障害のアラートから生成 AI で作成したスクリプトの適用によるメモリ使用率の減少判定をもちいた復旧時間の短縮

佐藤 健斗[1]　平尾 真斗[2]　串田 高幸[1]

概要：システム運用でのアラート対応では，監視担当者が障害の原因の調査や復旧作業を行う．課題は，アラートが通知された際に障害の復旧に時間がかかることである．提案手法は，監視システムから送信される JSON 形式のアラートを提案ソフトウェアが受信し，生成 AI をもちいて障害の対処を行う対処スクリプトを生成する．生成された対処スクリプトを監視対象に適用し，監視システムから取得したメトリクスをもとにしきい値が下回ったかどうかの判断を行う．しきい値を下回っていない場合，結果を生成 AI のプロンプトに反映し再生成を行う．評価実験では，3 種類の障害事例を対象として提案手法が障害を復旧できたかどうかを検証した．1 つ目は，Redmine の Pod のメモリ使用率がしきい値である 90%を超えた事例を再現し，10 回の試行を行った．その結果，10 回の試行のうち 8 回の試行では生成された対処スクリプトを適用させることでメモリ使用率が 90%を下回ることを確認できた．2 つ目は，CDSL で運用している Doktor の Node のメモリ使用率がしきい値である 90%を超えた事例を再現し，同様に 10 回の試行を行った．その結果，10 回のうち 4 回の試行でメモリ使用率が 90%を下回ることを確認できた．3 つ目は，Node のファイルシステム使用率が 90%を超えた事例を再現し，10 回の試行を行った．その結果，10 回のうち 1 回の試行でファイルシステム使用率が 90%を下回ることを確認できた．

1. はじめに

背景

クラウドコンピューティングやコンテナ技術の普及により，情報システムの規模は拡大し，運用管理の複雑性が増している [1]．こうした大規模かつ動的環境では，サービスの継続性を確保するために障害検知と予兆の把握を目的とした監視が重要である [2]．監視基盤は，メトリクスを収集するだけでなく，異常を迅速に検知するアラート機能を備えることで，障害対応の初動を支援する役割を果たしている [3]．特にサービスの継続性が重視されるサービスの環境においては，異常の早期発見や対応が運用コストの削減及び信頼性の向上につながるため，監視の重要性が高まっている [4]．

システムの監視を構成する要素であるアラートは，監視対象のメトリクスやログにおいて異常を検知した際に運用担当者へ通知を行い，迅速な初動対応を促す仕組みである [5]．一方で，アラートが過剰に発生した場合や誤検知が発生した場合には，運用担当者の作業負担が増大し，障害を見落とすリスクが高まる [6]．

システム監視の仕組みを支えるために，オープンソースの監視ソフトウェアである Prometheus が利用される [7]．Prometheus は時系列データベースを内蔵し，CPU の使用率やメモリの使用率のメトリクスを一定間隔で収集，保存することでシステムの状態を把握することができる [8]．また，クエリ言語である PromQL をもちいることで，メトリクスの集計やアラート条件の定義を行うことができる [9]．発生したアラートを効率的に制御，通知する役割として Alertmanager が利用される [10]．Alertmanager は重複通知の抑制や，Slack，メールへのルーティングを行い，運用における通知基盤を担っている [11]．さらに，収集したメトリクスを可視化するツールとしては Grafana が利用され，ダッシュボードの設計を行うことができる [12]．チケット管理システムは，プロジェクト管理ツールである Redmine が使用され，アラートをチケット化することで障害対応の進捗の記録や共有を行う [13]．運用では，これらの監視，可視化ツールとチケット管理システムを組み合わせることで，監視から障害対応までの一連の流れを管理している [14]．

東京工科大学のコンピュータサイエンス学部の研究室である Cloud and Distributed Systems Laboratory(以下

[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 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

CDSL と呼ぶ) では，10 台のサーバに VMware 社の ESXi を導入し，複数の Virtual Machine（以下 VM と呼ぶ）を配置している．これらのサーバの内 7 台は，研究室の学生の実験や作業を行う目的で使用されている．2 台は，DNS や DHCP の運用に使用され，1 台は外部向けのアプリケーション公開の用途に使用されている．CDSL では，運用環境の安定性を確保するために Prometheus，Alertmanager，Redmine をもちいた監視体制を構成し，通知されたアラートをもとに作成されたチケットを Redmine で管理している．運用しているシステムの障害発生時に通知されたアラートの調査や対処は監視担当の学生が行っている．図 1に，Doktor の worker-node2 でメモリ使用率が 90%を超えた際のアラートの通知から対処までの流れを示す．

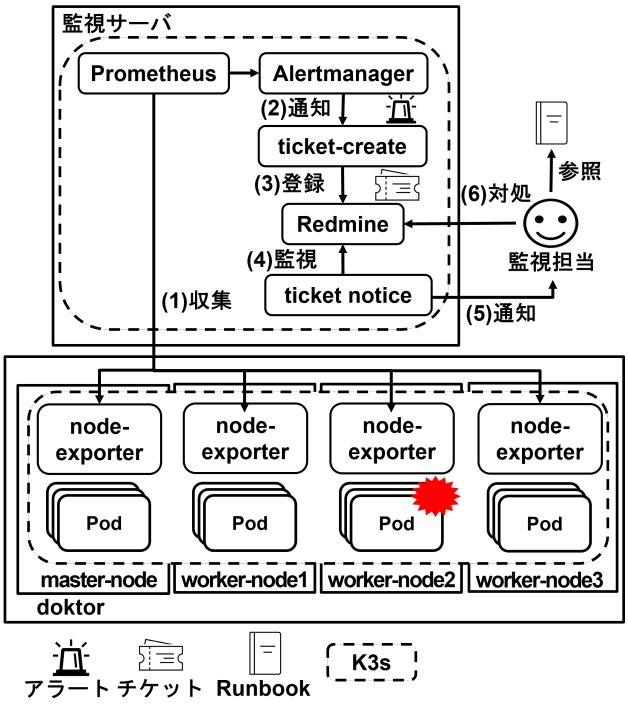

図 1: Doktor の worker-node2 でメモリ使用率が 90%を超えた際のアラートの通知から対処までの流れ

この監視体制の中で，実際に発生した障害の事例として 2025 年 9 月 1 日に CDSL が運用している Doktor のサイトを構成する worker-node2 の Node でメモリ使用率が 90%を超えたことを示すアラートが通知された．Doktor は，1 台の master-node と 3 台の worker-node で構成され，Kubernetes クラスタ上で運用されている．4 台の Node 上には，node exporter と Doktor のマイクロサービスを稼働させるための Pod が配置されている．監視サーバには，Prometheus，Alertmanager，ticket-create，Redmine，ticket notice が配置されている．監視担当の学生は，Runbook に沿って調査と対処を行った．調査では，メモリを使用している Pod の確認やプロセスのメモリ使用率の確認を行った．対応としては，メモリ使用率が高い Pod 以外を他の Node へ分散配置した．(1) で Prometheus が各 Node からメトリクスを収集している．(2) で Alertmanager によって，しきい値を超えた際にアラートを通知する．(3) で通知されたアラートをもとに ticket-create でチケットが作成され Redmine に登録される．(4) で ticket notice が Redmine にチケットが登録されたか監視する．(5) でチケットの登録を検知したら監視担当の学生に通知を行う．(6) で監視担当の学生が Redmine に登録されたチケットを Runbook を参照し，対処する．

