new relic

監視とは違う オブザーバビリティと SREの概要と潮流

New Relic 株式会社
Solutions Consultant
板谷 郷司

Safe Harbor

This presentation and the information herein (including any information that may be incorporated by reference) is provided for informational purposes only and should not be construed as an offer, commitment, promise or obligation on behalf of New Relic, Inc. (“New Relic”) to sell securities or deliver any product, material, code, functionality, or other feature. Any information provided hereby is proprietary to New Relic and may not be replicated or disclosed without New Relic’s express written permission.

new relic.

Solutions Consultant

Satoshi Itatani

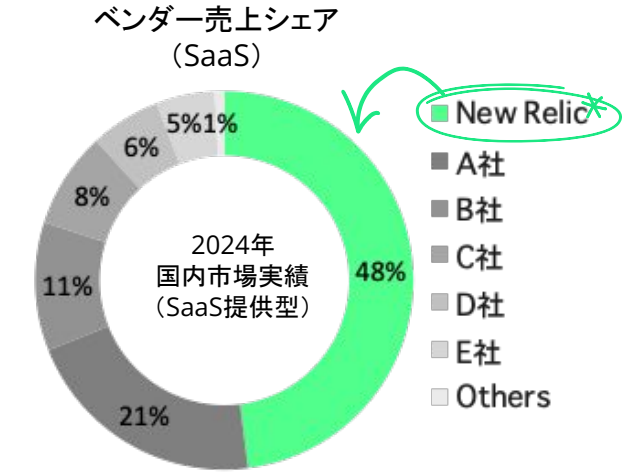
長年インフラエンジニアとして従事。
オンプレミスおよびクラウドでの大規模 Webインフラの
設計、構築、運用を専門とする。
物理、ネットワークからアプリまで
幅広い開発構築経験もあり、バックエンド全体の知識を有する。

最近のブームはWebコントロール型IoT鉄道模型

New Relicの概要

米国本社	New Relic, Inc.
事業内容	オブザーバビリティ・プラットフォームの提供
設立	2008年
CEO	アシャン・ウィリー(Ashan Willy)
従業員数	約2,400人
拠点数	17拠点

日本法人	New Relic 株式会社
設立	2018年8月



出典：株式会社テクノ・システム・リサーチ　(2025年10月)

はじめに

今日のゴール

- 監視とオブザーバビリティとの考え方の違いを理解する
- SREの概要について理解する
- インフラ上に乗っているデジタルサービス及びその開発について最近の考え方を理解する

Agenda

01	DevOpsが変えた監視の役割
02	運用と監視
03	オブザーバビリティとは
04	SREというプラクティス
05	オブザーバビリティとSRE
06	まとめ

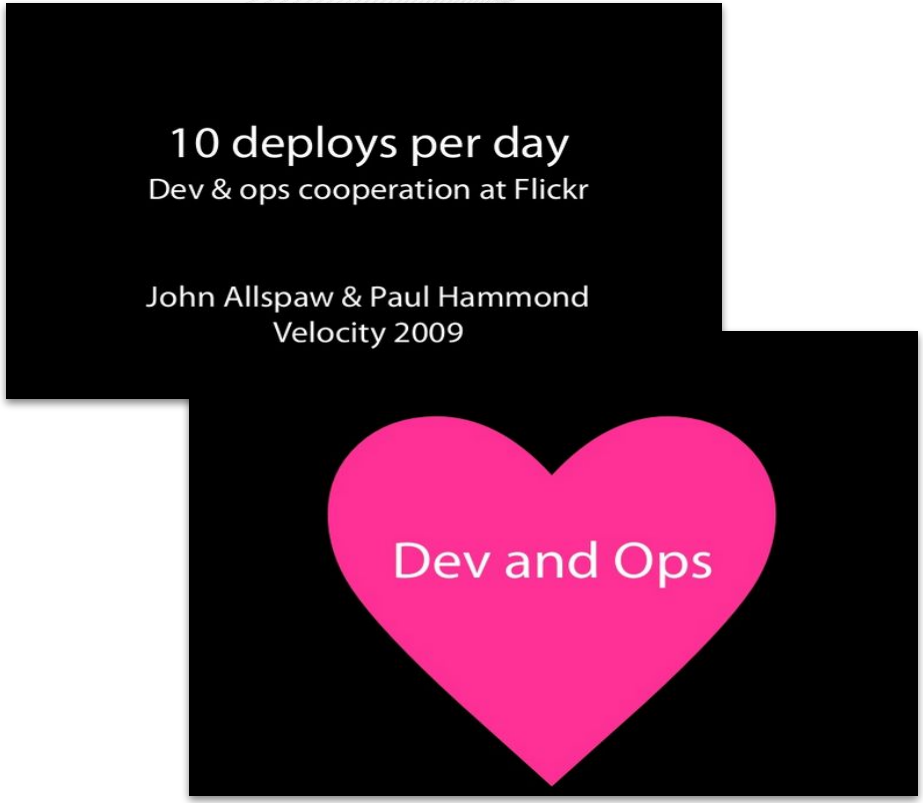
DevOpsが変えた監視の役割

new relic

DevOpsが変えた監視の役割

DevOpsてなに？

- きっかけは2009年Velocityカンファレンスでの Flickrの登壇資料
- DevとOpsで協力して1日10回デプロイできるようにしたという話



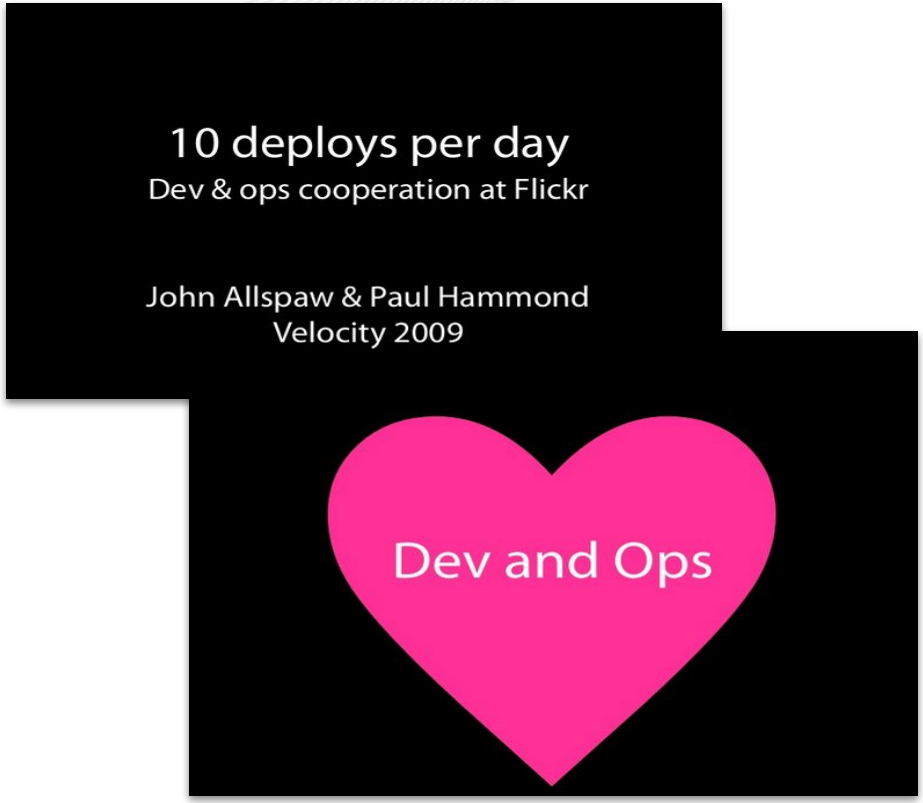
https://www.slideshare.net/jallspaw/10-deploys-per-day-dev-and-ops-cooperation-at-flickr

new relic

DevOpsが変えた監視の役割

DevOpsてなに？

- きっかけは2009年Velocityカンファレンスでの Flickrの登壇資料
- DevとOpsで協力して1日10回デプロイできるようにしたという話
  - インフラの自動化
  - 共有バージョン管理
  - One step build and deploy
  - 監視メトリクスの共有
  - etc
- DevとOpsが仲良くなるのが目的ではなくて、Opsの自動化やOpsの仕事をDevが巻き取って、**開発速度を向上** するのが目的



https://www.slideshare.net/jallspaw/10-deploys-per-day-dev-and-ops-cooperation-at-flickr

new relic

DevOpsが変えた監視の役割

そもそも何をしたかったんだっけ？

new relic

DevOpsが変えた監視の役割

そもそも何をしたかったんだっけ？

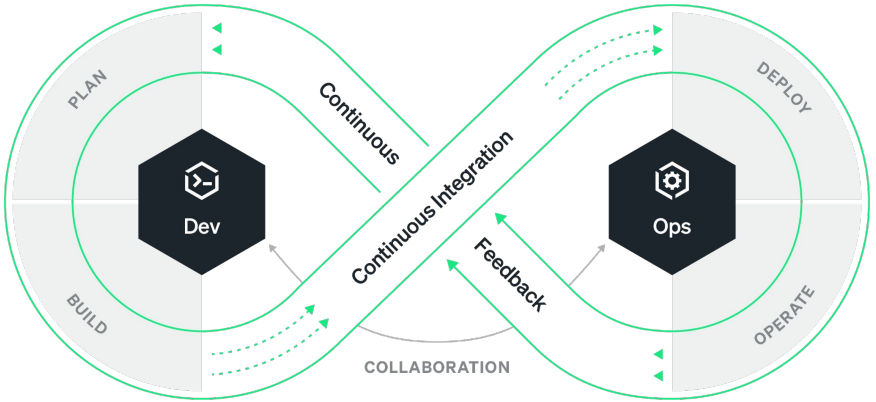
- ~~働かなくても、一生、毎月1000万振り込まれ続けたい~~
- 良いサービスを提供し社会に貢献し、5000兆円欲しい

new relic

DevOpsが変えた監視の役割

そもそも何をしたかったんだっけ？

- それぞれが5000兆円のためにやれるがあるはず
  - キャンペーン
  - パフォーマンス改善
  - 運用コスト最適化
- 正解のない時代、様々な施策を打つために、トライ&エラーを高速で回す必要がある
- 素早くリリースし、素早くシステムの状態やサービスの利用状況をチェックする必要性が高まっている



new relic

DevOpsが変えた監視の役割

現在の監視に求められているもの

- システムの状態だけはなく、エンドユーザやビジネスの状態も監視
  - PVやMAUといったビジネス上のKPIは、エンジニアも把握した方が良い
  - 共有する情報が多いとシンドイので、分かりやすい数字から
    - ビジネス→開発へは「その事業のドライバーとなっている数字」
    - 開発→ビジネスへは「そのKPIと相関がありそうな監視のデータ」
  - それらを共有できる仕組みを整える
  - 無理やり1つのツールやダッシュボードに収めようとしない

new relic

運用と監視

new relic

運用とは

- 働かせ用いること。物をうまく使うこと（引用：コトバンク）
- システム運用とは、企業などで情報システムが本来の能力を発揮し続けられるよう監視や管理、保守などを継続的に行うこと。（引用：e-Words）

new relic

運用とは

運用の重要性

- 月間の売上が300億のECサイトが有る場合
  - システム障害で24時間停止した場合の機会損失はいくらでしょうか？

運用とは

運用の重要性

- 月間の売上が300億のECサイトが有る場合
  - システム障害で24時間停止した場合の機会損失はいくらでしょうか？
  - 発生した障害や異常を早く、的確に検知する仕組み

運用とは

運用の重要性

- 月間の売上が300億のECサイトが有る場合
  - システム障害で24時間停止した場合の機会損失はいくらでしょうか？
  - 発生した障害や異常を早く、的確に検知する仕組み

監視

https://www.oreilly.co.jp/books/9784873118642/

new relic

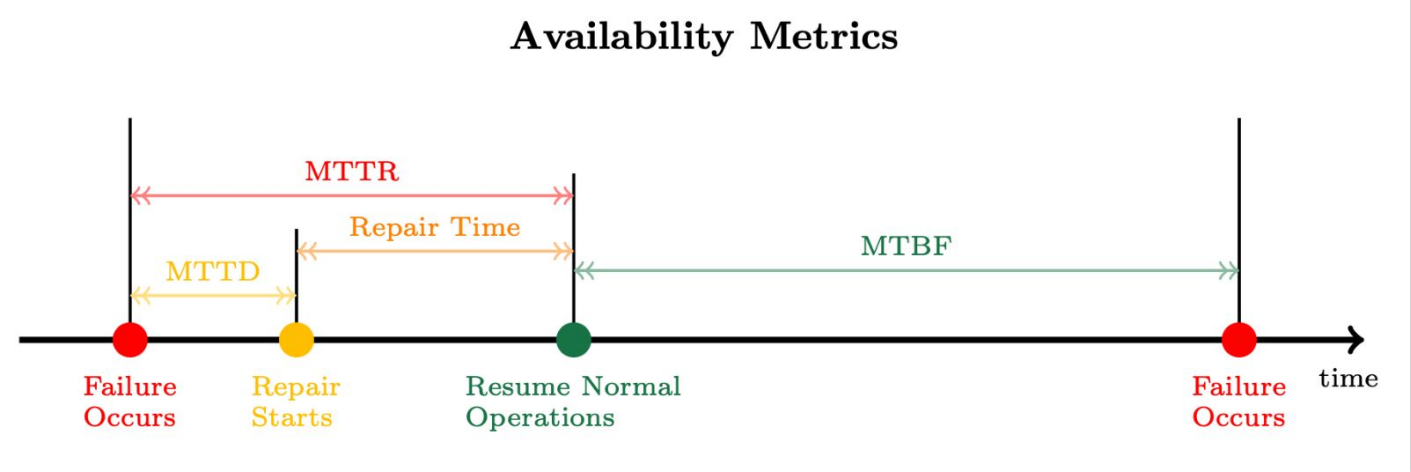
運用が目指すもの

new relic

運用が目指すもの

- **MTTD** (Mean Time to Detect) : 平均検出時間
  - 問題や障害が発生してから、それが検出されるまでに要する時間
- **MTTR** (Mean Time to Recovery) : 平均復旧時間
  - システムや機器が故障した際に、それを修理して正常な状態に戻すまでに要する時間
- **MTBF** (Mean Time Between Failures): 平均故障間隔時間
  - 故障と故障の間の時間。システムや機器が正常に稼働している期間

MTTD、MTTR、MTBF の関係を示す図

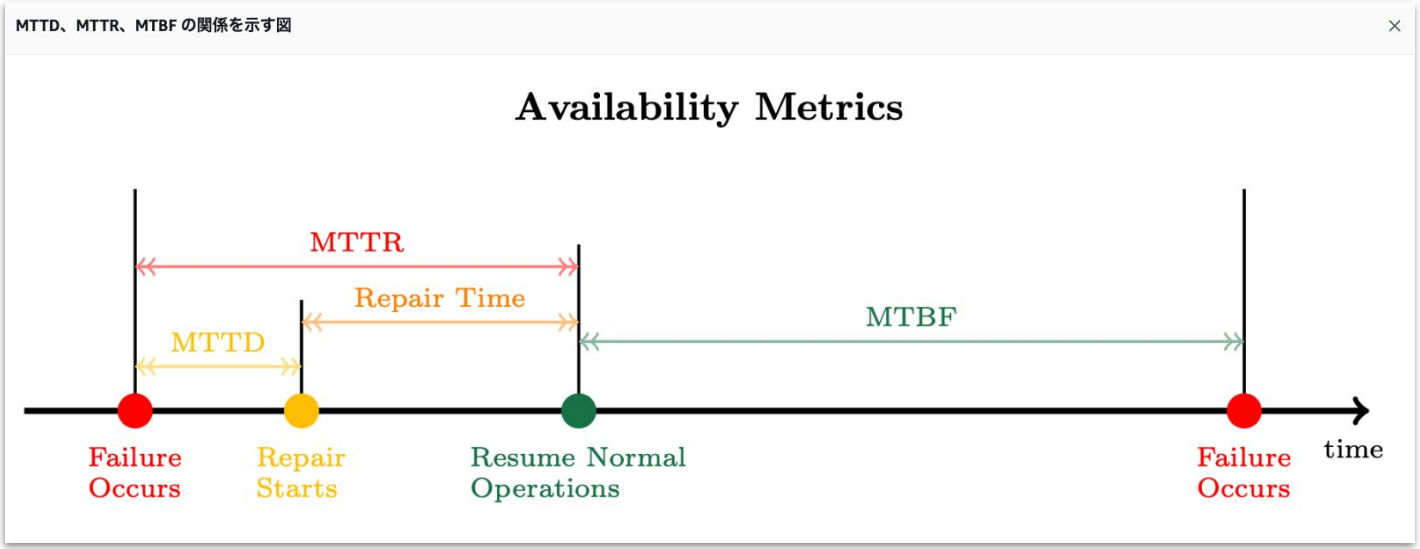


AWSホワイトペーパより引用
https://docs.aws.amazon.com/ja_jp/whitepapers/latest/availability-and-beyond-improving-resilience/understanding-availability.html

new relic

運用が目指すもの

- **MTTRをより短く**
- **MTBFをより長く**

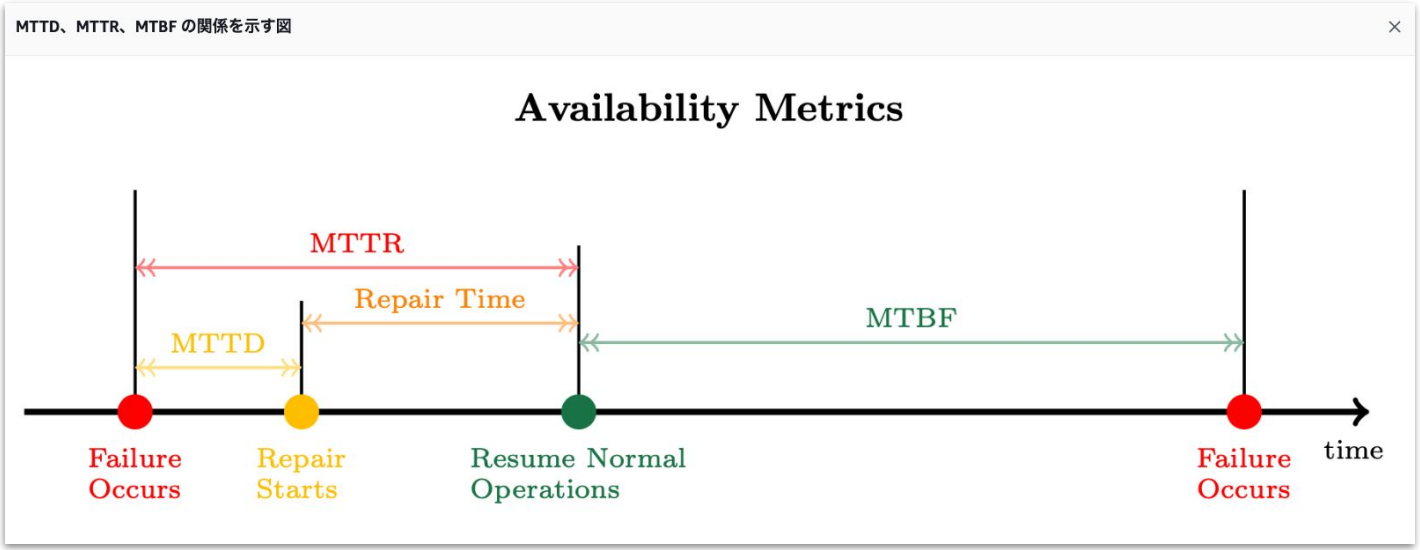


AWSホワイトペーパより引用
https://docs.aws.amazon.com/ja_jp/whitepapers/latest/availability-and-beyond-improving-resilience/understanding-availability.html

new relic

運用が目指すもの

- **MTTRをより短く** → **できるかぎり早く障害を収束させる**
- **MTBFをより長く** → **できるかぎり長く障害を起こさない**



AWSホワイトペーパより引用
https://docs.aws.amazon.com/ja_jp/whitepapers/latest/availability-and-beyond-improving-resilience/understanding-availability.html

