ログ取集・管理をメッセージング経由で適切に設定する手法の提案

木山 瑞基[1,a)]　河野 真治[2,b)]

概要：サービスを保守運用していく中でシステム障害は発生するものである. その障害がソフトウェアや機器の問題であったり外部からの攻撃などが原因である場合など挙げられ, その両方でサービスの安定運用を考えていく上でシステムの不調を早期に発見することが必要となってくる. また, 近年のシステムの大規模化・複雑化もあり, ログの設定は多種多様で都度適切な設定をするのは難しい. そこで, システムの監視とチャットツールを用いた対話型のアラート管理を提案する. システム監視はサーバー上で動作しているシステムの死活・リソース監視とログ収集のことを差す. 本稿ではシステム監視の構築とチャットツールを用いたアラートルール管理の実装をした.

1. システム運用におけるログ

近年のグローバル化やモバイル端末およびパソコンの普及により 24 時間 365 日, システムの稼働が求められる. だが, 物理機器の経年劣化によるトラブルやサービスのバージョンアップによるシステムトラブル, 外部からの攻撃などによるインシデントなどにより安定稼働の実現は難しい. ここでサービスのログは重大な役割を担う. システムの障害やユーザーの行動, 管理者の作業内容はログとして書き出され, 問題解決の重要な手がかりとなりえる. その為システム管理や障害対応を行う場合はログファイルを参照し正常動作の確認やトラブルシューティングを行う. しかしシステムの大規模化, 複雑化に伴いログ情報を手動で管理することは現実的ではない. また障害発生時に対象サーバーにアクセス出来ない場合にはそもそもログ自体を確認することができなくなる. このような問題にログ管理システムは有効だと考えられる

琉球大学工学部工学科知能情報コースには学生が主体となって管理するシステム (以下学科システムとする) が存在するこれらの学科システムは学内ネットワークや貸出用の仮想マシン、学内チャットツールなど研究や授業を円滑に進めるためのサービスを提供しているその為システムトラブルの予兆を検知することや外部から攻撃された時に管理者に通知を送信する機能が必要となる.

現在学科システムでは障害が発生した際にはユーザーが報告するか管理者がサーバーに入って確認するしか方法がない. さらに障害発生時に対象サーバーにアクセス出来ない場合エラーログを確認することができない. また監視システムを導入する際に死活監視・ログ収集では稼働しているサービスを対象として構築するが, アラート送信の機能は運用していく中で過不足が無いように調整が必要と考える . その為, アラートの送信を制御するアラートルールは組織全体で調整しながら運用する方針をとる . その際, 通常のアラートルール編集方法では, 作業者は編集後に作業をまとめる必要や第三者はその作業手順を探さないといけない問題があり属人化する恐れがある. そこで本稿では安定した運用のための学科システムに監視ツール及びログ収集サービスの実装の検討をする. また学科で使用しているチャットツールからアラートルールを編集する事で他者に情報が伝わり易くなると考えた . その為, 属人化を防ぐ為の mattermost のスラッシュコマンドを用いた変更方法の提案をする.

2. 技術概要

2.1 Prometheus

Prometheus は [7] オープンソースのメトリクスベースのモニタリングシステムである. 音声ファイル共有サービスを配信している SoundCloud 社によって 2012 年に開発されており,2015 年に一般に公開されている. 対象サービスから監視サーバーに対し情報を取得する pull 型を採用しており, 取得したデータは時系列データベースに保存される. 特徴としてそれぞれのデータに付与されているラベルごとに情報をまとめることが出来る. また、PromQL とい

[1] 琉球大学大学院理工学研究科工学専攻知能情報プログラム
[2] 琉球大学大学工学部工学科知能情報コース
[a)] oruta@cr.ie.u-ryukyu.ac.jp
[b)] kono@ie.u-ryukyu.ac.jp