課題

課題は，監視担当の学生によって行われる障害の復旧作業に時間がかかることである．CDSL で運用しているシステムでは，Prometheus によって収集されたメトリクスをもとに，設定されたしきい値を超えた際に Alertmanager からアラートが通知される．しかし，アラートを受信した後の調査や対処は学生が行っているため，知識や経験に依存している．例えば図 1の事例では，1 次調査に 2 時間 49 分，1 次対応に 1 時間 13 分，合わせて 4 時間 2 分の作業時間を要している．

各章の概要

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，提案した方式の実装について述べる．第 5 章では，提案方式の評価と分析について述べる．第 6 章では，提案方式の議論について述べる．第 7 章では，本稿のまとめについて述べる．

2. 関連研究

ICT システムの故障時に，オペレーターが実行した過去のログと復旧コマンドの対応を Seq2Seq モデルで学習し，新たな障害発生時に適切な復旧コマンドを推定する手法を提案した研究がある [15]．障害の検出や原因解析は既存手法により自動化されつつあるが，どのコマンドを実行するかの決定は人に依存していた課題に対して，モデルによって適切なコマンドの推定を目的としている．一方で，提案されているのは実行すべきコマンドの提示までであり，実行の判断や適用は人に依存している点に改善の余地がある．

OpenStack ベースのクラウド環境において，if-this-then-that 型のイベント駆動の自動化フレームワークを構築し，AI による意思決定モジュールが事前に用意したカタログから最適な復旧アクションを選択する自己修復システムを提案した研究がある [16]．この研究では，システムの信頼性を最大化し，ダウンタイムを最小化することを目的としている．一方で，パラメータの調整やモデルの最適化を人が行う設計になっている点に改善の余地がある．

Tier-2 クラスタにおいて Prometheus，Loki，Grafana に

よる監視システムに加えて，特定の既知障害に対してあらかじめ定義した簡易的な復旧アクションを自動で実行する仕組みを構築した研究がある [17]．この研究では，定常的な運用負荷を軽減し，人的介入を最小限に抑えることを目的としている．一方で，実行されるのは定義済みの障害に限定した簡易処理のため，新規の障害には人の介入が必要である点に改善の余地がある．

仮想化を利用したサーバ統合環境において，運用マニュアルを自然言語処理とデータ処理アルゴリズムにより効率的に分析し，自動化可能な手順の抽出と標準化を支援する手法を提案した研究がある [18]．この研究では，マニュアルの全文に対して高い計算コストがかかる課題を，効率的なアルゴリズムで克服している．一方で，抽出された手順は文書ベースであり，実際の運用時の動的なメトリクスの変化や異なる障害内容への柔軟な対応に改善の余地がある．

クラウド環境におけるインシデント管理の自動化を目的とした研究がある [19]．この研究では，ソフトウェアの開発ライフサイクルの各段階におけるコードや構成，監視データ，依存関係，トラブルシューティングのドキュメントを LLM に入力することで，依存関係の障害における根本原因の推定や監視対象のオントロジーの特定を行っている．一方で，最終的な対処の判断や実行はオンコールエンジニアが行う設計になっており，知識や経験に依存している点に改善の余地がある．

3. 提案

本研究では，監視システムから通知されるアラートの障害に対する復旧作業を迅速に行うことを目的とする．提案手法は，監視システムから通知されるアラートを初回のプロンプトに入力して生成 AI で対処スクリプトの生成を行う．生成された対処スクリプトを監視対象に適用させて，適用後に取得したメトリクスの値としきい値を比較することで復旧できたかの判定をする．復旧できていない場合，実行結果をもとにプロンプトを修正し，対処スクリプトの再生成を行う．

提案方式

図 2に提案方式の概要を示す．監視対象は，障害が発生した対象である．監視システムは，監視対象からメトリクスを収集し，しきい値を超えた際にアラートを通知する．提案ソフトウェアは，通知されたアラートをもとにプロンプトの生成，対処スクリプトの実行，対処スクリプトの評価を行う．生成 AI は，プロンプトをもとに対処スクリプトを生成する．提案方式では，(1) で監視対象からメトリクスを監視システムが収集している．(2) でメトリクスの値がしきい値を超えた場合に通知されるアラートを入力としてプロンプトの生成を行う．(3) で生成したプロンプトを生成 AI に渡す．(4) で生成 AI により対処スクリプトを生成し，(5) でアラートが通知された監視対象に適用させる．(6) で再度，監視対象からメトリクスを収集して，(7) で提案ソフトウェアに送信する．(8) で，その実行結果とメトリクスをもとにしきい値と比較して復旧できたかを判定する．復旧できていない場合は，実行結果のログやメトリクスをもとにプロンプトを再生成する．

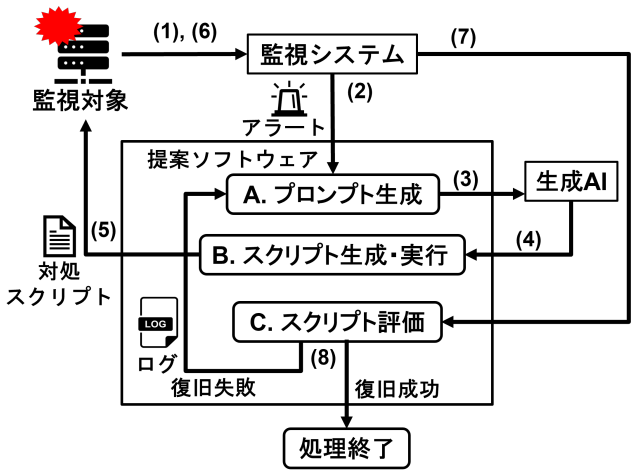

図 2: 提案方式の概要

A. プロンプト生成

プロンプトの生成は，監視システムから通知される JSON 形式のアラートを提案ソフトウェアで受信し，内容を保存する．アラートには発生時刻や障害が発生した対象，アラート名が含まれる．初回のプロンプトをコード 1に示す．

