作り方を**図 3** に示す．例として，ルールファイルを作成する時に用意する YAML ファイルのリソースは Deployment とする．ルールファイルの作成は kubectl apply コマンドが実行されたとき行われる．YAML ファイルが初めて kubectl apply コマンドで実行されたときステータスが Running 以外の時 YAML ファイルをコピーし保存する．この YAML ファイルが図 3 の Running にならない YAML ファイルとなる．同じ名前の YAML ファイルが kubectl apply コマンドで実行されステータスが Running になったとき再度 YAML ファイルをコピーする．この YAML ファイルが図 3 の Running になる YAML ファイルとなる．ルールファイルには間違えている箇所とステータスと Events を書き込む．間違えている箇所は，Running にならない YAML ファイルと Running になる YAML ファイルを diff コマンドで比較することで取得できる．ステータスと Events は，Running にならない YAML ファイルを kubectl apply コマンドで適用し作成した Pod から取得する．ステータスは kubectl get pod コマンドで取得し，Events は kubectl describe コマンドで取得できる．基礎実験の結果から，対象のステータスは Pending，Error，CrashLoopBackOff の 3 種類とする．

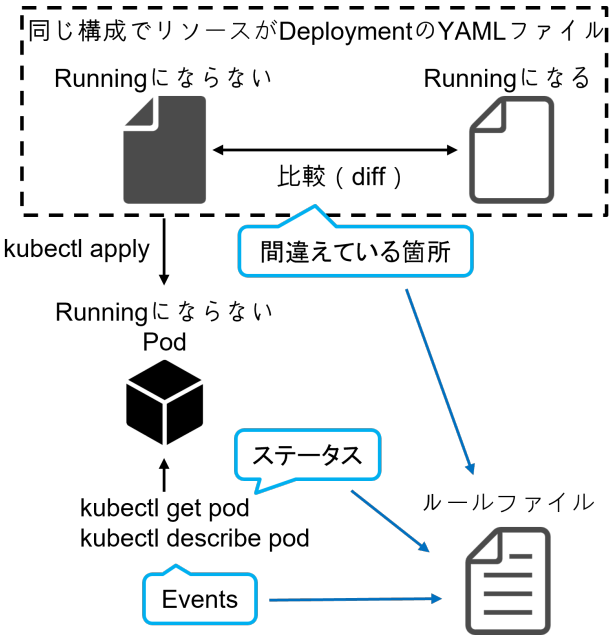

図 3 ルールファイルの作り方

エラー原因がある箇所をターミナル表示する方法は，エラー原因がある箇所を提示するステータスとエラー原因がある箇所を提示しないステータスで方法が異なるため 2 通りある．1 つ目のエラー原因がある箇所を Events で提示するステータスに対しては，kubectl describe pod コマン

[*1] https://kubeval.instrumenta.dev/
[*2] https://docs.kubelinter.io/

ドで取得できる詳細情報に含まれる Events の内容を日本語でターミナル表示する．エラー原因がある箇所を Events で提示するステータスのエラー原因がある箇所をターミナルに表示する方法を**図 4** に示す．Events にエラー原因の

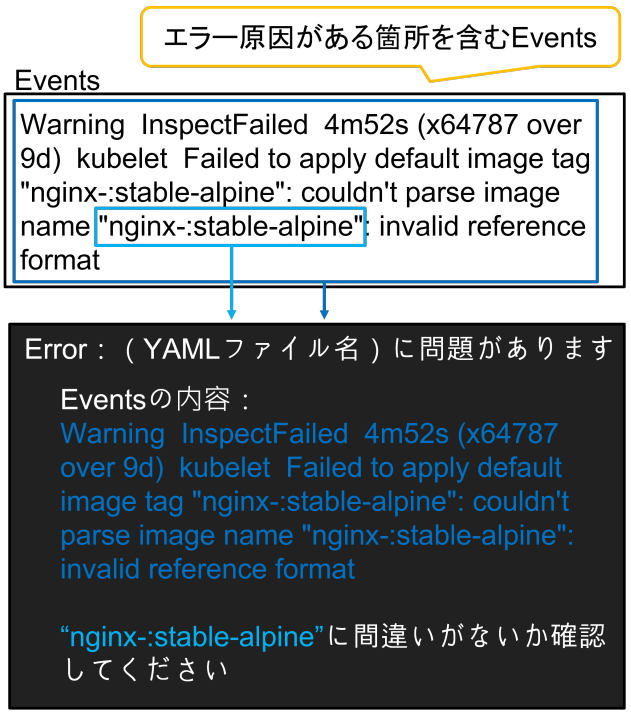

図 4 エラー原因がある箇所を Events で提示するステータスのときターミナルに表示する方法

ある箇所が分かる内容が含まれているため，Events の中にあるエラー原因がある箇所を取り出しターミナルに表示する．ダブルクォーテーションで囲まれている部分が Pod に適用されている YAML ファイルに書かれている内容から取り出されている．Events の内容で指摘されているこの部分がエラー原因のある箇所である．ターミナル表示では，Events の内容とこのダブルクォーテーションで囲まれている部分を確認するという内容を表示する．

