Google Cloud
Next
Tokyo

AI×運用

AIを活用した
障害対応の効率化実現方法

大田原 慶道

(Yoshimichi Ohtahara)

JIG-SAW株式会社
アカウントマネジメント本部
執行役員 本部長

クラウドアーキテクトとしてお客様の
クラウド活用における技術的な課題の
解決を支援する業務に従事

本セッションのゴール

“最新のシステム運用トレンド ”と

“AIを活用した障害対応事例 ”から

自社運用への AI活用を考える

（きっかけになれば ）

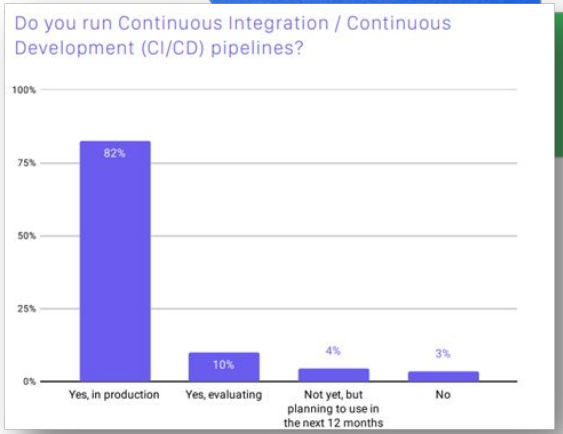
アジェンダ

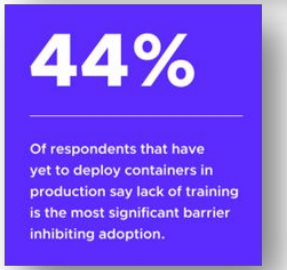
01 クラウドネイティブな環境におけるシステム監視の現在
02 AI活用で解決できる課題
03 AI導入事例
04 AIを運用で活用する際の注意点
05 まとめ

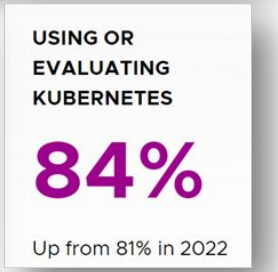
クラウドネイティブな環境におけるシステム監視の現在

最近のトレンドではコンテナ技術の活用が主流

- CNCFの2020の調査では、82%の回答者が「CI/CDパイプラインを使用している」と回答。
- 2022年の調査では、44%の回答者が「ほぼ全てのアプリケーションでコンテナ技術を活用している」と回答。
- 2023年調査においても、84 % の組織が「Kubernetes を使用している」と回答。







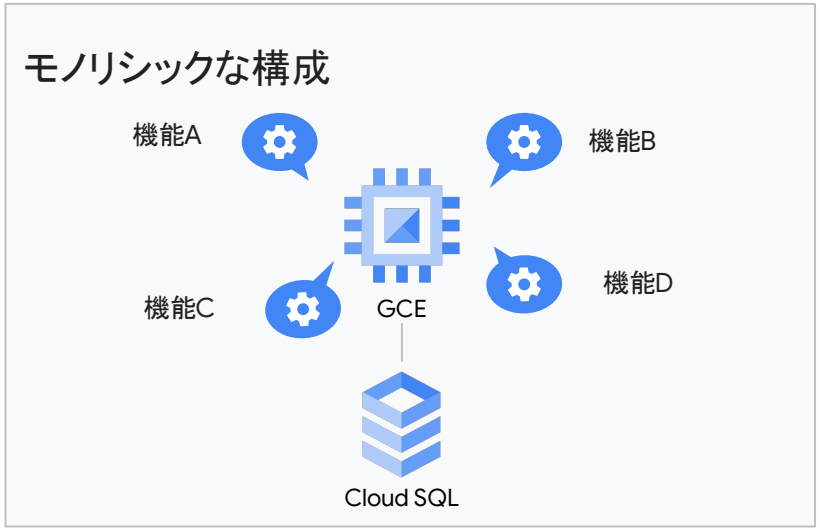
出典:
CNCF SURVEY 2020 (https://www.cncf.io/wp-content/uploads/2020/11/CNCF_Survey_Report_2020.pdf)
CNCF SURVEY 2022 (https://www.cncf.io/reports/cncf-annual-survey-2022/)
CNCF SURVEY 2023 (https://www.cncf.io/reports/cncf-annual-survey-2023/)