監視とは

監視とは、あるシステムやそのシステムのコンポーネントの振る舞いや出力をチェックし続ける行為である

https://www.oreilly.co.jp/books/9784873118642/

new relic

監視とは

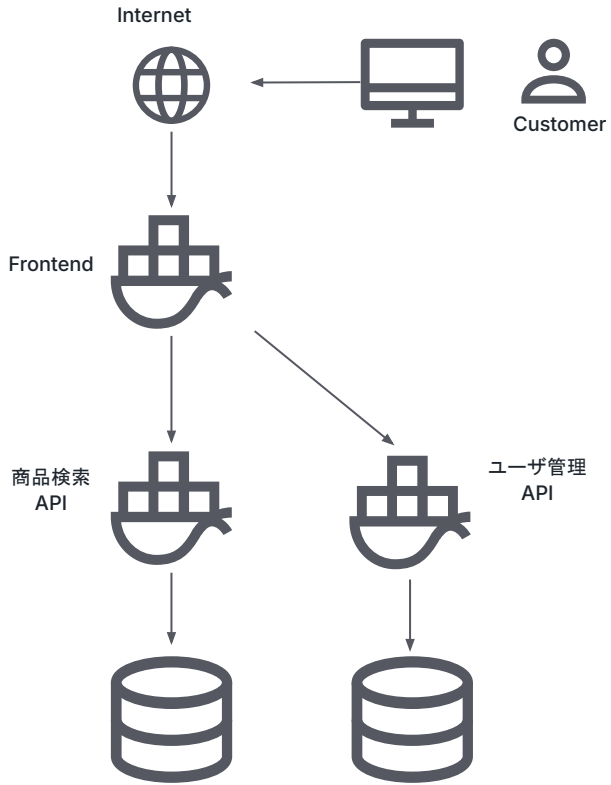
監視とは、あるシステムやそのシステムのコンポーネントの振る舞いや出力をチェックし続ける行為である

監視ポイントを決め、
その値がどう変化しているのかを
チェックする。
そして特定の値を逸脱した場合
にアラートとして通知する

https://www.oreilly.co.jp/books/9784873118642/

new relic

監視のデザインパターン

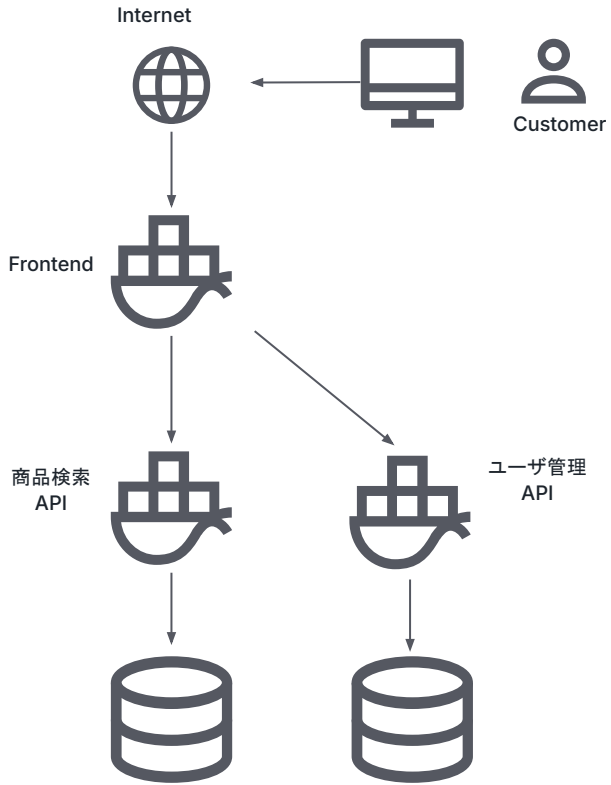


new relic

監視のデザインパターン

いくつかのフレームワーク

- USE Method
- RED Method
- Four Golden Signals



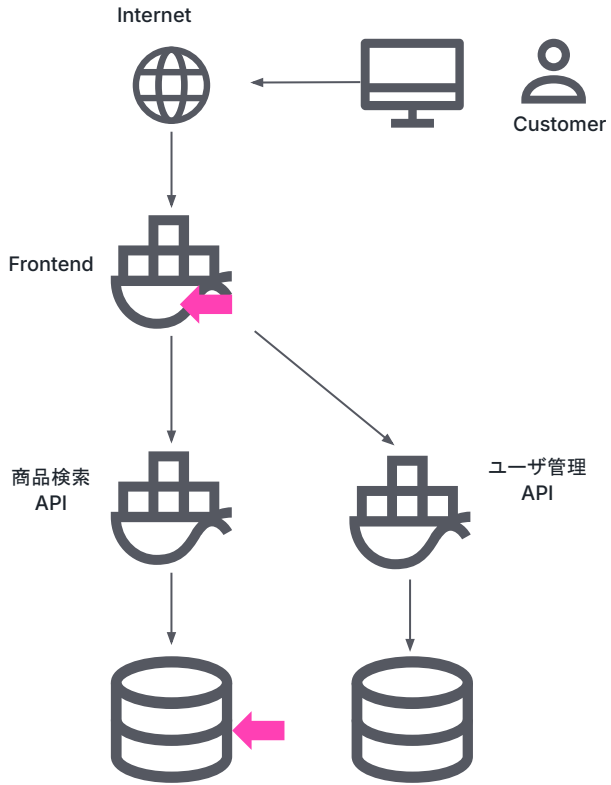
new relic

監視のデザインパターン

USE Method

- **U**tilization (利用率、忙しくなっている時間の割合)
- **S**aturation (行うべき作業の量、飽和の度合い)
- **E**rrors (エラーイベントの数)

- サーバやNWに用いられることが多い
  - CPU、メモリの使用率
  - ストレージの容量、Read/Writeキューの長さ
  - messageログに出ているエラー



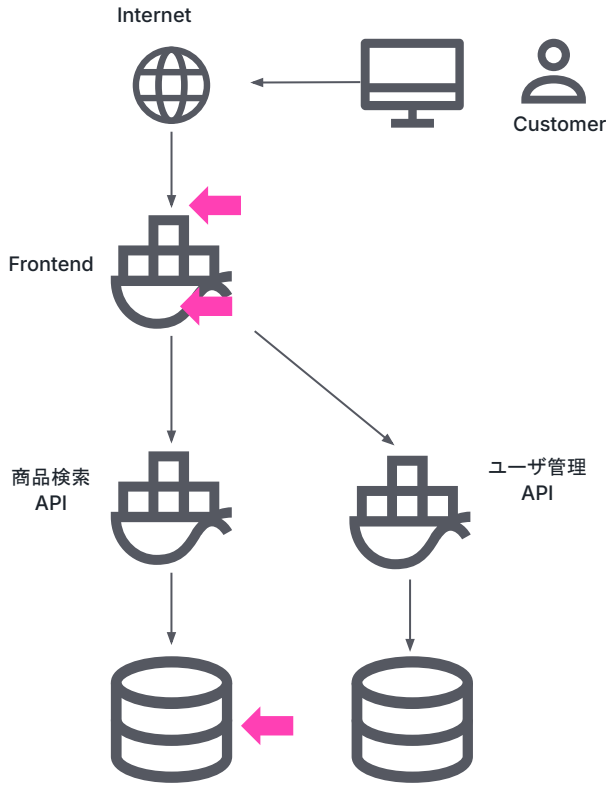
new relic

監視のデザインパターン

Four Golden Signals

- Latency(リクエストを処理するのにかかる時間)
- Traffic(システムに対するリクエストの量)
- Errors(失敗したリクエストの割合)
- Saturation(サービスがどれだけ飽和してるか)

- ユーザーが直接利用するシステムで、メトリクスを 4つだけ計測できるなら、この 4つに集中するのがよい。



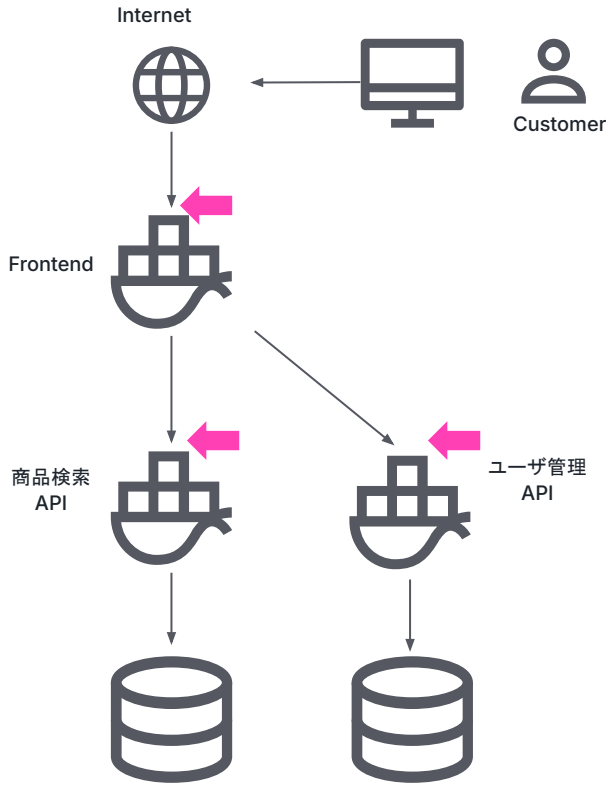
new relic

監視のデザインパターン

RED Method

- Rate (1秒あたりのリクエスト数)
- Errors (失敗したリクエストの数)
- Duration (リクエストの処理にかかった時間)

- USE Method に比べてBlackbox的
- Webサービスの外形監視と相性が良い。こういう値の方が気になりませんか？
  - 毎分当たりのリクエスト数
  - 5xxエラーの数
  - どのくらいの時間でリクエストを捌いてるのか

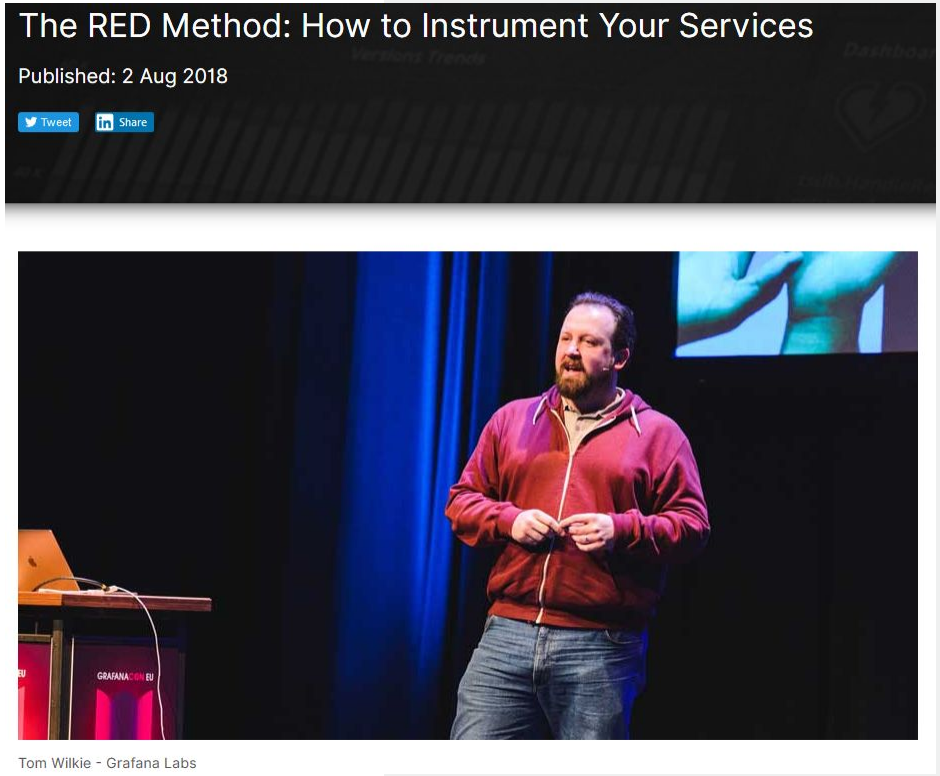


new relic

RED Method

サービスに関わる誰もがエラー率、リクエスト率、そしてそれらのリクエストのレイテンシの分布を理解する必要があります。

サービス監視の観点に一貫性を持たすことができ、開発側が意図していないコードのへの問い合わせも行えます。

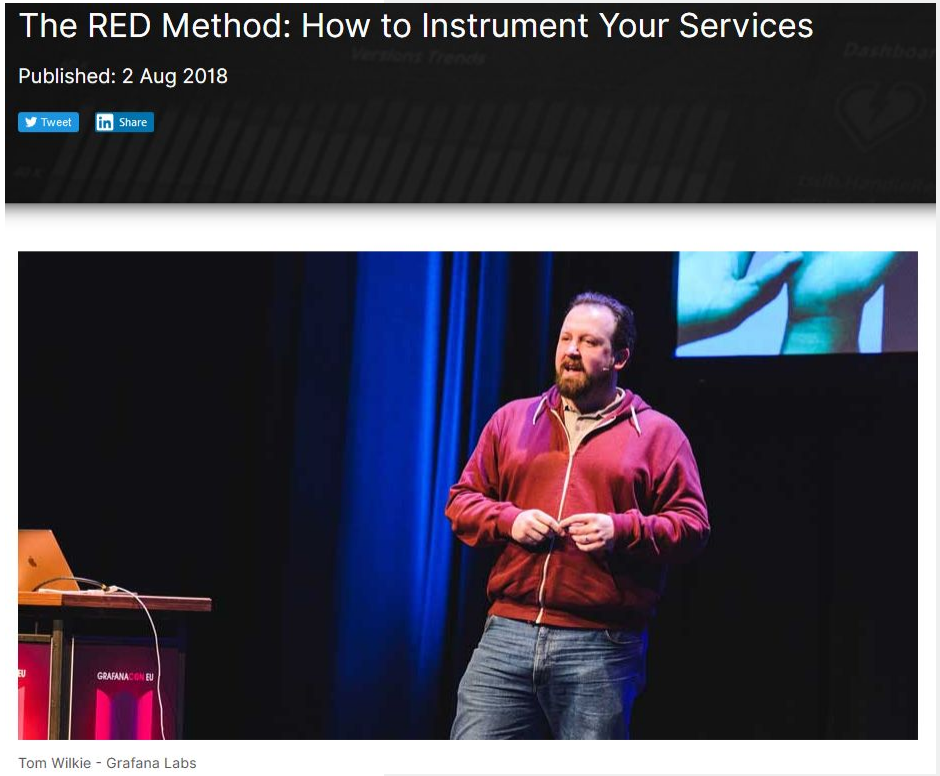


The RED Method: How to Instrument Your Services
https://grafana.com/blog/2018/08/02/the-red-method-how-to-instrument-your-services/

new relic

RED Method

RED Methodはユーザの満足度を示す優れた指標です。エラー率が高い場合、それは基本的にユーザーに影響があり、ページ読み込みエラーが発生します。表示時間が長いとWebサイトの表示が遅くなります。これらは、意味のあるアラートを作成し、**SLA** を測定するための優れた測定基準です。



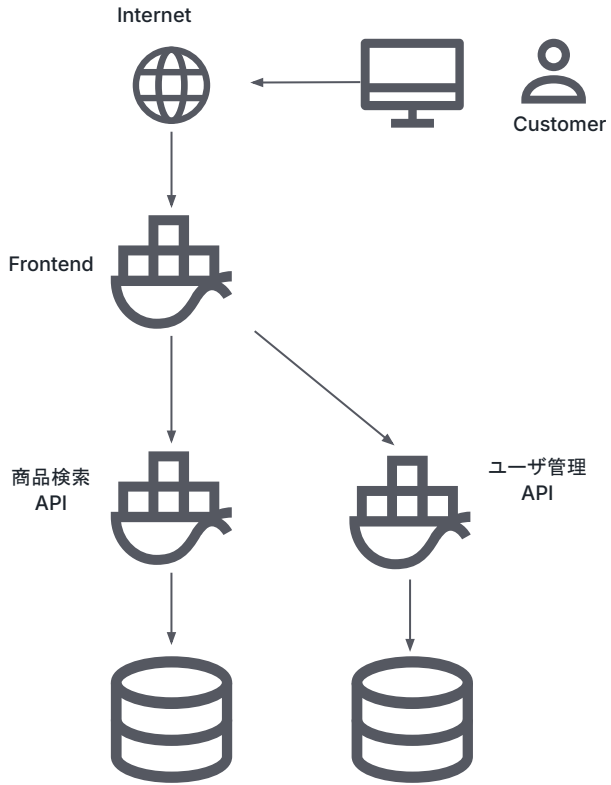
The RED Method: How to Instrument Your Services
https://grafana.com/blog/2018/08/02/the-red-method-how-to-instrument-your-services/

new relic

監視のデザインパターン

いくつかのフレームワーク

- RED Method
  - APIやWeb I/Fの健全性
- Four Golden Signals
  - サービスやシステムの全体的な健全性（アプリとインフラの状態）
- USE Method
  - サーバ、NW



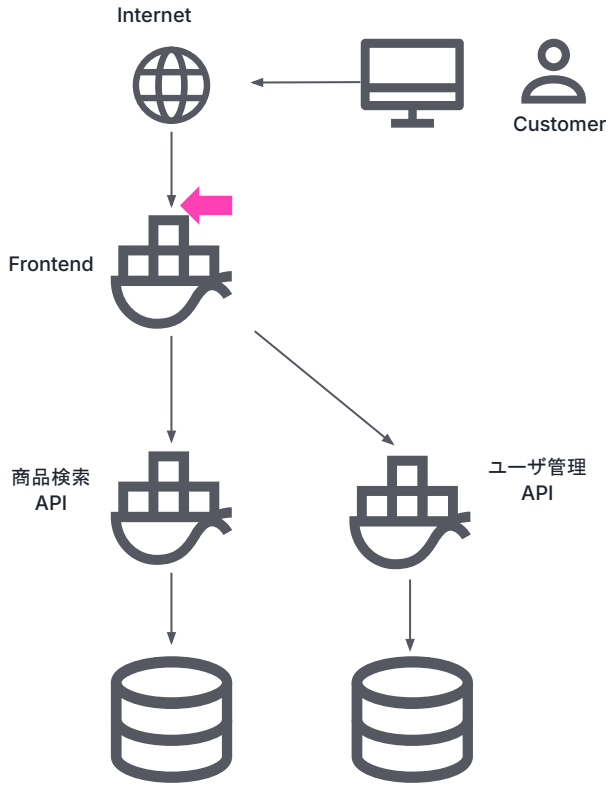
new relic

監視のデザインパターン

ユーザに近いインターフェースから

- 全体を俯瞰で見て、ユーザに近いところから
  - FrontのRED Method
    - Rate
    - Errors
    - Duration

そのアプリケーションが、やるべきことを上手く出来ているかどうかを監視する

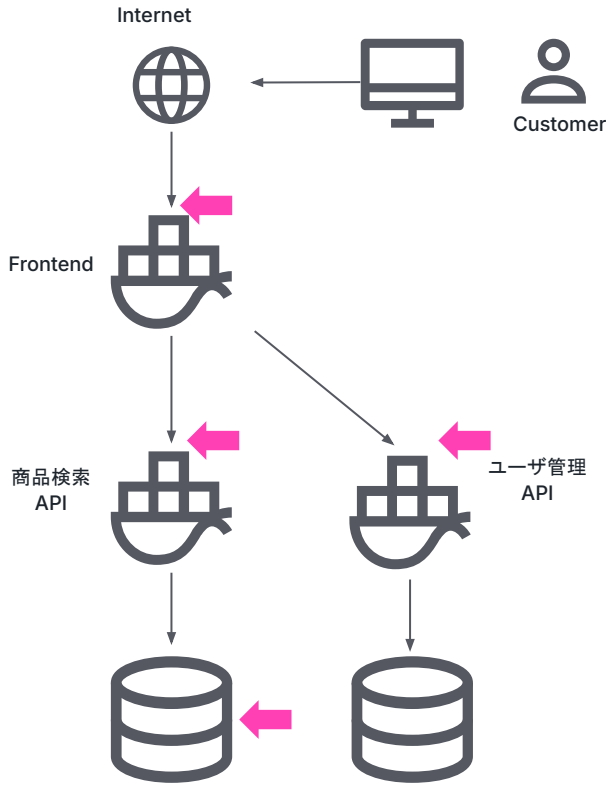


new relic

監視のデザインパターン

ユーザに近いインターフェースから

- その次は各インターフェースを見るようにする
- 各インターフェースの RED Methodが見られると良い
- ログやサーバのリソースといった内部の状態と関連付けて見られるように整備していく