2 つ目のエラー原因がある箇所を提示しないステータスに対しては，エラー毎にエラー原因がある箇所を分類し特定する．エラー原因がある箇所を Events で提示しないステータスでルールファイルを使用しエラー箇所をターミナルに表示する方法を**図 5** に示す．Events の中にエラー原因がある箇所が分かる内容が含まれていないため，ルールファイルを使用する．Pod のステータスと Events の内容からエラーが起きる箇所をルールファイルから取得する．Pod のステータスと Events の内容をルールファイルのステータスと Events と照合し，完全一致のとき Pod のステータスと Events の内容とエラー箇所をターミナルに表示する．

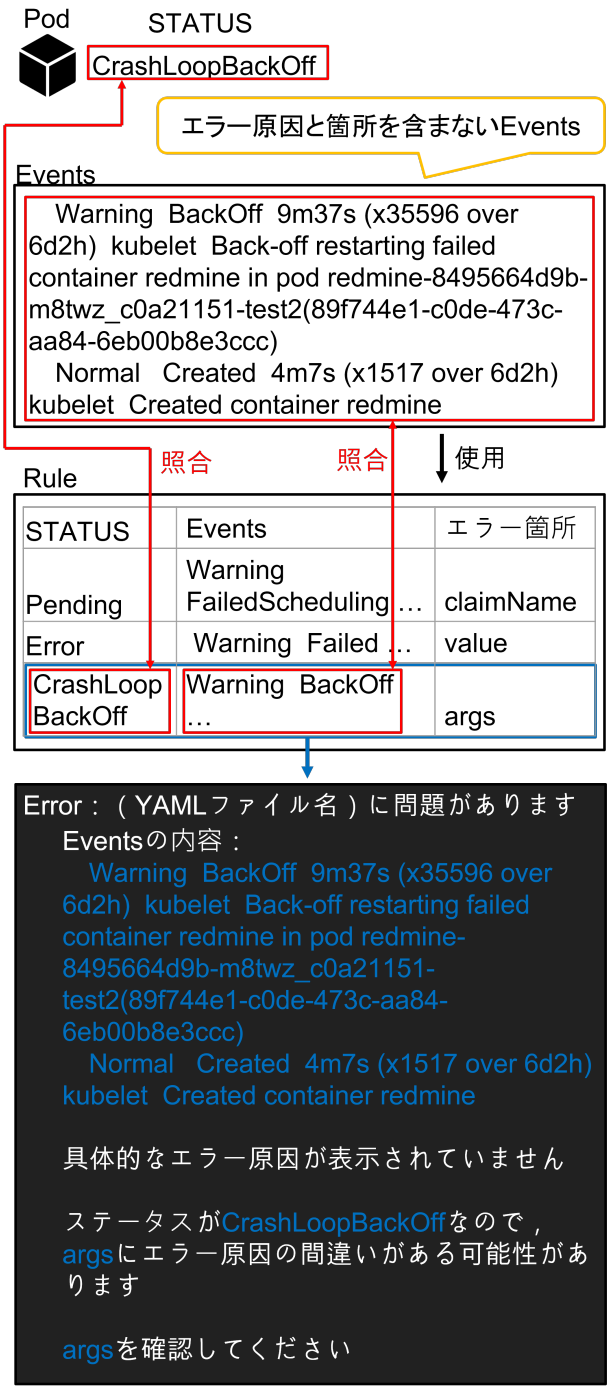

図 5 ルールファイルを使用してエラー箇所をターミナルに表示する方法

ユースケース・シナリオ

Kubernetes を使用してソフトウェアを配置する Site Reliability Engineering の実習であるプロジェクト実習 [IT・3] を受講している学生をユースケースとして想定する．エラーが起きてから学生が解決するまでの作業を**図 6** に示す．課題の中にはソフトウェアを作成するために Kubernetes を使用し Pod を作成するとき，エラーが起こる．Pod のステータスが Running 以外の時はエラーが起きている．エラーを解決するためにはエラー原因がある箇所を特定し，

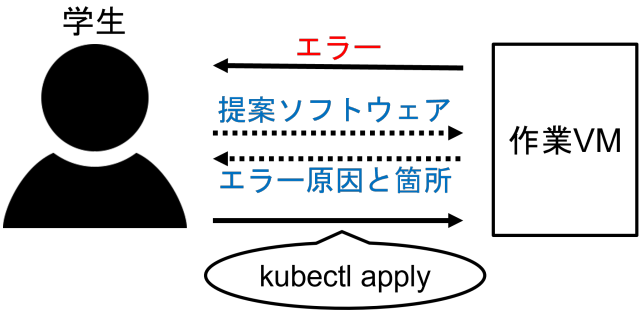

図 6 エラーが起きてから学生が解決するまでの作業

直す必要がある．提案ソフトウェアのルールファイルを使用することでエラー原因がある箇所が分かる．学生はエラー原因がある箇所が分かることでエラー原因を正確に直すことができ，kubectl apply コマンドを 1 回打つことでエラーを解決できる．