コード 1: 初回のプロンプト

```
# アラート情報
{alert_json}

# 環境情報
{environment_information}

# 現状の{監視対象の}使用率
before={BEFORE_VALUE}
threshold={THRESHOLD_VALUE}

# 前回試行結果
last_after=なし

# 要求
アラートを対処することができる bash スクリプトを生成してください
- 実行可能な kubectl コマンドやシステム設定変更を使用
- 出力は必ず ‘‘‘bash ‘‘‘ で囲み、スクリプトのみを返してください
```

初回のプロンプトには，アラートの内容や kubectl get all によるシステムの状態を把握したうえで対処方法を提案するように求めている．1 行目では，次の行でアラートの内容を与えることを示している．2 行目では，監視システムから取得した，障害の発生個所や発生時間が含まれる JSON 形式アラートを{alert_json} として記述している．4 行目では，次の行でシステムの状態を与えることを示している．5 行目では，クラスタの構成や対象の Pod の状態を{environment_information}として提示している．7 行目では，次の行で現在の監視対象のメモリやファイルシステムの使用率に関する内容を与えることを示している．8 行目で，アラート発生時のメモリやファイルシステムの使用率を{BEFORE_VALUE}として提示している，9 行目で，アラートのルールにもとづくしきい値を{THRESHOLD_VALUE} として提示している．11 行目では，次の行で前回の試行結果に関する内容を与えることを示している．12 行目では，初回のプロンプトのため{last_after}はなしになっている．ここでは，再生成を行った際の結果を示している．14 行目では，次の行で要求内容を与えることを示している．15 行目では，アラートの対処を行える Bash スクリプトの生成を要求している．16 行目では，kubectl コマンドによる操作を許可している．17 行目では，出力結果として bash で囲んだスクリプトの内容のみを返すように指定している．

B. スクリプト生成・実行

構築したプロンプトは生成 AI に送信され，対処スクリプトを返す．提案ソフトウェアは，対処スクリプトを内部で実行し，標準出力や標準エラーをログとして保存する．

C. スクリプト評価

提案方式では，対処スクリプトの有効性をメトリクスのしきい値との比較によって評価する．評価は，最初に対処スクリプトの実行前後に監視システムからメモリ使用率を取得する．次に，実行後のメトリクスがしきい値未満かどうかを判定する．しきい値未満だった場合には復旧できたと判定し，しきい値以上の場合は，復旧できていないと判定する．復旧できていない場合は，初回のプロンプトに実行結果のログを追加して対処スクリプトの再生成を行う．

ユースケース・シナリオ

本稿におけるユースケースとして，障害発生時にアラートが通知された際の復旧作業を想定する．例として Redmine の Pod のメモリ使用率が 90%を超えたことを示すアラートが通知された事例を想定する．図 3にユースケースシナリオを示す．

監視対象は，Redmine の Pod である．Node1 と Node2 は K3s をもちいたクラスタを構成する 2 台の Node である．cAdvisor は，Redmine のメモリ使用率を監視する．

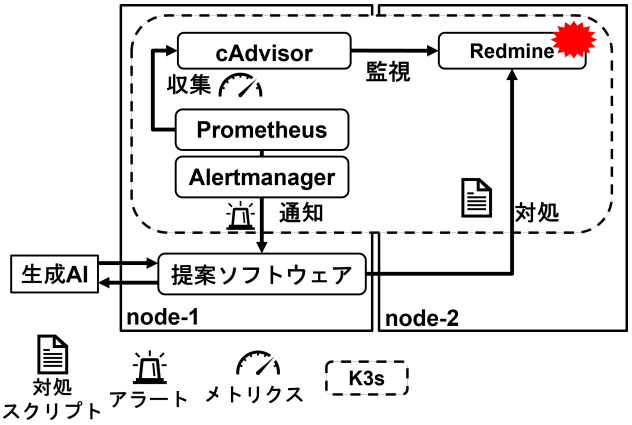

図 3: ユースケースシナリオ

Prometheus は，cAdvisor から Redmine のメトリクスを収集する．Alertmanager は，Prometheus によって収集したメトリクスが事前に設定されたしきい値を超えた場合にアラートを通知する．提案ソフトウェアは，Alertmanager から通知された JSON 形式のアラートを受け取り，その内容にもとづいてプロンプトを構築し，生成 AI により対処スクリプトを生成する．生成された対処スクリプトを監視対象に適用し，その結果として得られるメトリクスの値を参照することで，障害の復旧ができたかを判定する．復旧できていない場合は，対処スクリプトの実行結果やエラー内容，しきい値との差分をもとにプロンプトを再構築し，生成 AI に再入力する処理を繰り返す．

4. 実装

提案方式を実装するために，Python3.12.3 で Web アプリケーションを作成した．本アプリケーションは，Alertmanager から通知されるアラートを入力とし，生成 AI によって対処スクリプトを生成する．生成された対処スクリプトを実行し，その結果をしきい値と比較して復旧できたかを判定する．実装は，Flask をもちいて，HTTP リクエストを介してアラートを受け取る構成とした．使用したライブラリは，Flask，dotenv，google.generativeai，requests，os，json，datetime，subprocess，time，re である．図 4に実装する Alert_fix の概要を示す．

Prometheus は，収集したメトリクスの値が事前に設定されたしきい値を超える場合にアラートを作成する．Alertmanager は，Prometheus が作成したアラートを通知する．Node exporter は，監視対象の CPU やメモリのメトリクスを監視する．Alert_fix は，通知されたアラートを受信し，プロンプトの生成，Gemini による対処スクリプトの生成，対処スクリプトの適用，復旧の判定を行う．Gemini は，プロンプトをもとに障害の対処を行う対処スクリプトの生成を行う．本アプリケーションは，主に 4 つの機能で構成される．1 つ目は，アラートの受信である．アラートの受信では，Alertmanager から通知されたアラートを JSON 形

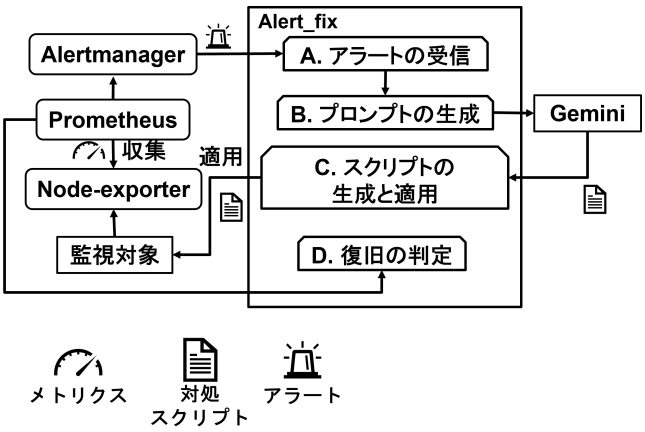

図 4: Alert_fix の概要

式で受け取る．2 つ目は，プロンプトの生成である．プロンプトの生成では，アラートを入力とした初回のプロンプトを生成する．3 つ目は，スクリプトの生成と適用である．スクリプトの生成と適用では，Gemini により対処スクリプトを生成し，アラートが通知された監視対象に適用させる．4 つ目は，復旧の判定である．復旧の判定では，監視対象に対処スクリプトを適用させた後のメトリクスの値をアラートルールのしきい値と比較して，復旧できたかの判定を行う．