う独自のクエリ言語を扱う事でアラート管理コンポーネントである Alertmanager にクエリを発行することができる．内蔵する式ブラウザからグラフ・ダッシュボードの作成やデータ検索ができるが作成したグラフは保存することができないことや汎用のダッシュボードシステムでは無い事から一般的には可視化ツールと組み合わせて運用される．

2.2 PromQL

Prometheus の時系列データを扱うことに特化したクエリ言語である．exporter によって付与されるラベルを用いることで柔軟な集計が可能である．グラフや Prometheus の式ブラウザを表示するのに使用したり,HTTP API を介して外部システムで利用することが出来る．

2.3 exporter

監視対象のデータを収集し Prometheus からのリクエストに応じて必要なデータを整形し Prometheus にレスポンスとして返すツールである．Prometheus に送信するデータに対しラベルを付与することができ情報の絞り込みが楽になる．サーバーの情報を収集する node_exporter や通信のエンドポイントを監視する blackbox_exporter など公式が提供しているもの以外に多くのサードパーティがサービスの exporter を提供しており, 自身で独自の exporter を作成することも可能である．

2.4 Alertmanager

Alertmanager [6] は Prometheus のコンポーネントであり, オープンソースソフトウェアとして公開されているアラート管理ツールである．アラートの重複排除、グループ化などによりアラートの送信を行うことができる．

2.5 Loki

Loki [3] は Prometheus に触発されたオープンソースのログ収集ツールである．特徴として Prometheus のようにログデータをラベル毎にまとめることができる．また,Prometheus と同様に LogQL という独自のクエリ言語を扱う事でアラート管理コンポーネントである Alertmanager にクエリを発行することができる．

2.6 Promtail

ログを収集して Loki に送信するツール．Prometheus の exporter のようにアプリケーションのログに対しラベルを付与することができる．

2.7 Grafana

収集されたデータ・ログをダッシュボートを用いてブラウザから可視化可能なツールである．自身でデータを収集をせずデータの可視化を行うためデータソースと組み合わせて使うのが一般的である．データソースとして Prometheus と loki を対応しており,LogQL を用いてカスタムしたログ情報を表示することが可能である．

2.8 Zabbix

Zabbix はオープンソースの統合監視ソフトウェアである．多数の監視機能をデフォルトの機能として提供しているほか、サーバー, ネットワーク, サービスを集中監視する為の監視機能, 障害検知機能, アラート送信機能, 可視化機能を提供している．また特徴としてエージェントを監視対象にインストールすることなく監視が可能であり, エージェントを使用することでサービスの詳細なステータスの監視も可能となる．

2.9 Mattermost

オープンソースのセルフホスティング式のチャットサービスである．類似プロダクトに Slack があり, 差別点としてはサーバーから自身で構築するためチャットに保存期間の上限が無い．また, 組織や企業などの開発者向けに作成されており、様々なツールとの結合が可能できる．

2.10 コンテナ型

仮想化技術の一つであり, 他の仮装技術との相違点はカーネルはホスト OS と共用で利用する点である．これにより他の仮想技術よりリソースが節約でき, 仮想環境の構築, 削除が高速でできる．

2.11 Docker

Docker.Inc. が開発したオープンソースのコンテナ管理ツールである．コマンドや専用の Dockerfile を用いて環境を構築することができ、また作成したイメージを登録することが出来る Docker Hub を用いることで環境を配布することもできる．

2.12 Podman

RedHat 社が開発した docker 互換のコンテナ管理ツールである．RHEL(Red Hat Enterprise Linux) の version7.6 以降からは docker のサポートが切れる事からコンテナ管理ツールとして採用された．

2.13 ハイパーバイザー型

仮想化技術の一つであり, ハードウェア上にハイパーバイザと呼ばれる仮想化ソフトウェアを動作させ, その上でゲスト OS を運用する．ホスト OS を不要とするがコンテナ型と比べて起動速度は低速となる．

2.14 KVM

KVM は (Kernel-based Virtual Machine) の略で linux カーネル上で動作する仮想化技術であり, カーネルをハイパーバイザとして機能させる.

