コンテナ監視のメトリクスとログの記録時間の照合による原因を記載したアラートの通知

坂井 萌桜[1]　平尾 真斗[2]　串田 高幸[1]

概要：監視システムは，対象に異常が起きた際にアラートを通知する目的で使用される．これらのシステムがメトリクスを取得し，監視することで，システムを安全に運用している．アラートの閾値や継続時間は監視システムの管理者の判断に依存している．課題は，システムの異常がアラートとして通知されず，異常に対する調査や対処ができないことである．提案は，監視システムにおいて監視対象のアプリケーションの HTTP ステータスコードが 200 以外である場合に，その原因箇所を監視対象のログを確認することで特定し，異常があった場合にアラートを通知する手法である．ログの確認は，Istio-proxy，アプリケーションコンテナ，ノードの順に行う．提案手法によるアラート通知は，監視システムに設定されたアラートルールとは別に行われる．基礎実験を，監視対象のアプリケーションの HTTP ステータスコードが 503 である 4 件のケースを対象として行い，原因箇所を特定できるかを検証した．また，ログ抽出の条件のうちログの記録時間に関する条件が，実際の異常に対してログを取得できる条件であるかを評価した．既存のアラートルールでは，HTTP ステータスコードが 200 以外の状態が 180 秒間継続した場合にアラートを通知する設定となっているため，これら 4 件のケースはアラートルールによるアラート通知の対象外である．結果は，4 件中 3 件のケースにおいてログにもとづいた原因箇所の特定ができた．

1. はじめに

背景

生活や社会活動，産業は重要インフラを含む IT システムに支えられており，これらへの依存は IT システムの発展によりさらに強まっていくとされている [1]．それに伴い，IT システムを安定的に運用することの重要性が高まっている．安定した IT システムの運用を実現するためには，システムの状態を継続的に把握する監視が不可欠である．システムが期待通りに動作し，高い可用性を維持するためには，効果的にシステムを監視する仕組みが求められる [2]．

システムの監視においては，システムから収集された各種メトリクスにもとづいてシステムの異常を検知し，必要に応じてシステムの管理者に通知を行う，アラートという仕組みが重要な役割を果たす．メトリクスとは，システム内で継続的に発生する CPU 使用率やメモリ使用量，レスポンス時間，エラー率を例とする，時系列データとして扱われる数値データのことである [2]．アラートは，システムの異常やパフォーマンスの低下を即座に把握し，それらに対し迅速な対応をするための手段となる．アラートが対象のシステムの異常に対し適切に対応するためには，メトリクスの閾値やアラートの通知のタイミングの設定を例としたアラートの通知条件の適切な設定が必要である．不適切なアラートは，誤検知や通知漏れによるアラート対応者の作業負荷の増加や，対応の遅延を引きおこす [3]．

東京工科大学コンピュータサイエンス学部の Cloud and Distributed Systems Laboratory（以下 CDSL と呼ぶ）では，物理マシンやシステム, アプリケーションを安定的に運用するために，監視ツールとして Prometheus と Grafana を使用している．

Prometheus とは，システムやアプリケーションのメトリクスを収集するオープンソースのソフトウェアである[*1]．Prometheus は，監視対象のメトリクスに対して柔軟にアラートルールを定義できる仕組みを提供しており，管理者は任意の閾値条件を設定し，外部の通知システムへアラートを送信するように構成することができる．このような柔軟性は，システムや運用の方針に応じたアラート設計を実現させるために有用であるが，閾値条件の妥当性や精度は管理者の経験や判断に完全に依存している [4]．

Grafana は Prometheus で収集したメトリクスの数値を

[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

[*1] https://prometheus.io/(参照 2025/7/21)

ダッシュボードで可視化することができるオープンソースのデータ可視化プラットフォームである．時間経過によるメトリクスの変動をグラフとして表示することができるため，システムの状態の把握や，アラートが通知された際の前後関係の分析に使用されている [5].

CDSL では，監視しているシステムのメトリクスを学生が Grafana で確認し，スプレッドシートに記録している．監視対象として，CDSL で運用している doktor がある*2．doktor は CDSL が外部に公開しているサイトであり，所属している学生が作成したテクニカルレポートが公開されている．これは，学生の研究成果を記載したレポートである．doktor は Kubernetes クラスタ上に構築されており，1 台のマスターノードと 3 台のワーカーノードから構成されている．これらのノードは研究室の外部にサイトを公開する目的で使用されている ESXi 上に配置されている．記録作業の流れの例を図 1に示す．

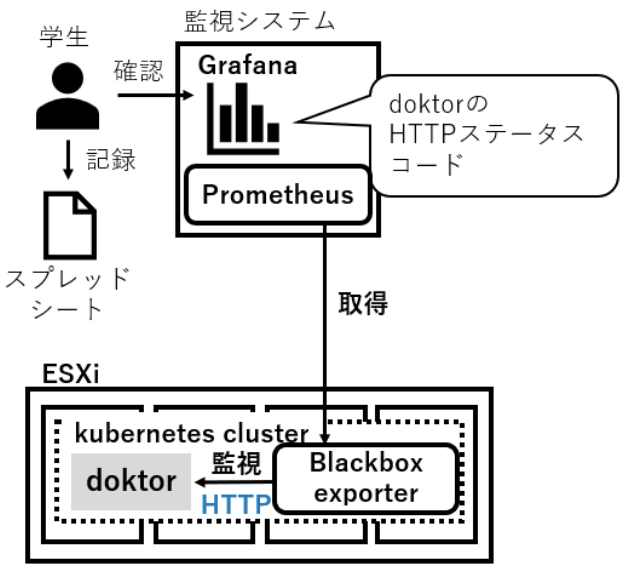

図 1: 監視作業の流れの例

doktor の HTTP ステータスコードについて記録する際には，Blackbox exporter，Prometheus，Grafana を使用する．Blackbox exporter は HTTP 経由で doktor にアクセスし，ステータスコードを取得する．Prometheus は Blackbox exporter が取得したステータスコードのメトリクスを収集し，時系列データとして保存する．Grafana は Prometheus から取得したメトリクスを可視化する．学生は Grafana に表示された doktor の HTTP ステータスコードを確認し，スクリーンショットを取得してスプレッドシートに記録する．

