ルールファイルの比較によるKubernetesのPod起動時に発生するエラー原因の特定と修正

手塚 雄星[1]　小山 智之[2]　串田 高幸[1]

概要：クラウド・分散システム研究室では，Prometheus や Alertmanager，Grafana を使い監視システムを構築している．監視システムのメトリクスがオペレーションミスやハードウェアの故障にともなうデータの消失を防ぐために，Prometheus のバックアップを実装している．バックアップの開発作業において開発者は YAML ファイルを作成し，kubectl apply コマンドを使い Pod を起動させる．Pod が起動する際にエラーが発生すると，開発者は Pod のログや失敗した YAML ファイルを確認し，エラーの原因を特定し YAML ファイルを修正する．開発者が Kubernetes の知識が不足していると，エラーの原因特定や YAML ファイルの修正に時間や手間がかかる．基礎実験では，開発作業にかかる時間の内訳を計測した．作業にかかる時間の合計は 2820 秒である．インターネット検索にかかる時間が合計 1680 秒で最も長かった．kubectl get コマンドを実行して確認する時間が合計 1020 秒で二番目に長かった．提案では，Pod の起動に失敗した YAML ファイルをもとに作成したルールファイルと YAML ファイルを比較することで開発作業で発生したエラーの原因特定にかかる時間を短縮する．実験環境では，Python の開発環境を構築する．評価方法では，提案手法で間違った箇所が修正された YAML ファイルのうち，正しく修正された YAML ファイルの割合を適合率で評価を行う．

1. はじめに

背景

コンテナの仮想化は現代の標準的な技術で使用されている [1]．コンテナとは，軽量な OS レベルの仮想化技術であり，アプリケーションとその依存関係をリソースが分離されたプロセス内で実行できるようにするものである [2]．クラウド・分散システム研究室 (CDSL) では，仮想化環境として VMware ESXi を複数台運用しており，そのうち「Lily」と名付けられた ESXi ホスト上に Prometheus の監視システムが構築されている．Prometheus は，オープンソースの監視のツールである [3]. Kubernetes とは，コンテナ化されたアプリケーションのデプロイ，管理，スケーリングを自動化するためのオープンソースのオーケストレーションプラットフォームである [4]．コンテナのソリューションやサービスを管理するために使用されて，宣言型の構成や自動化を促進する [5]．Pod，Deployment，Service のリソース定義を使い，同一ホスト上でのコンテナの連携や，ゼロダウンタイムでのローリングアップデート，サービスの負荷分散，永続ストレージの割り当てを構成する．また，コマンドラインツールの kubectl をもちいることで，容易な操作が可能である．CDSL の Prometheus の監視システムでは Kubernetes クラスタ上で稼働しており，合計 7 つの Pod によって構成されている [6]．具体的には，「Lily」上に配置された Prometheus 本体の Pod が 1 つ，Blackbox Exporter の Pod が 1 つ，VMware Exporter の Pod が 1 つ稼働している．別の ESXi ホストである「Iris」上の VM に配置された Node Exporter として 4 つの Pod が稼働している．このような構成により，仮想マシンや物理ノード，サービスの稼働状況，リソース消費量のメトリクスを収集や可視化する．収集されたメトリクスは，Grafana を用いて可視化され，CDSL 内のインフラ運用や監視に活用されている．

Grafana では Prometheus のデータが表示されない不具合が発生している．原因として，Prometheus のデータディレクトリの破損や，Pod の再スケジューリングによる一時的なデータ損失である．このような状況において，Prometheus に蓄積されたメトリクスデータの損失は，障害分析やパフォーマンス評価に支障をきたす [7]．CDSL では，Prometheus のメトリクスデータを定期的にバックアップする仕組みを構築している．Prometheus のバックアップには Kubernetes の CronJob リソースを使う．CronJob

[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

では PushGateway から取得した Prometheus のメトリクスを含むスナップショットデータを，定められたタイミングで自動的に保存する．PushGateway はアプリケーションのメトリクスを Prometheus に届けるための中継するソフトウェアである．バックアップ処理はシェルスクリプトにより実行される．バックアップを実装する開発者は，YAML ファイルを記述し，kubectl apply コマンドを使い，CronJob のリソースを作成する．YAML ファイルは，その簡潔さと可読性の高さから，設定ファイルやデータ保存で最も広く使用されている言語の 1 つである [8].

CDSL では，佐藤健斗さんが開発者として作業を行っている．（以降，佐藤さんとする）図 1は佐藤さんによる Prometheus をバックアップする開発作業を表している．Prometheus をバックアップする開発作業の流れは以下の

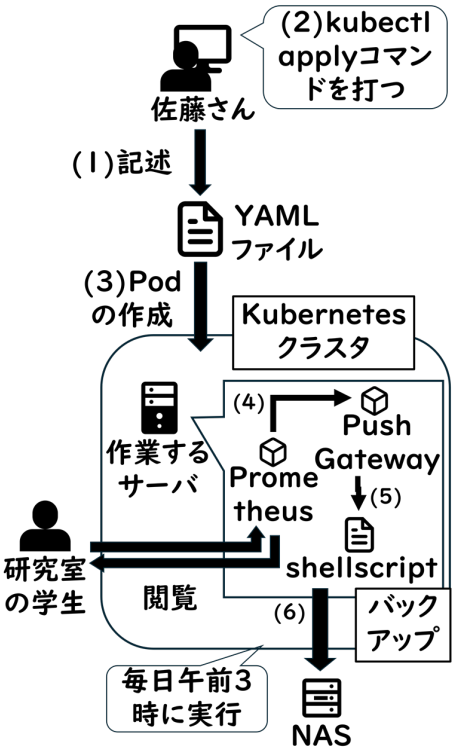

図 1　佐藤さんによる Prometheus をバックアップする開発作業

6 つである．

(1) YAML ファイルに記述する
(2) kubectl apply コマンドを実行する
(3) Pod を作成
(4) Prometheus から PushGateway にメトリクスを収集する
(5) shellscript が PushGateway にメトリクスを送信する
(6) CronJob で NAS にバックアップする

