ISUCON 本ゆる輪読会 #1

Chapter2 モニタリング

aoshima

2022 年 8 月 14 日

今日の資料

https://github.com/aoshimash/techresi-isucon-workshop

モニタリングとは

> *Web* サービスを提供する側にとってのモニタリングは、*Web* アプリケーションやそれらを提供する基盤となる部分の状態を計測するという意味合いで使われます。

モニタリングは高速化の対象がどのように遅くなっているのかを正しく把握するために重要な作業。

モニタリングに対する考え方

重要なこと

- 変わらない視点でモニタリングすること
- モニタリングする目的を確実に定めて、チーム内で共有すること
- メトリクスをダッシュボード化しておくこと（過去のデータをみたり複数のメトリクスの動きを同時に確認するため）

モニタリングの種類

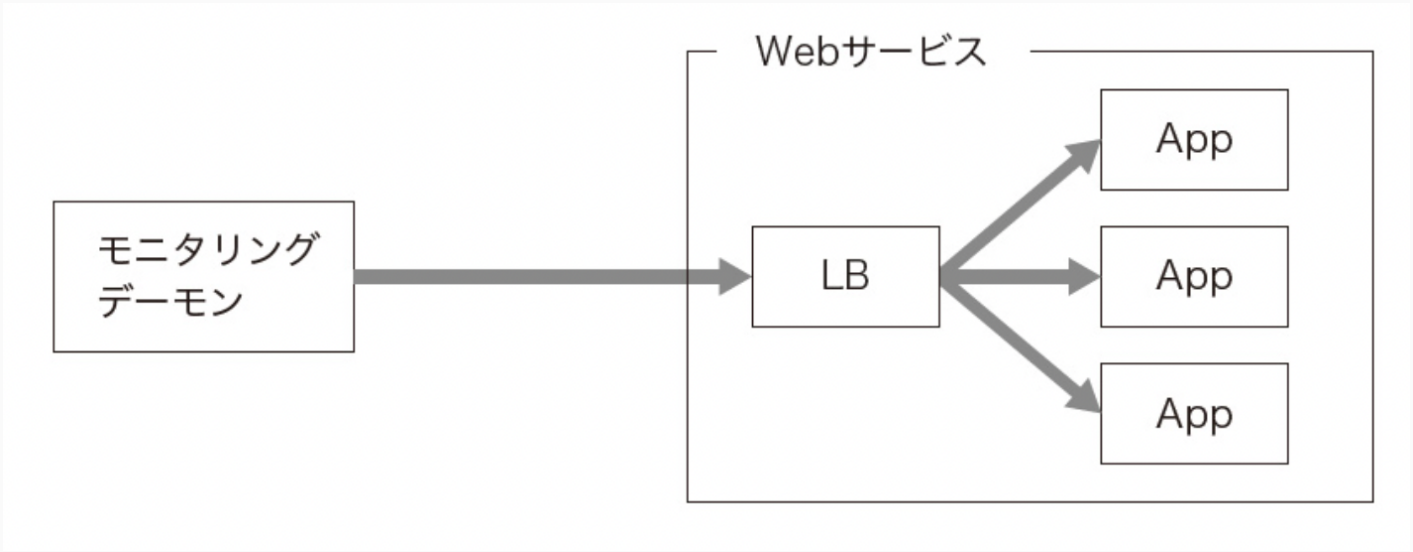
モニタリングは大きく２つに分けることができる。

外形監視
アプリケーションを外側からモニタリングする手法。サービスが正しく動作しているかを確かめることが主な目的。

内部監視
アプリケーションの内側からモニタリングする手法。ユーザーが見えない部分の状態をモニタリングし、それらが意図しない状態になっていないかを確かめることが主な目的。