A. アラートの受信

アラートの受信では，Prometheus と Alertmanager によって通知されたアラートを Flask アプリケーションのエンドポイントで受け取る．通知されるアラートは JSON 形式であり，監視対象の Pod 名や Namespace 名，メトリクス名が含まれている．受信したアラートは JSON 形式で保存される．

B. プロンプトの生成

プロンプトの生成では，アラートの内容をもとに生成 AI に送信するためのプロンプトを自動的に構築する．ここでは，アラートが示す異常状態に対して，その原因を調査し，復旧を行うための対処スクリプトを出力するようにプロンプトを生成する．

C. スクリプトの生成と適用

スクリプトの生成と適用では，Gemini に対して構築したプロンプトを送信し，応答として返されるテキストから対処スクリプト部分のみを抽出する．Gemini からの出力は，説明の文章とコマンドが混在している場合があるため，正規表現をもちいて対処スクリプト部分のみを取り出し，ファイルとして保存する．保存後に抽出した対処スクリプトを実行し，出力やエラーの内容を取得する．

D. 復旧の判定

復旧の判定では，対処スクリプトの実行後の状態を Prometheus から取得したメトリクス値と比較することで，復旧できたかを判定する．例えば，アラートが発生したメトリクスがメモリ使用率でしきい値が 90%の場合，対処スクリプトの実行後にメモリ使用率が 90%未満になっていれば，復旧ができたと判定する．復旧ができたと判定した場合は，そのまま処理を終了する．一方で，しきい値を下回らない場合には，対処スクリプトによって復旧ができなかったと判定する．復旧ができていないと判定した場合には，対処スクリプトの実行結果をプロンプトに追加し，再度プロンプトの生成から処理を行う．

5. 評価実験

評価実験では，提案手法により生成された対処スクリプトによってアラートが通知されるしきい値を下回ることができるかを検証した．1 つ目の実験では，2025 年 4 月 28 日に発生した Redmine の Pod においてメモリ使用率が 90%を超えた状況を対象とした．2 つ目の実験では，2025 年 9 月 1 日に発生した Doktor の Node においてメモリ使用率が 90%を超えた状況を対象とした．3 つ目の実験では，Node のファイルシステム使用率が 90%を超えた状況を対象とした．実験方法は，Prometheus から取得したメトリクスにもとづき，メモリ使用率のしきい値を 90%として，ファイルシステムの使用率も同様に 90%として，対処スクリプトの実行後にメモリ使用率がしきい値を下回るかどうかについて，有効性を評価した．

実験環境

Redmine の Pod を対象にした実験

1 つ目の実験は，Ubuntu24.04.1 LTS がインストールされた仮想マシン上で実施した．Redmine の Pod を監視対象にした実験環境を図 5に示す．

実験環境は 2 台の仮想マシンで構成されており，Node1 と Node2 は K3s をもちいたクラスタを構成する Node である．それぞれの Node のスペックは，vCPU が 2core，メモリが 4GB，ストレージが 30GB である．Redmine は監視対象である Pod を示す．cAdvisor は，Redmine の Pod のメモリ使用率を監視する監視エージェントである．Prometheus は，cAdvisor が監視している対象のメトリクスを収集する．Alertmanager は，Prometheus が収集したメトリクスがしきい値を超えた際に，アラートを通知する．Alert_fix は，通知されたアラートをもとに，障害の対処を行う．実験は，2025 年 4 月 28 日に発生した Redmine の Pod のメモリ使用率が 90%を超えた事例を再現する．事例を再現するために resources.limits.memory の値を Redmine をインストールしたときの 512Mi から 180Mi にして実験を行った．監視システムには Prometheus と Alertmanager を使

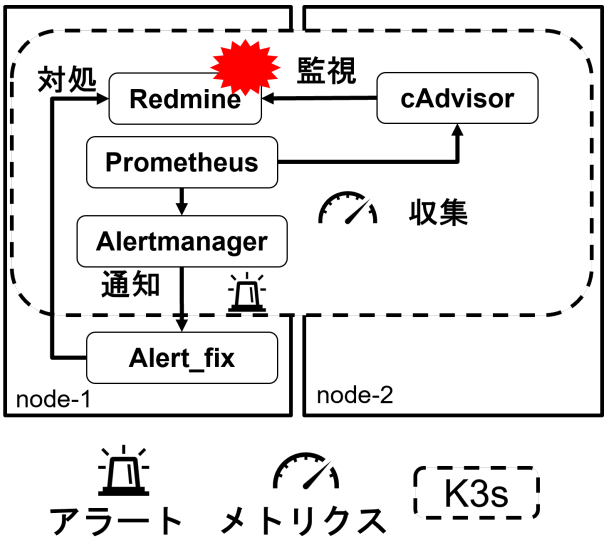

図 5: Redmine の Pod を監視対象にした実験環境

用し，Redmine の Pod のメモリ使用率を監視対象として設定した．アラートが通知された際に，提案ソフトウェアにアラートを通知して，対処スクリプトの生成と実行を行うようにした．

図 5は，Redmine が監視対象で，cAdvisor で監視している．Prometheus が cAdvisor からメトリクスを収集して事前に設定したしきい値を超えた際に Alertmanager から通知される．通知されたアラートは，Alert_fix で受信して正常な状態に戻す処理を行う．

Doktor の Node を対象にした実験

2 つ目の実験は，Ubuntu22.04.5 LTS がインストールされた仮想マシン上で実施した．Doktor の Node を監視対象にした実験環境を図 6に示す．