開発作業の流れから分けた 6 つの説明をする．（1）では，CronJob リソースの YAML ファイルを記述する．（2）では，kubectl apply コマンドを実行する．（3）では，YAML ファイルをもとにして Kubernetes クラスタに Pod を作成する．（4）では，Prometheus から PushGateway に対して pull してメトリクスデータを収集する．（5）では，shellscript が PushGateway に対して Push してメトリクスを送信する．（6）では，CronJob で NAS にバックアップを毎日午前 3 時に実行する．

課題

Kubernetes 環境で新しい Pod をデプロイする際，以下の手順を繰り返すことで原因特定に時間がかかる．

(1) kubectl apply -f prometheus-pv.yaml
(2) kubectl get pods，kubectl describe pod prometheus-7cd6ddc956-rs6jv -n monitoring，kubectl get cronjob
(3) YAML ファイルの修正

図 2は佐藤さんが開発作業で繰り返した手順を表している．佐藤さんは以下 3 つの手順を繰り返し行っていることが分

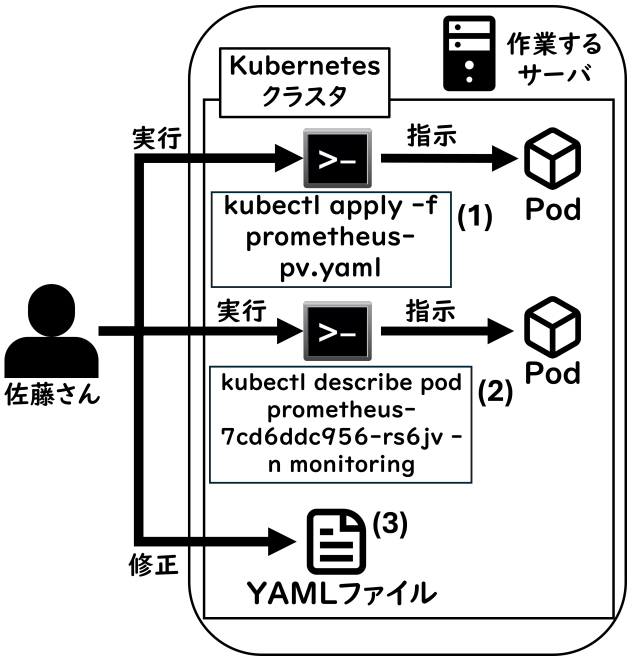

図 2　佐藤さんが開発作業で繰り返した手順

かった．(1) では，kubectl apply コマンドを実行して Pod を作成する．(2) では，kubect get pods コマンドや kubectl describe コマンド，kubectl get cronjob コマンドを実行して Pod の状態を確認している．(3) では，Pod が起動に失敗していた場合には YAML ファイルや Dockerfile，シェルスクリプトを修正する．

基礎実験

基礎実験では，佐藤さんがバックアップ作業をしている際に何の作業にどのくらいの時間をかけているのか記録した．図 3は佐藤さんが kubectl apply コマンドを打つまでに作業した内訳を積み上げ棒グラフで表している．図 3で

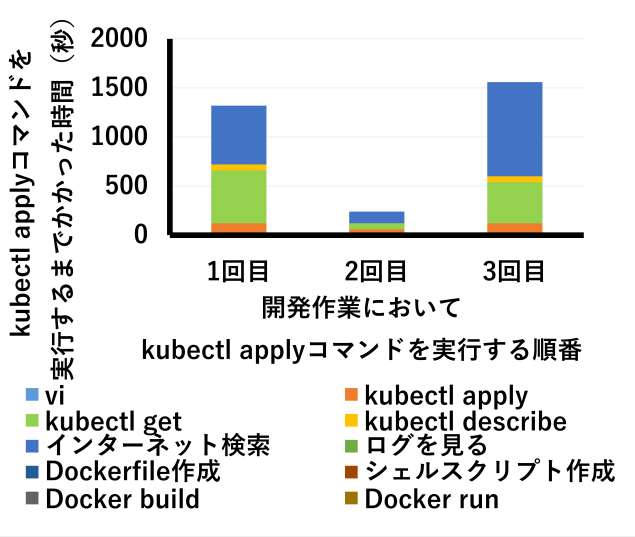

図 3 佐藤さんが kubectl apply コマンドを打つまでに作業した内訳

は，X 軸は個別の実験を表しており，単位に回目をとる．Y 軸は kubectl apply コマンドを実行するまでにかかった時間を表して，単位に秒をとる．凡例の黄土色は Docker run を表して，灰色は Docker build を表して，茶色はシェルスクリプト作成を表して，紺色は Dockerfile 作成を表して，緑色はログを見るを表して，青色はインターネット検索を表して，黄色は kubectl describe を表して，黄緑色は kubectl get を表して，橙色は kubectl apply を表して，水色は vi を表している．kubectl apply コマンドが 1 回目に実行するまでは，「インターネット検索」が 600 秒であり，「kubectl describe」が 60 秒であり，「kubectl get」が 540 秒になった．kubectl apply コマンドが 2 回目に実行するまでは，「インターネット検索」が 120 秒であり，「kubectl get」が 60 秒になった．kubectl apply コマンドが 3 回目に実行するまでは，「インターネット検索」が 960 秒であり，「kubectl describe」が 60 秒であり，「kubectl get」が 420 秒になった．作業にかかる時間の合計は 2820 秒である．「インターネット検索」が 2820 秒のうち合計 1680 秒かかっている．「kubectl get」が 2820 秒のうち合計 1020 秒かかっている．したがって，「インターネット検索」ことにかかる時間が合計 1680 秒で最も長い．「kubectl get」コマンドを実行して確認する作業にかかる時間が合計 1020 秒で二番目に長い．したがって，二番目に時間がかかっているインターネット検索の時間を短縮する．

各章の概要

第 2 章では，関連研究について説明する．第 3 章では，課題について解決するための提案方式について説明する．第 4 章では，提案した手法の実装について説明する．第 5 章では，評価実験として実験内容について説明する．第 6 章では，提案方式についての議論を説明する．最後に，第 7 章にて結論を説明する．

