障害アラートから生成AIのスクリプトの適用とメトリクスのしきい値を基準とした正常判定による復旧時間の短縮

佐藤 健斗[1]　平尾 真斗[2]　串田 高幸[1]

概要：アラートに対する対応は学生の手作業によって行われており，アラートの内容確認，原因調査，必要に応じた復旧作業までの一連の作業は個人の知識や経験に依存している．課題は，アラートの通知から監視担当の学生が行う復旧に時間かかかることである．本稿では，アラート通知に含まれるエラー内容やメトリクスを抽出，整形し，それをもとに対処スクリプトを生成するソフトウェアを提案する．提案手法では，監視システムから通知される JSON 形式のアラートをもとに，生成 AI に対してプロンプトを構築し，障害の復旧を行えるスクリプトを生成する．生成されたスクリプトを一時的に実行し，設定されたしきい値と比較して，メトリクスに異常があるかを判定する．異常がある場合は，再度プロンプトを構築する処理を行う．1 回目のプロンプトは，JSON 形式のアラートのみを入力として固定の文で構築し，2 回目以降は，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含めてプロンプトを更新する．基礎実験は，Redmine の Pod のメモリ使用率が 90%を超えた状況を対象として行った．その際，監視システムで設定されたメモリ使用率のしきい値を 90%として，生成されたスクリプトの実行によってメモリ使用率が 90%を下回るかどうかで生成されたスクリプトの判定を行った．その結果，スクリプトの実行後に対象のメトリクスが改善されたかどうかを基準とする方が，より正確な評価ができることが分かった．

1. はじめに

背景

IT システムは，クラウドサービスや分散システムの発展により，その規模や複雑さが急速に増大している [1]. このような環境下で，システムの安定稼働を維持するためには，継続的かつ包括的な監視が不可欠である．監視システムは，システムの状態やパフォーマンスをリアルタイムで把握し，異常を早期に検知することで，サービスの停止や性能の劣化を未然に防止する役割を果たしている [2]．特にアラートは，監視システムの重要な機能であり，障害発生時に運用担当者へ通知を行うことで迅速な対応を促している．適切に設定されたアラートは，運用担当者が障害を特定し，問題解決に向けて効果的な行動をするために利用される [3]．一方で，過剰なアラートや誤検知は運用担当者の負荷を増大させ，重要なアラートの見落としにつながるため，アラートの品質管理が重要である [4].

このような監視やアラート通知の仕組みを支えるソフトウェアとして，Prometheus や Alertmanager が使用されている．Prometheus は，システムやサービスの状態を数値化したメトリクスを一定間隔で収集し，時系列データベースに蓄積する監視ツールである [5,6]．これにより，CPU 使用率やメモリ使用率，ディスク使用率についての値を定量的に可視化し分析することができる．また，Prometheus にはクエリ言語である PromQL が備わっており，これをもちいて条件を指定することでアラートの発生条件を柔軟に定義することができる．Prometheus におけるアラートの管理は，Alertmanager で行うことができる．Alertmanager は，Prometheus が検出した異常な状態をトリガーとしたアラートを受け取り，運用担当者への通知を効率的に制御や管理をするためのコンポーネントである [7, 8]．Alertmanager の主な機能は，通知ルールの定義や Slack やメールへのルーティング，通知の重複の除去である．これにより，重要なシステムイベントが発生した際に，迅速かつ適切な対応を促進するための通知体制を構築することができる．アラートは，Prometheus 側で定義されたしきい値に達した際に，サービス名やクラスタ名，インシデントの種類を含んで Alertmanager に送信される．また，収集されたメトリクスを可視化するツールとして Grafana が使用される．Grafana は，取得したメトリクスに応じたダッシュボードを作成することができる [9].