4. 実装

提案より，Pod のステータスが Running 以外であるとき kubectl describe コマンドで取得できる詳細情報に含まれる Events の内容とルールファイルを照合し，ターミナルにエラー原因がある箇所を出力するプログラムを作成する．プログラムは Python で作成する．Kubernetes ライブラリにある client モジュールと config モジュール，subprocess ライブラリを使用する．Kubernetes ライブラリの client モジュールは，Kubernetes API を操作するためのクライアントモジュールである．このモジュールを使用して Python スクリプトから Kubernetes リソースを管理する．Kubernetes ライブラリの config モジュールは，Kubernetes クライアントを設定するためのモジュールである．このモジュールを使用して Kubernetes クライアントがどのクラスタに接続するかを設定する．subprocess ライブラリは，Python スクリプトからシェルコマンドを実行するためのライブラリである．

Pod の起動エラー原因がある箇所をターミナル表示するプログラムの構造を**図 7** に示す．

① kubectl apply コマンドを実行し Pod が作成される．
② Pod の情報を取得する．このとき取得する情報とは，kubectl get pod コマンドを実行したときに取得できる情報のことである．Pod の情報から Pod のステータスが分かる．ステータスが Running のとき，エラーは起きていないためプログラムを終了する．
③ ステータスが Running 以外のとき，kubectl get pod コマンドから取得した情報に含まれる Pod の名前と Pod のネームスペースを使用して，kubectl describe コマンドを実行する．kubetl describe コマンドで取得できる詳細情報に含まれる Events の内容を取得する．
④ Events でエラー原因がある箇所を提示しているか確認する．確認方法として，Events にダブルクォーテーションが含まれているかで判断する．含まれている場合は Events がエラー原因がある箇所を提示しているとし，含まれていない場合は Events がエラー原因がある箇所を提示していないとする．
⑤ Events でエラー原因がある箇所を提示するとき，Events の内容を日本語でターミナル表示する．
⑥ Events でエラー原因がある箇所を提示しないとき，ルールファイルを使用し，エラーごとにエラー原因がある箇所を特定する．特定したエラー原因がある箇所をターミナルで表示する．

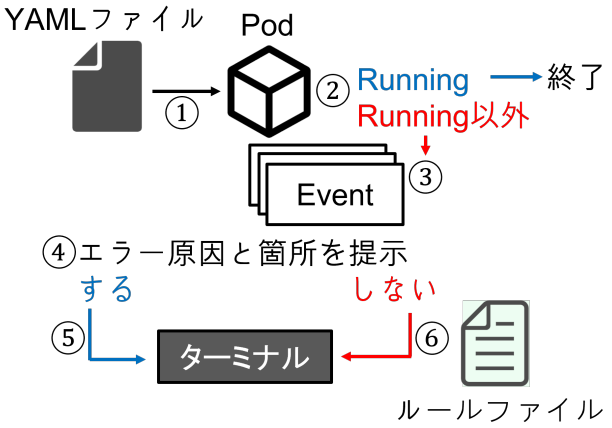

図 7 Pod の起動エラー原因がある箇所をターミナル表示するプログラムの構成

次にルールファイルを作成するプログラムの構造を説明する．ルールファイルには Pod のステータスと kubectl describe コマンドで取得できる Events とエラー原因がある箇所を書き込む．YAML ファイルを kubectl apply コマンドで作成した Pod のステータスが Running となるか Running 以外となるかで動作が異なる．

Pod のステータスが Running 以外となる YAML ファイルの場合の動作を**図 8** に示す．

① kubectl apply コマンドを実行し Pod が作成される．
② kubectl get pod コマンドでステータスを確認する．この図 8 では Pod のステータスは Running 以外とする．
③ ステータスが Running ではないとき，kubectl describe コマンドを実行し Pod の詳細情報に含まれる Events を取得する．
④ ステータスと Events を使用してルールファイルに既にこのエラー原因の箇所が分かるルールが記載されているか確認する．
⑤ ステータスと Events の内容がルールファイルに記載されていた場合，YAML ファイル名とエラー内容とルールファイルに記載されているエラー箇所をターミナルに表示する．
⑥ ステータスと Events の内容がルールファイルに記載