2. 関連研究

Golang と Helm SDK を活用して Kubernetes クラスタ内の非推奨（Deprecated）Helm リリースを自動的に検出や管理するコンテナ化アプリケーションを提案した研究がある [9]．Helm Go クライアントと Kubernetes Go クライアントを併用してクラスタ内の全リリース情報を収集した上で，それぞれのリリースが最新か否かを判断する．Helm リポジトリにおける非推奨状態かの判定や一覧表示することで，管理者がセキュリティリスクや互換性の問題を未然に察知や対処する構成となっている．手法は，CI/CD パイプラインへの統合可能性やカスタム Helm リポジトリ対応をする柔軟性を備えており，軽量なコンテナ構成と Golang の並列処理モデルにより，パフォーマンスとリソース効率を両立しながら，運用中クラスタの安定性と信頼性を高める支援ツールとして設計されている．

Isolation Forest による異常検知と LSTM ベースのシーケンスモデリングを組み合わせたハイブリッド機械学習モデルを用いた Kubernetes YAML ファイルの自動ミスコンフィギュレーション検出手法を提案した研究がある [10]．従来の静的ルールベース検証では捉えきれなかった構造的および意味的エラーを，高精度かつリアルタイムに検出することを目的である．CI/CD パイプラインに統合可能な軽量マイクロサービスとして設計されている．提案手法では，YAML ファイルの階層構造やトークン間依存関係を抽出し，特徴量を生成したうえで，Isolation Forest によりアウトライヤを検出し，LSTM により構成ルールの異常や論理的矛盾を学習して判定する．アプローチは，未知のエラーにも対応可能な柔軟性を持ち，運用中の Kubernetes 環境において高精度な構成検証を可能にするとともに，DevOps エンジニアへの自動フィードバックループや継続的学習機能も備えており，運用安定性の向上および設定作業の効率化に大きく貢献する．

教育現場においてアセンブリ言語による学生のプログラム提出物に対して，スタックエラーやメモリアクセスエラーを自動検出するツール群を提案した研究がある [11]．小型組み込みプロセッサ向けに設計されたシミュレーション環境を用いて，メモリアクセスの全履歴記録，スタック追跡，命令トレース，プログラムフロー解析の手法により，教員が学生の提出物を短時間かつ精度高く検査を可能にしている．この手法は，教育支援を主軸として，エラーの検出だけでなく予防的な観点も取り入れ，信頼性の高いアセンブリプログラムの育成を目的としている．

自然言語処理技術である UnnaturalCode を活用した構文エラー検出支援ツールを提案した研究がある [12]．従来のコンパイラが誤って複数箇所にエラーを報告してしまう問題に対し，正しくコンパイルされた過去のソースコードをもとに構築した n-gram 言語モデルを用いて「不自然」な

コード片を識別し，エラー箇所の絞り込みを行う．プログラマが修正すべき場所を効果的に特定でき，エントロピーの高いトークン列として提示された候補が，実際のエラー箇所に高確率で一致することが大規模な評価実験から示されている．UnnaturalCode は，コンパイル成功時に学習データを自動更新し，失敗時にはエラー補完情報として高エントロピー箇所を提示するインタラクティブな運用を実現しており，統計的アプローチにより構文解析支援の精度を高めた先進的なソフトウェアとして，構文エラー位置推定の新たな方法論を提示している．

3. 提案

図 4は提案ソフトウェアの概要を表している．提案ソフトウェアは，読み込みフェーズと比較フェーズ，修正フェーズで構成されている．読み込みフェーズでは，YAML ファイルと kubectl describe コマンドの結果を読み込む．(1) では，収集した YAML ファイルから，フィールド (<kind>, <name>) を取り出す．(2) では，提案ソフトウェアが kubectl describe <kind> <name>を実行する．実行した結果から EventType，EventReason，Status を取得する．(3) の読み込みフェーズでは，YAML ファイルの<kind>と kubectl describe コマンドの EventType，EventReason，Status の 4 種類の結果 (以降，エラーの詳細情報) を出力する．(4) の比較フェーズでは，読み込みフェーズで出力されたエラーの詳細情報とルールファイルを比較する．比較フェーズでエラーの詳細情報とルールファイルが一致した場合，エラーの詳細情報を修正フェーズに出力する．比較フェーズで一致しなかった場合，YAML ファイルの修正を行わずに終了する．(5) の修正フェーズでは，入力としてエラーの詳細情報と正解パターンを提案ソフトウェアに入れる．YAML ファイルと正解パターンが一致した場合, 正解パターンをもとにして YAML ファイルの修正をする．YAML ファイルと正解パターンが一致しなかった場合，問題点は見つかったが，修正方法がわからないのことをメッセージで出力にいれる．