[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

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 CDSL と呼ぶ) では，10 台の物理機器にブロードコム社の VMware 製品である ESXi が導入されており，Virtual Machine(以下 VM と呼ぶ) が配置されている．7 台の物理機器は，学生が作業や実験を行う目的で使用されている．2 台の物理機器は，DNS や DHCP，Jumpserver を運用するために使用されている．残りの 1 台は，外部に向けたサイトやアプリケーションを配置して公開するために使用されている．CDSL では，運用しているシステムの安定稼働を維持するために継続的な監視体制を構築している．監視の対象は，CDSL で稼働しているシステムの仮想化基盤である ESXi に対する ICMP を利用した疎通確認や各サーバのメモリ使用率やディスク使用率である．アラートの通知から対処までの流れを図 1に示す．この流れは，Prometheus，Alertmanager，チケット管理システム，対応体制の要素で構成されている．Prometheus と Alertmanager により構成されている監視システムは，アプリケーションやサービスのメトリクスの収集，監視を行う．設定されたしきい値を超えた場合は，Alertmanager がアラートを通知する．チケット管理システムでは，通知されたアラートがチケット作成システムによりチケット化され，Redmine に登録される．Redmine は障害に関するチケットや対応履歴を管理するツールである．対応体制は，監視担当の学生が，チケットとアラートの内容を参照して 1 次対応を行う．原因の特定が困難な場合は 2 次対応者，3 次対応者へとエスカレーションが行われる体制になっている．図 1では，Prometheus が監視対象からメトリクスを収集し，設定されたしきい値を超えた際に Alertmanager がアラートを通知する．その後，チケット作成システムでチケットが作成され，Redmine で管理される．監視担当の学生は，チケットに書かれた内容とアラートを参考に障害の対応を行う．障害の原因が特定できない場合や対処方法が判断できない場合は，2 次対応，3 次対応とエスカレーションが行われる．

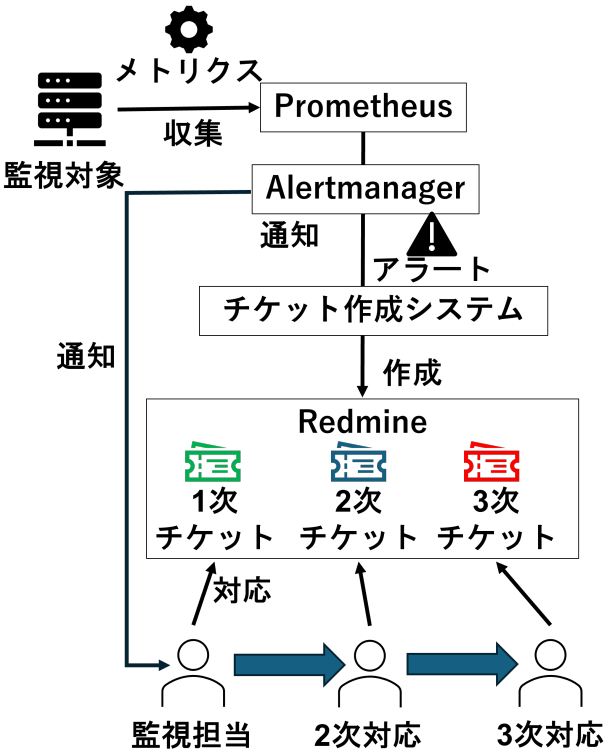

図 1: アラートの通知から対処までの流れ

課題

課題は，監視担当の学生によって行われる障害の調査，対処に時間がかかることである．例えば，2025 年 4 月 28 日に Redmine にアクセスできなくなる障害が発生した．障害発生時に，Redmine のメモリ使用率が 90%を超えたことを示すアラートと OOM-kill が発生したことを示すアラートが通知された．このアラートが通知された原因を調査した結果 Redmine に割り当てているメモリ量が不足していたことが分かった．対処としては，yaml ファイルに記述されていたメモリの Limits の値を 384Mi から 800Mi にあげた．この障害の解決までにかかった時間は，1 次調査で 1 時間 45 分，2 次調査で 16 分，対処までが 8 時間 46 分であった．各学生は，Slack に通知されたアラートの内容を確認し，Grafana やコマンドをもちいた調査によって原因を特定し，必要に応じて復旧作業を行っている．この対応は，障害発生時の初動として重要な役割を果たしている一方で，そのすべての判断が学生個人の知識や経験に依存しており，属人性が高くなっている．

さらに，対応時間外のアラートにおいては，即時の調査や復旧作業が行われていない場合があり，対応が翌週に持ち越されることがある．例えば，休日に発生したアラートについては，監視担当の学生が対応できる月曜日まで調査や復旧作業が実施されていない．また，類似しているアラートの発生に対しても同様の手順で原因の調査と復旧作業が行われており，作業の重複や効率の低下につながっている．

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

クラウド環境におけるインシデント対応の自動化を目的とした研究がある [10]．この研究では，生成 AI を活用してアラートから根本原因分析 (RCA) を行う支援ツールを