メトリクスを Grafana で確認し記録する作業の目的は，適切なアラート条件の設定である．適切なアラートとは，システムに異常がある時のみ通知されるアラートのことを指す．アラートルールの閾値条件が緩くアラートの感度が高い場合，アラートの数が増える．システムの異常がない場合にもアラートが通知されるようになり，真の異常を見逃すリスクと運用コストが増加する [6]．反対に閾値の設定が厳しくアラートの感度が低い場合，通知されるアラートの件数は減るが, システムの異常を見逃すケースが増加する．このように，システムの異常の検出と誤検知の削減はトレードオフの関係にあるため，適切な閾値条件の設定はシステムの安全な運用のために必要である [7].

CDSL において，アラートルールは所属している学生の判断にもとづいて設定されている．アラートの通知がされた際には CDSL 内で定めた対処の方法に従ってアラートの対処が行われる．

課題

課題は，CDSL で運用，管理されているシステムやアプリケーションにおいて，アラートルールの設定により監視対象の異常が通知されない場合があるということである．課題の概要を図 2に示す．

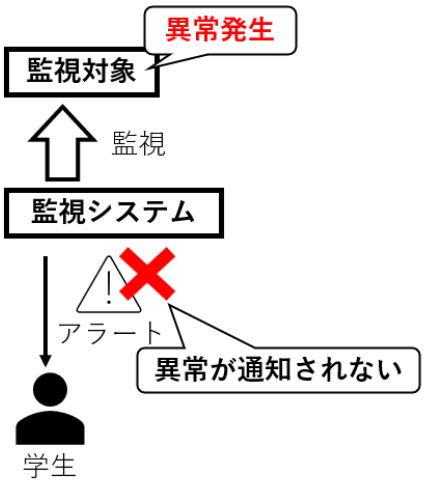

図 2: 課題の概要

監視対象において異常が発生していても，監視システムにおけるアラートルールの条件を満たさない場合はアラートが通知されない．

2025 年 6 月 16 日 0 時 0 分から 2025 年 6 月 18 日 23 時 59 分の 3 日間において，CDSL で運用，監視している doktor を構成するシステムに対する監視項目の 1 つである doktor front page HTTP status code で，ステータスコード 503 のメトリクスが計 11 件確認された．監視構成は，図 1に示した構成と同一である．HTTP ステータスコード 503 の発生日時と継続時間，および対応ノードを表 1に示す．11 件のうち 9 件はマスターノードで，2 件はワーカーノードで

*2 https://doktor.tak-cslab.org/(参照 2025/7/21)

確認された．3 台のワーカーノードを区別するため，表中においてはそれぞれワーカーノード 1，ワーカーノード 2，ワーカーノード 3 と表記する．

表 1: HTTP ステータスコード 503 の発生日時と継続時間，および対応ノード

日付	時刻	継続 (秒)	ノード
2025/6/16	4:13:00〜4:13:45	45	マスターノード
2025/6/16	4:39:30〜4:39:45	15	マスターノード
2025/6/17	7:57:00〜7:57:30	30	マスターノード
2025/6/17	8:04:45〜8:05:15	30	マスターノード
2025/6/17	8:28:15〜8:28:30	15	ワーカーノード 2
2025/6/17	13:13:00〜13:13:15	15	マスターノード
2025/6/17	14:21:00〜14:21:15	15	マスターノード
2025/6/17	14:29:30〜14:29:45	15	マスターノード
2025/6/17	15:42:15〜15:42:45	30	マスターノード
2025/6/17	20:22:15〜20:22:30	15	マスターノード
2025/6/18	19:28:15〜19:28:30	15	ワーカーノード 1

CDSL で運用している Prometheus では，doktor の HTTP ステータスコードに対して，200 以外の値が 180 秒間継続した場合にアラートを通知するというアラートルールが設定されている．この 11 件は，アラートルールを満たさないためアラートとして通知されなかった．なお，HTTP ステータスコードが一時的に 200 以外となる場合は，瞬間的な通信エラーによる一過性のものであるケースもあり，そのすべてを即座にアラートとして通知すると過剰にアラートの通知が行われることになる．実際，取得されたメトリクスの異常の全てがシステムの異常を意味するとは限らない [8].

各章の概要

第 2 章では関連研究について述べる．第 3 章では提案方式とユースケースシナリオについて述べる．第 4 章では実装について述べる．第 5 章では実験環境や提案方式と実験結果についての評価を述べる．第 6 章では議論を行う．第 7 章は本稿におけるまとめである．

2. 関連研究

アラートの閾値条件の設定が手動で直感的であるために通知される誤ったアラートや見逃されるシステム上の問題に対して，システムコンポーネントの挙動を分析することで動的な設定を生成することを提案している研究がある. [9]．過去のシステムのメトリクスを参考に，システムの通常時のメトリクスをモデル化することで，より柔軟で文脈に応じたアラート条件を動的に設定している．モデル化には変化点分析と時間的変化の把握，分類回帰木による分類を用いており，システムの直近の正常なメトリクスの特定，周期的な時間変化の特定，メトリクスの正常範囲の特定を行っている．この研究では，過去のメトリクスから動的な閾値の設定することによりアラート通知の閾値条件の設定を行うが，メトリクスの誤検知やノイズは考慮していない．

ネットワーク管理者の業務負荷と可用性に応じて，システムの異常を知らせるアラートの設定を自動化，スケジュール対応させることを提案している研究がある [10]. この研究ではアラートが通知された際，アラートの確認，対応をする人がいなければ意味がないとし，アラート対応者のスケジュール，応答できる件数を考慮してアラートの優先度やアラートの通知タイミングを調整する仕組みも提案している．可用性に焦点が当てられており，実運用におけるアラート処理の効率化を図っているが，アラートの閾値の設定段階においてどのようなメトリクスを基準とし，異常なメトリクスをどのように定義するのかという観点は明示的には扱われていない．