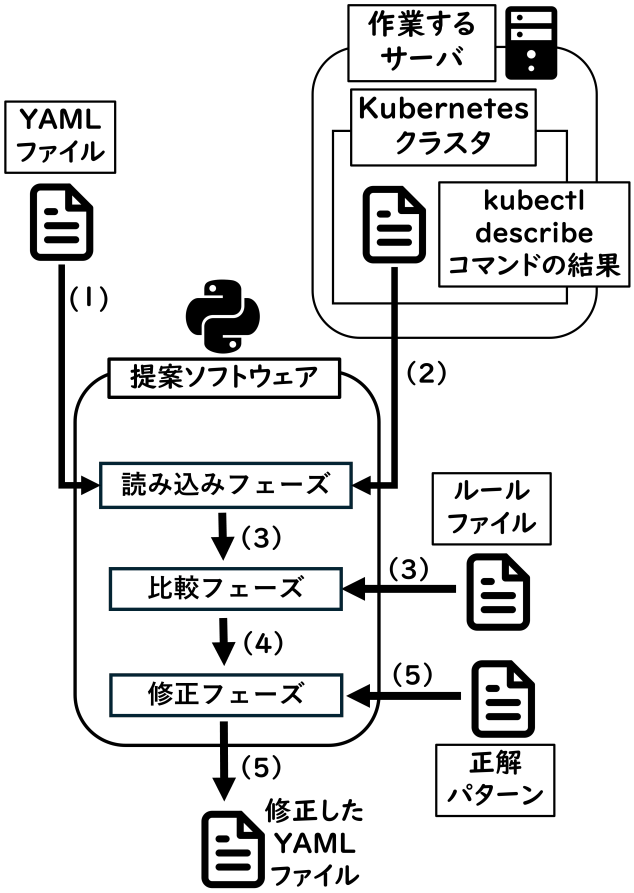

図 4　提案ソフトウェアの概要

ルールファイル

ルールファイルを作成する手順は以下の 4 つである．

1. Pod の起動に失敗する YAML ファイルを収集する
2. kubectl apply コマンドを実行する
3. kubectl describe コマンドの結果を記録する
4. CSV 形式のルールファイルに結果を書きこむ

1. では，に Pod の起動に失敗する YAML ファイルを収集する．datree の公式 GitHub には Datree リポジトリ[*1] に含まれる記述ミスのある YAML ファイルのデータセットがある．記述ミスのある YAML ファイルのデータセットはテストや学習目的として使われることが推奨されている．2. では，実際に発生するエラーを取得するために kubectl apply コマンドを実行する．3. では，kubectl describe コマンドの結果を記録する．4. では，CSV 形式のルールファイルに結果を書きこむ．出力情報は事前に収集した記述ミスがある YAML ファイルに対して kubectl apply および kubectl describe を実行したものである．ルールファイルは出力情報から抽出したフィールドを CSV 形式で構成されている．ルールファイルは，エラー発生時に YAML ファイルと kubectl describe コマンドを比較するための基準となるデータとして利用される．

図 5はルールファイルの内容を表している. 図 5では, 1 列目には kubectl describe コマンドの結果の Status が含まれる．2 行目のルールの Status は Running である．2 列目には YAML ファイルに記述している kind が含まれる．2 行目のルールの kind は CronJob である．3 列目には kubectl describe コマンドで表示された結果の EventType が含まれる．2 行目のルールの EventType は Warning である．4 列目には kubectl describe コマンドで表示された結果の EventReason が含まれる．2 行目のルールの EventReason は Failed である．

[*1] https://github.com/datreeio/datree (閲覧日:2025-07-15)

```
Status,Kind,EventType,EventReason
Running,CronJob,Warning,Failed
```

図 5 ルールファイルの内容

YAML ファイル

プログラム 1では，バックアップ作業で記述した YAML ファイルを表している．1 行目の apiVersion: batch/v1 は，リソースがバッチジョブであることを示す API バージョンを指定している．2 行目の kind: CronJob は，リソースの種類が CronJob であることを示している．3 行目の metadata: は，リソースのメタデータを定義している．4 行目の name: prometheus-backup は，CronJob の名前を prometheus-backup に設定している．5 行目の spec: は，CronJob の仕様を定義している．7 行目の schedule: ”*/1 * * * *” は，ジョブの実行スケジュールを cron 形式で指定しており，1 分ごとに実行される．8 行目の jobTemplate: は，CronJob が実行するジョブのテンプレートを定義している．9 行目の spec: は，ジョブテンプレートの仕様を定義している．10 行目の template: は，Pod テンプレートの定義している．11 行目の spec: は，Pod の仕様を定義している．12 行目の containers: は，Pod に含まれるコンテナのリストを定義する．13 行目の- name: prometheus-backup は，コンテナの名前を prometheus-backup に設定している．14 行目の image: c0a22169d8/backup-prometheus:latest は，使用するコンテナイメージを指定しており，Docker Hub 上の c0a22169d8/backup-prometheus の latest タグを使っている．15 行目の args: は，コンテナ起動時に渡す引数を定義するセクションである．16 行目の- ”/home/monitoring/prometheus-backup/prometheus-backup.sh” は，コンテナ起動時に実行するスクリプトのパスを指定している．17 行目の restartPolicy: OnFailure は，ジョブが失敗した場合にのみ Pod を再起動するポリシーを指定している．

プログラム 1 バックアップ作業で記述した YAML ファイル

```
1  apiVersion: batch/v1
2  kind: CronJob
3  metadata:
4    name: prometheus-backup
5  spec:
6    #schedule: "0 3 * * *"
7    schedule: "*/1 * * * *"
8    jobTemplate:
9      spec:
10       template:
11         spec:
12           containers:
13           - name: prometheus-backup
14             image: c0a22169d8/backup-prometheus
               :latest
15             args:
16             - "/home/monitoring/prometheus-
               backup/prometheus-backup.sh"
17           restartPolicy: OnFailure
```

kubectl descirbe コマンドの結果

プログラム 2は kubectl describe pod prometheus-backup-29187618-54d9v -n monitoring の結果の一部を表している．プログラム 2では，1 行目の Status: Running は，Pod が現在「実行中」と認識されている状態を示している．2 行目の Events: は，Pod に関連するイベントの一覧を表示するセクションである．3 行目の Type，Reason，Age，From，Message は，イベントの種類（Normal/Warning），理由，経過時間，発生元，詳細メッセージを示す列である．5 行目の Normal Scheduled 117s default-scheduler Successfully assigned monitoring/prometheus-backup-29187618-54d9v to monitoring-worker1 は，Pod がスケジューラによってノード monitoring-worker1 に正常に割り当てられたことを示している．6 行目の Warning Failed 70s (x4 over 113s) kubelet Error: failed to create containerd task... は，コンテナの起動に失敗したことを示している．エラーの原因は，指定されたスクリプト /home/monitoring/prometheus-backup/prometheus-backup.sh が存在しないこと示している．stat コマンドでファイルの存在確認ができず，no such file or directory が返されている．YAML ファイルの args