提案している．この提案は，アラートのタイトルや説明文を元に，関連するログやイベントを抽出し，その内容を言語モデルに入力して原因候補を出力することで，オンコール対応者の負担を軽減することを目指している．一方で，出力されるのは説明的な文章のため，対処にはオンコール対応者の知識に依存してしまう点に改善の余地がある．

クラウドネイティブな運用環境において，AI エージェントによる障害対応の自動化を目指した研究がある [11]. この研究では，障害の検知や調査，修復までの一連のプロセスを LLM ベースのエージェントが担う提案をしている．この提案は，各段階で取得されたシステムの情報をもとに，エージェントが必要な操作を自動実行する構成になっている．一方で，あらかじめ定義された実験環境に依存していることから，現実の構成環境に対する柔軟性に限界があり，実運用環境での適応には課題が残る．

LLM を使用してインシデントアラートの根本原因の特定を目指した研究がある [12]. この研究では，ログやメトリクス，アラートを統合し，データの分析や活用を容易にする RCA フレームワークを提案している．この提案により，原因候補を提示し，対応者の調査を支援している．一方で，出力が原因の特定のため，対処は対応者の知識に依存してしまう点に改善の余地がある．

クラウド環境における根本原因分析を目的とした研究がある [13]. この研究では，ツールによって拡張された LLM を自律的に動作させるエージェントとして使用し，ログやイベントの情報の収集，分析をしている．分析した結果をもとに原因や修正案，根拠をの特定までを一貫して行うフレームワークを構築している．しかし，出力は説明的な提案であり，実際の対処を行う操作までは自動化されていない点に改善の余地がある．

3. 提案

本提案の目的は，アラートの通知から復旧対応にかかる時間を短縮することである．そのため本研究では，監視システムから送信されるアラートの通知に対して復旧用のスクリプトを生成 AI により生成し，障害発生時の復旧作業を迅速化する手法を提案する．

提案方式

提案方式の概要を図 2に示す．監視システムは，監視対象を監視して，設定されたしきい値を超えた場合にアラートを通知される．生成 AI は，提案ソフトウェアで構築されたプロンプトをもとにスクリプトの生成を行う．提案方式は，正常状態のしきい値の特定，プロンプトの構築，スクリプトの適用，しきい値との比較と判定，スクリプトの保存で構成される．アラートの受信は，監視システムからアラートを受け取る．プロンプトの生成は受け取ったアラートを入力とした生成 AI へのプロンプトを構築する．実行と評価で，生成 AI による復旧用のスクリプトを一時的に実行し，実行後のメトリクスと設定されたしきい値の比較により復旧ができるかの判定を行う．例えば，監視システムでメモリ使用率が 90%を超えた際にアラートが通知されるようにしきい値を設定した場合，スクリプト実行後のメモリ使用率がが 90%以下に低下していれば，復旧できたと判定する．提案方式では，まず監視システムから監視対象の正常状態のしきい値の特定を行う．次に，通知される JSON 形式のアラートをもとに，生成 AI へのプロンプトを構築する．生成 AI はアラートの内容が含まれたプロンプトをもとに，原因の調査と復旧のためのスクリプトを生成する．得られた出力結果からは，復旧用のスクリプトのみを抽出し，障害の復旧ができるかを判定する．この際に，スクリプトによる復旧ができるかを判定するために，スクリプト適用後のメトリクスとしきい値を比較する．設定されたメトリクスのしきい値とは，アラートの種類ごとに事前に定められた正常に稼働している状態を示す指標のことである．スクリプトを一時的に実行し，その出力と設定されたしきい値を比較することで，実行後のメトリクスとしきい値との差分を抽出する．スクリプトによってメトリクスのしきい値を下回らなかった場合は，差分の内容となる改善されていない項目やエラーの内容を次回のプロンプトに追加し，生成 AI に再入力を行う．これにより，改善されたスクリプトが得られるようにする．この繰り返し処理は，スクリプトにより復旧することができると判定されるまで継続する．復旧できるスクリプトが生成された段階で，スクリプトファイルとして保存される．