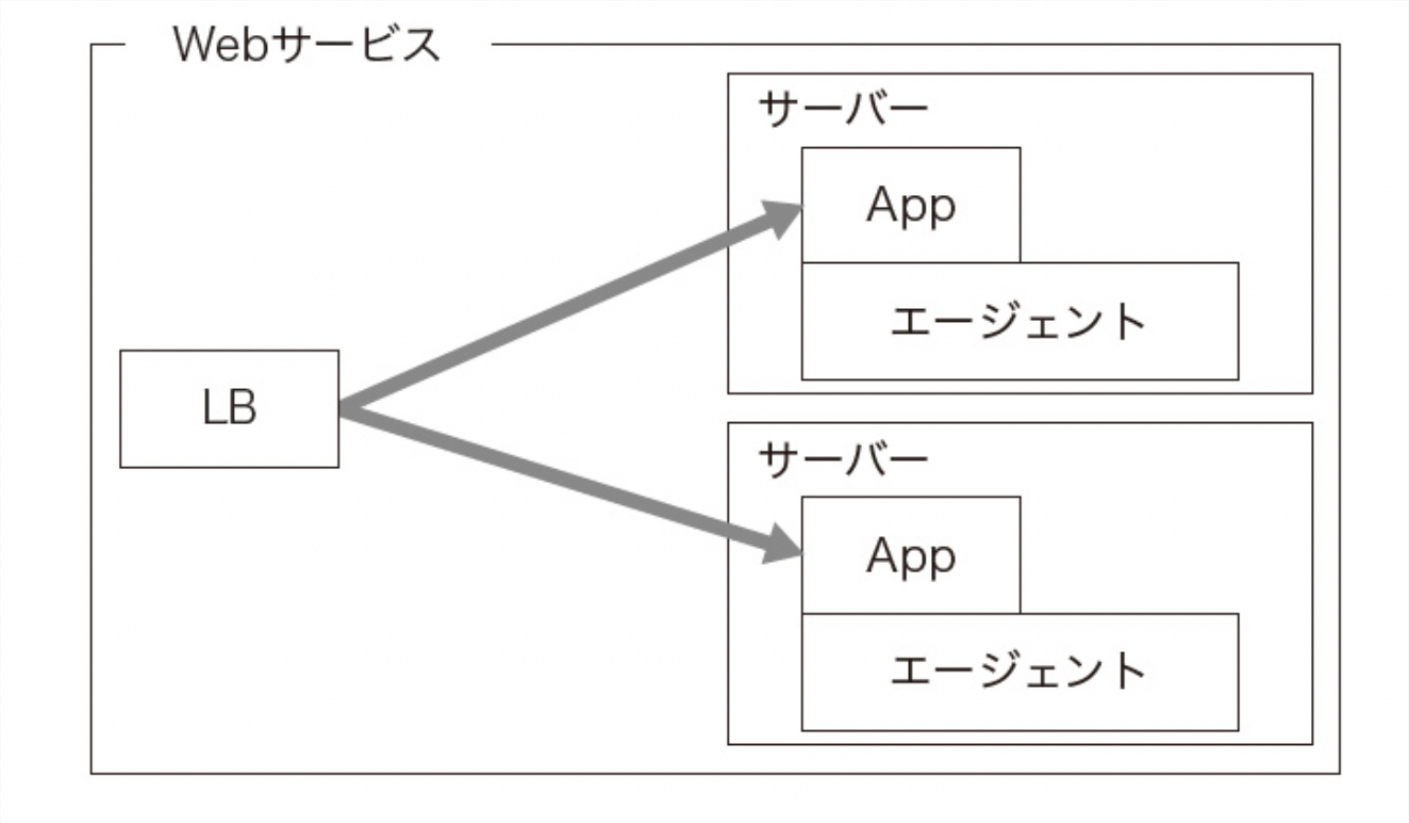
外形監視

提供している HTTP エンドポイントに対して実際に HTTP リクエストを行う。（ISUCON のベンチマークもこれ）



内部監視

Web アプリケーションや OS、ミドルウェアのメトリクスを確認する。



手動での監視

Linux コマンドで簡単な内部監視をしてみよう！

- top: CPU 使用率
- free: メモリ使用率

top 系コマンド

top とか free をそのまま使うより、こういうコマンド使うほうが便利。

- htop
- ctop
- gtop
- glances

他にも色々あるらしい。「top 系モニタリングツールまとめ」

top のヘッダ

- PID: プロセス ID
- USER: プロセスオーナー
- PRI: 優先順位
- NI: nice 値
- VIRT: 仮想メモリのサイズ
- RES: プロセスが実際に使っているメモリサイズ
- SHR: プロセスの共有メモリのサイズ
- S: プロセスの状態 (S: スリープ, R: 実行中, D: ディスクスリープ, Z: ゾンビ, T: トレースまたは一時停止, W: ページング)
- CPU%: 使用している CPU 時間のパーセンテージ
- MEM%: 使用しているメモリのパーセンテージ
- TIME+: プロセスがユーザとシステムに費やした時間