システム監視の役割の変容

旧来のモノリシックな構成では単一のコンポーネント毎に監視を行い、　障害時は再起動等の対応を行い復旧させた後、根本原因の為のログの調査等を行うのが一般的な監視オペレーションとなります。

Before 従来の監視

- 異常発生後の検知
- 単一のコンポーネントを監視
- 低いトレーサビリティ

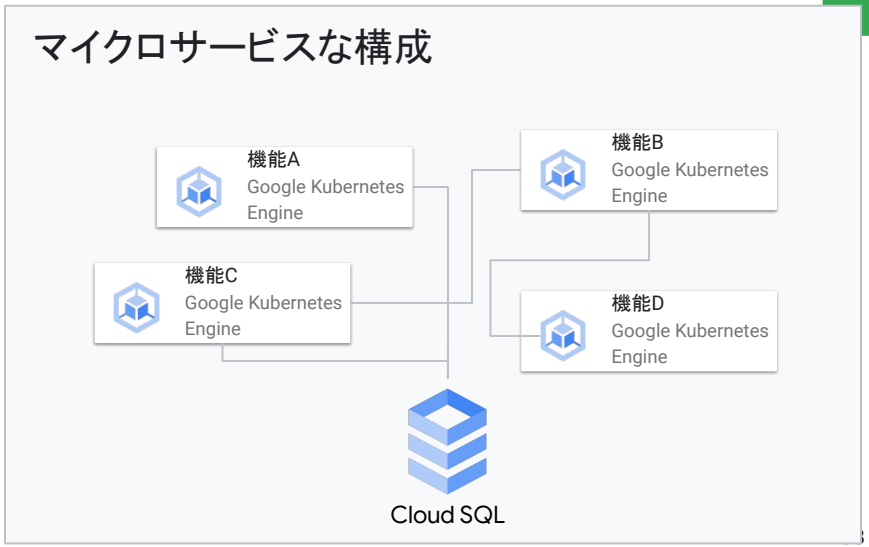


システム監視の役割の変容

クラウドネイティブでマイクロサービス化された構成では複雑な構成、 依存関係のあるマイクロサービスの健康状態を監視し、復旧に必要な調査が容易に行える高いオブザーバビリティが必要となります。

After クラウドネイティブな監視

- 異常予兆から検知
- システム全体の健康状態の監視
- 高いトレーサビリティ



AI活用で解決できる課題

クラウドネイティブな構成における課題

マイクロサービスアーキテクチャによる障害原因の複雑化

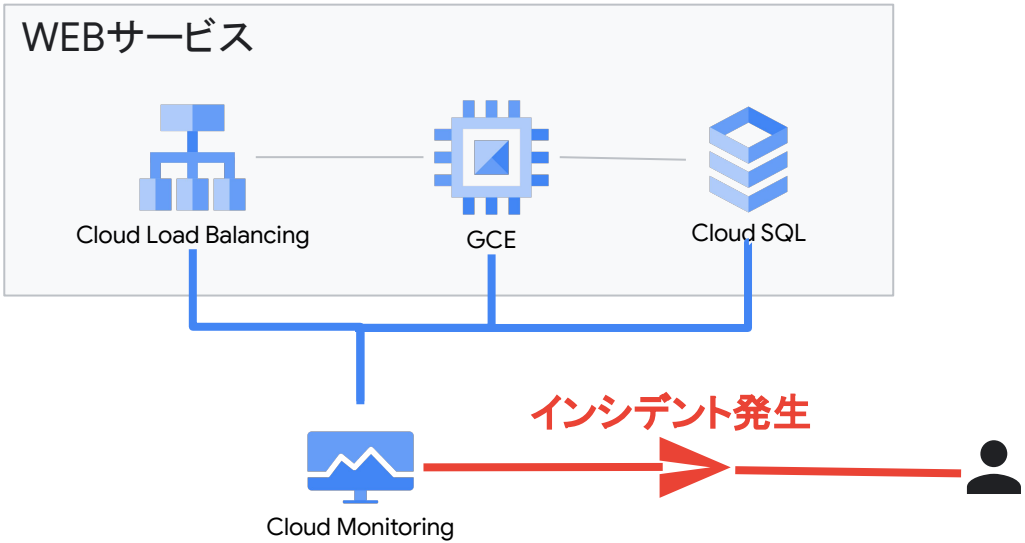
モノリシックな構成からマイクロサービス化された構成へ変わっていくなかで冗長性や可用性は高まってきている。その反面システムは、より複雑な構成となってきている。