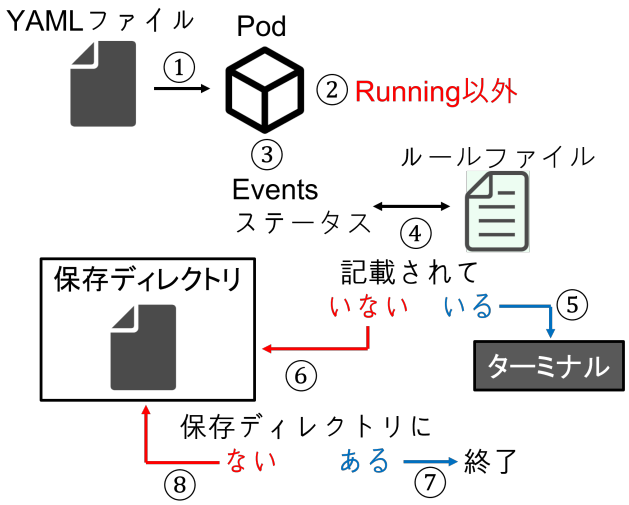

図 8 ルールファイル作成プログラムの Pod のステータスが Running 以外となる YAML ファイルの場合

されていなかった場合，YAML ファイル保存ディレクトリに同じ名前の YAML ファイルが保存されているか確認する．

⑦ YAML ファイル保存ディレクトリに YAML ファイルがあるなら終了する．

⑧ YAML ファイル保存ディレクトリに YAML ファイルがないなら YAML ファイル保存ディレクトリに YAML ファイルをコピーする．コピーした YAML ファイルは，はじめてステータスが Running 以外になった YAML ファイルである．

Pod のステータスが Running となる YAML ファイルの場合の動作を**図 9** に示す．

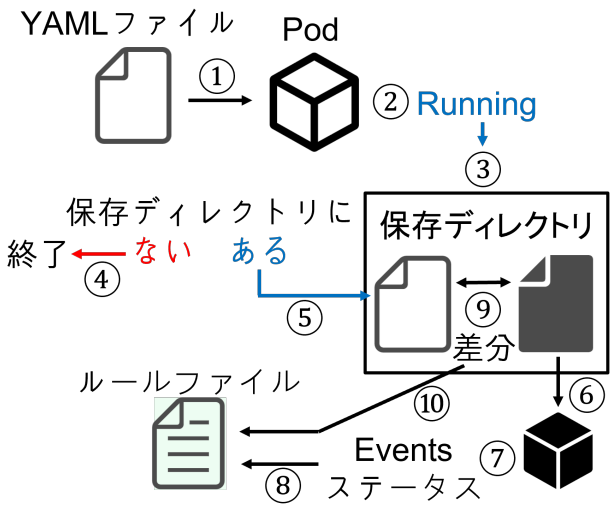

図 9 ルールファイル作成プログラムの Pod のステータスが Running となる YAML ファイルの場合

① kubectl apply コマンドを実行し Pod が作成される．

② kubectl get pod コマンドでステータスを確認する．この図 8 では Pod のステータスは Running とする．

③ YAML ファイル保存ディレクトリに同じ名前の YAML ファイルが保存されているか確認する．

④ YAML ファイル保存ディレクトリに YAML ファイルがないなら終了する．

⑤ YAML ファイル保存ディレクトリに YAML ファイルがあるなら YAML ファイル保存ディレクトリに YAML ファイルをコピーする．コピーした YAML ファイルは過去にステータスが Running 以外になったことがあり，ステータスが Running 以外になったときの YAML ファイルが YAML ファイル保存ディレクトリにある．

⑥ YAML ファイル保存ディレクトリに保存されている Running 以外になったときの YAML ファイルを kubectl apply コマンドで実行し Pod を作成する．

⑦ 作成した Pod に対して kubectl get pod コマンドと kubectl describe コマンドを実行しステータスと Pod の詳細情報に含まれる Events を取得する．

⑧ ステータスと Events をルールファイルに書き込む．

⑨ YAML ファイル保存ディレクトリにあるステータスが Running になる YAML ファイルと Running 以外になる YAML ファイルを diff コマンドで比較し差分を取得する．この差分はエラーが起きた原因がある行である．

⑩ 取得した差分の行のキーをエラー箇所としてルールファイルに書き込む．

5. 評価実験

エラー解決までに kubectl apply コマンドを実行した回数と提案プログラムで正しいエラー箇所を特定できているかの精度を評価する．実際にプロジェクト実習を受講している学生に行なってもらう．そのため評価実験は，後期のプロジェクト実習で行う．実習は 1 回 5 時間のため行う時間は 5 時間で人数は受講生が約 40 人のため，40 人を 2 つのグループに分け提案手法を使うグループと使わないグループを作り，提案手法の有無で違いがあるかを比較する．

まず，エラー解決までに kubectl apply コマンドを実行した回数の実験方法は，学生が kubectl apply コマンドを Pod が起動するまでに実行した回数で評価する．kubectl apply コマンドはエラーが発生したときから数え始め，Pod のステータスが Running になったとき終了する．実習中に実施し，課題を学生が解くときに提案方式を実装したソフトウェアを使用してもらうことで評価実験を行う．エラー解決までに kubectl apply コマンドを実行した回数は提案方式を実装したソフトウェア内で行う．kubectl apply コマンドを実行した回数をその時のルールファイルにあるルール数ごとに集計し，実行回数を比較する．