モニタリングツール

本番環境では手動でのモニタリングだけではなく、モニタリングツールと呼ばれるソフトウェアまたは SaaS を使うことが多い。モニタリングツールは次のような機能をもつ。

- メトリクスを自動で収集し、保存する
- 保存したメトリクスを Web ブラウザなどで時系列順に表示する・集計用のクエリなどで表示を切り替えられる
- メトリクスが特定の閾値に達すると通知を行う

モニタリングツールのアーキテクチャ

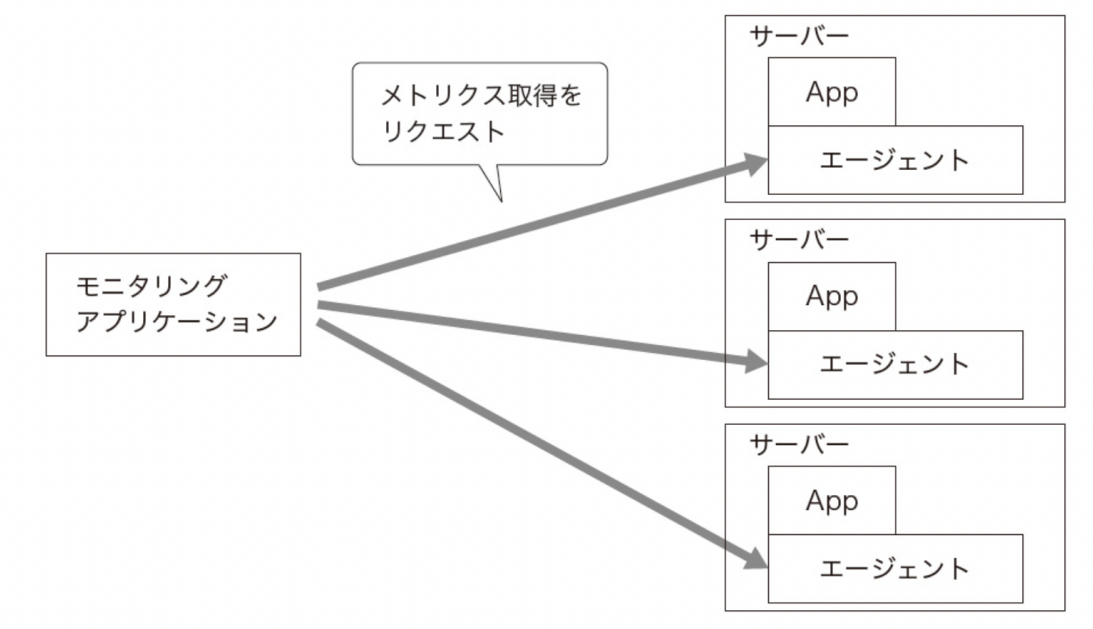
大きく分けて２つのアーキテクチャがある。

- プル型
- プッシュ型

どちらが優れているというわけではなく、特徴を理解して自分の環境に合ってるものを選べばいい。

プル型

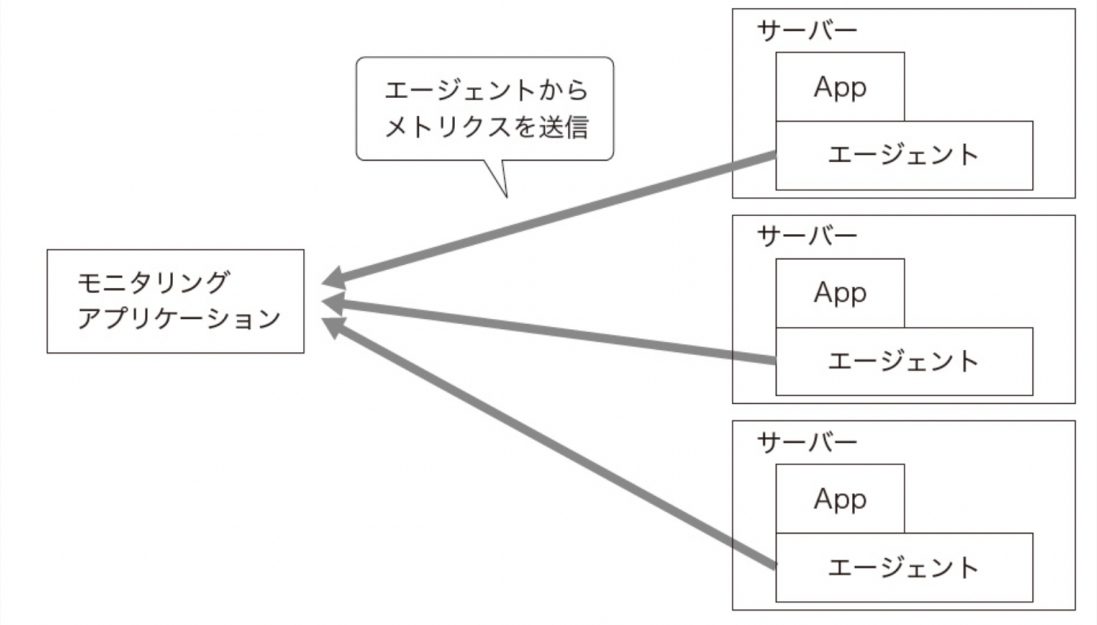
モニタリングアプリケーションからエージェントに対してメトリクス取得をリクエストするアーキテクチャ。



メリットはエージェント側の実装をシンプルにできること。（例: Prometheus, Nagios, Zabbix）

プッシュ型

エージェントからモニタリングアプリケーションにメトリクスを送信するアーキテクチャ。



メリットはモニタリングアプリケーション用のポート開放が要らないところと、モニタリング対象の増減時にモニタリングアプリケーション側の設定変更が要らないところ。(例: Datadog, Mackerl, Sensu)

Prometheus をさわってみよう

モニタリングツールの一例として Prometheus を実際にさわってみよう！
Prometheus の特徴

- プル型 (Pushgateway を使ってプッシュ型として動かすこともできる)
- ServiceDiscovery によってモニタリング対象の増減時に自動で対応できる
- セルフホストもできるし SaaS もある

node_exporter

node_exporeterh は Prometheus におけるエージェントの一つ。Linux におけるシステムメトリクスを取得することができ、Linux ホスト 1 台につき 1 つずつインストールされる。モニタリングアプリケーションである Prometheus からメトリクスを取得するためのリクエストを受け取ると、現在の状態を収集して HTTP レスポンスとして返却する。