アラートの誤通知の削減や有用なアラートの分類精度の向上を目的として，ログ内に内在するパターンの変化に着目し，エントロピーにもとづいて注目すべきログを抽出，分類する手法を提案している研究がある [11]．ログ内に含まれる単語を抽出し，その単語の出現回数にもとづいて対数エントロピーを計算することで，そのログが特徴的なログであるかを評価し重み付けを行っている．重みの変動が大きい区間をシステムの異常の兆候とみなすことで，アラートを通知する対象となるシステムの絞り込みを行っている．この研究ではログを用いることでシステムの異常を確認しているためシステムの異常に対し詳細な分析ができる一方で，リアルタイム性での対応という観点では制約も存在する．

3. 提案方式

本提案では, 監視対象としているシステムにおいて HTTP ステータスコードが 200 以外であることが確認された際に，その原因を特定し，監視対象システムに問題があると判断された場合にアラートを通知することを目的とする．提案手法の具体的な適用例として doktor をもちいる．doktor におけるアラートルールでは，HTTP ステータスコード 200 以外である状態が 180 秒間継続した場合にアラートを通知する設定になっている．

本提案において，メトリクスの異常とは，監視システムが取得した HTTP ステータスコードのメトリクスでステータスコードが 200 以外である値が確認された状態を指す．提案手法では，メトリクスの異常の原因を確認するためにまず監視システムが取得したメトリクスを起点とし，それに対応する監視対象システムのログを確認する．提案の概要を図 3に示す．

監視システムは，監視対象からメトリクスを取得し，アラートルールに従ってアラートを通知する．一方，提案方

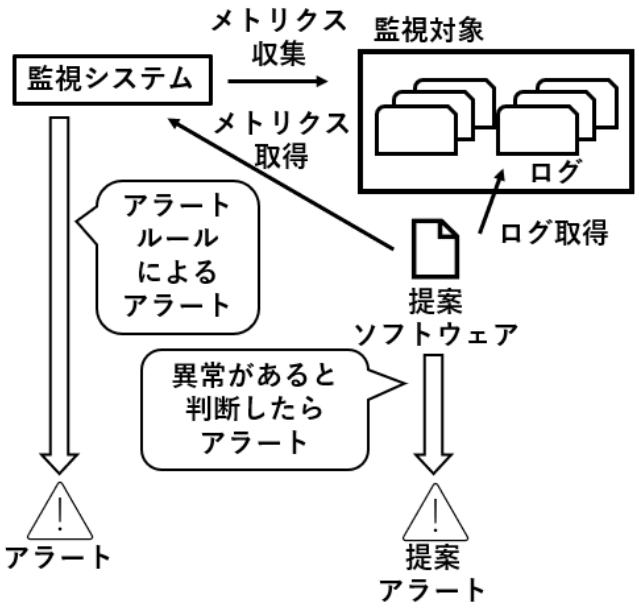

図 3: 提案の概要

式は監視システムからメトリクスを取得し，監視対象からログを取得することでメトリクスの異常の原因を確認する．その結果監視対象に原因があると判断された場合には，提案ソフトウェアがアラートルールとは別にアラートを通知する．この手法の対象には，監視システム，Istio-proxy，アプリケーションコンテナがある．監視システムは独立した Kubernetes クラスタ上に構成されており，Istio-proxy やアプリケーションコンテナは，監視システムとは別の Kubernetes クラスタ上で動作している．

ログを確認するフロー

Kubernetes クラスタ上に構成されている doktor を具体例としてもちいて，メトリクスで検知された異常に対してその原因を特定するためのログの確認フローを説明する．doktor では，外部からのリクエストは Istio-proxy を経由してアプリケーションコンテナへ到達する構成になっており，これらのコンテナは Kubernetes 上の pod 内で動作している．そのため，本提案では異常の原因箇所を推定するために，監視システムが取得するメトリクスを起点として，Istio-proxy のログ，アプリケーションコンテナのログ，pod の状態を順に確認するフローの設計を行った．ログを確認するフローを図 4に示す．

まず，Istio-proxy のログを確認する．Istio-proxy は，アプリケーションのコンテナと外部との通信を仲介する．監視システムのメトリクスにおいて HTTP に関する異常が観測された場合，そのリクエストは監視システムから Istio-proxy に対して送信されている．したがって，Istio-proxy のログにおいて監視システムからのリクエストに対応するものを確認対象とする．Istio-proxy のログメッセージでは通信先を確認することができる．ログメッセージに