次に，提案プログラムで正しいエラー箇所を特定できているかの精度の実験方法は，プロジェクト実習で起きたエ

ラーの YAML ファイルの内，原因があった箇所を正しく表示できた数で精度を計算し評価する．計算方法として，何割の YAML ファイルで正しくエラー箇所をターミナル表示できたか割合を算出する．プロジェクト実習で発生した Pod を作成する際，YAML ファイルは適用できたが Pod が起動しないというエラーが起きた時の YAML ファイルを全て集め，評価実験に使用する．提案プログラムで正しいエラー箇所を特定できているかは次の kubectl apply コマンドの実行時に Pod のステータスが Running になっているかを手作業で行う．

実験環境

実験環境はプロジェクト実習の環境である．実験環境の VM を**図 10** に示す．ルールファイルと YAML ファイル

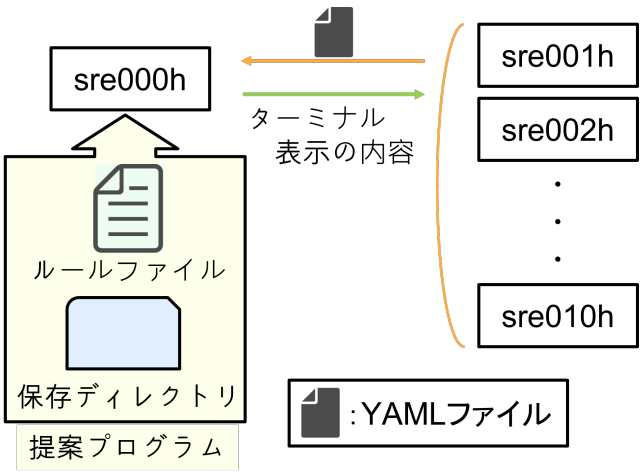

図 10 実験環境の VM

を保存するディレクトリを配置するためのクライアント VM を sre000h とする．kubectl コマンドを学生が実行する VM の sre とグループ番号と h の組み合わせの VM であり，プロジェクト実習のグループは 10 個あるため，クライアント VM が管理する VM は 10 個となる．学生が使用する VM からは YAML ファイルが送られ，クライアント VM からは提案プログラムの出力であるターミナル表示の内容を送る．

基礎実験

基礎実験として，プロジェクト実習 [IT・3] で学生が Kubernetes を使って Pod を立てるときにコンテナを適用させることはできるが Pod は起動しないというエラーが発生した YAML ファイルを集めた．14 回の実習の内 6 回の実習で 18 個の YAML ファイルが集まった．集めた YAML ファイルからエラー時の Pod のステータスと間違えている原因を特定した．

集めた YAML ファイルのエラー時の Pod のステータスのグラフを**図 11** に示す．図 11 からエラーが起きた時の Pod

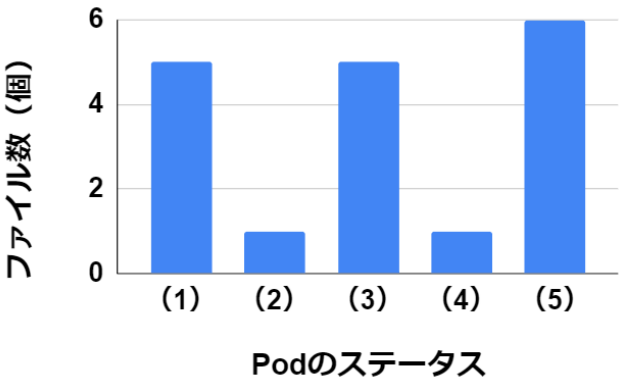

図 11 エラー時の Pod のステータス

のステータスは CrashLoopBackOff，CreateContainerConfigError，Error，InvalidImageName，Pending の 5 種類があった．図 11 の (1) のステータスは CrashLoopBackOff，(2) のステータスは CreateContainerConfigError，(3) のステータスは Error，(4) のステータスは InvalidImageName，(6) のステータスは Pending である．

各ステータスの YAML ファイル数と割合を**表 2** に示す．18 個の YAML ファイルの内，各ステータスの YAML

表 2 各ステータスの YAML ファイル数と割合

ステータス	YAML ファイル数 (個)	割合 (%)
(1)CrashLoopBackOff	5	約 27.8
(2)CreateContainer ConfigError	1	約 5.6
(3)Error	5	約 27.8
(4)InvalidImageName	1	約 5.6
(5)Pending	6	約 33.3

ファイル数と割合の内訳は CrashLoopBackOff が 5 個で約 27.8%，CreateContainerConfigError が 1 個で約 5.6%，Error が 5 個で約 27.8%，InvalidImageName が 1 個で約 5.6%，Pending が 6 個で約 33.3%であることが分かる．CrashLoopBackOff と Error と Pending が占める割合は約 88.9%であるため，CrashLoopBackOff と Error と Pending のエラーを解決できるとエラーが発生したとき約 88.9%を解決できる．よって CrashLoopBackOff と Error と Pending のエラー原因を解決する対象とする．集めた YAML ファイルの CrashLoopBackOff と Error と Pending のエラーが起きた原因を特定した．