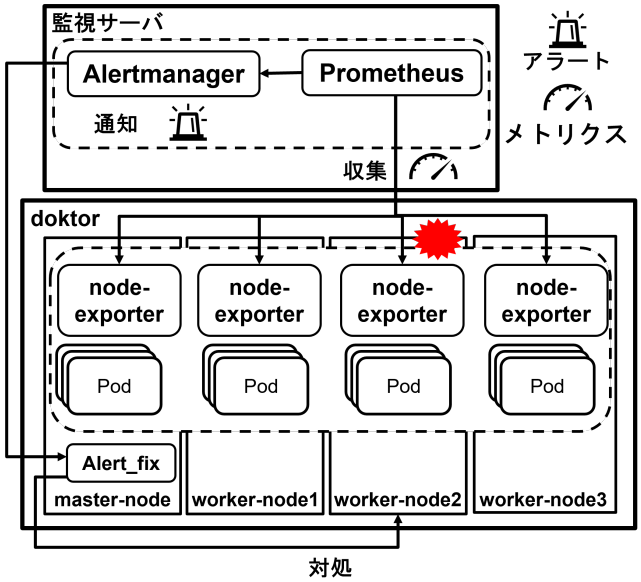

図 6: Doktor の Node を監視対象にした実験環境

実験環境は，1 台の master-node と 3 台の worker-node の 4 台で構成されている．master-node は，クラスタ内にある worker-node と Pod を管理するための Node である．worker-node1 から worker-node3 は，Pod を動作させるための Node である．それぞれの Node のスペックは，vCPU が 8core，メモリが 8GB，ストレージが 40GB である．worker-node 上の Pod は，Doktor のマイクロサービスを稼働させるための Pod である．

実験は，2025 年 9 月 1 日に発生した worker-node2 のメモリ使用率が 90%を超えた事例を再現する．事例を再現するために，worker-node2 に 19 個の Pod を配置した．19 個の中にメモリ使用率が 4514Mi を使用していた Pod を含めてメモリ使用率が 90%になるように配置した．Prometheus と Alertmanager は監視サーバ上で動作し，構成は 1 つ目の実験と同様である．提案ソフトウェアは，Doktor の master-node 上で動作し，Alertmanager から通知されたアラートを受信して対処スクリプトの生成と実行を行う．

図 6は，Doktor の Node が監視対象で，Prometheus が node exporter からメトリクスを収集している．監視対象に障害が発生した際に Alertmanager から提案ソフトウェアである Alert_fix に通知され，障害の対処を行う．

ファイルシステムを対象にした実験

3 つ目の実験では，Node のファイルシステムが 90%を超えた事例を再現した．Node のファイルシステムを対象にした実験環境を図 7に示す．実験の構成は図 5で示した内容と同様である．

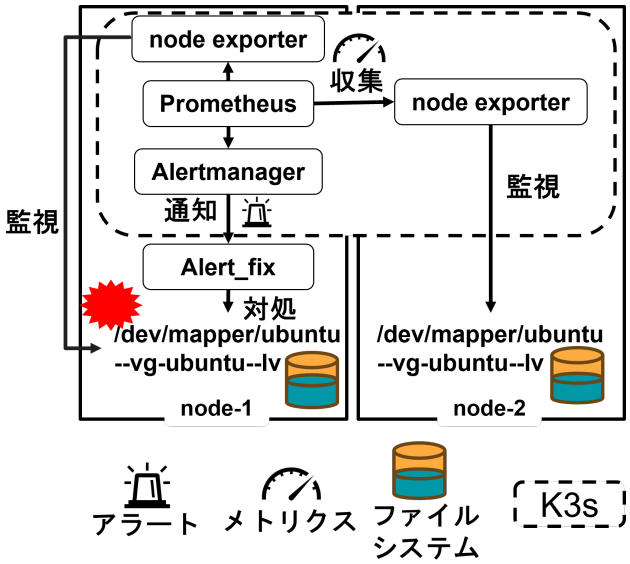

図 7: Node のファイルシステムを対象にした実験環境

図 7のファイルシステムは監視対象である．Node が 2 台あり，Node には/dev/mapper/ubuntu–vg-ubuntu–lv が存在する．実験では，/dev/mapper/ubuntu–vg-ubuntu–lv

のメモリ使用率が 90%を超えたことでアラートが通知された．node exporter は，Node のファイルシステムの使用率を監視する監視エージェントである．Prometheus は，node exporter が監視している対象のメトリクスを収集する．Alertmanager は，Prometheus が収集したメトリクスがしきい値である 90%を超えた際に，アラートを通知する．Alert_fix は，通知されたアラートを受信し，障害の対処を行う．

本実験では，事例を再現するために 512MB のファイルをファイルシステムの使用率が 90%を超えるまで作成した．Prometheus と Alertmanager は監視サーバ上で動作し，構成は 1 つ目の実験と同様である．Prometheus は node exporter からファイルシステムのメトリクスを収集し，使用率が 90%を超えた際に Alertmanager から提案ソフトウェアに通知される．提案ソフトウェアは通知を受信し，対処スクリプトの生成と実行を行う．また，実験は最大 5 回までの対処スクリプトの再生成を許容し，合計 10 回の試行を行った．各試行で同じ状態から始めるために，提案ソフトウェアの実行前に VM のスナップショットを取得し，各試行後にスナップショットからリストアすることで，ファイルシステムの使用率を約 91%の同一の状態から開始できるようにした．これにより，実行された対処スクリプトの影響を次の試行に引き継がないようにしている．

実験結果と分析

1 つ目の実験では，Redmine の Pod に割り当てられたメモリ使用率が 90%を超えた事例を再現し，Prometheus と Alertmanager によるアラートの通知をトリガーとして，提案ソフトウェアである Alert_fix による対処スクリプトの生成と適用後のメトリクスの確認を行った．この事例は，Redmine をインストールしたときに割り当てられているデフォルトの memory limits の値が低かったことにより，メモリ使用率が 90%を超えたアラートが通知された．本実験では，対処スクリプト適用後の Redmine の Pod のメモリ使用率がしきい値を下回るかを確認した．評価基準は，メモリ使用率が設定されたしきい値である 90%を下回るかどうかである．図 8に Redmine の Pod を対象にした試行回数と Pod のメモリ使用率の変化を示す．

図 8は，Redmine の Pod を対象として 10 回の試行を行った際の，対処スクリプトを適用した後のメモリ使用率の変化を示したものである．横軸は各試行において許容される最大 5 回までの対処スクリプトの再生成回数，縦軸は正常値に対するメモリ使用率の差分を表す．本実験は，Pod の memory の limits が 256MB で設定されており，そのうちの約 174MB が使用されている．使用されていた約 174MB を正常値として 68%と表した．この値を N と定義する．また，メモリ使用率が 90%を超えた際にアラートが発生するため，この値をしきい値として T=90%と定義した．正常