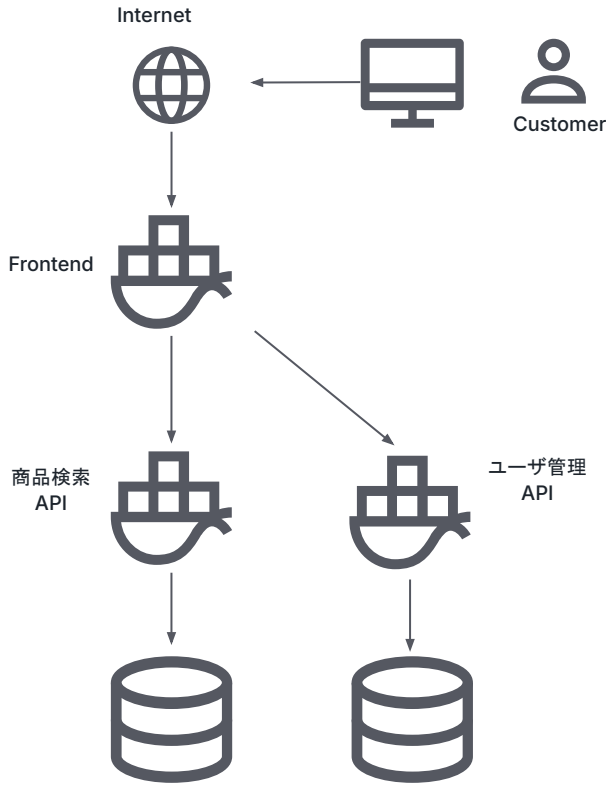


new relic

監視のデザインパターン

ツールを使う場合

- ユーザ目線での監視
  - 外形監視
  - RUM（Real User Monitoring）
- アプリケーションの RED Method
  - APM
- インフラの USE Method
  - CloudWatch metrics
  - エージェント

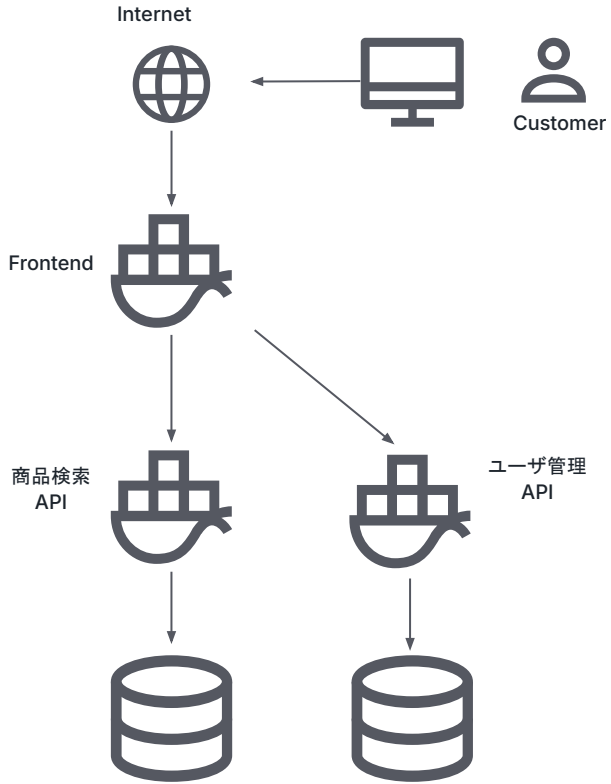


new relic

監視のデザインパターン

ツールを使う場合

- ユーザ目線での監視
  - **外形監視**
  - RUM（Real User Monitoring）
- アプリケーションの RED Method
  - APM
- インフラの USE Method
  - CloudWatch metrics
  - エージェント



new relic

監視のデザインパターン

ツールを使う場合

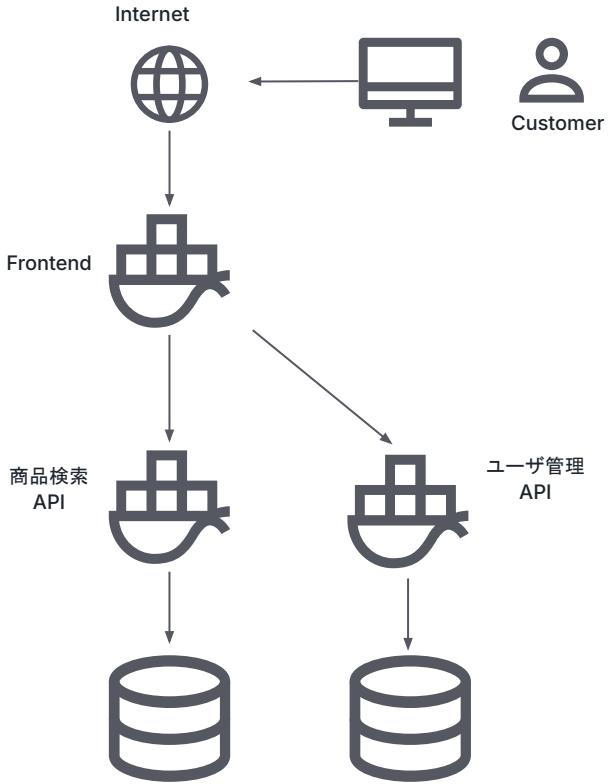
- ユーザ目線での監視
  - **外形監視**
    - サービスの外部からアクセスして、ユーザと同様の方法でアクセスして監視する方法
      - TOPページ
      - 特定ユーザでログインして操作

new relic

監視のデザインパターン

ツールを使う場合

- ユーザ目線での監視
  - 外形監視
  - **RUM（Real User Monitoring）**
- アプリケーションの RED Method
  - APM
- インフラの USE Method
  - CloudWatch metrics
  - エージェント

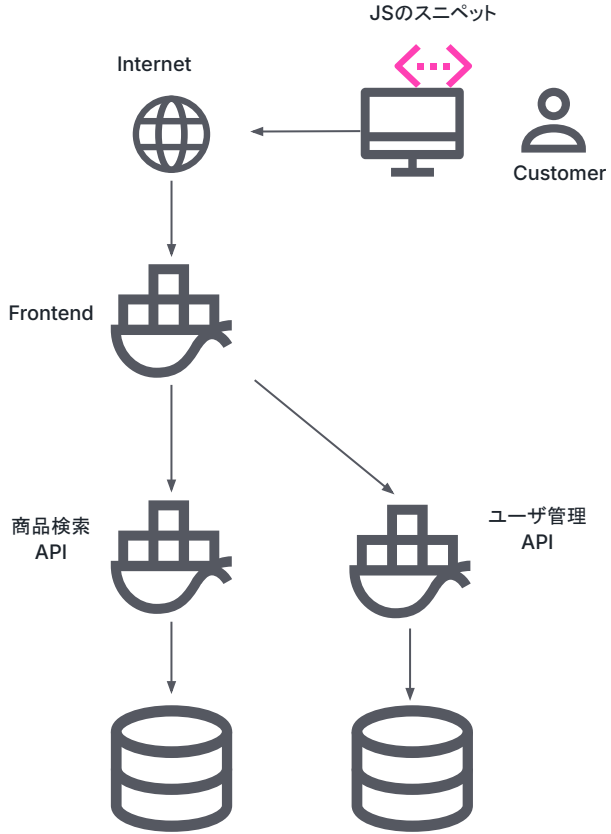


new relic

監視のデザインパターン

ツールを使う場合

- ユーザ目線での監視
  - 外形監視
  - **RUM（Real User Monitoring）**
    - 実際のユーザの行動データを取得
      - ページ毎のロード時間
      - JSのエラー
      - Core Web Vitals

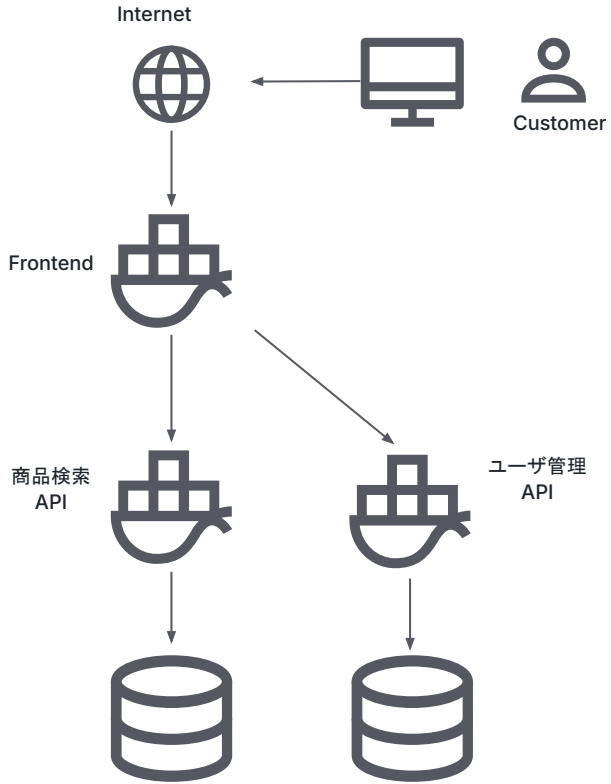


new relic

監視のデザインパターン

ツールを使う場合

- ユーザ目線での監視
  - 外形監視
  - RUM（Real User Monitoring）
- アプリケーションの RED Method
  - **APM**
- インフラの USE Method
  - CloudWatch metrics
  - エージェント

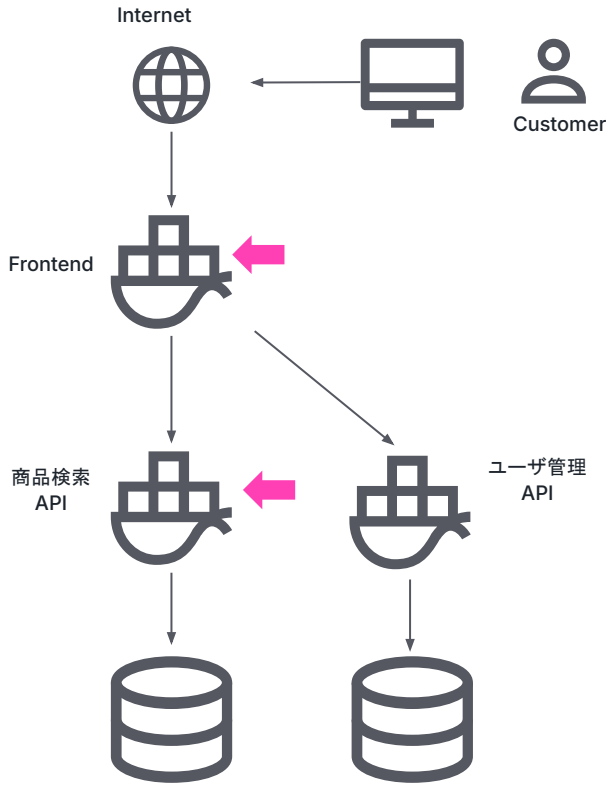


new relic

監視のデザインパターン

ツールを使う場合

- ユーザ目線での監視
  - 外形監視
  - RUM（Real User Monitoring）
- アプリケーションの RED Method
  - **APM**（**A**pplication **P**erformance **M**onitoring）
    - アプリケーションのパフォーマンスや状態を監視
    - メソッド、SQL単位の可視化が可能

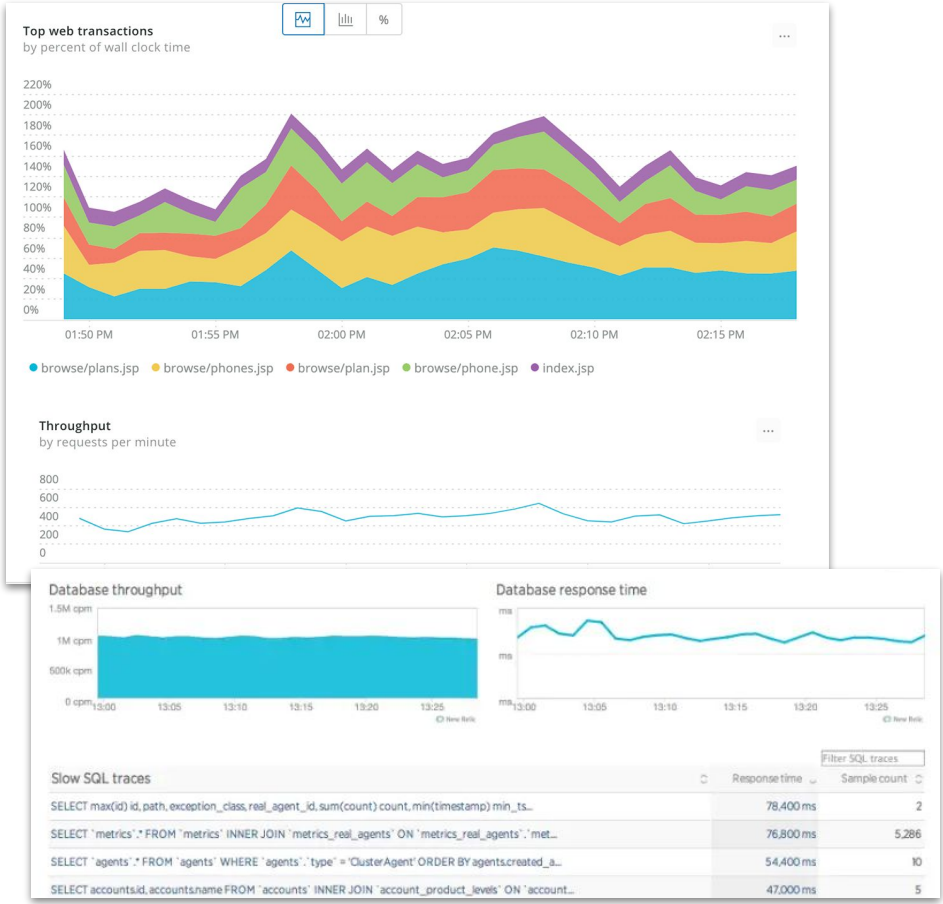


new relic

監視のデザインパターン

ツールを使う場合

- ユーザ目線での監視
  - 外形監視
  - RUM（Real User Monitoring）
- アプリケーションの RED Method
  - **APM**（**A**pplication **P**erformance **M**onitoring）
    - アプリケーションのパフォーマンスや状態を監視
    - メソッド、SQL単位の可視化が可能



new relic

監視のデザインパターン

通知戦略を考える

- 深刻度に応じた通知方法を考える
  - SlackやTeamsならチャンネルやメンションで使い分ける
  - **通知には「状況と受け取った人にやってほしいこと」を通知しましょう**
    - **その通知を受け取った人に、何をしてほしいですか？**

深刻度	対応方針の例
Disaster	深刻な障害。即時の情報共有、対応が必要
Critical	障害は発生したが即時対応は不要。翌営業日に対応
Warning	軽微な障害。5営業日以内に対応
Info	バックアップの状態など、知っておきたい情報

監視のアンチパターン

new relic

監視のアンチパターン

ツールに依存する

- ツールは加速装置です。導入それ自体を目的にしてはいけません
- ツールを導入することで解決する問題はありません
- 何を監視したいのかを明確にし、その目的を達成するための手段としてツールを選定・活用する
- 銀の弾丸はありません

監視のアンチパターン

役割としての監視

- 監視は役割ではなくスキル
- 監視するまで本番環境とは言えないようにする
- 組織的に監視の仕組みを作るチームが別だとしても、責任範囲は意識しましょう

監視とかよく分かんないから
インフラの人に…

監視のアンチパターン

とりあえず、何でも通知を出す

- その通知は担当者を叩き起こす必要はありますか
  - 深夜にCPU100％の通知を受け取った人の気持ち
  - エラーが起きてますと言われた人の気持ち
  - 通知はアクションを促すようにしましょう
- 多すぎる通知は人を麻痺させます
  - 「その通知は無視して」が日常化してないか

new relic

監視のアンチパターン

せっかくのインシデントを無駄にする

- インシデントは組織を強くするチャンス
- しっかり振り返りをしましょう
  - 非難しない
  - 人にフォーカスせず、モノやコトにフォーカス
  - その日、誰もが最善の行動を取ったという前提
  - インシデントが起きた原因を理解し、改善につなげる

ここまでのまとめ

- 運用はMTBFを増やし、MTTRを縮めるのがゴール
- デザインパターン
  - ユーザ目線での監視
    - RED Method
      - APM
      - 外形監視
  - 質の良いアラート
    - 何が起きてるのか分かる
    - 行動可能な通知を作る
- アンチパターン
  - 多すぎるアラート
  - インフラ偏重
  - チェックボックス監視

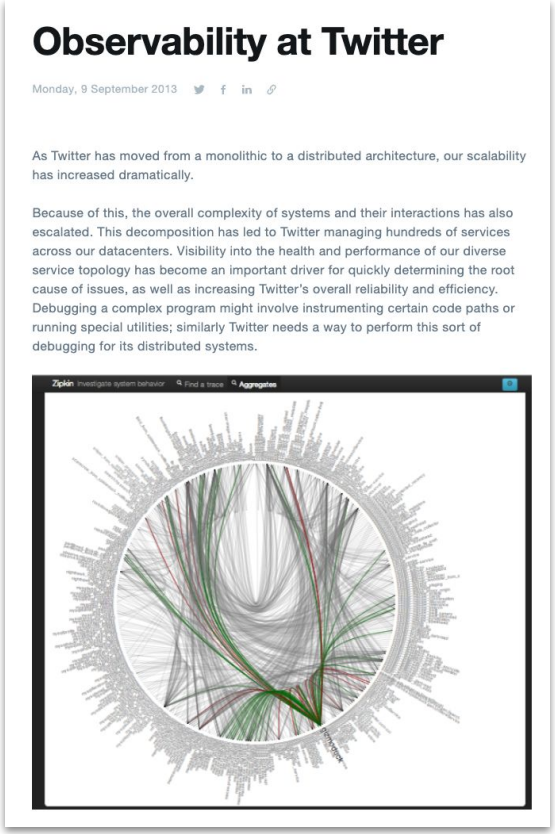
オブザーバビリティとは

new relic

オブザーバビリティとは

意味合いと歴史的経緯

- observe（観測）＋ability（能力）＝ observability（可観測性）
  - o11yと略されることもあります
- 制御工学理論で、システムの外部出力の情報から内部状態をどれだけ推測できるかという指標
- 2013年に「Observability at Twitter」という記事で、ITシステムに対してオブザーバビリティという言葉が使われた

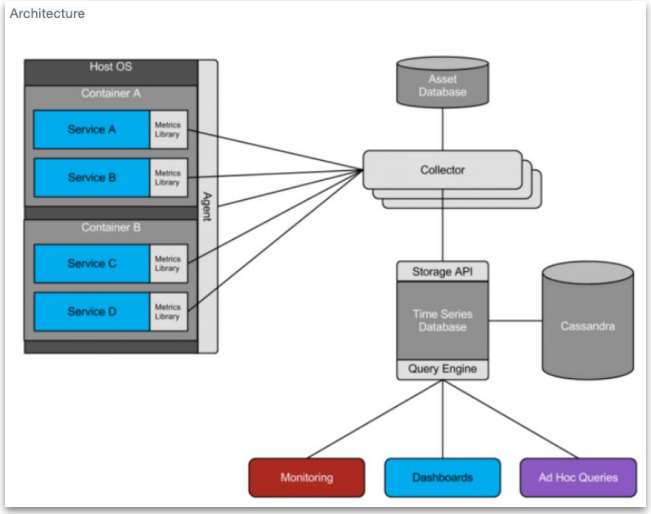


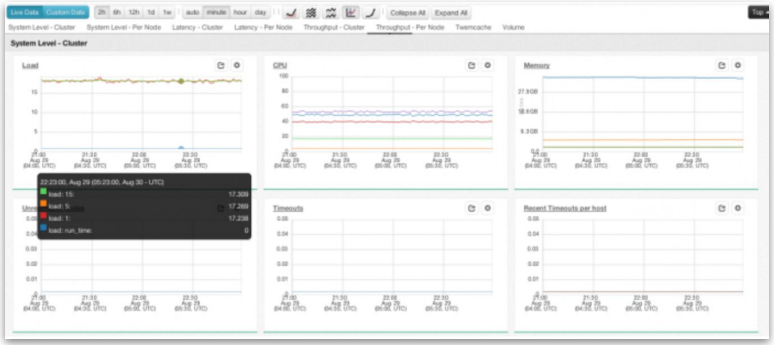
new relic

オブザーバビリティとは

Observability at Twitter

- Twitterはモノリシックアーキテクチャから分散アーキテクチャに移行し、スケーラビリティが向上
- 分散されたサービスは数百にのぼり、システムの複雑性と相互作用が増加。
- メトリクスの収集保存、クエリ処理、可視化、監視を統合し、エンジニアがシステムの状態をリアルタイムで把握できるようにした
- API成功率などのメトリクスを追跡し、異なるサービス間での影響を分析
- エンジニアは統一されたクエリ言語を使用して時系列データを可視化し、問題解析を行う。
- オブザーバビリティスタックは日々の運用監視とデータ分析に不可欠になっている。





https://blog.x.com/engineering/en_us/a/2013/observability-at-twitter

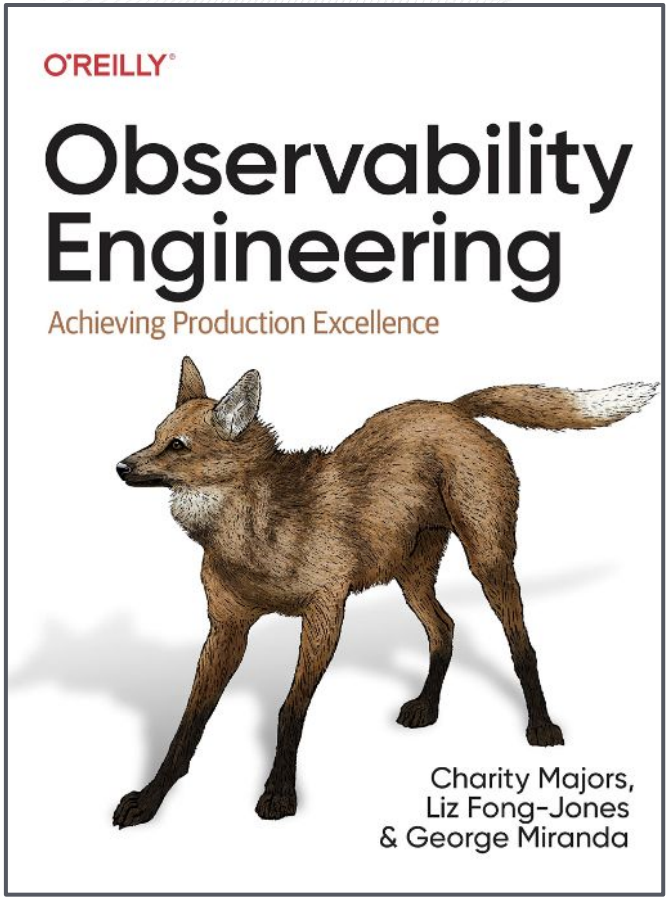
new relic

オブザーバビリティとは

『(意訳)**オブザーバビリティ**とは、システムの**あらゆる状態を理解し説明できる**特性です。

デバッグの事前定義や予測なしに、状態データをアドホックに調査し、**デバッグが可能に**することが求められます。

新しいコードのデプロイ不要で奇妙な状態も理解できる場合、オブザーバビリティが高いとされます。』



参考:Oreilly “Observability Engineering” 2022
https://www.oreilly.com/library/view/observability-engineering/9781492076438/