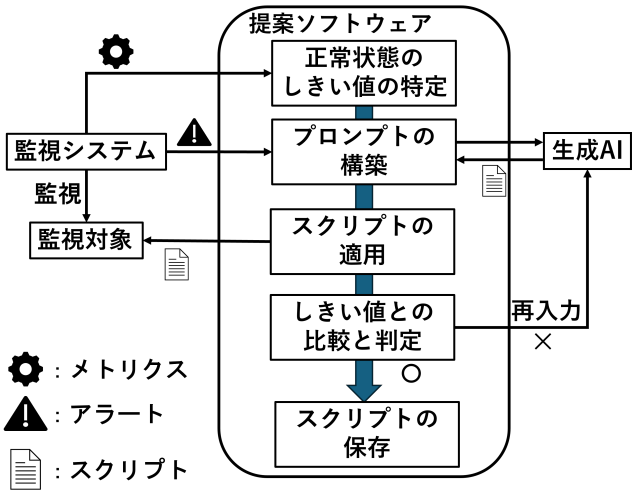

図 2: 提案方式の概要

正常状態のしきい値の特定

正常状態のしきい値の特定では，監視対象のメトリクスに対して，設定されているしきい値を監視システムから取得する．例えば，Pod のメモリ使用率が 90%未満が正常

な状態であると設定された値のことである．このしきい値は，アラートの種類ごとに定められている．

プロンプトの構築

プロンプトの構築では，受信したアラートをもとに，生成 AI に対してスクリプトの生成用のプロンプトを構成する．プロンプトには，監視システムから通知された JSON 形式のアラートと復旧するためのスクリプトの生成を依頼する文章を含めて構築する．初回の入力は，アラートのみで構築されるが，再入力時にはしきい値とメトリクスとの差分やエラー内容を含めてプロンプトを構築する．

スクリプトの適用

スクリプトの適用では，生成 AI から出力されたスクリプトを実行して，アラートに対しての復旧が行えるかを確認する．

しきい値との比較と判定

しきい値との比較と判定では，スクリプト適用後のメトリクスと，はじめに特定したしきい値を比較して，正常な状態になっていない場合は復旧ができていないと判定し，出力に含まれるエラー内容やしきい値との差分をプロンプトに追加して生成 AI に再入力する．

スクリプトの保存

スクリプトの保存では，復旧が成功したと判定されたスクリプトをファイルとして保存する．保存されたスクリプトは，同様のアラートが通知された際に適用することができる．

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 CDSL において運用しているシステムからアラートが通知された際の対応を想定する．図 3に提案ソフトウェアを使用したユースケースシナリオを示す．

監視システムは，CDSL 内の監視対象に対して継続的な監視を行っている．監視対象に対して設定されたしきい値を超えた際にアラートが通知され，提案ソフトウェアでアラートを JSON 形式でを受け取る．その後，受け取ったアラートをもとに生成 AI によって対処用のスクリプトを生成することで，対応を行う．

4. 実装

本提案方式を実装するために，Python3.12.3 上で Flask をもちいた Web アプリケーションを作成した．本実装では，Alertmanager から通知されるアラートを JSON 形式で受け取り，Gemini API をもちいてアラートに対する対処用のスクリプトを生成する処理を行う．使用したライブラリは，Flask，google-generativeai，re，os である．それぞれの用途を説明すると，Flask は Web サーバーの作成に使用され，Alertmanager からの POST リクエストの受信に使用する．google-generativeai は Gemini API の呼び出しに使用し，対話型 AI によるスクリプトの生成に使用する．re は生成されたテクストから bash スクリプトを抽出するための正規表現処理に使用する．os は環境変数から API キーの取得，生成したスクリプトファイルの保存や権限付与のファイル操作に使用する．