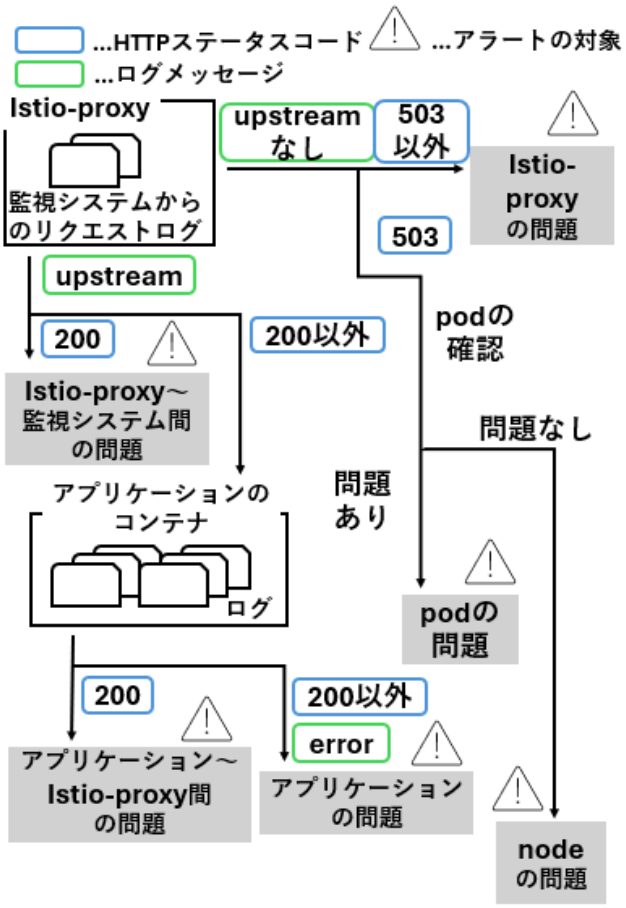

図 4: ログを確認するフロー

upstream という文字列が含まれている場合は，Istio-proxy がアプリケーションのコンテナと通信したログであると判断できる．

図 4の左側のフローでは，upstream を含むログの HTTP ステータスコードに着目する．監視システムのメトリクスで HTTP ステータスコードを取得した時刻よりも前の時間を確認対象の時間とする．ステータスコードが 200 であれば，アプリケーションのコンテナからの応答に異常はなかったと判断できる．この場合は，Istio-proxy から監視システムの問題と判断しアラートの通知を行う．ステータスコードが 200 以外の場合は，アプリケーションのコンテナが異常な応答を返している場合がある．これを確認するために，対応するアプリケーションコンテナのログを確認する．確認対象とするのは，Istio-proxy のログで 200 以外のステータスコードが出力された時刻から，1 分前までのログである．アプリケーションコンテナのログにおいて，同一リクエストに対する応答がステータスコード 200 であれば，アプリケーションでは異常な処理は行われておらず，アプリケーションと Istio-proxy 間の通信に問題があったと判断しアラートの通知を行う．アプリケーションコンテナのログにも 200 以外のステータスコードが記録されている，もしくはログメッセージに error という文字列が含ま

れる場合，アプリケーション自体が異常な応答を返したと判断しアラート通知の通知を行う．

図 4の右側のフローでは，Istio-proxy のログに upstream が含まれていない場合を扱う．これは，Istio-proxy がアプリケーションコンテナとの通信自体に失敗していることを意味している．この場合，HTTP ステータスコードが 503 であれば，アプリケーションと通信できず，応答も得られなかった状況であるため，pod の状態確認を行う．pod の確認では，該当 pod が存在しているか，状態が Running であるか，異常なイベントが発生していないかをチェックする．これらに問題がある場合，pod の問題としてアラート通知の対象とする．特に異常が見られない場合，node の問題と判断する．node の問題とは，監視システムからのリクエストに対して Istio-proxy が 503 エラーを返した場合において，pod にも異常が確認されなかった場合を示す．node の問題と判断された場合もアラートを通知する．ステータスコードが 503 以外である場合，Istio-proxy 内部での問題とみなし，Istio-proxy の問題としてアラートを通知する．

Istio-proxy のログとアプリケーションのログの確認

Istio-proxy のログにおいて，upstream という文字列が含まれており，HTTP ステータスコードが 200 以外であるログが確認された場合，アプリケーションコンテナのログを照合することで異常の原因がアプリケーションにあるか確認する．Istio-proxy のログとアプリケーションコンテナのログの対応を確認する方法について図 5に示す．

upstream という文字列が含まれており，かつ HTTP ステータスコードが 200 以外であるログが確認された Istio-proxy と同一 pod 内のアプリケーションコンテナのログを確認対象とする．これは，Istio-proxy はアプリケーションのコンテナと同一の pod 内に配置され，通信を仲介しているためである．Istio-proxy のログにおいて異常が記録された時刻から 1 分前までの範囲を対象とし，アプリケーションコンテナのログを確認する．アプリケーションコンテナのログに HTTP ステータスコード 200 以外が記録されている，もしくはログメッセージに error という文字列が含まれる場合，そのログは Istio-proxy のログにおいて，HTTP ステータスコードが 200 以外であったログと同一の原因にもとづくものであると判断する．これは，Istio-proxy がアプリケーションコンテナから返されたレスポンスを中継しその内容をログとして記録している構造上，その 2 つのログのステータスコードに不一致が生じることは基本的にないためである．