3. 現在の学科システム

本章では現在の学科システムの問題点及び監視システムを導入する際に発生するであろう 問題点について述べる.

3.1 問題点

現在学科システムにはシステム監視、ログ収集、アラート送信などの異常検知の機能が 無く, システム障害や攻撃の早期発見・事後対応が困難である. 実際に 2021 年 8 月にはレンタルサーバーと基幹サーバーで障害が発生した. その際に 利用者からサーバー上で動作しているサービスが使用出来ないという報告を受けて障害に気づいた. また, 該当サーバーが物理故障しており, 原因調査の為ログを確認したかったがサーバーにアクセス出来なかった為調査開始に時間がかかった. また,10 月から 11 月にかけては学科で運用している Gitlab が脆弱性を突かれ攻撃された. 実際に攻撃を受ける前に総当たり攻撃が失敗していたりユーザーに対して警告メールが送信されていたが攻撃に使用されているのに気づいたのは報告を受けてからだった. 本稿では, 障害の早期発見や事後対応を円滑に進める為に監視システムの提案を行う.

3.2 監視システムを運用する上での課題

監視システムを運用していく中でアラートルールは通知される量やその精度に応じて調 整する必要があり, 障害は個人ではなく組織として対応する為, 全ての変更はグループ全 体が理解しているのが好ましい. しかし通常では一人がサーバーにアクセスして CLI 上での変更が必要となる. 上記の方法では第三者に編集内容を伝える方法としてログなどにまとめるしか無く, また第三者はログを自分で探す必要がある. その為学科で使用しているチャットサービスである mattermost からアラートルールを 変更する事で属人化を防ぐ. 上記の方法だとアラートルールを共有する形で編集できる為, 作業ログをまとめる必要や第三者がそれらのログを探す手間が省ける. これにより上述した問題点を改善することが出来, 属人化するリスクを抑えることが出来ると考える

4. 監視システムの構成

サービスでは学科システムのシステムの監視やログの収集を行う。この監視システムの構成を図に示し。概要を以下で説明する。

4.1 監視システム

本実験では全ての監視システムをコンテナ上に構築しており,docker-compose.yml ファイルから podman でコンテナを作成する形を取っている. また、各サービス間の通信は各コンテナに ip を振るのではなくコンテナ名を指定することで行っている。

サービスの死活監視は prometheus, ログ収集は loki, それらのデータ可視化は grafana, アラート送信は alertmanager を用いて構築している. システム監視の構成を 1 に示す. exporter,alertmanager は prometheus のコンポーネントとなっており,exporter で各サービスのメトリック情報を prometheus に対して送信している. また,prometheus の独自のクエリ言語である promQL で設定した条件を満たした際に alertmanager に対してアラートを送信する.

使用するクエリ言語が logQL に変更しているが loki も promethrus 同様にログを収集し, アラートを送信している.

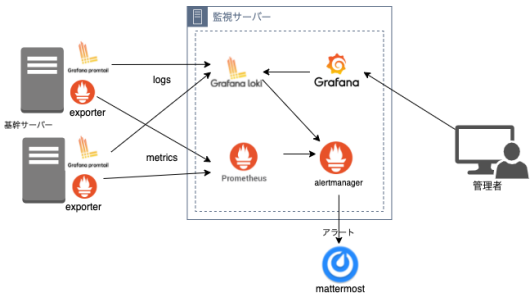

図 1: 構成図.

4.2 サービス監視

各サービスのリソース・死活監視は promethrus を用いて行っている. prometheus は収集したデータをブラウザで表示する機能を持っている. 図 2 に各 expoter サーバーから prometheus に集めた情報をブラウザを通じて表示する様子の一例を示す.