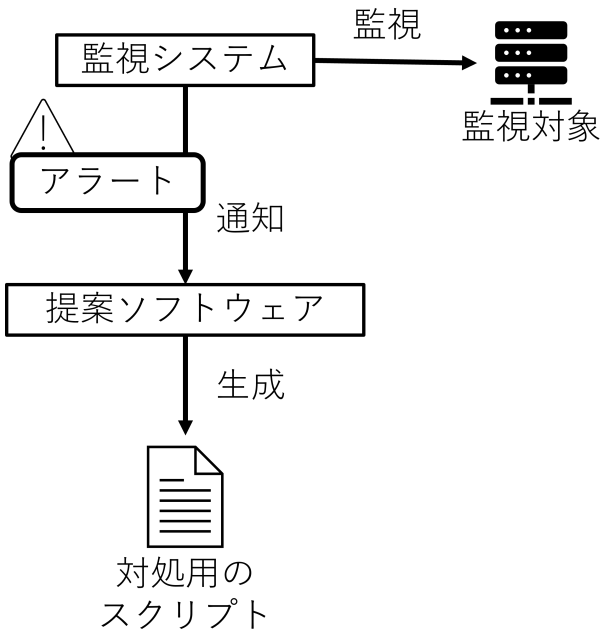

図 3: ユースケースシナリオ

開発した提案ソフトウェア (app.py) の概要を図 4に示す．

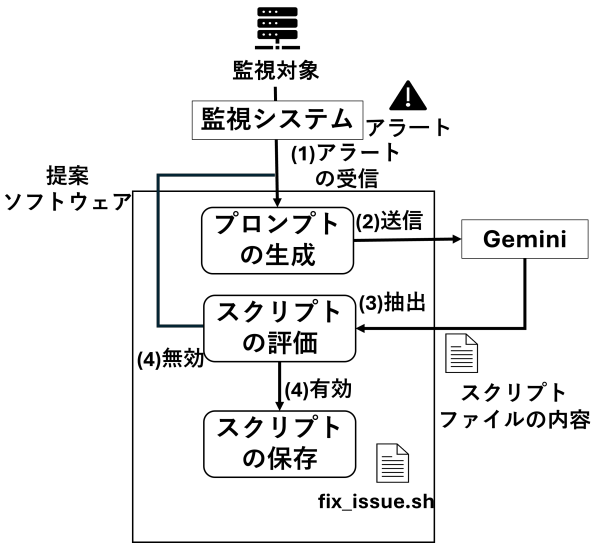

図 4: app.py の概要

図 4では，監視システムから JSON 形式のアラートを受け取り，スクリプト生成用のプロンプトとして整形され，

Gemini API に送信する流れを表している．Gemini からの応答を受け取り，その中から bash スクリプトを正規表現で抽出する．次に，抽出したスクリプトを評価関数で判定し，有効と判定された場合にスクリプトファイルとして保存，実行権限を付与し処理を終了する．無効と判定された場合には，評価結果をもとにプロンプトを再生成し，再度 Gemini にスクリプトの生成を依頼する．この繰り返しは，スクリプトの実行結果におけるエラー数の削減を指標として行う．最後に有効なスクリプトが得られた時点で処理を終了し，その過程をログとして出力を行う．

提案ソフトウェアである app.py の主な関数と役割について Gemini への入力としては，JSON 形式のアラートを含むプロンプトを構築し，問題の根本原因の調査，対処に必要なシェルスクリプトの生成を行う．生成されたレスポンスからは対処用のスクリプト部分を抽出し，そのスクリプトの有効性を評価する．評価は，スクリプトに含まれるコマンドやスクリプトの実行結果である標準出力やエラー出力にもとづいて行う．例えば，kubectl patch や systemctl restart の対処に有効と判断されるコマンドが含まれているかを確認する．また，スクリプトの実行時のエラーの有無を判定する．スクリプトが有効と判定された場合には，そのスクリプトを fix_issue.sh として保存する．一方で，無効と判定された場合には，前回の生成内容と評価結果を踏まえてプロンプトを生成し直し，Gemini に再送信してスクリプトの生成を繰り返す．

また提案ソフトウェアである app.py で使用される関数は，extract_script，script_effective，generate_script，regenerate_prompt，save_script がある．

extract_script は，Gemini API からの応答テキストに含まれる bash スクリプトを抽出する関数である．Markdown 形式のコードブロックを正規表現で検出し，#!/bin/bash から始まるスクリプト部分のみを返す．

script_effective は，抽出されたスクリプトがアラートに対して有効な対処を行うものかを判定する評価関数である．スクリプト内に含まれるコマンドを検査し，定義された有効なコマンドとの一致にもとづいて有効性を判定する．

generate_script は，整形したプロンプトを Gemini API に送信し，スクリプトの生成を依頼する関数である．

regenerate_prompt は，無効と判定されたスクリプトが出力された場合に，評価結果をフィードバックとして組み込み，プロンプトを動的に再生成する関数である．この関数により，対処用のスクリプトが生成されるまで繰り返し処理が行われる．

save_script は，裕子と判定されたスクリプトをファイルとして保存して，必要に応じて実行権限を付与する関数である．保存時にはファイル名の重複を避ける処理も含まれている．