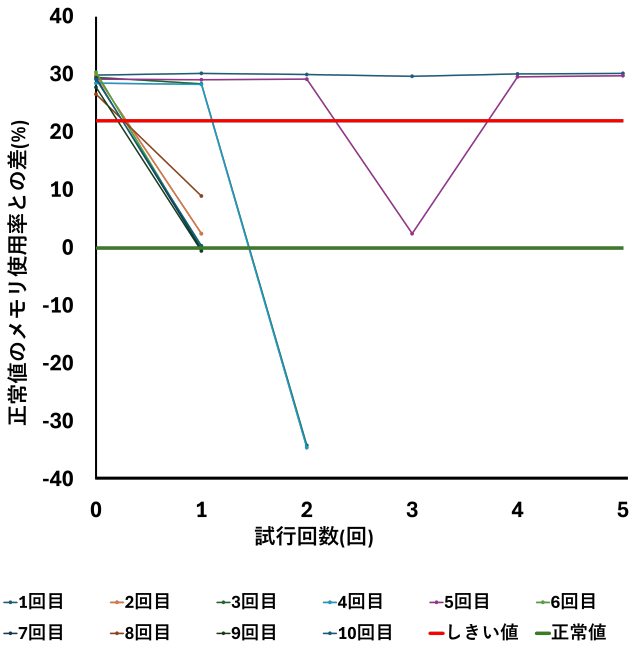

図 8: Redmine の Pod を対象にした試行回数と Pod のメモリ使用率の変化

値としきい値の差 D を式 1で表す．

$$D = T - N \tag{1}$$

縦軸における 0%の値は，正常値 N を示し，22%の位置はしきい値 T を示している．したがって，縦軸の値が 0%に近づくほど，対処スクリプトによってメモリ使用率を正常値に近づけられたことを示している．アラート受信時点のメモリ使用率は，約 94%から約 97%程度の範囲で変動しており，すべての試行が同一の初期値から開始をしていない．本稿では，初期値の変動を前提として，各試行において生成された対処スクリプトの適用後にメモリ使用率がしきい値を下回るかどうかを評価した．図 8から，提案ソフトウェアによって生成された対処スクリプトの実行結果により，メモリ使用率がしきい値を下回った場合と下回らなかった場合を確認できる．1 回目と 5 回目の試行では，最大 5 回の再生成を行ったが，メモリ使用率は 90%を下回らなかった．一方，その他の試行でメモリ使用率がしきい値を下回る結果となった．しきい値を下回った要因としては，生成された対処スクリプトに Pod に割り当てられた memory limits の引き上げを行う内容が記述されていたことである．具体的には，3 回目と 4 回目が 180Mi から 512Mi に変更されていた．2 回目，6 回目，7 回目，8 回目，9 回目，10 回目は，180Mi から 256Mi に変更されていた．この結果から，提案ソフトウェアによる対処スクリプトの生成と実行によるアラートの対処を 10 回のうち 8 回行えたことを確認できた．

2 つ目の実験は，1 つ目の実験と同様に Node のメモリのしきい値は 90%で 10 回の試行を行った．図 9に Doktor

の Node を対象にした試行回数と Node のメモリ使用量の変化を示す．

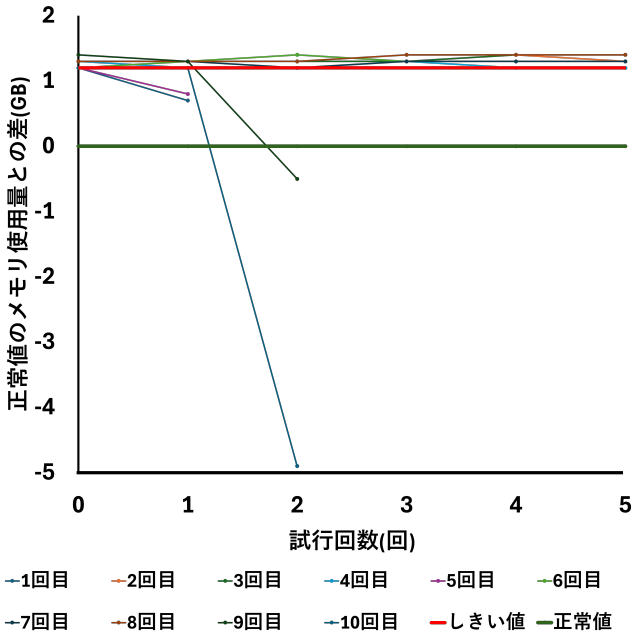

図 9: Doktor の Node を対象にした試行回数と Node のメモリ使用量の変化

図 9は，Doktor の Node を対象として 10 回の試行を行った際の，対処スクリプト適用後におけるメモリ使用量の変化を示したものである．横軸は最大 5 回まで許容される対処スクリプトの再生成回数，縦軸は対処スクリプト適用後のメモリ使用量と正常値との差を表す．縦軸の 0 は，実験前のアラートが通知されない状態を正常値として 6GB を示している．しきい値は，VM に割り当てられた 8GB の 90%である 7.2GB であり，正常値との差である 1.2GB の位置にしきい値を示す線を示している．Node のメモリ使用量の正常値を N，しきい値を T，N と T の差を D とする．対処スクリプト適用後のメモリ使用量が正常値と比べてどの程度の差があるかを求めた式は，式 1と同様である．

図 9において，縦軸の値が 0 に近づくほど，正常値である 6GB に近いメモリ使用量に対処できていることを示す．一方で，0 を下回る値は，正常値よりもメモリ使用量が低くなる対処を行なったことを示す．

アラート受信時点のメモリ使用量は，約 7.2GB から約 7.4GB 程度の範囲で変動しており，すべての試行が同一の初期値から開始をしていない．図 9と同様にメモリ使用量の初期値の変動を前提として，各試行において生成された対処スクリプトの適用後にメモリ使用量がしきい値である 7.2GB を下回るかどうかを評価した．

図 9から，10 回の試行のうち 4 回の試行でメモリ使用量がしきい値を下回った．それぞれしきい値を下回った要因としては，1 回目の試行は，drop_caches の実行によるキャッシュの解放，5 回目の試行は，Pod のスケールアップとスケールダウンによる再デプロイやキャッシュの解放，9 回目の試行は，最もメモリを消費していた Pod の再起動，10 回目の試行は，node drain による Pod の削除と再スケジュールである．これらは，いずれも一時的な対処であり，メモリの恒久的な削減や根本的な負荷の軽減を行えていない．一方で，2 回目，3 回目，4 回目，6 回目，7 回目，8 回目の試行ではメモリ使用量がしきい値を下回らなかった．

3 つ目の実験では，Node のファイルシステムの使用量を対象として，10 回の試行を行った．Prometheus は node exporter からファイルシステムのメトリクスを収集し, 使用量が 27GB を超えた際に Alertmanager から提案ソフトウェアに通知された. 提案ソフトウェアは通知を受信して対処スクリプトを生成し, 実行した. 対処スクリプトの再生成は最大 5 回まで許容した. 図 10にファイルシステムを対象にした試行回数とファイルシステム使用量の変化を示す．