集めた YAML ファイルの CrashLoopBackOff と Error と Pending のエラーが起きた原因のグラフを**図 12** に示す．図 12 から CrashLoopBackOff と Error と Pending のエラーが起きた原因は，スペースがない，設定が足りない，インデントがあっていない，文字が多い，文字が足りない，似ている文字の間違いという間違いの 6 種類があった．図 12 の (1) のエラーが起きた原因はスペースがない

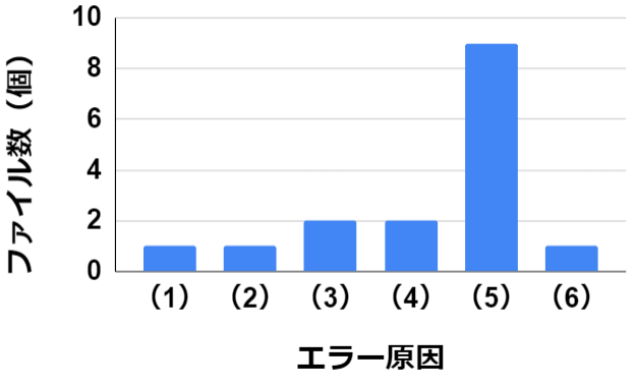

図 12 CrashLoopBackOff と Error と Pending のエラーが起きた原因

という問題であり，(2) のステータスは設定が足りないという問題であり，(3) のステータスはインデントがあっていないという問題であり，(4) のステータスは文字が多いという問題であり，(5) のステータスは文字が足りないという問題であり，(6) のステータスは似ている文字の間違いという問題である．

エラーが起きた原因のインデントがあっていないという問題は，行の段落が YAML 形式で書けていないという間違いである．エラーが起きた原因の文字が足りていないという問題は，PersistentVolumeClaim を PersistenVolumeClaim と書いてあり単語に t が足りてないことや，名前を指定する際にハイフンが無いという文字が足りてない間違いである．実際に学生が間違えた例をソースコード 1 に示す．1 行目は，環境変数の名前を設定している．CASSANDRA SEEDS という名前にしている．2 行目は，Cassandra クラスタのシードノードを設定している．cassandra0.cassandra.svc.cluster.local が設定されている．ソースコード 1 の 2 行目に書かれている cassandra0.cassandra.svc.cluster.local でエラーが起きた原因の間違いがある．正しい Cassandra クラスタのシードノードの名前は cassandra-0.cassandra.svc.cluster.local である．環境変数に指定する値は正確に記述する必要がある．エラーが起きた原因の似ている文字の間違いという問題は，metadata を metadate のように a を e と書いてしまうことやハイフンをアンダーバーと書いてしまう間違いである．エラーが起きた原因の文字が多いという問題は，nginx を nginx-と書いてあり余分な文字が書かれているという間違いである．

また，CrashLoopBackOff と Error と Pending ごとのエラー原因の YAML ファイル数を**表 3** に示す．エラー原因が Pending のとき，(4) の文字が多いという問題が 1 個と (5) の文字が足りないという問題が 4 個と (6) の似ている文字の間違いという問題が 1 個であった．(5) の文字が足りないという問題というエラー原因の 4 個の YAML ファイルの内，エラー箇所が同じ YAML ファイルはなかったためエラー箇所数は 4 個である．そのため Pending のエラー原因を解決するために必要なルール数は，(4) の文字が多いという問題で 1 個と (5) の文字が足りないという問題で 4 個と (6) の似ている文字の間違いという問題で 1 個のため，6 個となる．エラー原因が Error のとき，(3) インデントがあっていないという問題が 1 個と (5) の文字が足りないという問題が 4 個であった．(5) の文字が足りないという問題というエラー原因の 4 個の YAML ファイルの内，エラー箇所が同じ YAML ファイルが 2 個ずつあったためエラー箇所数は 2 個である．そのため Error のエラー原因を解決するために必要なルール数は，(3) インデントがあっていないという問題で 1 個と (5) の文字が足りないという問題で 2 個のため，3 個となる．エラー原因が CrashLoopBackOff のとき，(1) のスペースがないという問題が 1 個と (2) の設定が足りないという問題が 1 個と (3) のインデントあっていないという問題が 1 個と (4) の文字が多いという問題が 1 個と (5) の文字が足りないという問題が 1 個であった．YAML ファイルが複数個あるエラー原因が無い．そのため CrashLoopBackOff のエラー原因を解決するために必要なルール数は，(1) のスペースがないという問題で 1 個と (2) の設定が足りないという問題で 1 個と (3) のインデントあっていないという問題で 1 個と (4) の文字が多いという問題で 1 個と (5) の文字が足りないという問題で 1 個のため，5 個となる．