5. 評価実験

評価実験では，提案手法により生成されたスクリプトの有効性と評価関数の判定基準がスクリプトの生成に与える影響を確認する．CDSL では，監視担当の学生がアラートやチケットの内容を確認し，原因の調査や対処を手作業で行っており，知識や経験に依存した対応となっていた．これに対して，本提案では生成 AI による対処用のスクリプトが生成と有効性の判定の仕組みを導入している．実験では，スクリプト内で実行されたコマンドの数を指標として記録し，復旧に必要な操作量を評価する．

実験環境

実験で使用した環境を図 5に示す．

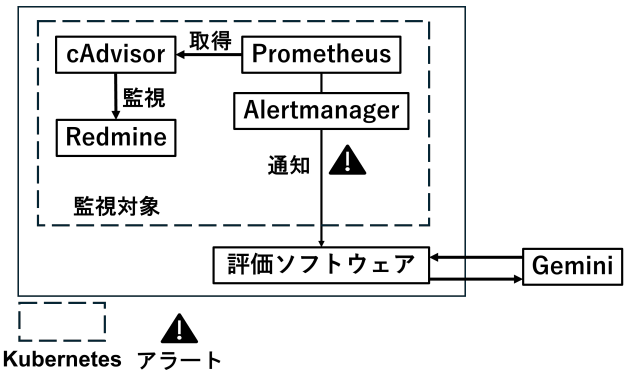

図 5: 実験環境

実験は，Ubuntu24.04.2 がインストールされた仮想マシンで行った．仮想マシンは 2 台で構成されており，構成は，vCPU が 2 コア，メモリが 4GB，ストレージが 30GB である．本実験では，K3s による Kubernetes クラスタを構築した．監視システムには，Prometheus と Alertmanager を使用してアラートの通知を行い，Redmine の Pod を監視対象として設定した．実験で通知されたアラートは 2 項目ある．Redmine にアクセスできない障害で通知されたアラートを表 1に示す．

表 1: Redmine にアクセスできない障害で通知されたアラート

監視項目	監視内容	しきい値
メモリ使用率	Pod のメモリ使用率を監視する	90%以上
OOM-kill の発生	Pod 内で OOM-kill が発生したかを確認	1

Prometheus は監視対象の監視を行い，しきい値を超え

た際に Alertmanager にアラートの通知を行う．Alertmanager は，Prometheus から受け取った内容を監視担当者に通知を行う．生成 AI モデルは，Google の Gemini 1.5 Flash モデルを使用している．提案ソフトウェアでは，Alertmanager から通知を受け取り，アラートの内容とプロンプトを Gemini に送信することでスクリプトの生成を行う．また，生成されたスクリプトの有効性を評価し，必要に応じてプロンプトを動的に再度生成する処理を行う．

実験結果と分析

本実験では，2025 年 4 月 28 日に発生した Redmine にアクセスできなかった事例を再現する．この事例では，Prometheus でメモリ使用率が 90%を超えた際にアラートを通知する条件と Pod 内で OOM-kill が発生した際にアラートを通知する条件の 2 つの監視条件を設定した．この障害時に，Redmine に割り当てられた Pod でメモリ使用率が 90%を超えたことを示すアラートと OOM-kill が発生したことを示すアラートの 2 つのアラートが通知された．実験では 1 回目のプロンプトを固定して Gemini 1.5 Flash モデルを使用してスクリプトの生成と評価を行った．1 回目に使用したプロンプトをプロンプト 1に示す．

プロンプト 1: 1 回目に使用したプロンプト

```
1 アラート情報に基づいて、問題の原因を分析し、
  適切な対処スクリプト（fix_issue.sh）を
  生成してください。
2
3 アラートの情報：
4 << ここにJSON形式のアラートの内容を挿入 >>
5
6 スクリプトの目的:
7 - アラートの根本原因を解消すること
8
9 出力形式: [fix_issue.sh]
  から始めてください。
```

また，提案方式において評価関数のしきい値がスクリプトの生成結果に与える影響を分析するため，2つの条件を設定し，それぞれの出力結果を比較する基礎実験を行った．基礎実験では，生成されたスクリプトを実行して，対象となるメトリクスが設定したしきい値を下回るかどうかをもって，スクリプトの有効性の評価について確認した．今回の実験では，メモリ使用率が 90%を下回るかどうかを確認した．2つの条件の比較について表 2に示す．