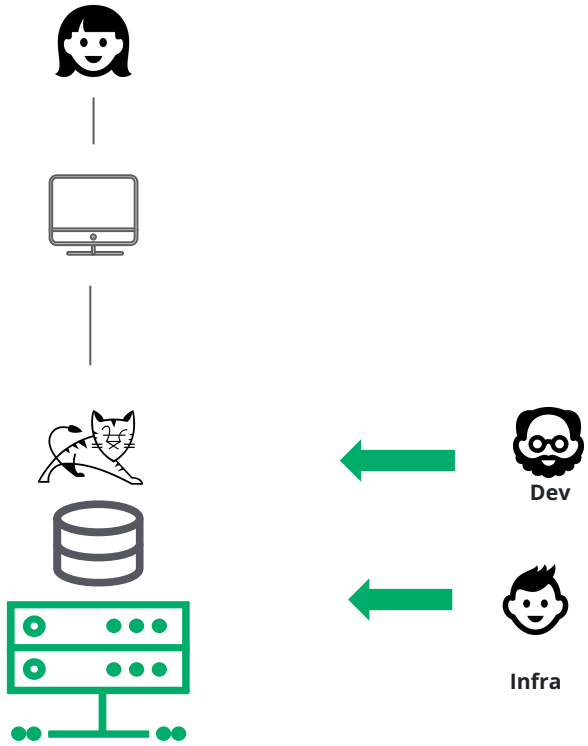
new relic

モノリシックなシステム

アプリがモノリシックかつ基盤が密結合だったため、リソースが枯渇しなければ大きな問題が発生しなかった

サーバのリソースやプロセスを死活を見ていれば、システムの健全性を把握できた

Devメンバーもサーバにログインすれば容易にデバッグできた

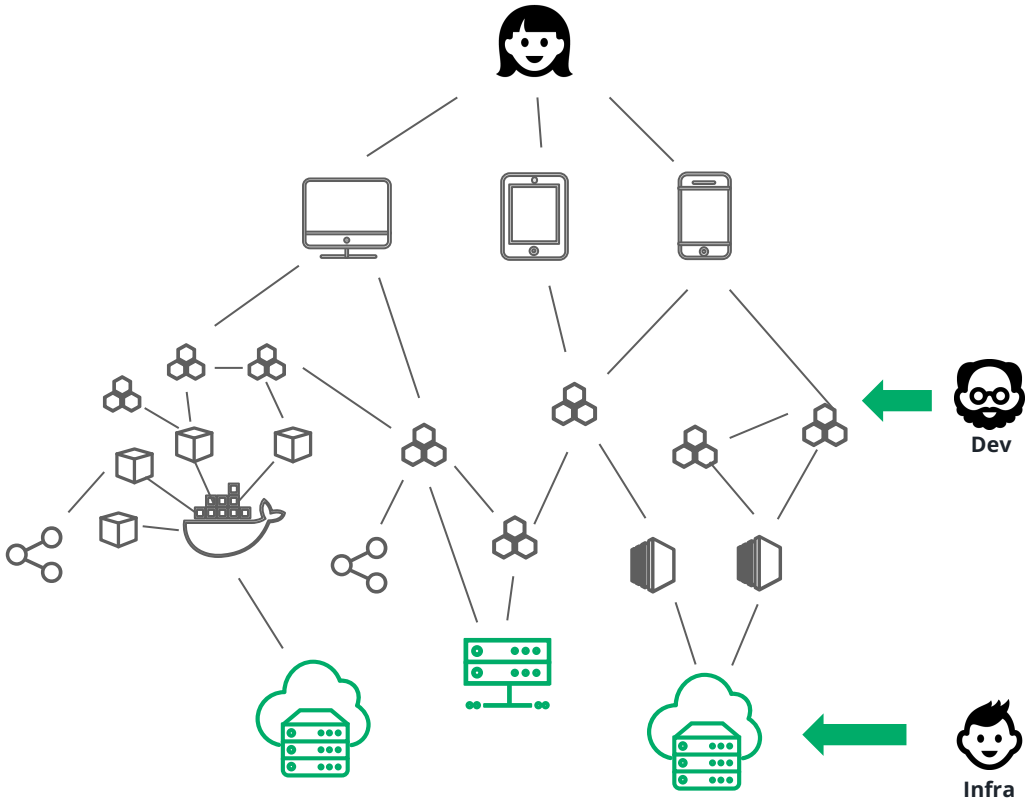


クラウドとマイクロサービス

アプリがマイクロサービス化かつ基盤と疎結合なため、基盤リソースと関係なく問題が発生しうる

一部コンポーネントに障害が起きていても全体としては動いている

サービスの健全性を容易に把握できない世界



マイクロサービス：大きなアプリケーションを小さな独立したサービスに分割する設計手法。各サービスは独立して開発・運用可能で、全体の柔軟性とスケーラビリティを向上させる手法

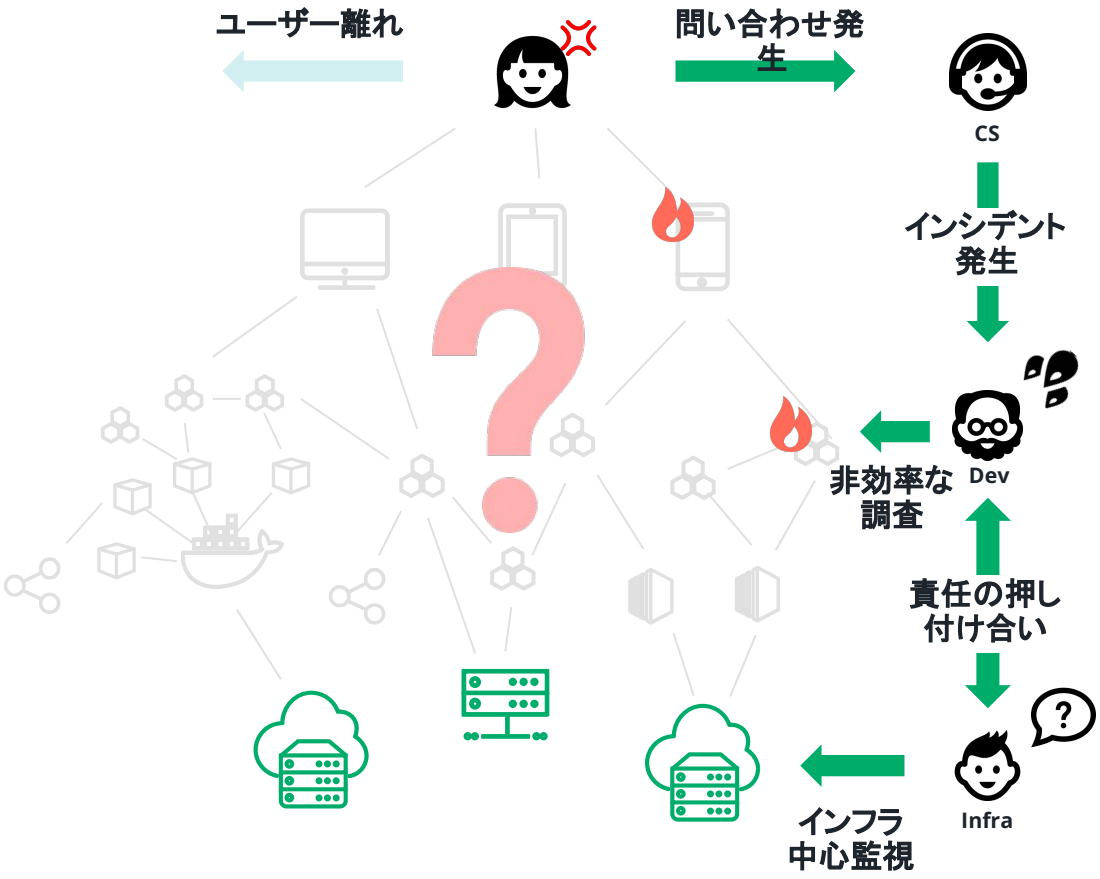
リアクティブな障害対応

責任の押し付け合い

非効率な調査

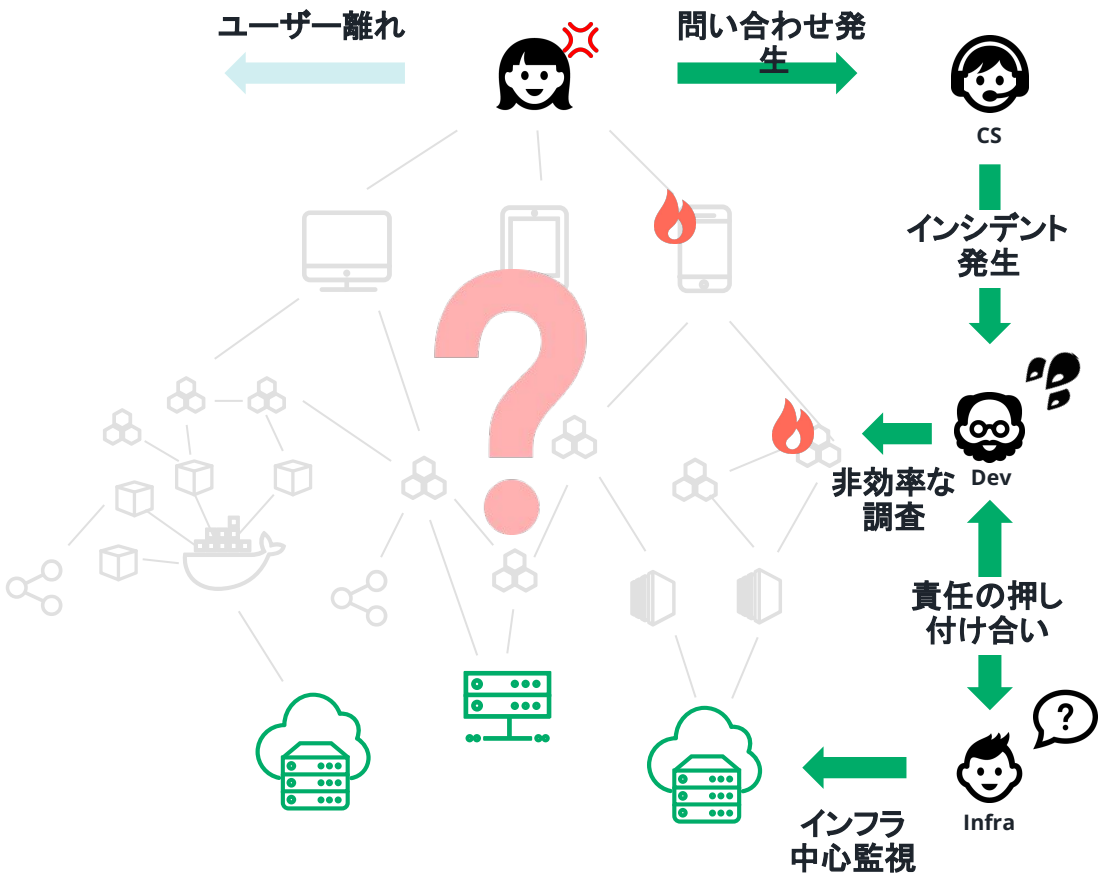
インシデント対応増加

顧客満足度低下



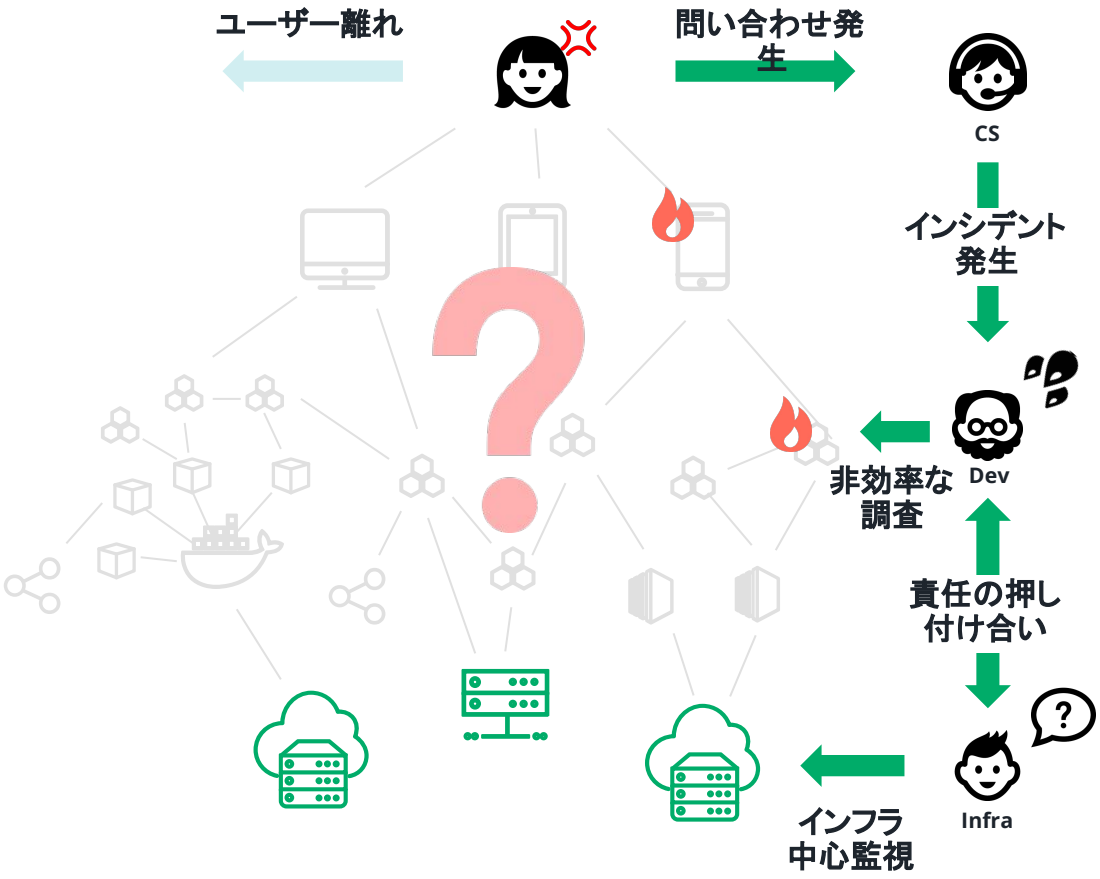
new relic

それらの問題を解決するため、システムの状態をリアルタイムに把握でき、システムの振る舞いを調査、理解できる能力が注目されるようになった

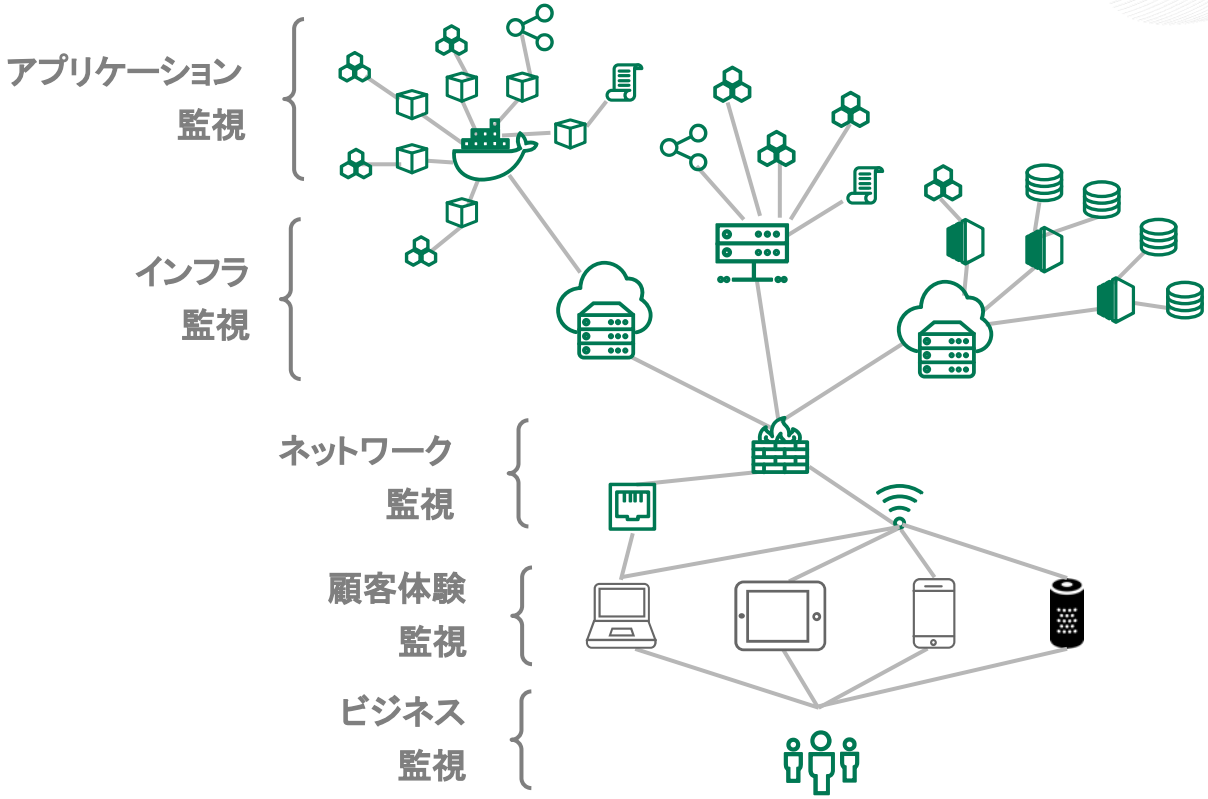


それらの問題を解決するため、システムの状態をリアルタイムに把握でき、システムの振る舞いを調査、理解できる能力が注目されるようになった

システムの異常にすぐに気づき、素早く原因にたどり着くことが重要



オブザーバビリティ（可観測性）



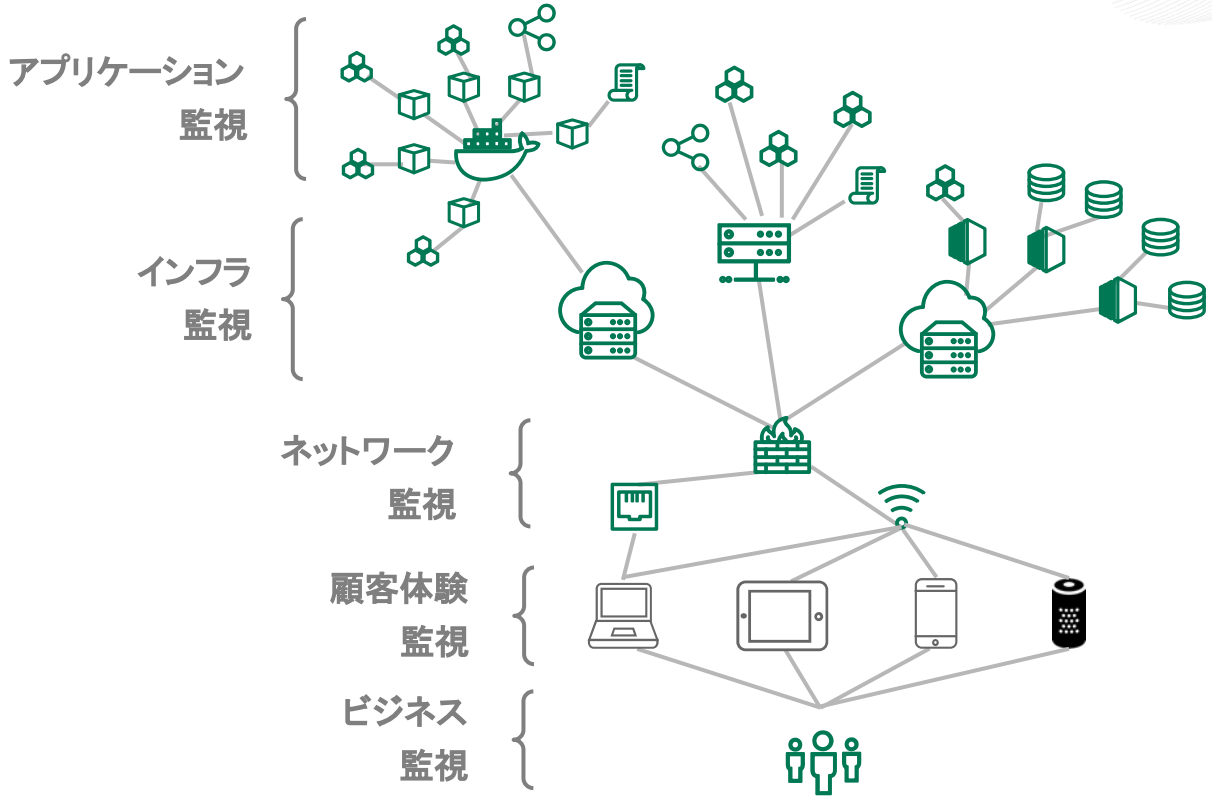
システム全体を計装

監視は個々のデータを必要に応じて収集・チェックするが、オブザーバビリティは全てを収集する

“サービスに支障をきたさないようにするためには、アプリケーションのあらゆる側面を観察、分析し、あらゆる異常を認識してすぐに修正する必要があります。” (出典：CNCF)