プログラム 2 kubectl describe pod prometheus-backup-29187618-54d9v -n monitoring の結果の一部

```
1  Status: Running
2  Events:
3    Type Reason Age From Message
4    ---- ------ ---- ---- -------
5    Normal Scheduled 117s default-scheduler
     Successfully assigned monitoring/prometheus-
     backup-29187618-54d9v to monitoring-worker1
6    Warning Failed 70s (x4 over 113s) kubelet
     Error: failed to create containerd task:
     failed to create shim task: OCI runtime
     create failed: runc create failed: unable to
      start container process: exec: "/home/
     monitoring/prometheus-backup/prometheus-
     backup.sh": stat /home/monitoring/prometheus
     -backup/prometheus-backup.sh: no such file
     or directory: unknown
7    Normal Pulled 70s kubelet Successfully
     pulled image "c0a22169d8/backup-prometheus:
     latest" in 1.294s (1.294s including waiting)
     . Image size: 3643355 bytes.
8    Warning BackOff 43s (x7 over 110s) kubelet
     Back-off restarting failed container
     prometheus-backup in pod prometheus-backup
     -29187618-54d9v_monitoring(95534006-402a
     -4285-8e55-8dd9491195c1)
```

で指定されたシェルスクリプトが，コンテナの中に存在しないため no such file or directory になっている．7 行目の Normal Pulled 70s kubelet Successfully pulled image ”c0a22169d8/backup-prometheus:latest” in 1.294s... は，指定されたコンテナイメージが正常に pull されたことを示している．8 行目の Warning BackOff 43s (x7 over 110s) kubelet Back-off restarting failed container prometheus-backup... は，コンテナの起動失敗により Kubernetes が再起動を試みているが，一定時間待機する「BackOff」状態に入っていることを示している．プログラム 3は Dockerfile を表している．プログラム 3では，1 行目の FROM alpine: 3.21 は，ベースイメージとして Alpine Linux のバージョン 3.21 を使用することを指定している．2 行目の RUN mkdir backup は，コンテナ内に backup ディレクトリを作成するコマンドを実行している．3 行目の COPY prometheus-backup.sh / は，ホスト側にある prometheus-backup.sh スクリプトをコンテナのルートディレクトリ / にコピーしている．シェルスクリプトを実行する場合には，/prometheus-backup.sh をパスでする必要がある．4 行目の RUN chmod +x prometheus-backup.sh は，コピーされたスクリプトに実行権限を付与している．5 行目の ENTRYPOINT [”bash”, ”prometheus-backup.sh”] は，コンテナ起動時に実行されるコマンドを指定している．

プログラム **3**　Dockerfile

```
1 FROM alpine: 3.21
2 RUN mkdir backup
3 COPY prometheus-backup.sh /
4 RUN chmod +x prometheus-backup.sh
5 ENTRYPOINT ["bash", "prometheus-backup.sh"]
```

比較フェーズと修正フェーズ

比較フェーズでは，エラーの詳細情報とルールファイルを比較する．修正フェーズでは，比較フェーズで一致したルールが存在する場合に実行する．修正が可能であるかは，事前に定義された「正解パターン」に照らして判断する．プログラム 4はプログラム 1を修正した YAML ファイルを表している．プログラム 4では，プログラム 1のバックアップ作業で記述した YAML ファイルの 16 行目を「/home/monitoring/prometheus-backup/prometheus-backup.sh」から「/prometheus-backup.sh」に変更している．変更した理由は，記述した YAML ファイルの args のパスを間違えており，YAML ファイルの args で指定されたシェルスクリプトがコンテナが存在しないからである．

図 6は正解パターンを表している．図 6の正解パターンは，1 列目には describe.EventsMessage が含まれる．2 行目の describe.EventsMessage は no such file or directory である．2 列目には command が含まれる．2 行目の command は

プログラム **4**　プログラム 1を修正した YAML ファイル

```
1  apiVersion: batch/v1
2  kind: CronJob
3  metadata:
4    name: prometheus-backup
5  spec:
6    #schedule: "0 3 * * *"
7    schedule: "*/1 * * * *"
8    jobTemplate:
9      spec:
10       template:
11         spec:
12           containers:
13           - name: prometheus-backup
14             image: c0a22169d8/backup-prometheus
                :latest
15             args:
16             - "/prometheus-backup.sh"
17           restartPolicy: OnFailure
```

```
describe.EventsMessage,command
no such file or directory,docker run -it {im
age_name} find / -name {file_name}
```

図 6　正解パターン

docker run -it {image_name} find / -name {file_name} である．正解パターンの `describe.EventMessage` と `kubectl describe` の結果が一致する場合には，修正可能と判定して command で確認し，YAML ファイルの該当箇所を修正する．含まれていない場合には，修正できないと判定する．正解パターンは，Kubernetes の公式ドキュメントを生成 AI に読み込ませることで，フィールド間の整合性に関するルールを抽出して作成する．抽出されたルールは，人間の手によって内容の妥当性を確認して，問題がないと判断したものを正解ファイルとして登録する．

関連研究と比較

Golang で構築されたアプリケーションを用い，Kubernetes クラスター内の Helm リリースの状態を分析し，非推奨（Deprecated）になっているリリースを自動検出する仕組みを提案している研究である [9]．提案方式と比較すると，開発作業で YAML ファイルを記述した際のエラー原因の特定と修正に対応している．入力として YAML ファイル，kubectl describe コマンドの結果，そしてルールファイルの 3 つをもちいることで，明確な問題解決を可能としている．ルールファイルにエラーの原因が含まれている場合には，直接 YAML ファイルを修正し，そうでない場合には修正しない．これらの一連の処理により，エラーの原因特定と修正にかかる時間を大幅に短縮している．

Isolation Forest と LSTM ベースの機械学習モデルを用いて，Kubernetes における YAML ファイルの構文的およ

び構造的なミスコンフィグレーション（誤設定）を自動的に検出するシステム「Vela」を提案している研究である [10]．提案方式と比較すると，YAML ファイルの誤りの検出にとどまらず，エラー原因に応じた修正と広範な知識を活用して対処の提示をする．具体的には，入力として YAML ファイル，kubectl describe コマンドの結果，そしてルールファイルの 3 つを用い，出力としてルールファイルに原因がある場合には修正された YAML ファイルを生成し，ルールファイルに原因がない場合には，修正しない．この構成により，問題が事前に定義されたパターンに起因しているか否かに応じて対応方法を自動的に選択し，トラブル対応をしている．