条件 A では，df や free，top の確認のコマンドがスクリプトに含まれていれば有効と判定する基準である．一方で条件 B では，sysctl や ulimit，kubectl patch のシステム設定の変更を伴うコマンドを含む場合のみを有効と判定する基準である．この基準は，作成さるスクリプトに設定の変更を行えるコマンドが含まれていない場合は，確認のみで終わるスクリプトが作成されるためである．条件 A では，ディスクの容量不足やメモリの高負荷に対する確認のコマンドやログの削除，メモリの解放を行う軽微な対処コマンドを含むスクリプト生成され，有効と判定された．一方条件 B では，システム設定の変更を伴う対処コマンドが含まれるスクリプトのみを有効としたため，1 回の試行では対処のスクリプトの生成に失敗したため，再度エラーの内容や実行結果をもとにプロンプトの生成が必要と判定された．しかし，有効と判定された条件 A でも実際にスクリプトを実行してもメモリ使用率がしきい値を下回ることはなく，復旧に至らなかった．この結果は，スクリプトに含まれるコマンドの種類だけではしきい値を下回る改善に至らなかったが，スクリプトの実行後に対象のメトリクスが改善されたかどうかを基準とする方が，より正確な評価ができることが分かった．

表 2: 条件の比較

	条件 A	条件 B
評価条件	確認系コマンドを含む	システム設定変更コマンドのみ
評価関数による判定	有効	生成失敗
実行結果によるメモリ使用率の改善	メモリ使用率が 90%以上のまま	スクリプトなし
実行結果による判定	無効	無効

6. 議論

本稿では，Alertmanager から通知されるアラートを JSON 形式のまま Gemini API のプロンプトとして入力している．しかし，実際のアラートには，生成 AI にとって不要な情報や曖昧な表現が含まれているため，対処を行えるスクリプトが生成されないケースも発生した．特に，対象リソースや以上の具体的な状況が明示されていない場合は，再起動による対処にとどまるスクリプトが出力された．この問題を解決するためには，アラートの種類や重要度に応じて必要な情報を抽出，整形し，AI にとって有効な形に加工して入力する前処理が重要である．例えば，対象の Pod 名やリソースの使用状況，過去の対応履歴を併せて入力することで，より正確かつ有効な対処を行えるスクリプトを生成ができる．

本稿では，アラートにもとづいて障害の対処を行えるス

クリプトを生成する方式を提案し，実装において Gemini API をもちいた生成モデルによる出力を利用した．この方式は，人による調査や判断に代わり，生成モデルによる迅速な対処案を実現する提案である．しかし，この方式は生成された出力の内容に強く依存しており，その限界について検討する必要がある．生成モデルは高い言語生成能力を有している一方で，出力される内容には一貫性や正確性が保証されるわけではない．例えば，あいまいな説明的な文章が含まれる場合や無効なコマンド，不要な手順が含まれるケースが存在した．この問題を解決するためには，生成モデルの出力を補完，検証する仕組みの導入が必要である．具体的には，実行環境の構成や障害内容にもとづく条件の事前の確認，生成されたスクリプトの構文チェックや解析を行う．生成モデルの出力に対して段階的に評価，修正するプロセスを整備することで安定した障害対処を行うことができる．

本稿では，アラートが通知された際に，対処を行えるスクリプトを生成するソフトウェアを提案した．このソフトウェアでは，毎回アラートにもとづいて，新たにスクリプトを生成しており，同一あるいは類似したアラートに対しても再びスクリプトの生成が行われる構成になっている．しかし，この方式では，過去に発生した障害に対する対処の手段やスクリプトが蓄積，再利用されないため，同じ障害に対しても何度も類似したスクリプトを生成し直す問題が生じる．また，過去に対処した内容が活用されないため，生成されるスクリプトの精度が安定せず，毎回試行錯誤を繰り返す処理になってしまう．この問題を解決するためには，過去のアラートと生成されたスクリプトをペアで蓄積し，それらをデータベースに格納する．新しいアラートが通知された際には，まず蓄積されたアラートとの類似度を判定し，既存の対応履歴から対処が可能なスクリプトがある場合は新しく生成せずに実行する仕組みを導入することで障害対処の迅速化ができる．さらに，過去のスクリプトの実行結果や有効性も記録しておくことで，対処を行う精度の継続的な改善にもつながる．