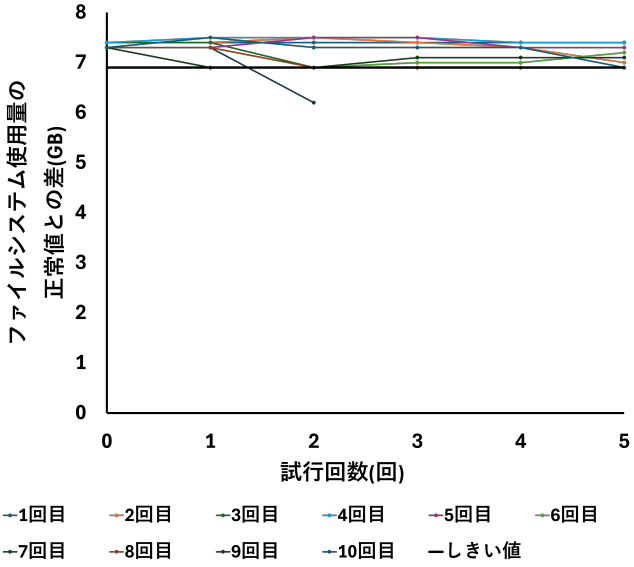

図 10: ファイルシステムを対象にした試行回数とファイルシステム使用量の変化

図 10は，Node のファイルシステム使用量を対象として 10 回の試行を行った際の，対処スクリプト適用後におけるメモリ使用量の変化を示したものである．横軸は最大 5 回まで許容される対処スクリプトの再生成回数，縦軸は対処スクリプト適用後の使用量と正常値との差を表す．縦軸の 0 は，正常値である 20GB を示している．しきい値は，27GB であり，正常値との差である 7GB の位置にしきい値を示す線を示している．Node のファイルシステム使用量の正常値を N，しきい値を T，N と T の差を D とする．対処スクリプト適用後のファイルシステム使用量が正常値と比べてどの程度の差があるかを求めた式は，式 1と同様である．

図 10において，縦軸の値が 0 に近づくほど，正常値である 20GB に近いファイルシステム使用量に対処できていることを示す．

3 つ目の実験の結果，10 回の試行のうち 7 回目のみファイルシステム使用率が 90%を下回った．90%のしきい値を下回った要因としては，Docker の不要なイメージの削除によって約 1GB の領域が削減されたことである．一方でしきい値を下回らなかった要因は，APT キャッシュや/var/tmp の一時ファイルを削除対象としたが十分な容量を削除できなかったことや，イメージのみを対象とする削除で containerd のデータ領域や K3s 関連のディレクトリに対しては十分な削除ができなかったことである．

3 つの実験結果から，提案ソフトウェアは Pod のように対象範囲が限定され，改善すべき対象が明確なアラートに対しては，対処スクリプトの実行によりしきい値を下回る結果が得られることが確認できた．一方で，Node やファイルシステムのように複数の要因や大容量な領域が分散している対象では，しきい値を下回る回数が減少することが確認できた．特にファイルシステムの実験では，削除対象が軽微な領域に限定され，主要なディレクトリに十分なアプローチができなかったことにより，10 回の試行のうち 1 回のみしきい値を下回った．このことから，対象が限定的で原因が明確な場合には，提案ソフトウェアが有効である一方，対象範囲が広く複雑な場合には，削除対象の選定や原因の分析を行う仕組みが必要であることが分かった．

6. 議論

本稿では，監視システムから通知されるアラートをもとに，生成 AI によって対処スクリプトを生成し，その有効性を評価した．1 つ目の実験では，メモリ使用率がしきい値を超えたアラートに対して 10 回の試行を行い，そのうち 8 回の試行でメモリ使用率がしきい値である 90%を下回ったが，2 回はしきい値を下回らなかった．2 つ目の実験では，10 回の試行のうち 4 回は，しきい値を下回ったが，6 回はしきい値を下回らなかった．3 つ目の実験では，10 回の試行のうち 1 回は，しきい値を下回ったが，9 回はしきい値を下回らなかった．1 つ目の実験は，Pod に対するリソースの障害であったのに対して，2 つ目の実験は，node のメモリ使用率に複数の Pod やプロセスが関与していたため，アラートから具体的な原因を特定することが難しかったと考えられる．また，生成 AI による対処スクリプト生成の有効性は，確認できたが，環境の構造やアラートの内容に依存して再現性が低下すると考えられる．この課題に対して，アラートに加えて，関連する Pod のメトリクスやログ，過去の対処履歴を入力に組み込むことで，プロンプトを生成 AI がより細かい判断を行えるようにする．

本稿では，アラート発生時に生成された対処スクリプトを実行し，その結果をもとに再生成プロンプトを構築する処理を導入した．これにより，対処スクリプトの有効性を評価し，改善を繰り返す再生成を行った．しかし，3 つの実験において，5 回まで再生成を繰り返しても，復旧できる対処スクリプトに近づくとは限らなかった．この結果から，試行回数の増加では対処スクリプトの精度は向上せず，誤った方針による対処が繰り返される場合もあることが示された．原因としては，生成 AI が過去の失敗した内容を十分に理解できずに，同様の対処を再度出力してしまうことがあげられる．この課題を解決するためには，再生成時のプロンプトの設計を整理する必要がある．具体的には，前回の対処スクリプトとその実行結果との差分に加えて，どの操作が無効だったのか，どのリソースが影響を受けたのか失敗要因を明示することで，生成 AI が修正すべき方向性を適切に学習できるようにする．

本稿では，アラートの通知をトリガーとして生成 AI が自動的に対処スクリプトを生成し，その対処スクリプトを実行する仕組みを実装した．これにより，人がコマンドを確認することなく迅速な初期対応が行える．しかし，生成 AI が出力する対処スクリプトには，誤作動やシステムを破壊する操作のリスクが存在する．例として，2 つ目の実験では，node drain を実行する対処スクリプトが生成された．そのため，安全性の確保と自動実行の両立が必要である．この課題に対して，対処スクリプトの静的解析による構文の検証や，実際のシステムに影響を与えないサンドボックス環境で試行することで，安全性と自動化の両立ができる．

7. おわりに

システム運用において，監視担当者はアラートの内容を確認し，原因の特定や復旧作業を行っている．課題は，アラートの通知から対応までに時間がかかることである．本稿では，アラートをもとに生成 AI が対処スクリプトを生成し，その実行結果に応じてプロンプトを更新して再生成を行う仕組みを提案する．提案手法では，Alertmanager から通知されたアラートを JSON 形式で Flask アプリケーションが受け取り，Gemini で対処スクリプトを生成する．生成された対処スクリプトの実行後のメトリクスと Prometheus から取得したしきい値を比較することによって復旧できたかを判定し，復旧できていない場合には，対処スクリプトの再生成を行う．評価実験では，Redmine の Pod のメモリ使用率が 90%を超える事例と Doktor の Node のメモリ使用率が 90%を超える事例の 3 つを再現し，提案ソフトウェアで 10 回の試行を行った．その結果，Redmine を対象とした実験では，10 回の試行のうち 8 回の試行でメモリ使用率がしきい値を下回った．具体例として，対処スクリプト実行前のメモリ使用率が約 97.9%であったものが，対処スクリプト実行後に約 70.5%に低下した．この改善は，生成された対処スクリプトによって Redmine の Pod の memory