ユースケース・シナリオ

佐藤さんが Prometheus のバックアップ作業を行っている場面を想定する. 図 7はユースケース・シナリオを表している．Kubernetes 環境で新しい Pod をデプロイする際，3 つの作業を繰り返すことで，kubectl apply コマンドで Pod を起動して，Pod の状況を確認をして，YAML ファイルを修正するを繰り返すことで原因特定に時間がかかる．(1) では，kubectl apply コマンドを実行して Pod を作成する．(2) では，kubect get pods コマンドや kubectl describe コマンド，kubectl get cronjob コマンドを実行して Pod の状態を確認している．(3) では，提案ソフトウェアを用いて YAML ファイルを修正している．提案ソフトウェアはルールファイルにエラーの原因がある場合に YAML ファイルの修正をして，エラーの原因がない場合に修正しない．これにより，Pod の状況を確認してから kubectl apply コマンドを実行するまでの作業時間を短縮する．したがって，Pod の状況を確認してから kubectl apply コマンドを実行するまでの作業時間を短縮することによって，開発作業にかかる全体の時間が短くなる．

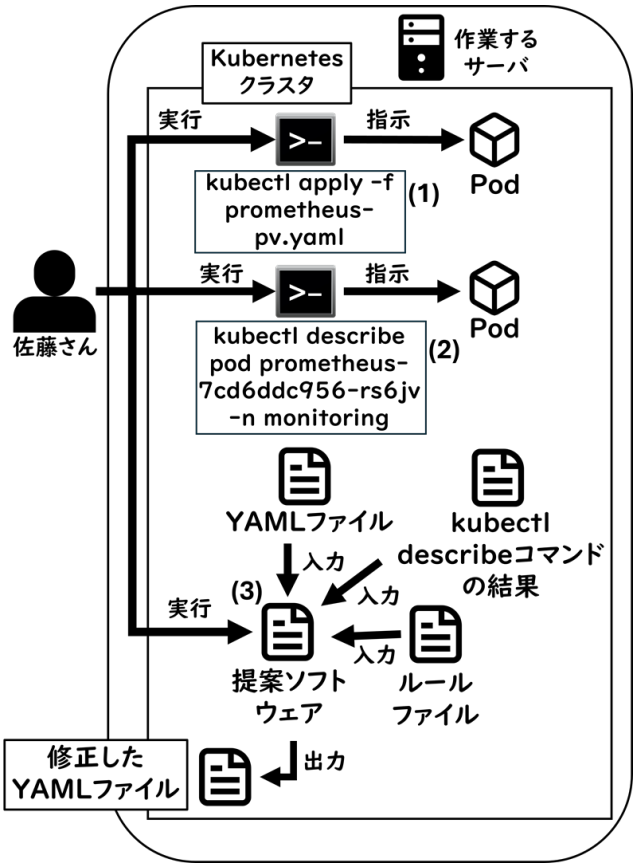

図 7　ユースケース・シナリオ

4. 実装

ソフトウェアは Python 3 で実装する．実行する環境は Python 3.12.3 である．図 8はプログラムの構成を表している． Ubuntu の環境では，ホームディレクトリの/home/c0a22103/に，複数の Python スクリプトが格納されている．ディレクトリ直下には，2 つのファイルの monitor_k8s_yaml.py と apply_and_log.py がある．さらに，ホームディレクトリの中にはサブディレクトリの practice があり，中にはファイルである my-kubectl-1.py，my-kubectl-2.py，my-kubectl-3.py がある．

```
/home/c0a22103/
├── monitor_k8s_yaml.py
├── apply_and_log.py
├── practice/
    ├── my-kubectl-1.py
    ├── my-kubectl-2.py
    └── my-kubectl-3.py
```

図 8　プログラムの構成

my-kubectl-1.py

プログラム 5は my-kubectl-1.py を表している．sys.argv を使用してコマンドラインから渡された引数を取得している．取得した引数のうち，最初の要素であるスクリプト名は除外し，実際に使用される引数のみを抽出している．抽出した引数を空白で区切って連結し，標準出力に表示することで，コマンドラインインターフェースを操作する時の引数の受け渡し動作を可視化している．

プログラム 5　my-kubectl-1.py

```
1 c0a22103@c0a22103-practice:~/c0a22103-kubectl/
  practice$
2 python3 my-kubectl-1.py apple banana
3 apple banana
```

my-kubectl-2.py

プログラム 6は my-kubectl-2.py を表している．subpro-

cess.run() 関数を使用して kubectl get pod -A を実行している．出力を capture_output=True および text=True のオプションにより文字列として取得している．得られた出力は result.stdout に格納されて，print() 関数で標準出力に表示することで，コマンドラインインターフェース上に Pod 情報を一覧表示する．

プログラム 6　my-kubectl-2.py

```
1 c0a22103@c0a22103-practice:~/c0a22103-kubectl/
  practice$ python3 my-kubectl-2.py
2 NAMESPACE NAME READY STATUS RESTARTS AGE
3 default nginx-deployment-96b9d695-518hc 1/1
  Running 1 (4d2h ago) 48d
4 default nginx-deployment-96b9d695-clb8q 1/1
  Running 1 (4d2h ago) 48d
5 default nginx-deployment-96b9d695-wpxxp 1/1
  Running 1 (4d2h ago) 48d
6 elastic fb-filebeat-fr2rg 1/1 Running 0 42m
```

my-kubectl-3.py

プログラム 7は my-kubectl-3.py を表している． my-kubectl-1.py と my-kubectl-2.py を合わせたものが my-kubectl-3.py である．コマンドライン引数からスクリプト名を除いた内容を取得し，kubectl コマンドとして実行する．実行結果は文字列として取得され，標準出力に表示される．