表 3 CrashLoopBackOff と Error と Pending ごとのエラー原因の YAML ファイル数

ステータス ＼ エラー原因	Pending	Error	CrashLoop BackOff
(1) スペースがない	0	0	1
(2) 設定が足りない	0	0	1
(3) インデントがあっていない	0	1	1
(4) 文字が多い	1	0	1
(5) 文字が足りない	4	4	1
(6) 似ている文字の間違い	1	0	0

以上の結果から，集計した YAML ファイルの内ステータスが CrashLoopBackOff と Error と Pending になる YAML ファイルのエラー原因を解決するために必要なルールは，14 個である．

基礎実験の実験環境

基礎実験の実験環境を**図 13** に示す．実験は研究室で管理されており，研究室に在籍している人は誰でも使える共

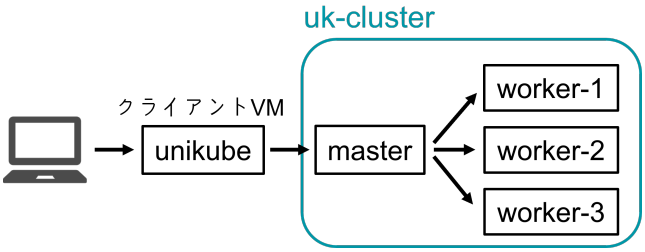

図 13 実験環境である Kubernetes クラスターの構成

有 Kubernetes クラスターの unikube で行った．unikube は Kubernetes の軽量版として改良されたプラットフォームである k3s で構築されている．共有 Kubernetes クラスターは，unikube クライアントである VM と unikube クラスターである uk-cluster で構成されている．ユーザーがアクセスするときは，unikube クライアントである VM にログインして kubectl コマンドを実行する．クラスターには 4 つのノードがあり，マスターノード (master) が 1 つとワーカーノード (worker-1, worker-2, worker-3) が 3 つある．マスターノードはワーカーノードを管理している．unikube クライアントである VM は，CPU が 2 コア，メモリが 4GB，ストレージが 100GB で構成されている．マスターノードは，VM は CPU が 8 コア，メモリが 12GB，ストレージが 50GB で構成されている．ワーカーノードは，VM は CPU が 8 コア，メモリが 8GB，ストレージが 50GB で構成されている．

6. 議論

本稿の提案では，kubectl describe コマンドで取得した Pod の詳細情報のうち Events のみを取り出している．しかし，Events の中には異常終了したエラーだけでなく通常終了した内容もあるため，エラーの原因特定に必要のない部分も含まれている．例えば，kubectl describe コマンドで取得した Pod の詳細情報のうち Events にある Normal がエラーの原因特定に必要のない部分である．そのため Warning を含む行のみ取り出し，エラー原因特定に使用する．

本稿の提案にあるルールファイルでは，1 度目のエラーに対して何も解決方法を提示しないため YAML ファイルを自力で直すしかない．そのため，1 度目のエラーにも対応できるように，似ているエラーを探し提示する．似ているエラーの探し方は，1 度目のエラーの Events に含まれる単語とルールファイルに保存されている 1 度目のエラーのステータスの Events に含まれる単語を照合し，最も一致した単語が多かった Events のエラー箇所をターミナルに表示する．例えば，1 度目のエラーのステータスが CrashLoopBackOff のとき，ルールファイルに保存されているルールの内ステータスが CrashLoopBackOff のルールが対象となる．1 度目のエラーの Events に含まれる単語と対象となったルールの Events に含まれる単語で一致する単語を照合し，最も一致する単語が多かった Events のエラー箇所がターミナルに表示される．

本稿の提案にあるルールファイルの作成方法は，新しいエラーが出るたびにルールが追加され，1 つのルールで 1 つのエラーしか対応できないためエラーの数だけルールが増えてしまう．そのため，少ないルールで多くのエラーに対応できるように，ステータスとエラー箇所が同じルールは Events を複数にしルールは 1 つにまとめることでルール数を減らすことができる．

7. おわりに

プロジェクト実習 [IT・3] では VM 内で Kubernetes を使用しソフトウェアを構築するが，そのとき Pod を作成し起動する必要がある．Pod を起動するときエラーが起きることがあり，エラーを解決するためにエラー原因がある箇所を特定する必要があるが手間になる．提案として，kubectl describe コマンドで取得できる詳細情報に含まれる Events にエラー原因がある箇所があるときは Events の内容を日本語でターミナルに表示する．kubectl describe コマンドで取得できる詳細情報に含まれる Events にエラー原因がある箇所がないときは，ルールファイルを使用しエラー箇所を特定する．基礎実験では，プロジェクト実習 [IT・3] でエラーが起きる YAML ファイルを 14 回の内 6 回の実習で 18 個集めた．集めた YAML ファイルで作成された Pod のステータスを集計した結果，CrashLoopBackOff と Error と Pending の割合で約 88.9%を占めた．CrashLoopBackOff と Error と Pending がの Pod を対象としてエラー原因を分類し，集計した結果一番多かった問題は，文字が足りないという問題であり 9 個で約 56.3%だった．以上のことから，CrashLoopBackOff と Error と Pending のエラー原因を解決すると，エラーが発生したとき約 88.9%を解決できる．また，その原因の約 56.3%は文字が足りないという問題であり，集計した YAML ファイルの内ステータスが CrashLoopBackOff と Error と Pending のエラー原因を解決するために必要なルールは，14 個である．評価は，エラー解決までに kubectl apply コマンドを実行した回数と提案プログラムで正しいエラー箇所を特定できているかの精度を評価する．評価実験は，後期のプロジェクト実習でプロジェクト実習を受講している学生に，課題を解くときに提案方式を実装したソフトウェアを使用してもらうことで行う．エラー解決までに kubectl apply コマンドを実行した回数は提案方式を実装したソフトウェア内で行い，提案プログラムで正しいエラー箇所を特定できているかは次の kubectl apply コマンドの実行時に Pod のステータスが Running になっているかで判断する．