通知するアラート

アラート通知の対象は，図 4に示すように，アプリケーションに問題がある場合である．提案手法によって特定さ

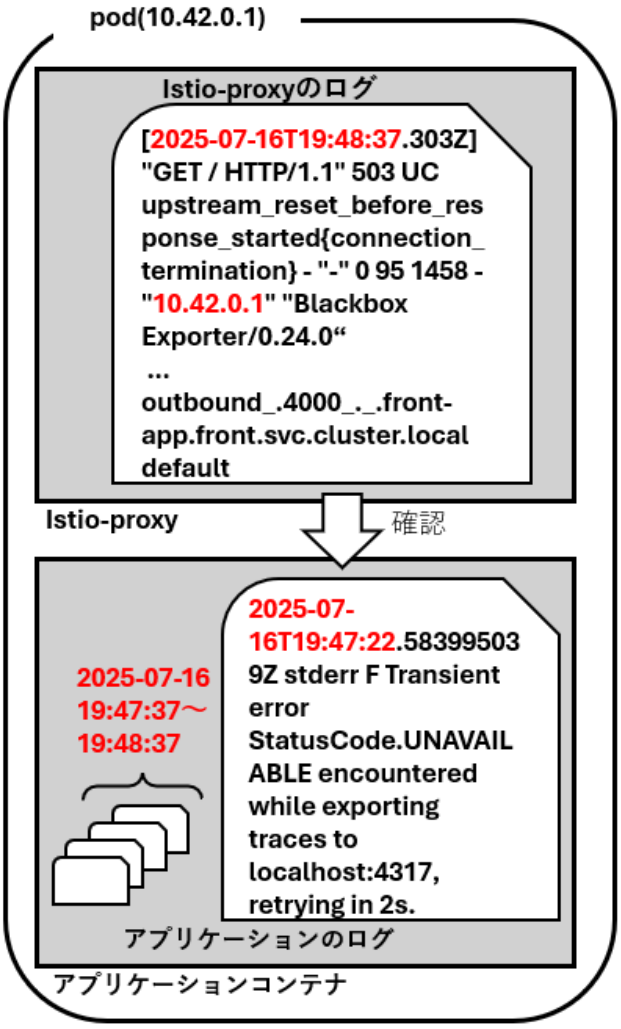

図 5: Istio-proxy のログとアプリケーションコンテナのログの対応を確認する方法

れた異常の原因箇所，原因の特定に使用されたログを記載したアラートを通知する．

ユースケース・シナリオ

本稿では，CDSL で運用し外部向けに公開されているサイトである doktor をユースケースとする．doktor は Kubernetes クラスタ上に構築されており，利用者が Web ブラウザからフロントページへアクセスする構成である．本提案では，監視システムが取得したメトリクスをもとに異常を検知し，提案ソフトウェアがログを取得し，アラートルールを設定する．メトリクスが，提案ソフトウェアに実際のシステムの異常と判断された場合，システムの運用を担当する学生にアラートが通知される．学生はその内容を確認して調査・対応を行う．図 6に提案ソフトウェアを使用した本稿のユースケースシナリオを示す．

doktor の利用者がフロントページを閲覧しようとしたが, アクセスできなかった．同時刻，doktor を監視しているシステムは，HTTP ステータスコードが 200 以外であ

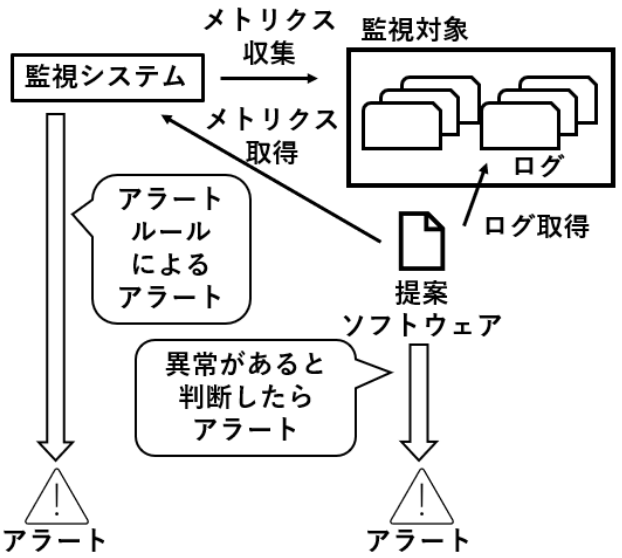

図 6: ユースケースシナリオ

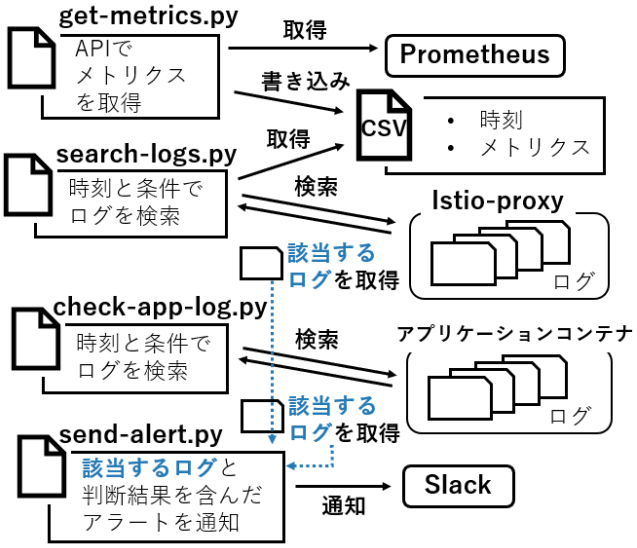

図 7: 実装における処理の流れ

るメトリクスを取得した．提案ソフトウェアはこの異常なメトリクスを確認した時刻を起点として対応するログを確認，取得する．また，提案ソフトウェアは取得したログをもとに異常の原因箇所を判断し，ログを記載したアラートを通知する．通知されたアラートをもとに学生が調査と対応を行うことで，利用者が doktor を閲覧することができるようになる．提案適用前は, 異常が発生してもアラートルールの条件を満たさない場合，学生にアラートが通知されない状態であった．本提案によりアラート通知が行われるようになる．

4. 実装

本稿では，監視システムで取得されたメトリクスと Istio-proxy のログ，およびアプリケーションコンテナのログを照合し，原因を判断したうえでアラートを通知する処理について実装を行った．本実装では，4 つの Python スクリプトを作成した．処理の流れを図 7に示す．

1 つ目のスクリプトである get-metrics.py は Prometheus の API をもちいて，doktor の HTTP ステータスコードのメトリクスを doktor を構築している 4 つのノードから 15 秒間隔で取得する．取得したメトリクスは，取得した時刻とともに CSV 形式で保存する．2 つ目のスクリプトである search-logs.py は，get-metrics.py によって作成された CSV ファイルから，doktor の HTTP ステータスコードについてのメトリクスが 200 以外である時刻を取得する．その後，取得した時刻と提案手法の条件を用いて Istio-proxy のログに対して検索を行う．条件に該当するログを取得する．3 つ目のスクリプトである check-app-log.py は，search-logs.py によって出力された Istio-proxy のログにおける記録時間をもとに pod 内のアプリケーションコンテナのログを検索する．Istio-proxy のログが記録された時間の直前 1 分間を対象にアプリケーションコンテナのログを確認し，HTTP ステータスコードが 200 以外であるログ，または error という文字列を含むログを検出した場合，該当するログを取得する．4 つ目のスクリプトである send-alert.py は，search-logs.py および check-app-log.py によって取得されたログを入力として，異常の有無を判断し，該当するログおよびその判断結果を含むアラートを Slack の Incoming Webhook をもちいて通知する．

5. 評価実験

評価実験として，監視システムで HTTP ステータスコードにおいて 200 以外のメトリクスが取得された時刻に対応する Istio-proxy のログとアプリケーションコンテナのログが，本提案の条件によって抽出されるかを確認する．実験で使用している Prometheus は，HTTP ステータスコードが 200 以外である状態が 180 秒間継続した場合に異常と判断してアラートを通知するようなアラートルールを設定している．評価実験において，ログを確認する対象の時間として以下の条件を定めている．