検証環境の立ち上げ

検証環境用の docker-compose を example ディレクトリに用意したので、まずは次のコマンドで環境を立ち上げる。

```
$ docker-compose up -d
```

これで、node_exporter を含めていくつかのコンテナが立ち上がる。curl で node_exporter に GET してみる。

```
$ curl localhost:9100/metrics
```

ちなみに、この HTTP レスポンスのフォーマットは OpenMetrics という標準化されたフォーマット。

node_exporter で取得できるメトリクス

node_exporter で取得できるメトリクスはここで確認できる。
https://github.com/prometheus/node_exporter
Prometehus に保存するメトリクスが増えるほど、サーバーの負荷が増え、ディスク容量も圧迫するため、どのメトリクスを取得するのか取捨選択は必要。

Prometheus ダッシュボード

http://localhost:9090

PromQL

PromQL（Prometheus Query Language）をテキストボックスに入力してグラフを出力する。

CPU 利用時間を表示するクエリ

```
avg without(cpu) (rate(node_cpu_seconds_total{mode!="idle"}[1m]))
```

Grafana の紹介

Grafana を使うと Prometheus のデータを使ってダッシュボードを作ることができて便利！
http://localhost:3000
初期ユーザーのパスワードは admin/admin
ダッシュボードのサンプルは、”Configuration > Data sources > Prometheus > Dashboards” から「Promethus 2.0 Stats」を import するとみることができる。

モニタリングの注意点

プロファイラとかちゃんと使うと良さそう。Go だと pprof とかかな？

ログに対するモニタリング

パフォーマンスチューニングの文脈では、

- アクセスログに記録されるリクエストごとのレイテンシ
- エラーログ

などがとくに大事