7. おわりに

クラウドや仮想化技術の普及により，IT システムの構成は複雑かつ大規模になっている．このような環境においては，継続的な監視によるシステムの状態を把握し，異常を迅速に検知，対処を行うことが重要となっている．CDSL では Prometheus や Alertmanager を使用したシステムの監視を行っているが，アラートの対応は学生が手動で実施している．課題は，アラートの通知時の対応を学生が行っているため，復旧に時間がかかることである．本稿では，課題を解決する提案手法として，アラートにもとづき対処を行うスクリプトを生成するソフトウェアの開発を行った．提案したソフトウェアは，Alertmanager から通知されたアラートをもとに Gemini を使用して復旧用のスクリプトを生成する．生成されたスクリプトの実行結果と定義された理想の状態を比較し，有効性の判定を行う．理想の状態と一致した場合は有効と判定され，それ以外の場合は，再度エラー内容や比較した際の差分をプロンプトに含めてスクリプトの生成を繰り返すことで，対応の属人性を排除し，迅速かつ効率的な障害対応を行う．基礎実験では，Redmine の Pod に対してメモリ使用率が 90%を超える状況を再現した．復旧用のスクリプトがしきい値を下回るように改善できるかを確認した．実験の結果は，スクリプトの実行後に対象のメトリクスが改善されたかどうかを基準とする方が，より正確な評価ができることが分かった．

参考文献

[1] Pourmajidi, W., Steinbacher, J., Erwin, T. and Miranskyy, A.: On challenges of cloud monitoring, *arXiv preprint arXiv:1806.05914* (2018).

[2] Liu, S., Zhou, Y., Ying, L., Tian, Y., Zhang, J., Zhou, S., Cui, W., Lin, Q., Moscibroda, T., Zhang, H. et al.: RCInvestigator: Towards Better Investigation of Anomaly Root Causes in Cloud Computing Systems, *arXiv preprint arXiv:2405.15571* (2024).

[3] Jiang, G., Chen, H., Yoshihira, K. and Saxena, A.: Ranking the importance of alerts for problem determination in large computer systems, *Proceedings of the 6th international conference on autonomic computing*, pp. 3–12 (2009).

[4] Ahmed, T., Shah, A., Kolla, M. and Yellasiri, R.: Reduction of Alert Fatigue using Extended Isolation Forest, *2021 International Conference on Forensics, Analytics, Big Data, Security (FABS)*, Vol. 1, pp. 1–5 (online), DOI: 10.1109/FABS52071.2021.9702617 (2021).

[5] 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).

[6] 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).

[7] 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).

[8] 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).

[9] 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).

[10] Chen, Y., Xie, H., Ma, M., Kang, Y., Gao, X., Shi, L., Cao, Y., Gao, X., Fan, H., Wen, M., Zeng, J., Ghosh, S., Zhang, X., Zhang, C., Lin, Q., Rajmohan, S., Zhang, D. and Xu, T.: Automatic Root Cause Analysis via Large Language Models for Cloud Incidents, *Proceedings of the Nineteenth European Conference on Computer Systems*, EuroSys '24, New York, NY, USA, Association for Computing Machinery, p. 674–688 (online), DOI: 10.1145/3627703.3629553 (2024).

[11] Chen, Y., Shetty, M., Somashekar, G., Ma, M., Simmhan, Y., Mace, J., Bansal, C., Wang, R. and Rajmohan, S.: AIOpsLab: A Holistic Framework to Evaluate AI Agents for Enabling Autonomous Clouds (2025).

[12] Sarda, K., Namrud, Z., Watts, I., Shwartz, L., Nagar, S., Mohapatra, P. and Litoiu, M.: Augmenting Automatic Root-Cause Identification with Incident Alerts Using LLM, *2024 34th International Conference on Collaborative Advances in Software and COmputiNg (CASCON)*, pp. 1–10 (online), DOI: 10.1109/CASCON62161.2024.10838171 (2024).

[13] Wang, Z., Liu, Z., Zhang, Y., Zhong, A., Wang, J., Yin, F., Fan, L., Wu, L. and Wen, Q.: Rcagent: Cloud root cause analysis by autonomous agents with tool-augmented large language models, *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pp. 4966–4974 (2024).