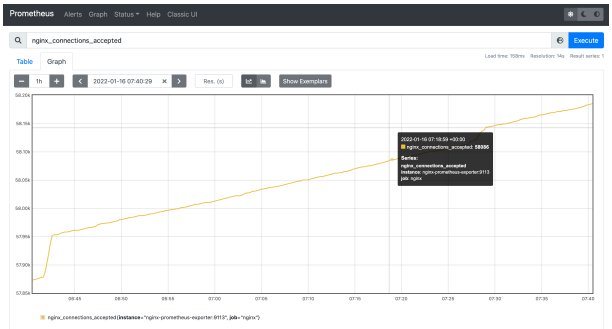

図 2: promethrus でのグラフを用いたデータの可視化.

しかし,prometheus のグラフでは表示のカスタマイズや可視化したダッシュボードの登録, ログイン認証をサポートしていない. そこでデータの可視化は grafana を用いた. また, これらのダッシュボードは自身でカスタムしたり grafana に登録されてあるダッシュボードをインポートすることで, よく使用する情報を登録することが可能である. 以下の図 3 が grafana のダッシュボードの例である. このように複数の情報を可視化することができる.

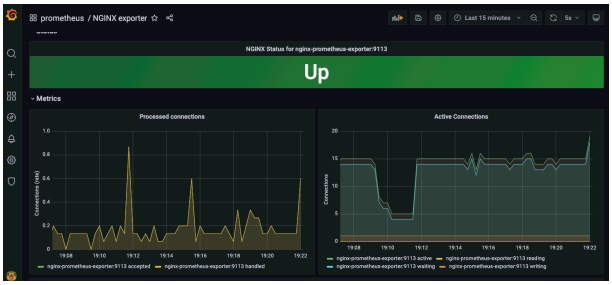

図 3: grafana でカスタムされたダッシュボード.

4.3 ログ収集

障害が発生したサーバーがアクセス可能であればログを確認することはできるが, 何らかの原因でアクセス出来ない場合はログを確認する手段がない. そこで各サーバーのログを収集する事でサーバーの死活状態に関わらず原因調査が可能である. また,grafana では logQL を用いて絞り込みが出来る為特定までにかかるコストが低くなると考える. 提案システムでは, ログ収集には loki を用いている. loki 自体に収集したログを表示する 機能は無く,grafna と連携してログの可視化を行う. また、loki と grafana でのログの可視化でも一般的には, カスタムダッシュボードを使用して可視化を行う.

4.4 アラート送信

システムのリソース状態や死活状態, ログ情報は安定してシステムを運用する上で把握しておくべき情報である. しかし, 人間が 24 時間 365 日稼働しているシステムを監視し続けるのは現実的ではない. その為指定したログが出力されたりサービスが停止した際に管理者に通知する仕組みが必要でありそれがアラート機能である. また,prometheus,loki 共に設定ファイルに alertmanager の UR を記入することで連携が可能である. またアラートルールのファイル形式について grafana の公式サイトは「Loki alerting rules are exactly the same, except they use LogQL for their expressions. 」[3] と明言しており, 使用する QL を除いてアラートルールの記述方法は変わらない事からここでは loki のアラートルールのみを紹介する. ソースコード 4.2 が loki のアラートルールファイルである. このコードの 7 行目がアラートを制御する部分であり 5 分周期でログの有無を確認しログが生成されていたら alertmanager にクエリを送信するようなコードになっている.

```
groups:
  - name: amane-fail2ban-log
    rules:
    - alert: amane-fail2ban-log
      annotations:
              message: "{{$labels.job}}␣has␣log"
      expr: |
              count_over_time(({job="amane_fail2ban_log
                  "})[5m]) >= 0
      for: 5m
```

Listing 1: loki のアラートルールファイル

5. 対話型アラート編集システムの構築

監視システム運用の際に発生するアラートルールの属人化を防ぐためのスラッシュコマンドを用いたアラートルールの編集方法を提案する.

5.1 構成

監視システムと同一のサーバーに API サーバーを立て mattermost から送信された GET,POST リクエストに対し処理を行う。図 5.1 に構成図を示す.