- HTTP ステータスコードが 200 以外であるメトリクスが取得された時刻の，直前 15 秒間の Istio-proxy のログ
- Istio-proxy のログが得られた時刻の，直前 1 分間のアプリケーションコンテナのログ

本提案におけるログ抽出の条件の妥当性を評価するために，世界標準時での 2025 年 7 月 16 日 15 時 0 分 0 秒から 2025 年 7 月 17 日 14 時 59 分 59 秒の 24 時間において，監視システムで HTTP ステータスコードにおいて 200 以外となった 4 回のケースを対象として提案手法を適用した．対象としたメトリクスの異常が発生した時刻を以下に示

す．また，いずれも HTTP ステータスコード 503 を示していた．
(1) 7 月 16 日 19 時 49 分 0 秒
(2) 7 月 17 日 5 時 52 分 30 秒
(3) 7 月 17 日 8 時 0 分 0 秒
(4) 7 月 17 日 8 時 56 分 30 秒

実験環境

doktor と監視システムの環境を図 8に示す．監視対象として CDSL で運用し外部向けに公開されているサイトである doktor を使用する．

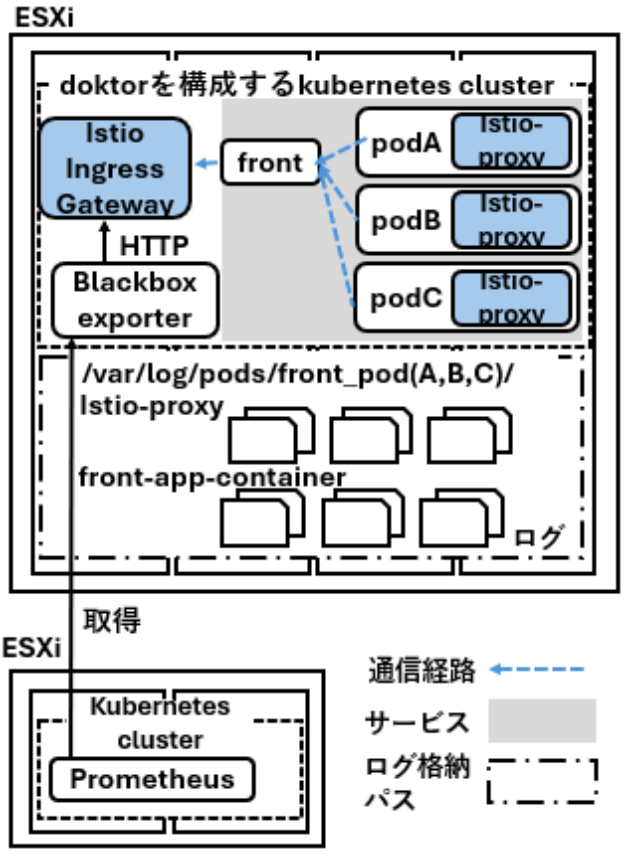

図 8: doktor と監視システムの環境

doktor が配置されているクラスタは 4 つの仮想マシン上に構成されており，全ての仮想マシンに Ubuntu22.04 がインストールされている．各仮想マシンは vCPU 8core，RAM 8GB，SSD 40GB であり，さらに分散ストレージである Rook-Ceph 用として追加で SSD が 40GB ずつ割り当てられている．doktor が配置されているクラスタは K3s による kubernetes クラスタであり，マスター 1 台，ワーカー 3 台の構成である．アプリケーションとして，front サービスが配置されており，3 つの fornt の pod により構成されている．各 pod にはサイドカーコンテナとして Istio-proxy が配置されており，Istio Ingress Gateway を通じて外部からのアクセスを受け付ける構成である．doktor を監視する Prometheus は Kubernetes クラスタ上に構成されており，各仮想マシンには Ubuntu24.04 がインストールされている．仮想マシンは vCPU 4core，RAM 8GB，SSD 40GB である．Prometheus が配置されているクラスタも K3s による Kubernetes クラスタであり，マスター 1 台，ワーカー 1 台の構成である．Prometheus および Blackbox exporter をもちいて，doktor に対して HTTP リクエストを送信し，返された HTTP ステータスコードをもとに監視を行っている．監視対象のログは各 pod の Istio-proxy のログとアプリケーションコンテナのログをもちいる．

実験結果と分析

実験結果を，Istio-proxy のログの確認とアプリケーションコンテナのログの確認の 2 つに分けて記述する．まず，Istio-proxy のログの確認について評価を行う．実験の対象とした 4 つのケースについて，Istio-proxy のログにメトリクスの異常の原因と判断できるログが含まれていることを確認した．これらのケースでは，いずれも同様のログメッセージが記録されていた．例として 2025 年 7 月 16 日 19 時 49 分 0 秒のケースにおいて確認されたログをログ 1に示す．この時記録されていた upstream reset before response started は，リクエストに対するレスポンスが返される前に接続がリセットされたことを表している．

ログ 1: 確認された Istio-proxy のログ

```
[2025-07-16T19:48:37.303Z] "GET␣/␣HTTP/1.1"
503 UC
upstream_reset_before_response_started{
connection_termination} - "-" 0 95 1458 - "
10.42.0.1" "Blackbox␣Exporter/0.24.0" "
fc594902-0098-41e5-890b-05a5c6f84fd9" "
192.168.201.8" "10.42.0.101:8000" inbound
|8000|| 127.0.0.6:47499 10.42.0.101:8000
10.42.0.1:0 outbound_.4000_._.front-app.
front.svc.cluster.local default
```

また，HTTP ステータスコード 503 が記録された時刻と，その原因と考えられるログが生成された時刻，2 つの時刻の差を表 2に示す．本提案では，Prometheus が Blackbox exporter にスクレイプする時間の間隔が 15 秒であるため，HTTP ステータスコードが 200 以外であるメトリクスが取得された時刻の直前 15 秒間の Istio-proxy のログを対象にしているが，実験の結果ログは 15 秒以上前に出力されていた．