new relic

オブザーバビリティ（可観測性）



システム全体を計装

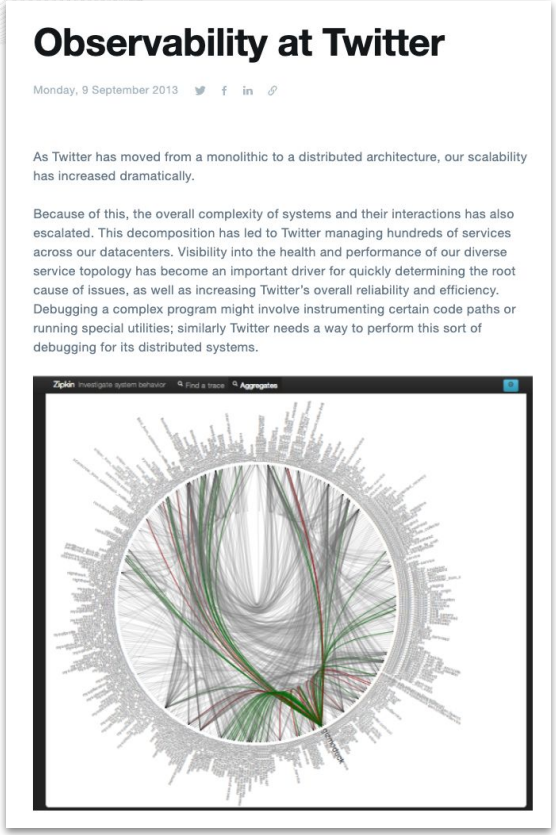
包括的にシステムの状態を把握することで以下のようなことが可能に

- 顧客体験の把握
- ビジネスKPIの把握
- サービスの品質を把握
- サービス品質とKPIの相関を把握

new relic

オブザーバビリティとは

- 時代背景
  - 複雑化したシステム
  - 高機能になった監視システム
- オブザーバビリティは、システムへの疑問にどれだけ早く正確に答えられるかという能力
  - いつもと違う動きがないか
  - 特定ユーザの行動や満足度
  - 機能をリリースした前後のシステムの振る舞い変化



new relic

オブザーバビリティで重要なもの

Primary Signals

- Logs
- Metrics
- Traces
- Dumps
- Profiles

https://github.com/cncf/tag-observability/blob/main/whitepaper.md

オブザーバビリティで重要なもの

Primary Signals

- **Logs**
  - ログはシステムの特定のイベントを記録するためのデータです。これは通常、システムが特定のアクションを実行するたびに生成されます。ログはエラーメッセージ、警告、情報メッセージなどを含むことがあります。また、ログはシステムが何をしたか、何が問題であったか、何が起こったかを把握するための重要なツールです

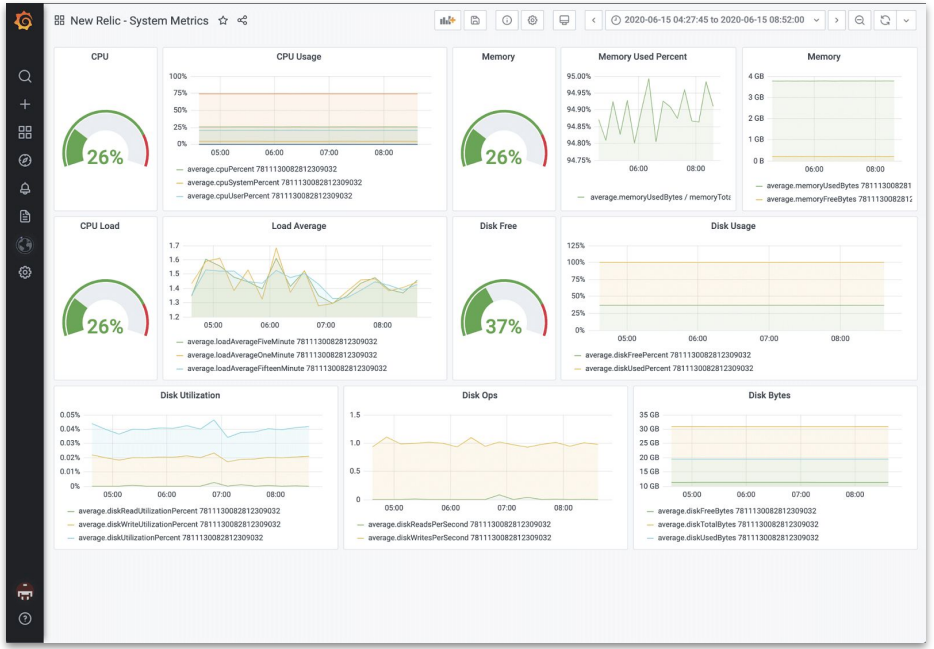
https://github.com/cncf/tag-observability/blob/main/whitepaper.md

new relic

オブザーバビリティで重要なもの

Primary Signals

- **Metrics**
  - メトリクスは一定期間にわたるシステムの状態を数値で表現したものです。これは通常、リアルタイムまたは近リアルタイムで収集され、時間の経過とともにシステムのパフォーマンスを追跡するために使用されます。例えば、CPU使用率、メモリ消費量、リクエスト数などがメトリクスに含まれる可能性があります。



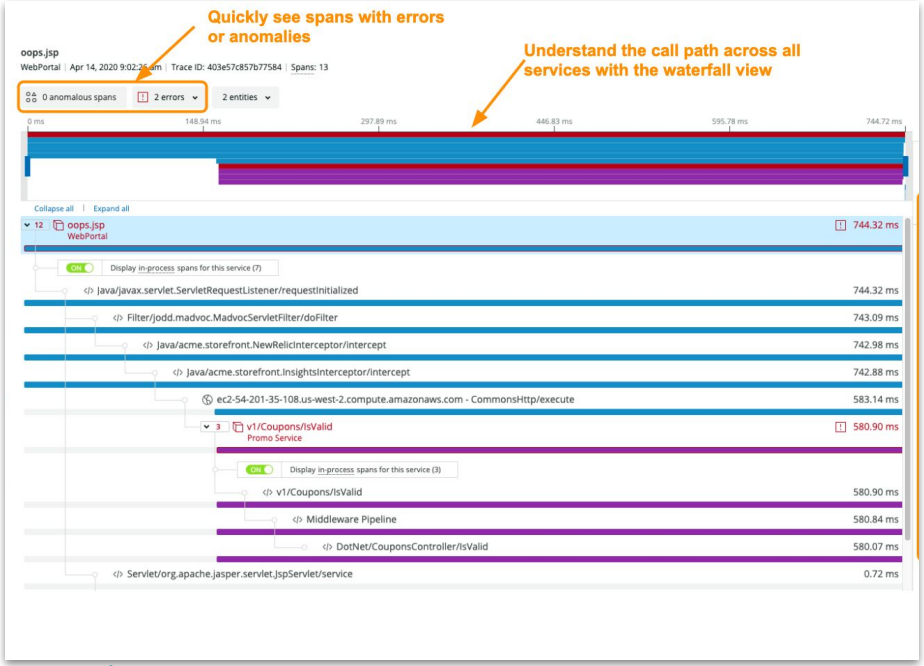
https://github.com/cncf/tag-observability/blob/main/whitepaper.md

new relic

オブザーバビリティで重要なもの

Primary Signals

- **Traces**
  - トレースはシステム内での一連の関連イベントを追跡するためのデータです。これは通常、分散システムにおけるリクエストのパスを追跡するために使用されます。トレースは、リクエストがシステム内をどのように移動し、どのサービスが関与し、それぞれのサービスでどれくらいの時間がかかったかを明らかにすることができます。



https://github.com/cncf/tag-observability/blob/main/whitepaper.md

new relic

オブザーバビリティで重要なもの

Primary Signals

- **Dumps**
  - ダンプはシステムの特定の瞬間の状態をキャプチャするためのデータです。これは主にバグのトラブルシューティングやパフォーマンス問題の解析に使われます。ダンプには、システムのメモリ、プロセッサの状態、スタックトレース、その他の診断情報が含まれることがあります。

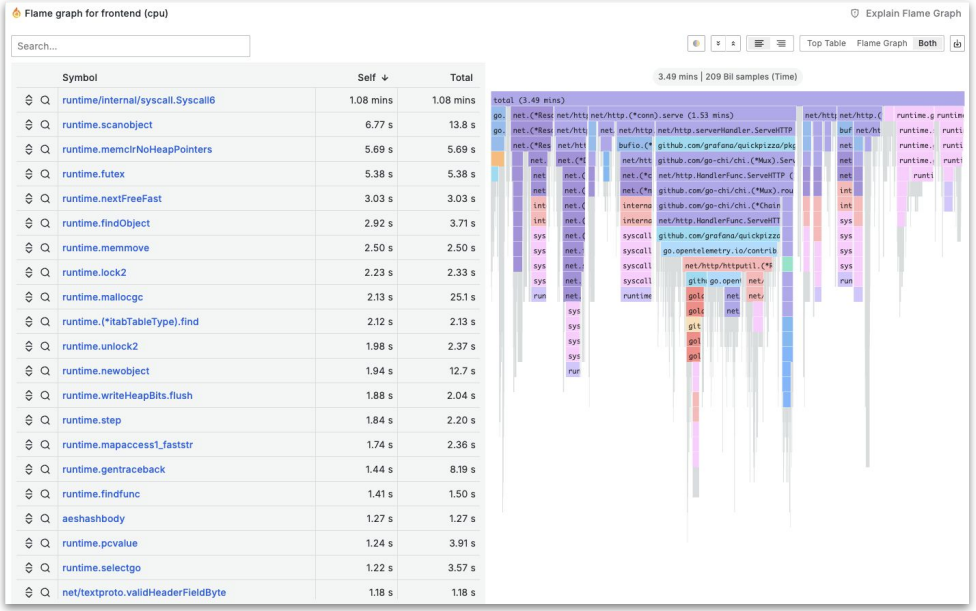
https://github.com/cncf/tag-observability/blob/main/whitepaper.md

new relic

オブザーバビリティで重要なもの

Primary Signals

- **Profiles**
  - プロファイルは、一定期間にわたるシステムの詳細なパフォーマンス情報を提供します。これは通常、コードレベルのリソース消費（例えば、メモリ使用量、CPU時間など）を追跡するために使用されます。プロファイルを使用すると、パフォーマンスのボトルネックや最適化の機会を特定することができます。



https://github.com/cncf/tag-observability/blob/main/whitepaper.md

new relic

オブザーバビリティで重要なもの

Primary Signals

- Logs
- Metrics
- Traces
- Dumps
- Profiles

- すべてを揃える必要はなく、必要に応じて観測できるものを増やしていく事が重要

https://github.com/cncf/tag-observability/blob/main/whitepaper.md

オブザーバビリティがもたらすもの

new relic.

オブザーバビリティがもたらすもの

本番リリースが怖い

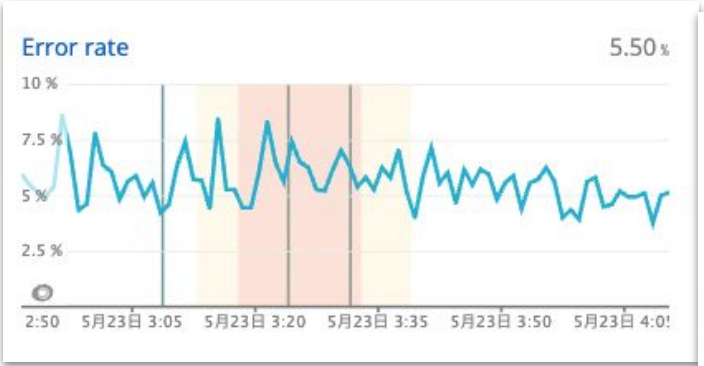
- 変更障害発生率はエリートと呼ばれる企業でも、0-15%
- 金曜にリリースして土日に障害対応。。。

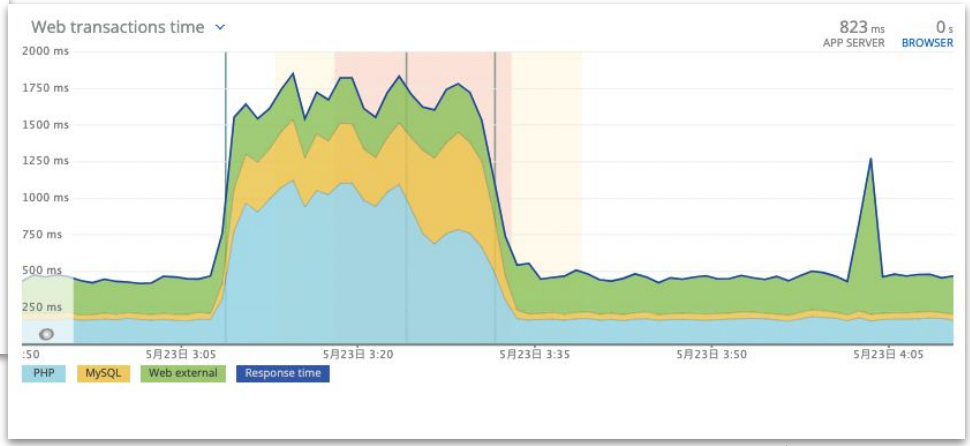
オブザーバビリティがもたらすもの

本番リリースが怖い

- 変更障害発生率はエリートと呼ばれる企業でも、0-15%
- 金曜にリリースして土日に障害対応。。。

デプロイ前後のシステム変化をすぐに確認



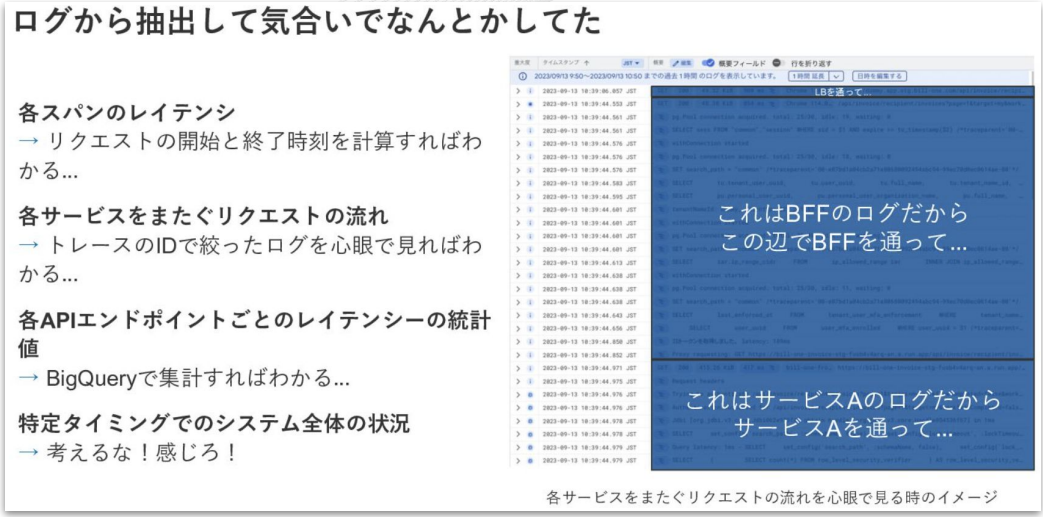


new relic

オブザーバビリティがもたらすもの

トラブルシューティングの民主化

- かつての運用は低レイヤのメトリクスや各種ログから、原因を 推測 するしかなった
- 必然的に経験値の高い人が重宝され、アプリケーション、アーキテクチャに詳しい人が障害対応、デバッグにおける最後の砦になり、運用が属人化する
- 勘や経験に頼った運用 になり、システム運用が職人になる