プログラム 7　my-kubectl-3.py

```
1 c0a22103@c0a22103-practice:~/c0a22103-kubectl/
  practice$ alias kubectl="python3 my-kubectl-3.
  py"
2 c0a22103@c0a22103-practice:~/c0a22103-kubectl/
  practice$ kubectl get pods
3 NAME READY STATUS RESTARTS AGE
4 nginx-deployment-96b9d695-5l8hc 1/1 Running 1
  48d
5 nginx-deployment-96b9d695-clb8q 1/1 Running 1
  48d
6 nginx-deployment-96b9d695-wpxxp 1/1 Running 1
  48d
```

monitor_k8s_yaml.py

プログラム 8は monitor_k8s_yaml.py を表している．対象となる監視ディレクトリに対して，一定間隔（10 秒）でファイルの更新状況をチェックし，新規作成または内容が更新された YAML ファイルを検出すると，タイムスタンプ付きでバックアップディレクトリへコピーを保存する．os.listdir() を用いて監視対象ディレクトリ内のファイル一覧を取得し，拡張子が.yaml のファイルに限定して処理を行っている．os.path.getmtime() によって取得した最終更新時刻を記録や比較し，更新されたファイルがあれば shutil.copy() を使用してバックアップを作成する．バックアップファイルには現在時刻を付与することで，履歴管理を容易にしている．

プログラム 8　monitor_k8s_yaml.py

```
1 c0a22103@c0a22103-practice:~/c0a22103-kubectl$
2 python3 monitor_k8s_yaml.py
3 Monitoring /home/c0a22103 for changes...
4 Backup created: /home/c0a22103/k8s-backup
  /20250716_070543_nginx-deployment.yaml
5 Backup created: /home/c0a22103/k8s-backup
  /20250716_070543_nginx-service.yaml
```

apply_and_log.py

プログラム 9は apply_and_log.py を表している．指定されたディレクトリ内の YAML ファイルを一括して kubectl apply で適用し，結果および関連情報をログファイルとして記録や保存する自動化ツールである．まず，k8s-backup ディレクトリ内に存在する.yaml ファイルをすべて取得し，それぞれのファイルに対して kubectl apply コマンドを実行する．適用後には，対象リソースに関する詳細情報を取得するために kubectl describe を併用し，リソースの状態やエラーメッセージを収集している．これらの情報は，ファイル名とタイムスタンプを組み合わせたログファイルとして k8s-logs ディレクトリに保存される．

プログラム 9　apply_and_log.py

```
1 c0a22103@c0a22103-practice:~/c0a22103-kubectl$
2 python3 apply_and_log.py
3 Applied /home/c0a22103/k8s-backup/20250711
  _031355_nginx-deployment.yaml, log saved to /
  home/c0a22103/k8s-logs/20250716
  _070108_20250711_031355_nginx-deployment.yaml.
  log
4 Applied /home/c0a22103/k8s-backup/20250523
  _034537_nginx-service.yaml, log saved to /home
  /c0a22103/k8s-logs/20250716
  _070109_20250523_034537_nginx-service.yaml.log
5 Applied /home/c0a22103/k8s-backup/20250707
  _005739_nginx-deployment.yaml, log saved to /
  home/c0a22103/k8s-logs/20250716
  _070109_20250707_005739_nginx-deployment.yaml.
  log
```

5. 評価実験

評価実験では，提案手法で間違った箇所が修正された YAML ファイルのうち，正しく修正された YAML ファイルの割合を適合率で評価を行う．適合率 (Precision) を式 (1) に示す．

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1}$$

式 (1) では，分子が True Positive（以降，TP とする）で分母が TP と False Positive（以降，FP とする）の和で表している．TP は提案手法で間違った箇所が正しく修正された YAML ファイルの数である．FP は提案手法で間違った箇所が誤って修正された YAML ファイルの数である．図 9は評価までの全体的な流れを表している．評価実験を行う手順は以下の 7 つである．(1) のクローンでは，リモートリポジトリから git clone コマンドを実行してローカルリポジトリにクローンする．(2) の選出では，クローンしたディレクトリ内の YAML ファイルから間違いがあるパターン作成用 YAML ファイルを選出する．(3) の作成では，パターン作成用 YAML ファイルをもちいてルールファイルを作成する．(4) の入力では，開発環境で記述した YAML ファイルと kubectl describe コマンドの結果，ルールファイルの 3 つを提案ソフトウェアに入れる．(5) の出力では，提案ソフトウェアから修正した YAML ファイルが出力する．(6) の入力では，修正した YAML ファイルを評価ソフトウェアに入れる．(7) の出力では，評価ソフトウェアが適合率を出力する．

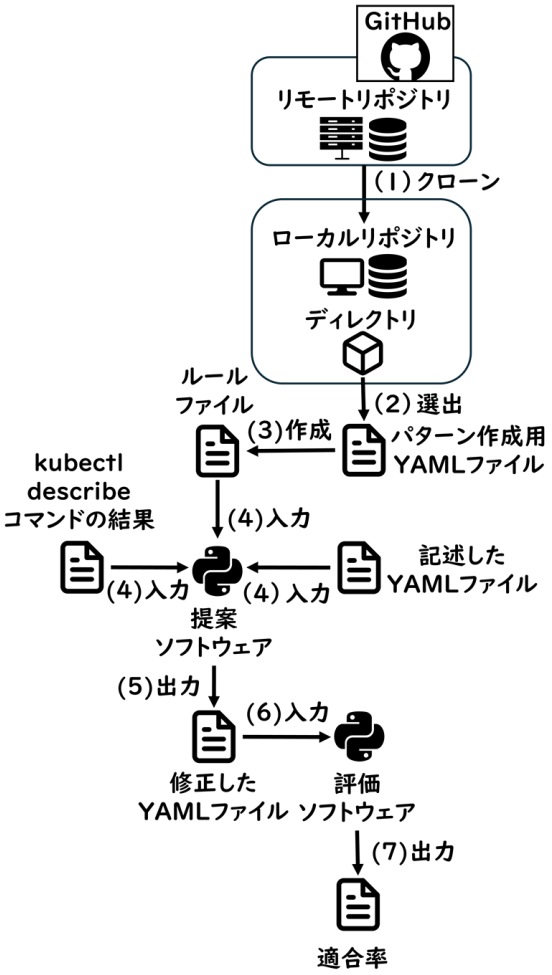

図 9　評価までの全体的な流れ

実験環境

まず，仮想マシンを 1 台作成する．次に，Ubuntu をインストールして，Kubernetes クラスタを構築する．そして，Python をインストールする．

- vCPU：2Core
- メモリ：2GB
- ストレージ：30GB
- OS：Ubuntu 24.04

図 10は実験環境を表している．図 10では，GitHub のリモートリポジトリからローカルリポジトリにクローンして，作業するサーバに複製している．ローカルリポジトリのディレクトリ内には，記述ミスをした YAML ファイルがある．提案ソフトウェアを作業するサーバに作成した．ソフトウェアは Python 3 で実装する．実行する環境は Python 3.12.3 である．