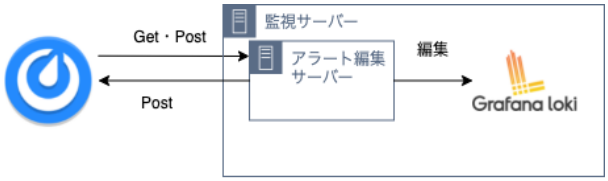

図 4: 構成図.

mattermost のスラッシュコマンドは/から始まるコマンドを打つ事で GET/POST リクエストが送ることができる. また引数には POST リクエストの Body を指定することが出来る. 提案手法ではスラッシュコマンドに作成した WEB API の URL を紐付けた.

5.2 コマンド一覧

以下で本研究で作成したコマンドの説明を行う. 想定している操作は全てアラートルールの編集に関するものでありそれぞれ第一引数によって処理を分けている.
それらの処理の一覧を以下に示す。

アラートルールを追加する場合は/alert add $alertname $label $pattern $time を入力する.

表 1: アラートルール変更時の入力内容

add $alertname $label $pattern $time	ルール追加
list	ルール一覧取得
delete	ルール削除

引数の意味はそれぞれ$alertname はアラートが送信される時のアラート名,$label はアラートに紐付いているラベル,$pattern はログに含まれていた際にカウントする文字列,$time はアラートを送信する周期を指定している.
図??に alert add コマンドを使用してアラートを追加した様子を示す.

/alert list は登録したアラートルールを表示する. 上記のコマンドは引数なしでコマンドを入力すると登録されているアラート名のみが表示される. アラートルール全体の詳細が見たい場合は list の後ろに all コマンドを入力することで確認することが出来る. また, アラートルールの詳細が見たい場合は list の後ろにアラート名を入力することで指定したアラートルールのみが表示できる.

また, アラートルールを削除したい場合は delete コマンドを打つ事によってアラートルールが削除される.

6. 今後の課題

本稿で構築した監視システムおよび対話型のアラート編集システムについての課題を挙げていく.

6.1 監視システムの運用

本稿ではコンテナによる構築を行い実際に学科のシステムを監視できたが構築場所が VM 上のみとなった. 本番環境で動作させるにはクラウドサーバーとオンプレの両方で構 築し冗長性を保つ必要がある. また収集した情報は監視サーバー上にのみある. その為定期的にディスクサーバーにデータを送信し監視サーバーでは容量が膨らまないようローテーションする必要がある.

6.2 監視システムの冗長化

一般的に監視システムを構築する際は冗長化構成を作成しどちらのサーバーがダウンし ても監視し続けられる体制を整える. しかし本研究ではオンプレミスでのみ構築している. その為クラウドにセカンダリを構築することで冗長化を図る必要がある.

6.3 必要なログ・アラートの選択

本稿で提案した手法によりアラートルールの設定を共有しやすくなった. しかし

7. まとめ

本稿ではシステムの保守・運用を円滑に行う為に学科システムに監視システムと対話型のアラート編集システムを構築した。システム管理チームは教員と学生が中心となっており, 学生は学部一年から所属しても 4 年, もしくは 6 年で卒業してしまう事から他者に共有できる形でアラートルールを編集する事でチーム全体としての理解が深まると考えた. 今後, 本監視システムを本番環境にデプロイし実際に運用する中で問題点を改善していく. またログを管理する際にどのようなログを異常と判断するか, 管理者に適切なアラートを出すシステムの構築を行う

参考文献

[1] Brazil, B.: 入門 Prometheus–インフラとアプリケーションのパフォーマンスモニタリング, O'Reilly Japan (2019).
[2] Julian, M.: 入門監視–モダンなモニタリングのためのデザインパターン, O'Reilly Japan (2019).
[3] Labs, G.: Grafana Loki.
[4] Labs, G.: Grafana: The open observability platform | Grafana Labss.
[5] Labs, G.: Promtail | Grafana Labs.
[6] Prometheus.io: Alertmanager | Prometheus.
[7] Prometheus.io: Prometheus - Monitoring system & time series database.