参考：勘に頼らず原因を □つけるためのオブザーバビリティ https://speakerdeck.com/sansantech/sansan-20230929

new relic

オブザーバビリティがもたらすもの

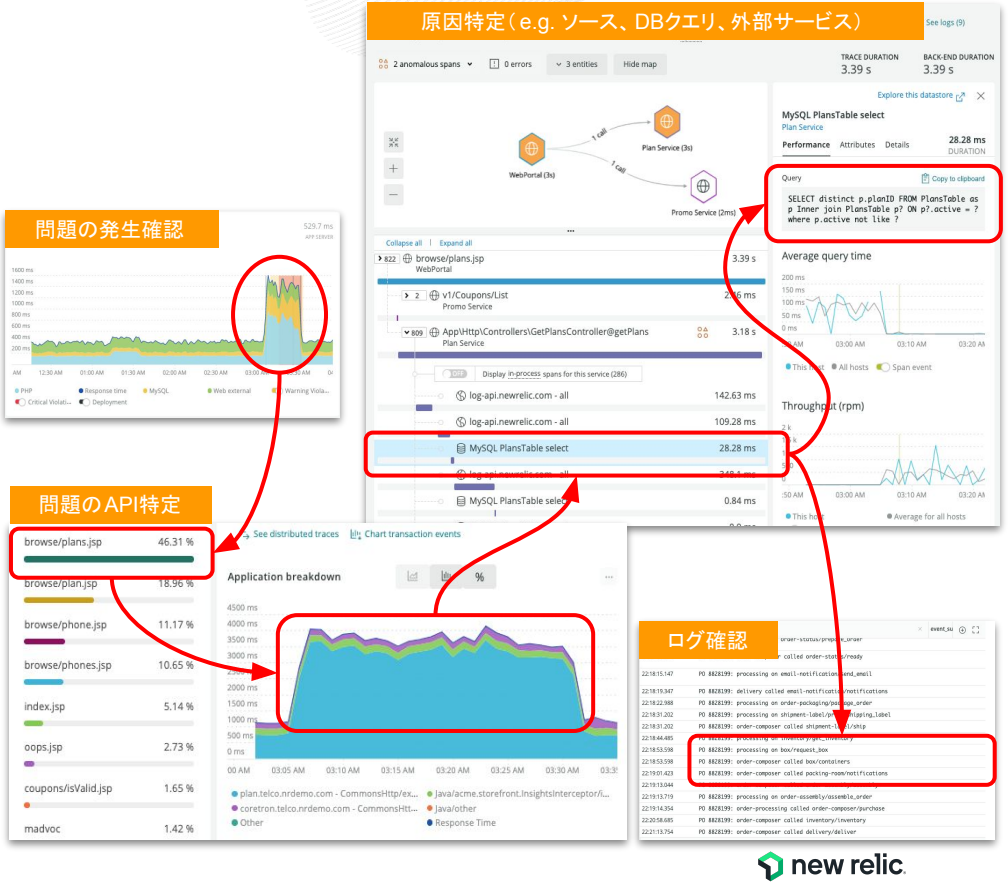
トラブルシューティングの民主化

- インフラからクライアントサイドまで、システムのスタック全体に渡って一つのプラットフォームで迅速に問題解決 が可能

- ソースコードやDBクエリの詳細なレベルまで原因特定ができ、根本解決が可能

- *オブザーバビリティツールでは、チーム内の最□のデバッガーは、通常、もっとも好奇□の強いエンジニアです。*

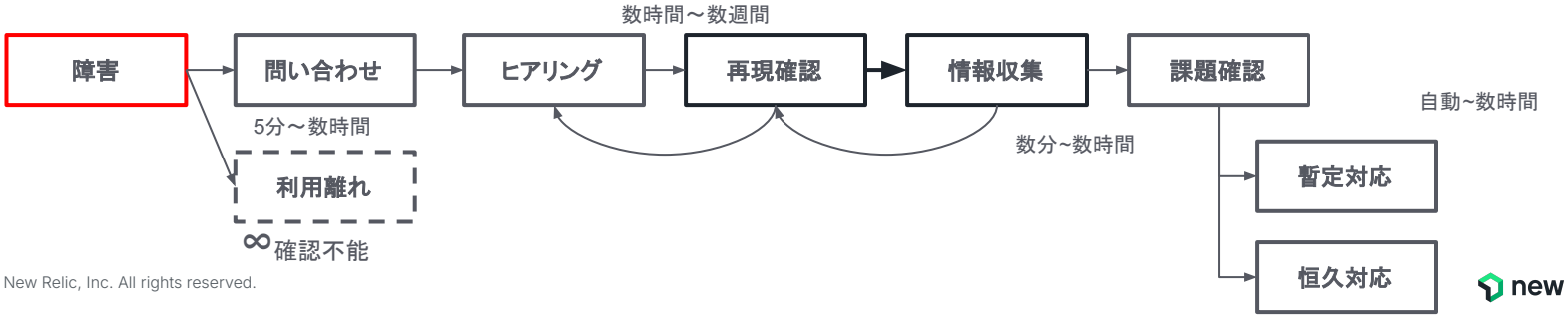
参考：Oreilly “Observability Engineering” 2022
https://www.oreilly.com/library/view/observability-engineering/9781492076438/



オブザーバビリティがもたらすもの

例えばこんな問い合わせが来ました

- ECサイトにおいてあるユーザから「昨日の 22時にスマホアプリで買い物をしていたが、途中で突然アプリが落ちた」という問い合わせが来た
  - そのユーザが昨日の 22時にどういう操作をしてエラーになったのか
  - アプリケーションクラッシュの原因はなにか
  - 注文処理が正常に終わっているか

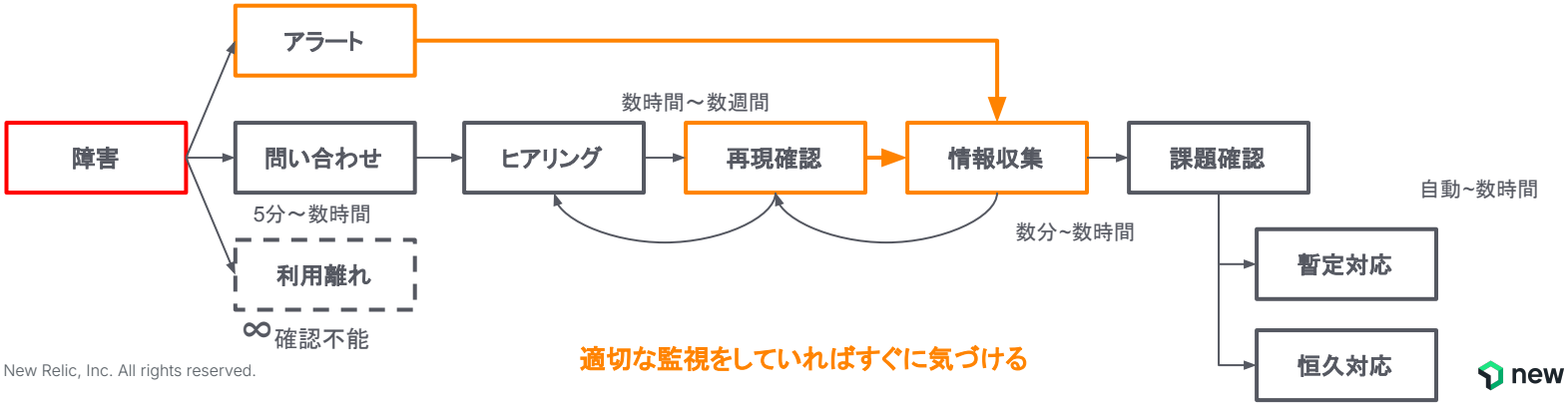


new relic

オブザーバビリティがもたらすもの

例えばこんな問い合わせが来ました

- ECサイトにおいてあるユーザから「昨日の 22時にスマホアプリで買い物をしていたが、途中で突然アプリが落ちた」という問い合わせが来た
  - そのユーザが昨日の 22時にどういう操作をしてエラーになったのか
  - アプリケーションクラッシュの原因はなにか
  - 注文処理が正常に終わっているか



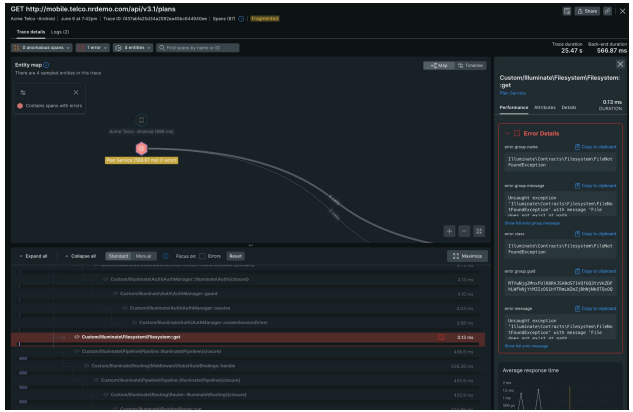
適切な監視をしていればすぐに気づける

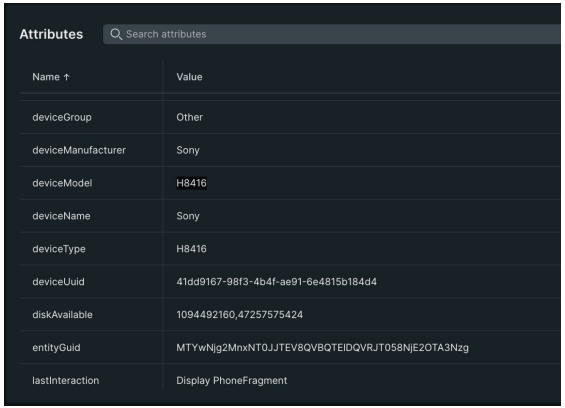
new relic

オブザーバビリティがもたらすもの

例えばこんな問い合わせが来ました

- ECサイトにおいてあるユーザから「昨日の 22時にスマホアプリで買い物をしていたが、途中で突然アプリが落ちた」という問い合わせが来た
  - そのユーザが昨日の 22時にどういう操作をしてエラーになったのか
  - アプリケーションクラッシュの原因はなにか
  - 注文処理が正常に終わっているか





オブザーバビリティが高いと、
エラーの原因を特定できる

new relic

オブザーバビリティがもたらすもの

プロダクトマーケットフィット

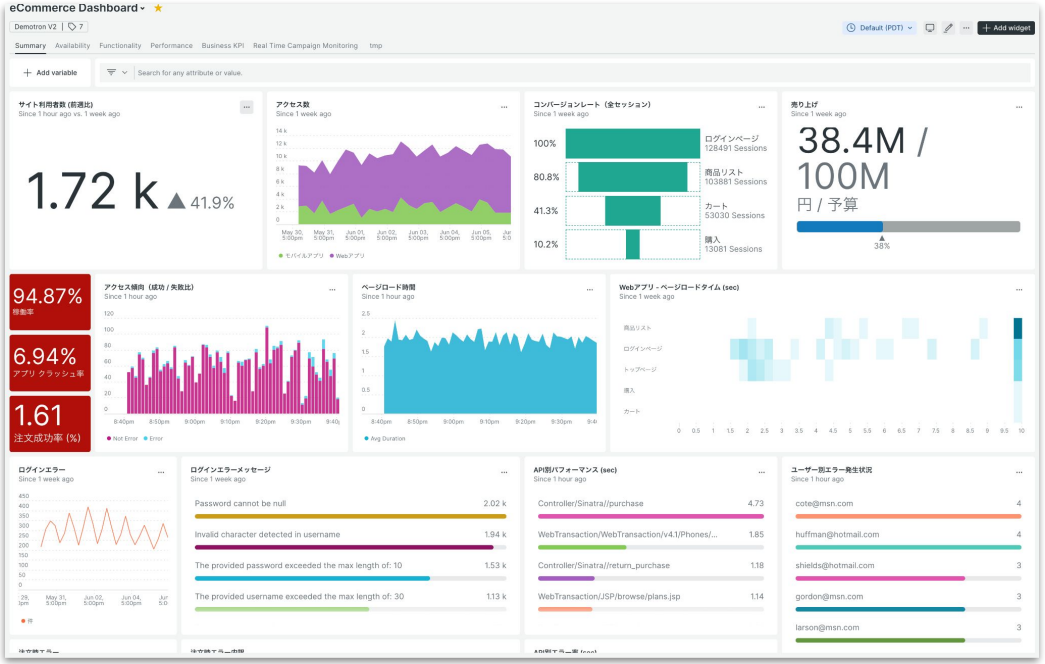
- 営業や企画「自分らの製品は市場にに受け入れられてるのかな？」
- エンジニア「自分らの作った機能は使ってもらえてるんだろうか？」

オブザーバビリティがもたらすもの

プロダクトマーケットフィット

- 営業や企画「自分らの製品は市場にに受け入れられてるのかな？」
- エンジニア「自分らの作った機能は使ってもらえてるんだろうか？」

サービスの品質とビジネスKPIを可視化



new relic

オブザーバビリティは誰のもの？

ITシステムにおける登場人物


運用

システムの安定稼働がミッション。小規模な組織では開発やインフラメンバーが兼任することも


開発

アプリケーションの開発がミッション。担当機能のパフォーマンスが気になる


ビジネス層

アプリケーションの売上向上がミッション。新機能やエンドユーザーの動向が気になる


ユーザ

アプリケーションのエンドユーザー

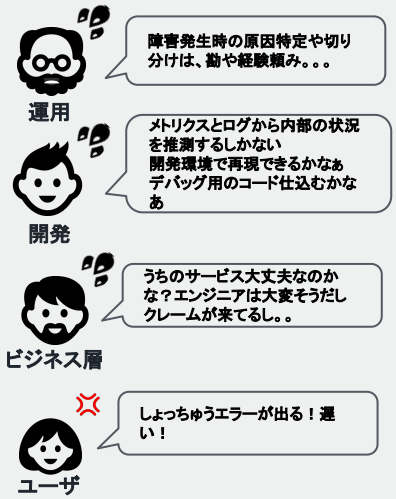
new relic

オブザーバビリティは組織の能力

- 組織でオブザーバビリティを高めれば、サービスに関わる人とエンドユーザーが幸福になる

オブザーバビリティの成熟度

単純なメトリクスだけを監視



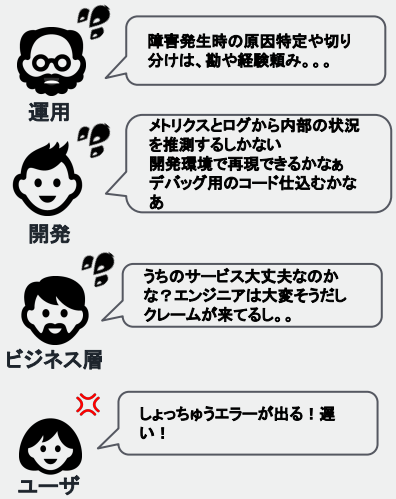
new relic

オブザーバビリティは組織の能力

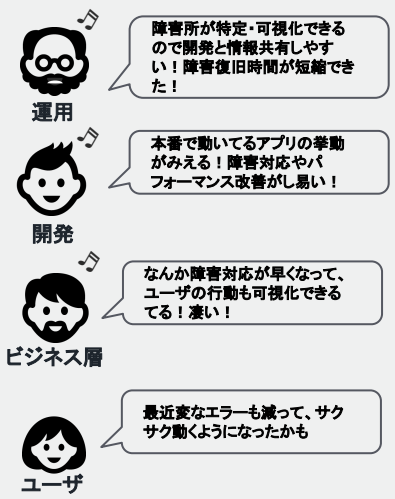
- 組織でオブザーバビリティを高めれば、サービスに関わる人とエンドユーザーが幸福になる

オブザーバビリティの成熟度

単純なメトリクスだけを監視



APMなどでアプリケーション・ユーザ動向のリアルタイムモニタリング



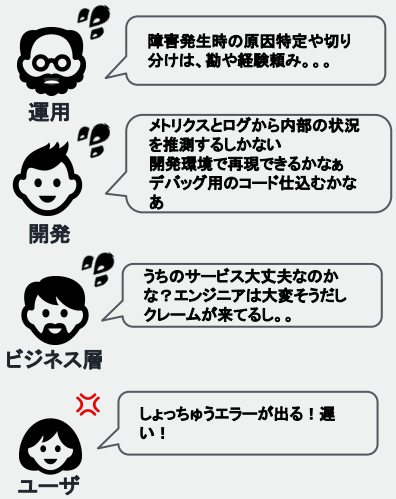
new relic

オブザーバビリティは組織の能力

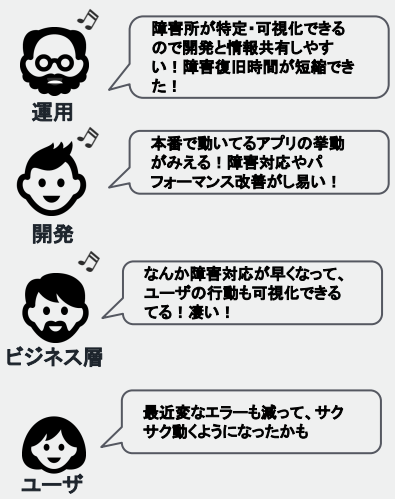
- 組織でオブザーバビリティを高めれば、サービスに関わる人とエンドユーザーが幸福になる

オブザーバビリティの成熟度

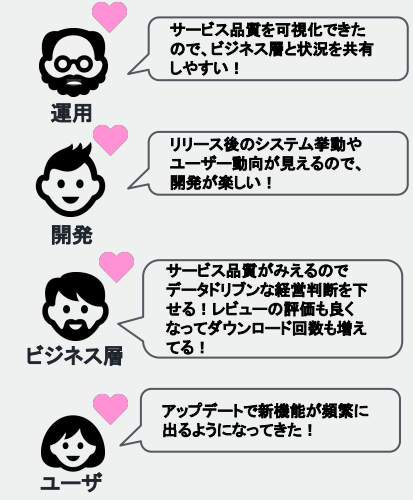
単純なメトリクスだけを監視



APMなどでアプリケーション・ユーザ動向のリアルタイムモニタリング



ビジネスKPI、サービスレベルやユーザ満足度の定量化・可視化

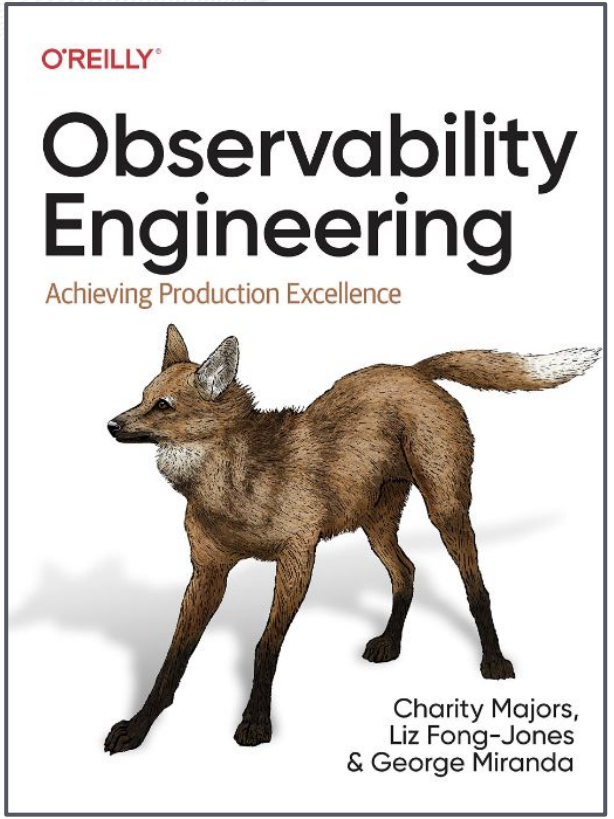


（再掲）オブザーバビリティ とは

『（意訳）**オブザーバビリティ** とは、システムの**あらゆる状態を理解し説明できる** 特性です。

デバッグの事前定義や予測なし に、状態データをアドホックに調査し、**デバッグが可能に** することが求められます。

新しいコードのデプロイ不要 で奇妙な状態も理解できる場合、オブザーバビリティが高いとされます。』



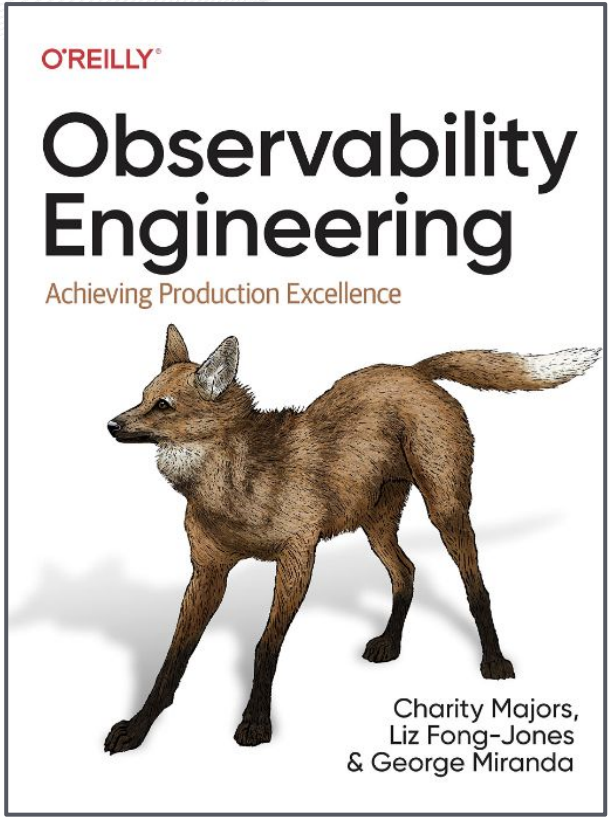
参考：Oreilly “Observability Engineering” 2022
https://www.oreilly.com/library/view/observability-engineering/9781492076438/

new relic

オブザーバビリティの目指す方向

オブザーバビリティのゴール

- **持続可能なシステムとエンジニアの生活の質**
  - オブザーバビリティが確保されたシステムは、所有しやすく、維持しやすいものになり、結果として、そのシステムを所有するエンジニアの QOLは向上します
- **顧客満足度の向上によるビジネスニーズへの対応**
  - オブザーバビリティによって、エンジニアリングチームは開発するサービスと顧客の関係をより良く理解できます。理解することによって、エンジニアは顧客のニーズを把握し、顧客に喜ばれるパフォーマンス、安定性、機能を提供できます



参考：Oreilly “Observability Engineering” 2022
https://www.oreilly.com/library/view/observability-engineering/9781492076438/

new relic

まとめ

オブザーバビリティとは

- オブザーバビリティとは、システムのあらゆる状態を理解し説明できる能力
  - システムへの**「なぜ起きたのか」「なにが起きているのか」** という疑問にどれだけ答えられるか

- 複雑になった現在のシステムにとっては重要な特性
  - いろいろなデータを収集し、関連付けて調べられるようにする
    - Log,Metrics,Trace,Dump,Profile,etc