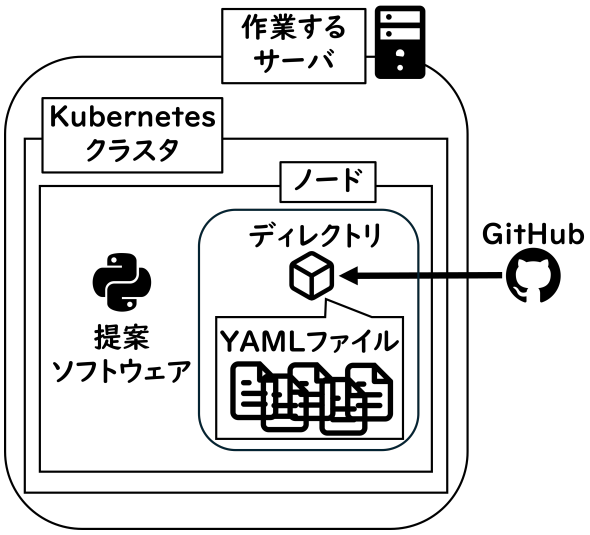

図 10　実験環境

6. 議論

提案方式では，ルールファイルと YAML ファイルを比較を行う．Datree の公式 GitHub リポジトリから Pod の起動に失敗する YAML ファイルを収集し，YAML ファイルと kubectl describe コマンドの結果をもとにルールファイルを作成する手法である．つまり，ルールファイルを作成するために YAML ファイルを収集する必要がある．この方式は，YAML ファイルの中でも間違えている場合の YAML ファイルを収集する必要がある．しかし，間違えている YAML ファイルのデータセットが公開されていないため，データセットを作成する必要がある．この問題に対して，インターネットの上に公開されている YAML ファイルを収集する．また，GitHub や Qiita をもちいて YAML ファイルを集める．さらに，自分の Kubernetes クラスタ

を構築して無料で公開することで YAML ファイルを収集する．収集した YAML ファイルに対して kubectl apply コマンドを実行して起動するか確認する．

提案方式では，ルールファイルにはエラーの詳細情報が 4 つ含まれている．ルールファイルに含まれているフィールドの種類を増やすことでルールの件数を増やせるが，ルールファイルと YAML ファイルの適合率が減少する．ルールファイルに含まれている情報のフィールドを減らすことでルールの件数を減らせるが，ルールファイルと YAML ファイルの適合率が増加する．この問題に対して，部分一致を取り入れる必要がある．ルールファイルに含まれているフィールドの種類を増やして部分一致を取り入れることで，ルールの件数を増やした際の完全一致がしない課題が解決する．

7. おわりに

課題では，バックアップを構築する過程において Pod の状況を確認してから kubectl apply コマンドを実行するまでに時間がかかっている．基礎実験では開発作業にかかる時間の内訳を計測した．作業にかかる時間の合計は 2820 秒である．インターネット検索にかかる時間が合計 1680 秒で最も長かった．kubectl get コマンドを実行して確認する時間が合計 1020 秒で二番目に長かった．提案では，ルールファイルと YAML ファイルを比較することでエラー原因を特定して YAML ファイルを修正する．ルールファイルの作成では，GitHub や Qiita から kubectl apply コマンドで Pod の起動に失敗した YAML ファイルを収集してディレクトリに保管する．YAML ファイルをもとに作成したルールファイルを CSV 形式で保存しておく．実験環境では，仮想マシンに Ubuntu を構築して Python の開発環境を用意する．評価方法として，提案手法で間違った箇所が修正された YAML ファイルのうち，正しく修正された YAML ファイルの割合を適合率で評価を行う．

参考文献

[1] Queiroz, R., Cruz, T., Mendes, J., Sousa, P. and Simões, P.: Container-based virtualization for real-time industrial systems—a systematic review, *ACM Computing Surveys*, Vol. 56, No. 3, pp. 1–38 (2023).

[2] Menouer, T.: KCSS: Kubernetes container scheduling strategy, *Journal of Supercomputing*, Vol. 77, pp. 4267–4293 (online), DOI: 10.1007/s11227-020-03427-3 (2021).

[3] Elradi, M. D.: Prometheus Grafana: A Metrics-focused Monitoring Stack, *Journal of Computer Allied Intelligence(JCAI, ISSN: 2584-2676)*, Vol. 3, No. 3, p. 28–39 (online), DOI: 10.69996/jcai.2025015 (2025).

[4] Kubernetes, T.: Kubernetes, *Kubernetes. Retrieved May*, Vol. 24, p. 2019 (2019).

[5] Moravcik, M., Kontsek, M., Segec, P. and Cymbalak, D.: Kubernetes - evolution of virtualization, *2022 20th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pp. 454–459 (online), DOI: 10.1109/ICETA57911.2022.9974681 (2022).

[6] Rabenstein, B. and Volz, J.: Prometheus: A Next-Generation Monitoring System (Talk), Dublin, USENIX Association (2015).

[7] Zhu, Z., Chamberlain, J., Wu, K., Starobinski, D. and Liu, Z.: Approximation-First Timeseries Monitoring Query At Scale (2025).

[8] Predoaia, I., Kolovos, D., Garcia-Dominguez, A., Lenk, M., Ebel, W. and Burkl, J.: Towards Processing YAML Documents with Model Management Languages, *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, MODELS Companion '24, New York, NY, USA, Association for Computing Machinery, p. 970–979 (online), DOI: 10.1145/3652620.3688219 (2024).

[9] 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).

[10] Sivaraman, H.: Automated Detection of Misconfiguration in Kubernetes YAML Files Using Machine Learning, *INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH AND CREATIVE TECHNOLOGY*, Vol. 9, No. 1 (online), DOI: 10.5281/zenodo.14250656 (2023).

[11] 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).

[12] 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).