参考文献

[1] Chebotko, A., Kashlev, A. and Lu, S.: A Big Data Modeling Methodology for Apache Cassandra, *2015 IEEE International Congress on Big Data*, pp. 238–245 (online), DOI: 10.1109/BigDataCongress.2015.41 (2015).

[2] Dhingra, S., Sharma, S., Kaur, P. and Dabas, C.: Fault tolerant streaming of live news using multi-node Cassandra, *2017 Tenth International Conference on Contemporary Computing (IC3)*, pp. 1–5 (online), DOI: 10.1109/IC3.2017.8284310 (2017).

[3] Han, J., E, H., Le, G. and Du, J.: Survey on NoSQL database, *2011 6th International Conference on Pervasive Computing and Applications*, pp. 363–366 (online), DOI: 10.1109/ICPCA.2011.6106531 (2011).

[4] Kebbani, N., Tylenda, P. and McKendrick, R.: *The Kubernetes Bible: The definitive guide to deploying and managing Kubernetes across major cloud platforms*, Packt Publishing Ltd (2022).

[5] German, K. and Ponomareva, O.: An Overview of Container Security in a Kubernetes Cluster, *2023 IEEE Ural-Siberian Conference on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT)*, pp. 283–285 (online), DOI: 10.1109/USBEREIT58508.2023.10158865 (2023).

[6] 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).

[7] Phuc, L. H., Phan, L.-A. and Kim, T.: Traffic-Aware Horizontal Pod Autoscaler in Kubernetes-Based Edge Computing Infrastructure, *IEEE Access*, Vol. 10, pp. 18966–18977 (online), DOI: 10.1109/ACCESS.2022.3150867 (2022).

[8] Lehtinen, K.: Scaling a Kubernetes Cluster (2022).

[9] Wennerström, W.: Active Assurance in Kubernetes (2021).

[10] Kenny, J. and Knight, S.: Kubernetes for HPC Administration., Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States); Sandia ⋯ (2021).

[11] 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).

[12] 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).

[13] 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).

[14] Muddinagiri, R., Ambavane, S. and Bayas, S.: Self-Hosted Kubernetes: Deploying Docker Containers Locally With Minikube, *2019 International Conference on Innovative Trends and Advances in Engineering and Technology (ICITAET)*, pp. 239–243 (online), DOI: 10.1109/ICITAET47105.2019.9170208 (2019).

[15] Schrettenbrunner, J.: Migrating Pods in Kubernetes, PhD Thesis (2020).

[16] Rasheed, S., Dietrich, J. and Tahir, A.: Laughter in the Wild: A Study Into DoS Vulnerabilities in YAML Libraries, *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pp. 342–349 (online), DOI: 10.1109/TrustCom/BigDataSE.2019.00053 (2019).

[17] Khalel, M. M., Arul Pugazhendhi, M. and Raj, G. R.: Enhanced Load Balancing in Kubernetes Cluster By Minikube, *2022 International Conference on Smart Technologies and Systems for Next Generation Computing (ICSTSN)*, pp. 1–5 (online), DOI: 10.1109/ICSTSN53084.2022.9761317 (2022).

[18] Pulcinelli, L. E. G., Pedroso, D. F. and Bruschi, S. M.: Conceptual and Comparative Analysis of Application Metrics in Microservices, *2023 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)*, pp. 123–130 (online), DOI: 10.1109/SBAC-PADW60351.2023.00028 (2023).

[19] Kuznetsov, M. and Firsov, G.: Syntax Error Search Using Parser Combinators, *2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus)*, pp. 490–493 (online), DOI: 10.1109/ElConRus51938.2021.9396311 (2021).

[20] Lundager, M. and Berqvist, A.: Finding and Resolving Common Code Style Errors in Java (2024).

[21] Li, S. and Tan, G.: Finding bugs in exceptional situations of JNI programs, *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, New York, NY, USA, Association for Computing Machinery, p. 442–452 (online), DOI: 10.1145/1653662.1653716 (2009).