- 組織でオブザーバビリティを高めることで、システムのユーザーと関わる人間を幸福にできる

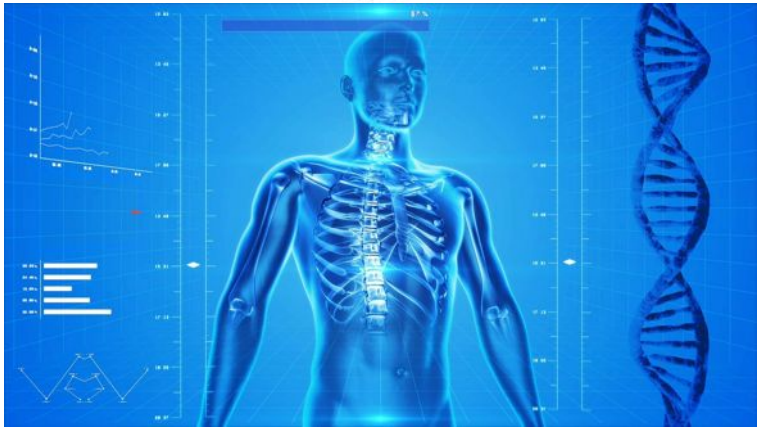
まとめ

監視とオブザーバビリティ

- ここまで話をしてきたオブザーバビリティと監視の違いを整理しましょう

監視とオブザーバビリティ

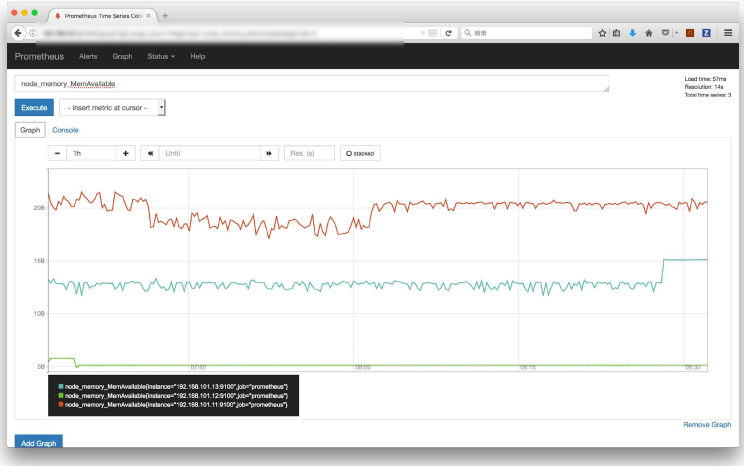
監視
システムの状態を継続的にチェックする行為



オブザーバビリティ
システムの状態を理解・説明できる能力

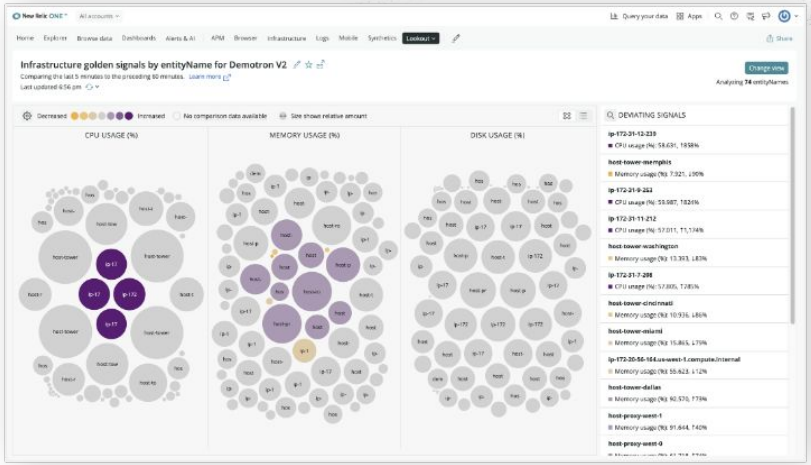
監視とオブザーバビリティ

監視



- あらかじめ見るものを決め、それに合わせてデータを収集する
- 一定のしきい値を超えたら異常とするよう設定
- 個々のデータは独立

オブザーバビリティ



- あらゆるデータを収集する
- いつもと異なる振る舞いを調査、解析
- データの関連付けと可視化を行い、システムの全容を把握できる

new relic

監視とオブザーバビリティへ

監視

- 明らかに病気の症状が出る
- 症状が出たら医者にかかる

オブザーバビリティ

- 各種センサーからのデータを保存
- データは自動で分析され可視化される
- いつもと異なる値が出たら生活に注意する
- 筋肉量の計測など、身体の向上にも使える

new relic

監視とオブザーバビリティへ

監視

オブザーバビリティ

監視とオブザーバビリティはお互いを否定するものではなく、相互に補完しあう関係

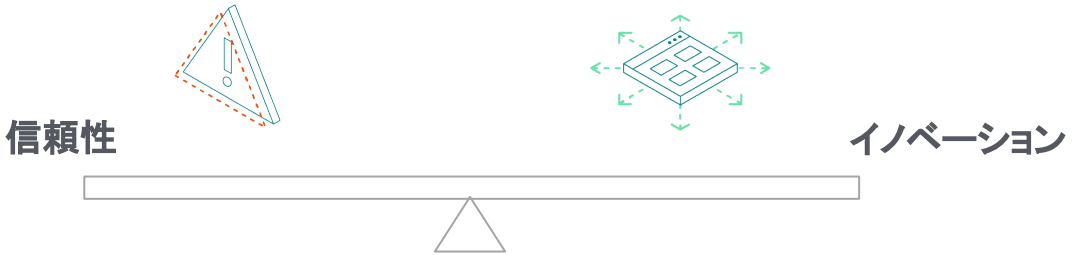
- 収集したデータを使って、異常がないかチェックするのが **監視**
- 監視で異常が見つかった場合、調査する能力が **オブザーバビリティ**

SREというプラクティス

SRE について

> “SRE チームは、サービスの可用性、レイテンシ、パフォーマンス、効率性、変更管理、モニタリング、緊急対応、キャパシティプランニングに責任を負います。”
>
> 出典: SRE サイトリライアビリティエンジニアリング(Oreilly, 2017)

常に新機能を追加しているサービスにとって、機能追加(=変更)と信頼性はトレードオフ



サイトリライアビリティエンジニアリングは、**信頼性におけるリスクとイノベーションの速度**および、**サービス運用効率性というゴール**とのバランスを取ることを目指すプラクティス

ビジネスアジリティの必要性

今後さらにDX・デジタル化が進む中で企業は**柔軟かつ変化に飛んだデジタルビジネスの展開が求められる**世界に。

■調査結果の概要

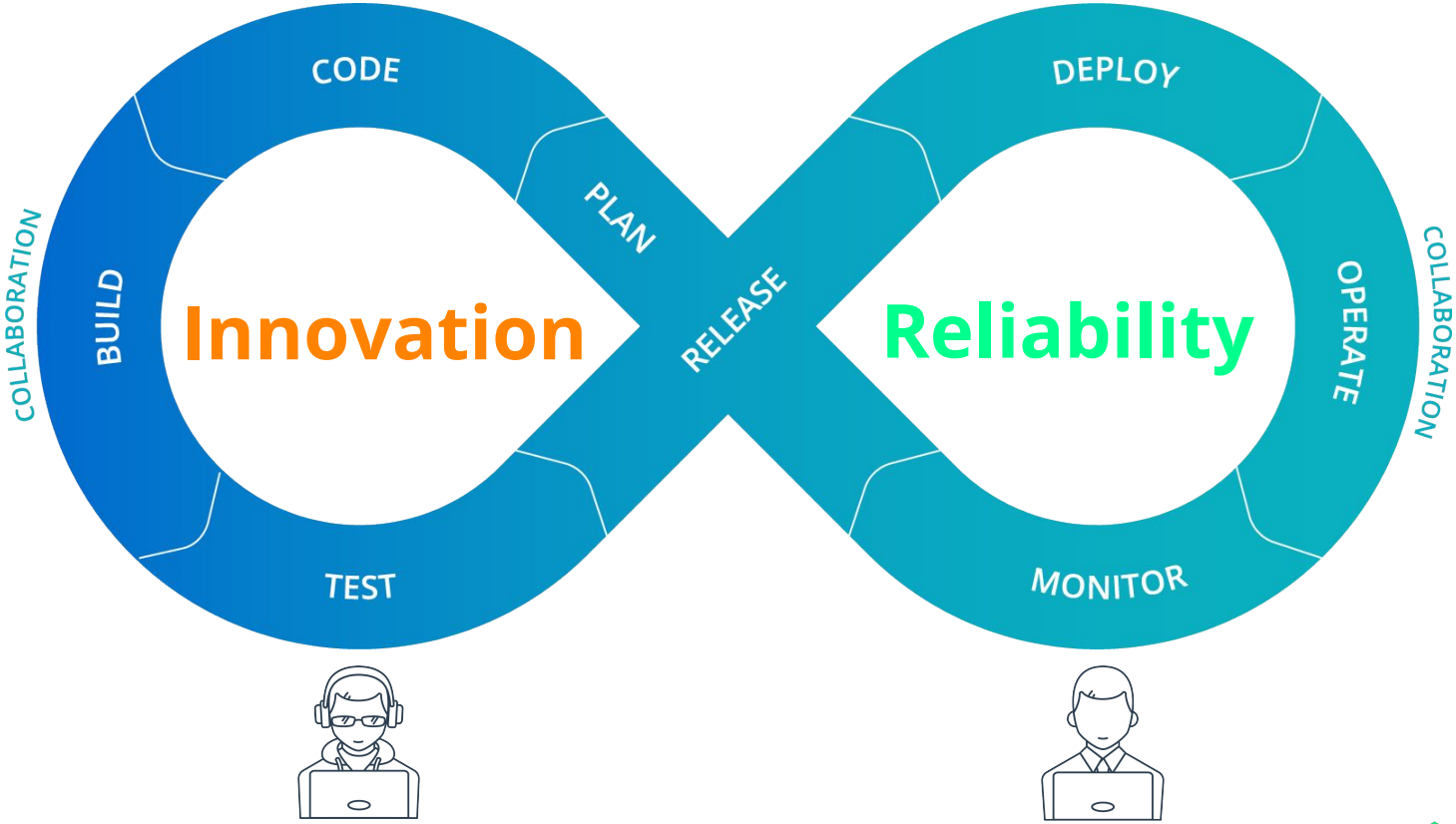
■DXの国内市場（投資金額）

	2020年度	2030年度予測	2020年度比
製造	1,620億円	5,450億円	3.4倍
流通／小売	441億円	2,455億円	5.6倍
金融	1,887億円	6,211億円	3.3倍
医療／介護	731億円	2,115億円	2.9倍
交通／運輸	2,780億円	1兆2,740億円	4.6倍
不動産	220億円	970億円	4.4倍
自治体	409億円	4,900億円	12.0倍
社会インフラ／建設／その他業界	499億円	2,078億円	4.2倍
営業・マーケティング	1,564億円	4,500億円	2.9倍
カスタマーサービス	410億円	802億円	195.6%
コミュニケーション	760億円	2,290億円	3.0倍
戦略／基盤	2,500億円	7,446億円	3.0倍
合計	1兆3,821億円	5兆1,957億円	3.8倍

引用:『2022 デジタルトランスフォーメーション市場の将来展望 市場編／ベンダー戦略編』富士キメラ総研
https://www.fcr.co.jp/pr/22025.htm

new relic

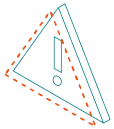
ビジネスアジリティとソフトウェアライフサイクル



new relic

“信頼性”を計測・評価する

今、
どちらを優先すべきか？



信頼性

VS

イノベーション

イノベーションを推進するか否かを判断するためには、
サービスの信頼性 の状態を計測し、その結果を**評価する必要がある**

信頼性を評価可能なものにするためには？

評価可能な信頼性: SLO (サービスレベル目標)

- SLO とは、**サービスの信頼性の目標レベル** を示すものであり、
  信頼性に関してデータを元に意思決定をする上で鍵となる目標値
- SLO を定めることによって、それに逸脱しないという**明確な基準**を持って、
  新機能のリリースを推進することができる
- SLO は運用チーム、開発チーム、プロダクトチームの**共通言語として活用** できる

チーム種別	SLO を定めるメリット
プロダクト	新機能の信頼性に対するコストをリアルタイムに知り、 優先順位付け ができる
開発	エラーバジェットの範囲内でよりスピーディーに機能をリリース することができる
運用	闇雲にアラート対応している状態から、データを元に信頼性を維持 することができ、またその取組みを他チームと共有することができる 1つ1つのリリースを気にかけるのではなく、エラーバジェットをキープしながらより信頼性を高める取り組みに専念することができる

SLO/SLI/エラーバジェット

- **SLO (サービスレベル目標：Service Level Objectives)**
  - 信頼性に関する問題を検知するためのしきい値
  - ex. Web ページへのアクセスの成功率（SLI) が 99.9%

- **SLI (サービスレベル指標：Service Level Indicator)**
  - ユーザーの観点からサービスを計測したメトリクス（SLO を満たすための計測値）
  - ex. Web ページへのアクセスの成功・失敗。応答時間が　N秒以内。

- **エラーバジェット**
  - どれだけ信頼性が損なわれてもビジネスとして許容できるかを示す指標
    許容される不具合の量(時間ベース、イベントベース)
  - 100% - SLO
  - ex. SLO が 99.9% の場合、0.1%

ユーザーの信頼性の数値化 (SLI, SLO)

重要なユーザー体験

- サービスが意図通り動いているか？
- 提供できているか？
- ユーザーにストレスなく提供できているか

ユーザー体験を評価する方法

- サイトが落ちてないか？
- 応答速度が遅くないか？

サービスレベルを測るための指標 (SLI)

- HTTP リクエストの成功割合
  (応答コードが 200 OK のもの)
- レスポンス応答が 3秒以内の割合

目標とするサービスのレベル (SLO)

- HTTP リクエスト成功率が 99% まで
- 3秒以内のアクセスが 95% まで

ユーザーが満足しているか評価する尺度

個々の SLI に対する具体的な目標値

ユーザージャーニーの定義と アーキテクチャの確認

ユーザージャーニー

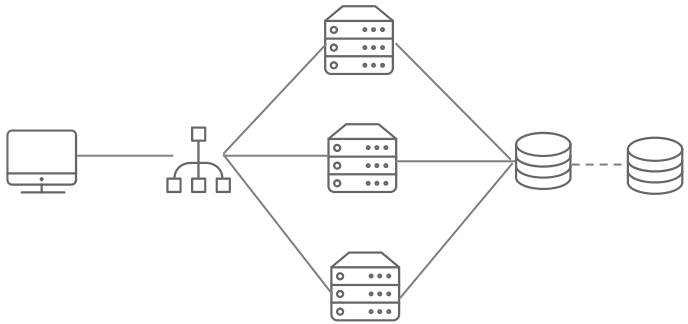
ユーザーがサービスを利用する際の一連の動作

例. New Relic のユーザージャーニー（の一例）

1. ログイン画面を開く
2. ログインし、New Relic のページに行く
3. APMのメニューを開く
4. 詳細を確認したいアプリを選ぶ
5. ・・・

アーキテクチャ

サービスを提供するシステムの構成要素



SLI の定義

大前提: サービスを利用するユーザーが期待しているようなことを指標とする

- 予測可能なものであることが望ましい (ユーザーの満足度と SLI が比例する)
- 上の条件を満たすために、Valid Event(検査する総イベント)に対し、Good Event(総イベントのうち、“よい”と定義されたイベント)の割合で示す手法が一般的

$$\text{SLI} = \frac{\text{Good Event(“良い”イベント)}}{\text{Valid Event(総イベント)}}$$

SLI の定義

SLI の候補となる項目の一覧（SLI メニュー）

サービスの種類	SLIの種類	説明
Request/ Response	可用性(Availability)	正常に応答したリクエストの比率 どのリクエストを対象にするのか、“正常”とは何かの定義が重要 ユーザージャーニーから離脱してしまうケースを想像し、正常を0か1で評価できるものを選択する
	遅延(Latency)	しきい値より早く応答したリクエストの比率 95%や99%で確認するのが一般的、ただし傾向を知るために75%も見る場合も
	品質(Quality)	特定の品質を満たしたリクエストの比率 過負荷や障害等でサービスがデグレする設計の場合、デグレしていないレスポンスを見るためのもの、“degraded”というフラグを立てたりして計測
データ処理	新鮮さ(Freshness)	ある特定の時間をしきい値にして、それより最近に更新されたデータの比率
	正確性 (Correctness)	正しい値の出力につながったデータ処理への入力レコードの比率
	カバレッジ (Coverage)	バッチ: ターゲット量以上のデータを処理したジョブの比率 ストリーム処理: ある時間ウィンドウ内に処理に成功した入力レコードの比率
ストレージ	Durability(耐久性)	書き込まれたレコードのうち、正しく読み出せるものの比率

SLI の定義

Core Web Vitals

種別	概要
LCP Largest Contentful Paint 最大視覚コンテンツの表示時間	読み込みのパフォーマンスを測定するための指標
INP Interaction to Next Paint 入力から応答までの遅延時間	ページ全体のレスポンス性に関する指標
CLS Cumulative Layout Shift 累積レイアウト シフト数	視覚的な安定性を測定するための指標

参考情報: https://web.dev/i18n/ja/vitals/

Golden Signals

モニタリングする方法や対象を計画する際に推奨される
4 つのシグナル

- レイテンシ
  - サービスがリクエストの処理にかける時間
- トラフィック（スループット）
  - サービスに対する要求の量
- エラー（可用性）
  - サービスが失敗する割合
- 飽和度（リソース利用率）
  - サービスのリソースがフル使用にどれだけ近いかを示す尺度

SLO 策定の例

参考情報:
https://landing.google.com/sre/workbook/chapters/slo-document/

SLO の定義

定めたSLIに対して、目標値を設定する

- 現状のサービスの状態が十分信頼性を満たしている場合は、
  現状の値よりも悪化しないことを目標とした値を設定

- 現状のサービスが信頼性に欠けていると判断する場合は、
  ユーザーが満足するであろう理想的な値を設定

SLO の定義

高すぎる目標は高コスト

Uptime	Daily	Weekly	Monthly	Yearly
99%	14 分 24 秒	1 時間 40 分 48 秒	7 時間 12 分	3 日 15 時間 36 分
99.9%	1 分 26 秒	10 分 5 秒	43 分 12 秒	8 時間 45 分 36 秒
99.99%	9 秒	1 分	4 分 19 秒	52 分 34 秒
99.999%	1 秒未満	6 秒	26 秒	5 分 15 秒

99.9% - 人が調査、修正、解決するのに十分な時間がある
99.99% - 自動化を実装して、停電を検出し、リダイレクトし、セルフヒーリングを実行する必要がある
99.999% - 分散システムのうち、ごく一部の機能だけが使えなくなる程度

SLI の計測

一般的にはユーザに近い方が望ましいが、システム構成やみたい観点に応じて選択する

遠

ユーザーとの距離

近

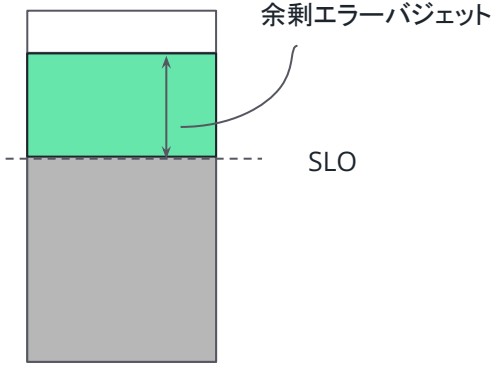
観測対象	メリット	デメリット
各種テキストログ	柔軟な情報出力が可能	ロギングロジックを編集するためのコーディングの負荷 リアルタイム性の欠如(中長期的な分析に向く)
アプリケーションパフォーマンス	収集が容易 リアルタイムに観測が可能	複雑なユーザージャーニーとの関連付けが難しい
ロードバランサからのデータ	収集が容易 (クラウドプロバイダも提供している)	ステートレスなデータしか収集できず、トラッキング不可能
外形監視	ユーザージャーニーの把握が簡単	全てのユーザー体験を把握できるわけではない
UI状況の観測	ユーザー体験を最も正確に知ることができる	不確定要素(ユーザーの利用環境等)のノイズが入る