の limits を引き上げる操作を実行したためである．Doktor を対象とした実験では，10 回の試行のうち 4 回の試行でメモリ使用率がしきい値を下回った．この改善は，キャッシュの解放や Pod の削除のコマンドを実行したためである．Node のファイルシステムを対象とした実験では 10 回のうち 1 回の試行でファイルシステム使用率が 90%を下回ることを確認できた．これにより，提案手法がアラートをもとに対処スクリプトを生成し実行することによる改善をすることができた．

参考文献

[1] Birje, M. N. and Bulla, C.: Cloud monitoring system: basics, phases and challenges, *International Journal of Recent Technology Engineering (IJRTE)*, Vol. 8, No. 3, pp. 4732–4746 (2019).

[2] Zou, Z., Xie, Y., Huang, K., Xu, G., Feng, D. and Long, D.: A Docker Container Anomaly Monitoring System Based on Optimized Isolation Forest, *IEEE Transactions on Cloud Computing*, Vol. 10, No. 1, pp. 134–145 (online), DOI: 10.1109/TCC.2019.2935724 (2022).

[3] Elradi, M. D.: Prometheus & Grafana: A Metrics-focused Monitoring Stack, *Journal of Computer Allied Intelligence (JCAI, ISSN: 2584-2676)*, Vol. 3, No. 3, pp. 28–39 (2025).

[4] Hrusto, A., Runeson, P., Engström, E. and Ohlsson, M. C.: Advancing Software Monitoring: An Industry Survey on ML-Driven Alert Management Strategies, *2024 50th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 435–442 (online), DOI: 10.1109/SEAA64295.2024.00073 (2024).

[5] Yu, Q., Zhao, N., Li, M., Li, Z., Wang, H., Zhang, W., Sui, K. and Pei, D.: A survey on intelligent management of alerts and incidents in IT services, *Journal of Network and Computer Applications*, Vol. 224, p. 103842 (online), DOI: https://doi.org/10.1016/j.jnca.2024.103842 (2024).

[6] Hu, W., Chen, T. and Shah, S. L.: Detection of Frequent Alarm Patterns in Industrial Alarm Floods Using Itemset Mining Methods, *IEEE Transactions on Industrial Electronics*, Vol. 65, No. 9, pp. 7290–7300 (online), DOI: 10.1109/TIE.2018.2795573 (2018).

[7] Sukhija, N. and Bautista, E.: Towards a Framework for Monitoring and Analyzing High Performance Computing Environments Using Kubernetes and Prometheus, *2019 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pp. 257–262 (online), DOI: 10.1109/SmartWorld-UIC-ATC-SCALCOM-IOP-SCI.2019.00087 (2019).

[8] Chen, L., Xian, M. and Liu, J.: Monitoring System of OpenStack Cloud Platform Based on Prometheus, *2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL)*, pp. 206–209 (online), DOI: 10.1109/CVIDL51233.2020.0-100 (2020).

[9] Mart, O., Negru, C., Pop, F. and Castiglione, A.: Observability in Kubernetes Cluster: Automatic Anomalies Detection using Prometheus, *2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pp. 565–570 (online), DOI: 10.1109/HPCC-SmartCity-DSS50907.2020.00071 (2020).

[10] Klymash, M., Zablotskyi, S. and Pohranychnyi, V.: Improving Alerting in the Monitoring System Using Machine Learning Algorithms, *2024 IEEE 17th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*, pp. 150–153 (online), DOI: 10.1109/TCSET64720.2024.10755868 (2024).

[11] Ujiie, R., Kholodilo, V., Northern, B. and Ulybyshev, D.: Secure Monitoring and Notification System for Cloud Infrastructures, *SoutheastCon 2022*, pp. 787–792 (online), DOI: 10.1109/SoutheastCon48659.2022.9764125 (2022).

[12] Siddiqui, I., Pandey, A., Jain, S., Kothadia, H., Agrawal, R. and Chankhore, N.: Comprehensive Monitoring and Observability with Jenkins and Grafana: A Review of Integration Strategies, Best Practices, and Emerging Trends, *2023 7th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, pp. 1–5 (online), DOI: 10.1109/ISMSIT58785.2023.10304904 (2023).

[13] Sarkan, H. M., Ahmad, T. P. S. and Bakar, A. A.: Using JIRA and Redmine in requirement development for agile methodology, *2011 Malaysian Conference in Software Engineering*, pp. 408–413 (online), DOI: 10.1109/MySEC.2011.6140707 (2011).

[14] Yu, Q., Zhao, N., Li, M., Li, Z., Wang, H., Zhang, W., Sui, K. and Pei, D.: A survey on intelligent management of alerts and incidents in IT services, *Journal of Network and Computer Applications*, Vol. 224, p. 103842 (online), DOI: https://doi.org/10.1016/j.jnca.2024.103842 (2024).

[15] Ikeuchi, H., Watanabe, A., Hirao, T., Morishita, M., Nishino, M., Matsuo, Y. and Watanabe, K.: Recovery command generation towards automatic recovery in ICT systems by Seq2Seq learning, *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–6 (online), DOI: 10.1109/NOMS47738.2020.9110370 (2020).

[16] Arora, R., Kumar, A., Soni, A. and Tiwari, A.: AI-Driven Self-Healing Cloud Systems: Enhancing Reliability and Reducing Downtime through Event-Driven Automation, *Preprints*, (online), DOI: 10.20944/preprints202408.1860.v1 (2024).

[17] Simili, E., Stewart, G., Roy, G., Skipsey, S. and Britton, D.: A hybrid system for monitoring and automated recovery at the glasgow tier-2 cluster, *EPJ Web of Conferences*, Vol. 251, EDP Sciences, p. 02047 (2021).

[18] Onodera, S., Asaoka, M., Yanase, T. and Namba, I.: Study on efficient analyzing method of operation manuals for runbook automation, *2014 IEEE Network Operations and Management Symposium (NOMS)*, pp. 1–15 (online), DOI: 10.1109/NOMS.2014.6838370 (2014).

[19] Goel, D., Husain, F., Singh, A., Ghosh, S., Parayil, A., Bansal, C., Zhang, X. and Rajmohan, S.: X-lifecycle learning for cloud incident management using llms, *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, pp. 417–428 (2024).