次に Istio-proxy のログからそれと対応するアプリケーションコンテナのログを確認できるかを評価する．本提案の第 1 段階で抽出された Istio-proxy のログの時刻を基準として，その直前の 1 分間に出力されたアプリケーションコンテナのログを対象とし，HTTP ステータスコードが 200

表 2: HTTP リクエストの
確認ができるまでの
記録時刻と Istio-proxy ログの出力時刻の時刻差

日付	HTTP リクエストの確認ができるまでの記録時刻	Istio-proxy のログの時刻	時刻差 (秒)
2025/7/16	19:49:00	19:48:39	21
2025/7/17	5:52:30	5:52:08	22
2025/7/17	8:00:00	7:59:40	20
2025/7/17	8:56:30	8:56:08	22

以外であるログ，または error という文字列を含むログが出力されているかを確認した．表 3に，Istio-proxy のログの記録時刻とそれに対応して確認されたアプリケーションコンテナのログの記録時刻，および時刻差を示す．2025 年 7 月 16 日 19 時 49 分 0 秒のケースにおいては，対応するアプリケーションコンテナのログが存在していない．確認対象の時間を Istio-proxy のログの時刻の直前 3 分に変更し再度確認を行ったが，該当するログは得られなかった．この結果については，提案手法にしたがって，Istio-proxy からアプリケーションコンテナ間における通信あるいは処理上の問題が発生したと判断した．

表 3: Istio-proxy とアプリケーションコンテナのログの時刻差

日付	Istio-proxy のログの時刻	アプリケーションコンテナのログの時刻	時刻差 (秒)
2025/7/16	19:48:39	記録なし	N/A
2025/7/17	5:52:08	5:51:22	46
2025/7/17	7:59:40	7:58:46	54
2025/7/17	8:56:08	8:55:35	33

対象となったケースにおけるアプリケーションコンテナのログのうち，HTTP ステータスコードが 200 以外であるログおよび error という文字列を含むログの件数を表 4に示す．ログの確認対象とした時間は，抽出された Istio-proxy のログの時の直前の 1 分間である．

表 4: アプリケーションコンテナのログにおいて HTTP ステータスコードが 200 以外であるログと error を含むログの件数

日付	確認した時刻	200 以外のログの件数	error ログの件数
2025/07/16	19:47:39〜19:48:39	0	0
2025/07/17	5:51:08〜5:52:08	0	5
2025/07/17	7:58:40〜7:59:40	0	2
2025/07/17	8:55:08〜8:56:08	1	6

2025 年 7 月 17 日 5 時 52 分 30 秒，2025 年 7 月 17 日 8 時 00 分 00 秒，2025 年 7 月 17 日 8 時 56 分 30 秒のケースにおいて，それぞれ error という文字列を含むログは 5 件，2 件，6 件確認された．これらの 3 つのケースにおいて確認された error を含むログは計 13 件であり，いずれも同様のログメッセージが記録されていた．例として 7 月 17 日 5 時 52 分 30 秒のケースにおいて確認されたログをログ 2に示す．これは，トレースデータの送信処理において，一時的に宛先サーバである localhost:4317 に接続できなかったことを示しており，アプリケーション内でのエラー発生を示している．なお，retrying in で記録されている時間は一定ではなく，リトライ処理にかかっている時間の変化によって変化する．

ログ 2: error という文字列を含むアプリケーションコンテナのログ

```
2025-07-17T05:51:22.889601661Z stderr F
Transient error StatusCode.UNAVAILABLE
encountered while exporting traces to
localhost:4317, retrying in 32s.
```

HTTP ステータスコードが 200 以外のログは，7 月 17 日 8 時 56 分 30 秒のケースにおいて 1 件確認された．確認されたログの内容をログ 3に示す．これは，クライアントからの HEAD メソッドによるリクエストに対して，アプリケーションがそのメソッドを許可していなかったことによるものである．本稿の提案においては，アプリケーションの問題と判断されるが，このように，HTTP ステータスコードが 200 以外である場合にアプリケーションの異常とは限らないケースが含まれていた．

ログ 3: HTTP ステータスコード 200 以外を含むアプリケーションコンテナのログ

```
2025-07-17T08:56:04.284839959Z stdout F INFO
: 127.0.0.6:34385 - "HEAD␣/␣HTTP/1.1" 405
Method Not Allowed
```

以上の結果より，Istio-proxy のログの確認については，本提案におけるログ確認の対象時間の条件を 15 秒から延長する必要があると考えられる．アプリケーションコンテナのログに関しては，4 つのケースのうちの 3 つのケースにおいてログが確認できた．しかし，アプリケーションコンテナのログを異常と判断する条件において，HTTP ステータスコードおよびログメッセージのみにもとづく本稿の判断基準について検討が必要であることが示された．

6. 議論

評価実験は，図 4に示す提案手法のうち，Istio-proxy のログに upstream という文字列を含むケースに対応するフローに限定して実施している．このため，Istio-proxy 自体に問題がある場合や，pod,node に起因する異常に対応する他のフローについては追加実験による評価が必要である．

評価実験において，Istio-proxy のログに異常が記録されていた一方で, それに対応するアプリケーションコンテナのログは確認できないケースもあった．これは，現在の監視環境においてはアプリケーションコンテナのログの出力まで確認することができるとは限らないことを示している．本提案では，Istio-proxy のログとアプリケーションコンテナのログを確認することでリクエストが正常に到達したかを判断できるケースもある．さらにアプリケーション側に Istio-proxy からのリクエスト受信時にアクセス記録としてログを出力するように設定を追加することで，実際にリクエストが到達していたかどうかを明示的に判断ができる．