エラーバジェットの活用

エラーバジェット → 余裕がある → 機能のリリースなど

目標にしているサービス品質を超えている

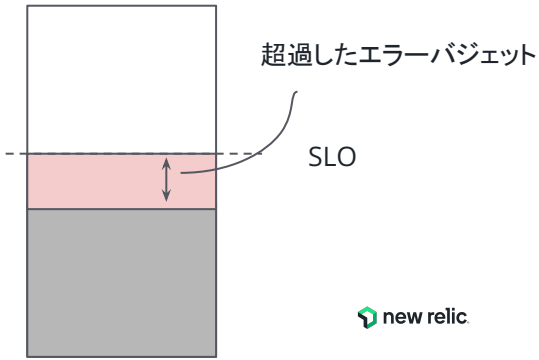
コンテンツを
リッチにしよう



エラーバジェット → ない・超過 → 信頼性回復のための修正

本来、提供したいサービスが提供できていない

不具合やエラーの
修正へ



new relic

SLI/SLOの定期的な見直し

- SLOの変更
  - 今設定しているSLOを満たしていてもユーザの満足度につながっていない場合
  - SLO違反が発生してもユーザ影響が認められない場合
- SLIの実装の変更
  - なるべくユーザの体験に近い方法に実装を変更する等

重要なのは、**ユーザーの声を可能な限り集めながら**、**それに沿った SLI/SLO を検討し続ける**こと

SLI、SLO を定義して活用するステップ

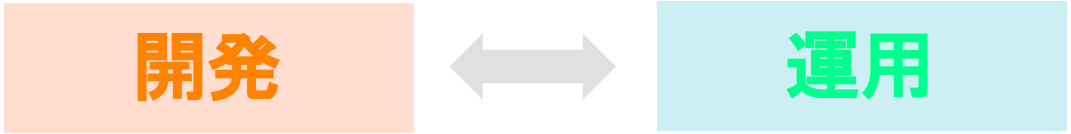
1. 対象となるサービスのユーザージャーニーを定義、システム構成を確認
2. SLIメニュー等を参考に対象サービスのSLIを定義
3. 定めたSLIに基づいてSLOを定義
4. SLIを計測して現状をリアルタイムに把握
5. エラーバジェットを活用し、信頼性を高める
6. 定期的にSLI/SLOを見直す

new relic

オブザーバビリティと SRE

ビジネスアジリティを高めるために

	変更頻度・イノベーション Velocity	信頼性 Reliability
顧客起点の期待	新しい価値が頻度高く提供	安全にいつでも利用可能
ビジネス観点	価値提供スピードの高速化	セキュアで可用性高い サービス提供
エンジニア観点	顧客価値に集中した開発	問題対応時間の最小化



new relic

信頼性とベロシティの両立を実現するために

ビジネス・顧客への価値提供を最大化するために

変更頻度・イノベーション
Velocity

信頼性
Reliability

顧客起点の期待

新しい価値が頻度高く提供

安全にいつでも利用可能

ビジネス観点

価値提供スピードの高速化

セキュアで可用性高い

エンジニア観点

顧客価値に集中した開発

問題対応時間の最小化

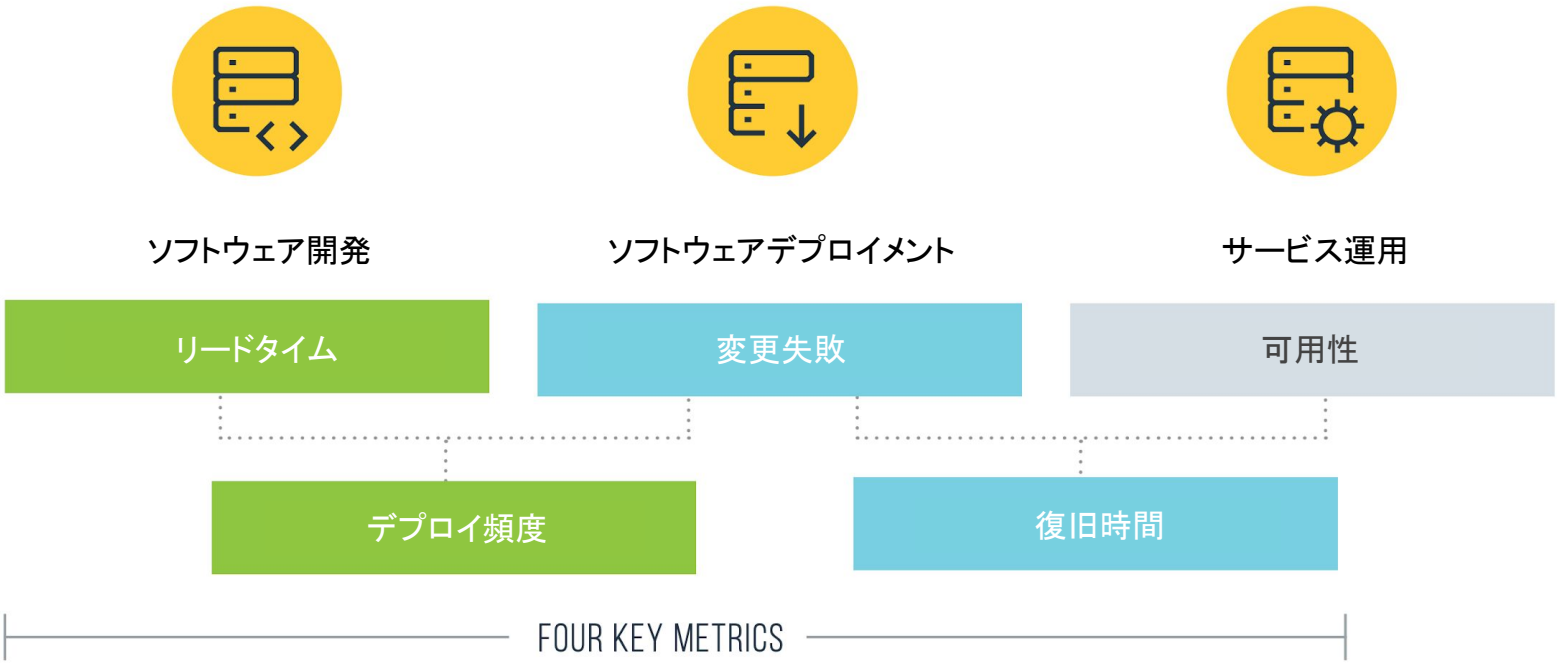
ビジネスアジリティ・

顧客価値向上に主眼

定量的な共通指標で会話

new relic

DevOps の Key Metrics



new relic

成長企業とその他の企業の差

new relic

Service Level のタイプ

SLI

Service Level Indicator

計測値

SLO

Service Level Objective

目標値

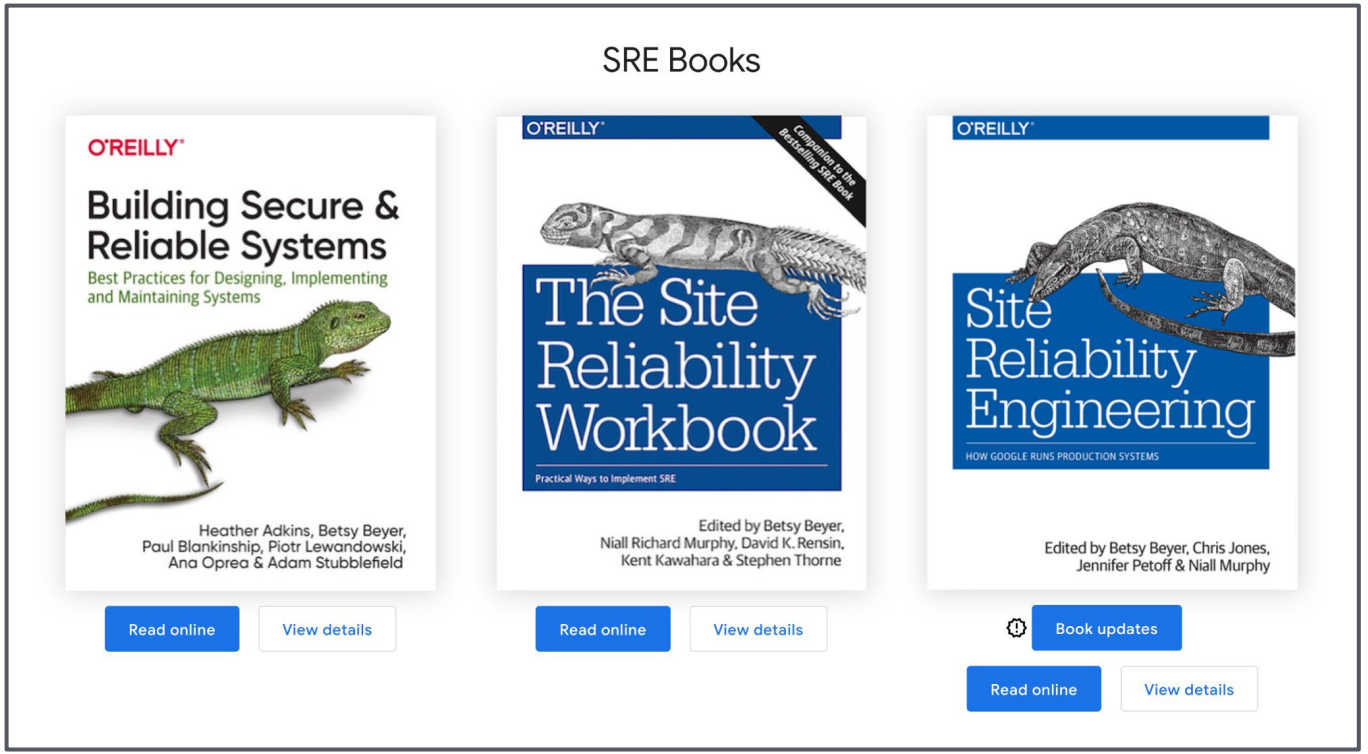
SLA

Service Level Agreement

契約値

SREがフォーカスするのは**SLIとSLO**

Google SREとは



new relic

SRE（Site Reliability Engineering）とは

SRE とは
“ソフトウェアエンジニアが
運用を設計したらどうなるのか？ ”
ということ

– Benjamin Treynor Sloss, Vice President, Engineering, Google

- “SREはシステムの信頼性に重きをおく”
- ”SREとはDevOpsに独自拡張を加えた1プラクティスと考えることもできる”
- “一般的に、SREチームは可用性、レイテンシー、パフォーマンス、効率、変更管理、監視、緊急対応、サービスのキャパシティプランニングに責任をもつ”

※”運用にかけていいリソースは活動の50%まで、**50%以上は信頼性を向上するためのコード生成を行う** こと“

new relic

サービスの信頼性とは

”ここでは、信頼性とは「[システムが]求められる機能を、定められた条件の下で、定められた期間にわたり、障害を起こす ことなく実行する確率」です。これは[Oco12]の定義に従っています。”

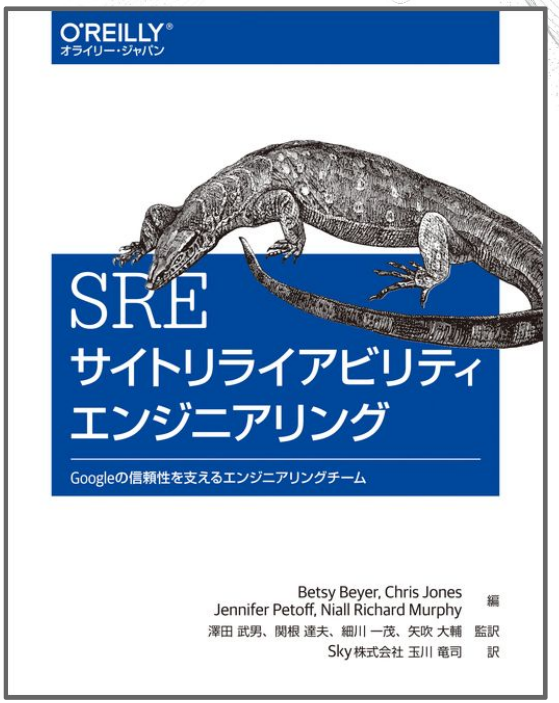
Oco12:P. O'Connor and A. Kleyner,「Practical Reliability Engineering」, 5th edition: Wiley, 2012.

サービスの信頼性とは
”障害を起こすことなく実行する確率 ”

new relic

SRE（Site Reliability Engineering）の原則とは

1. リスクを受け入れる
2. SLO サービスレベル目標
3. 苦労（トイル）をなくす
4. 分散システムの監視（モニタリング）
5. 自動化
6. リリースエンジニアリング
7. シンプル



new relic

よくあるSREの誤解

🥇 SREの目的は
信頼性の向上だ

🥉 SLOは
高い方が良い

🥈 SREチームを作れば
それで良い

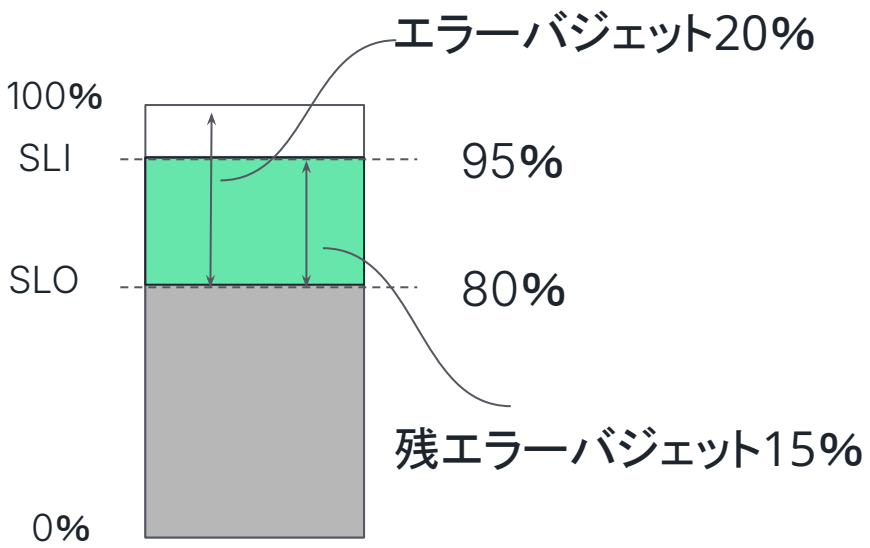
🎖️ オブザーバビリティ
インフラ監視のみ利用

new relic

🥇 SREの目的は信頼性の向上だ

🥉 SLOは高い方が良い

SLI/SLO/Error Budgetsとは？



SREにおける信頼性とは、

- 100%の信頼性は100%ありません！
- エラーバジェットの範囲内でアジリティを高め自動化やトイル削減しよう！

new relic

🥈 SREチームを作ればそれで良い

SREチームを作る時の誤解

- **インフラチームを名前だけ変えてSREチーム**発足
- SREチームを作ったから**任せておけば大丈夫**
- **SREチームがSLI/SLOを設定して運用**しようとしてもうまくいかない

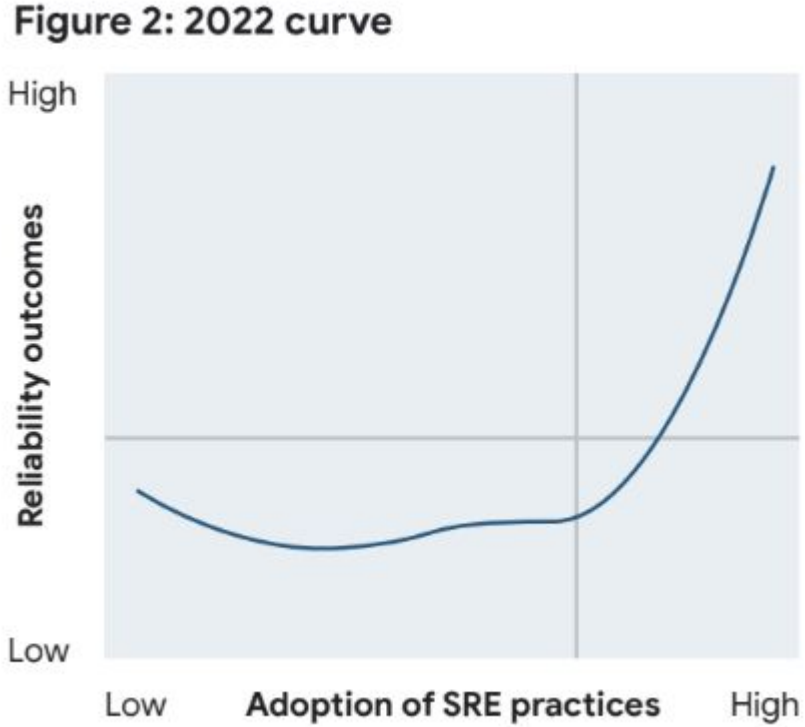
SLOの設定には各チームとの対話

サービスレベル目標（SLO）を単に設定することが目的ではなく、それを通じて組織全体でビジネスアジリティを高めるための方向性を共有し、**対話を促進することが重要**です。

あえてそのために**”SREチーム”を作らない戦略でうまくいっている企業も**増えています。

2022 DORA Report SREのJカーブ

SRE を軽く実践するだけのチーム（SRE 導入の初期段階など）にはメリットがないだけでなく、**ユーザーが経験する信頼性について後退が見られる場合** があります。しかし、こうした実践がより深く浸透してからは、変曲点に達し、信頼性エンジニアリングの能力を継続的に改善することで信頼性によるメリットが大きく向上することがわかります。



https://cloud.google.com/blog/ja/products/devops-sre/sre-in-the-2022-state-of-devops-report?hl=ja

new relic

🎖️ オブザーバビリティをインフラ監視に利用

インフラ監視のみという誤解

オブザーバビリティに
チャレンジを決めて、

・インフラ監視 SaaS化
・ログ監視 SaaS化

して満足してしまう

オブザーバビリティ

1. **問題の把握**
2. **原因の特定**
3. **原因の修正**

ができる状態をシステムと組織に獲得することが重要

SREでは

フロント・バックを含めたフルスタック かつソフトウェアライフサイクル全体 で活用するのが重要

オブザーバビリティの重要性

オブザーバビリティの役割
オブザーバビリティは、サービスレベルが低下時に、迅速に問題を検知、特定、解決する能力を提供 します。これにより、信頼性を損なうことなくビジネスアジリティと開発速度を向上させることが可能になります。

このオブザーバビリティは、SREが信頼性を守りながらソフトウェアリリース速度を向上させるために無くてはならいものとなり**日本においても 2020年頃から現在にかけてSREチームにおいて広くオブザーバビリティが普及** した

new relic

SREを組織的に実践

SREとオブザーバビリティの実践は、単なる技術的な取り組みではなく、**組織全体での共同努力が必要** です。このプロセスでは、サービスレベルのインジケーターを設定し、継続的な監視と改善を行いながら、ビジネスの目標に合わせて信頼性の基準を調整します。

SREは、**特定のチームだけの役割（ SREer）ではなく、組織全体で取り組むべき実践（SREing）**です。サービスレベルを共通の目標として設定し、開発、運用、ビジネスチームが協力して、これを達成するためのコミュニケーションと合意形成が必要です。

まとめ

new relic

監視とオブザーバビリティ

SRE

イノベーションと信頼性のバランスを取るためのエンジニアリングプラクティス

オブザーバビリティ

システム状態の観測性を高めること、もしくは観測性を高めたシステムの状態・性質

監視

観測されたデータからポイントを絞って異常を検出。
行動可能な通知を行う

監視とオブザーバビリティ

SRE

イノベーションと信頼性のバランスを取るためのエンジニアリングプラクティス

オブザーバビリティ

システム状態の観測性を高めること、もしくは観測性を高めたシステムの状態・性質

監視

観測されたデータからポイントを絞って異常を検出。
行動可能な通知を行う

- 監視とオブザーバビリティは共存し、相互に補完しあう関係
- SREはソフトウェアエンジニアベースの運用プラクティス
- 100%の可用性は求めず、システムの信頼性をベースとした運用方法
- エンジニアだけではなく、ビジネスと連携しながら運用していく

監視とオブザーバビリティ

SRE

イノベーションと信頼性のバランスを取るためのエンジニアリングプラクティス

オブザーバビリティ

システム状態の観測性を高めること、もしくは観測性を高めたシステムの状態・性質

監視

観測されたデータからポイントを絞って異常を検出。
行動可能な通知を行う

- 監視とオブザーバビリティは共存し、相互に補完しあう関係
- SREはソフトウェアエンジニアベースの運用プラクティス
- 100%の可用性は求めず、システムの信頼性をベースとした運用方法
- エンジニアだけではなく、ビジネスと連携しながら運用していく

ユーザーから求められるシステムとして
あるべき姿を考えながら観測・監視する

new relic

参考文献

- 入門監視
- オブザーバビリティ・エンジニアリング
- SRE サイトリライアビリティエンジニアリング
- SREの探求
- サイトリライアビリティワークブック
- SREをはじめよう
- SLO サービスレベル目標
- LeanとDevOpsの科学
- システム運用アンチパターン
- SREの知識地図

new relic

new relic

Thank you.

Satoshi itatani
sitatani@newrelic.com