評価実験について，Istio-proxy のログに upstream reset before response started connection terminationt というログメッセージを含む 503 エラーのログが記録されていたにもかかわらず, Prometheus の監視結果には反映されていない場合がある．これは，Prometheus のメトリクスの取得間隔が 15 秒に設定されていることが原因であると考えられる．Prometheus は一定間隔で監視対象のメトリクスをスクレイピングしているため，監視対象で異常が発生していてもその異常が Prometheus のメトリクスの取得タイミングと重ならなければメトリクスとして観測できない．したがって，本提案の手法においても一部の異常を見逃している場合がある．Istio-proxy のログに異常が記録されているものの，Prometheus の監視結果に反映されないという現象は，監視メトリクスだけに依存した監視の限界を示している．Prometheus の設定を変更することで最小 1 秒間隔でのスクレイピングもできるが，スクレイピングの頻度を上げることは監視対象システムの負荷増加に繋がる [12]. したがって，異常の検知精度や原因特定の精度を高めるためには本提案のようにメトリクスをトリガーとする方式に加え，Istio-proxy のエラーログそのものをトリガーとして用いる方式も有効であると考えられる．

提案について，HTTP ステータスコードに異常が検出された際に Istio-proxy とアプリケーションコンテナのログを段階的に確認することで，異常の原因特定とアラート通知の判断を行っている．この確認フローでは，即時にアラートを通知するため誤検知によってアラートが増加するケースが考えられる上，アラートの内容に誤りがある場合がある．特に，Istio-proxy のログにおいて HTTP ステータスコードが 200 以外であってもそれがアプリケーションの異常によるものか，通信経路上の一時的な問題によるものかは，現状の監視体制では原因の詳細な切り分けはできない．実際に評価実験では，Istio-proxy のログに 503 が記録されていてもアプリケーションログに対応するエラーが記録されていないケースが存在した．この場合，本稿の提案手法では実際の異常の原因とは異なる原因を記載したアラートの通知が増加する. したがって，アラートの通知に関しては即時通知かつ単一の判断基準によるアラートではなく，検出された異常の性質や，その正確性，頻度に応じたアラート通知の仕組みが求められる．例えば，原因が確実に特定できない場合はアラートとして通知するのではなく，保留状態として記録し，後に出力されたログやメトリクスを照合して判断を行う方式がある．これにより，原因が明確に特定できていない状態ではアラートとして通知せず，異常の原因が確実になったタイミングでアラート通知を行うことができるようになり，誤通知や過検知の抑制に繋がると考えられる．

7. おわりに

本稿の課題は，監視対象の異常がアラートとして通知されず，それに対する調査や対応ができない事である．提案は，監視システムにおいて監視対象のアプリケーションの HTTP ステータスコードが 200 以外である場合に，その原因箇所を監視対象のログを確認することで特定し，異常があった場合にアラートを通知する手法である．ログの確認は，Istio-proxy，アプリケーションコンテナ，ノードの順に行う．提案手法によるアラート通知は，監視システムに設定されたアラートルールとは独立して行われる．2025 年 7 月 16 日 15 時 0 分 0 秒から 2025 年 7 月 17 日 14 時 59 分 59 秒における 24 時間に発生した，監視メトリクス上の HTTP ステータスコードが 503 である 4 件の異常を対象とした評価実験の結果，3 件についてはログに基づいて原因箇所を特定することができた．

参考文献

[1] Burton, S., McDermid, J., Garnett, P. and Weaver, R.: Safer Complex Systems : An Initial Framework (2021).

[2] Rivera, C. and Martinez, A.: Enhancing Reliability Through Effective System Monitoring, *Science and Technology*, Vol. 8 (2024).

[3] Vinnakota, K. and Kolla, M.: Creating Effective Alerts for Monitoring Distributed Systems, *International Journal of Computer Trends and Technology (IJCTT)*, Vol. 73, No. 5, pp. 172–178 (online), DOI: 10.14445/22312803/IJCTT-V73I5P122 (2025).

[4] Gadala, M., Strigini, L. and Ayton, P.: Improving Human Decisions by Adjusting the Alerting Thresholds for Computer Alerting Tools According to User and Task Characteristics (2021).

[5] Jani, Y.: Unified Monitoring for Microservices: Implementing Prometheus and Grafana for Scalable Solutions, `https://www.theseus.fi/handle/10024/512860` (2022). Bachelor's thesis.

[6] Ghafouri, A., Abbas, W., Laszka, A., Vorobeychik, Y. and Koutsoukos, X.: Optimal Thresholds for Anomaly-Based Intrusion Detection in Dynamical Environments (2017).

[7] Zohrevand, Z. and Glässer, U.: Should I Raise The Red Flag? A comprehensive survey of anomaly scoring methods toward mitigating false alarms (2020).

[8] Correia, L., Goos, J.-C., Klein, P., Bäck, T. and Kononova, A. V.: Online model-based anomaly detection in multivariate time series: Taxon-

omy, survey, research challenges and future directions, *Engineering Applications of Artificial Intelligence*, Vol. 138, p. 109323 (online), DOI: https://doi.org/10.1016/j.engappai.2024.109323 (2024).

[9] Venkateswaran, P., Malapati, A., Natu, M. and Sadaphal, V.: Towards next-generation alert management of data centers, *2016 8th International Conference on Communication Systems and Networks (COMSNETS)*, pp. 1–2 (online), DOI: 10.1109/COMSNETS.2016.7440016 (2016).

[10] Perdices, D., García-Dorado, J. L., Ramos, J., De Pool, R. and Aracil, J.: Towards the Automatic and Schedule-Aware Alerting of Internetwork Time Series, *IEEE Access*, Vol. 9, pp. 61346–61358 (online), DOI: 10.1109/ACCESS.2021.3073598 (2021).

[11] Cinque, M., Corte, R. D. and Pecchia, A.: Entropy-Based Security Analytics: Measurements from a Critical Information System, *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 379–390 (online), DOI: 10.1109/DSN.2017.39 (2017).

[12] Andreolini, M., Colajanni, M., Pietri, M. and Tosi, S.: Adaptive, scalable and reliable monitoring of big data on clouds, *Journal of Parallel and Distributed Computing*, Vol. 79-80, pp. 67–79 (online), DOI: https://doi.org/10.1016/j.jpdc.2014.08.007 (2015). Special Issue on Scalable Systems for Big Data Management and Analytics.