求められる対応スキルの高度化

障害発生時の原因の特定や復旧プロセスは単純なマニュアル化が難しくなってきており人のスキルに依存した対応と原因特定や改善の為に高いオブザーバビリティ(可観測性)を求められる場面が増えてきている。

クラウドネイティブな構成における課題

Before：従来の(モノリシック)障害対応事例

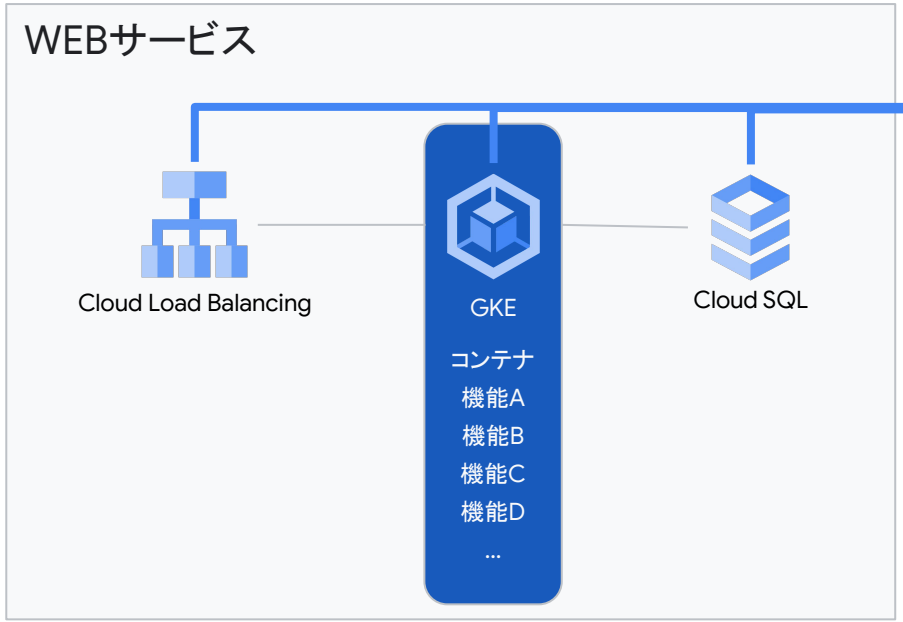


- ランブックの対応
  - GCE再起動
  - Cloud SQL再起動
- ログ調査(復旧優先で事後調査)
  - システムログ
  - アプリケーションログ
  - SQLログ

概ね再起動で復旧する
シンプルな構成

クラウドネイティブな構成における課題

After：マイクロサービス化された複雑な障害対応事例



- 障害原因の特定
  - 問題のコンテナ(pod)の特定
  - 障害原因から復旧対応策の検討
- ログ調査(復旧の為の調査)
  - システムログ
  - アプリケーションログ
  - SQLログ

原因特定の難しさと
スキル依存が高い構成

クラウドネイティブな構成における課題

オブザーバビリティ (可観測性)を実現する

マイクロサービスアーキテクチャで複雑な構成であってもシステム全体の健康状態をリアルタイムで把握でき、異常が発生した場合でも詳細な調査が可能なトレーサビリティを実現することが重要です。

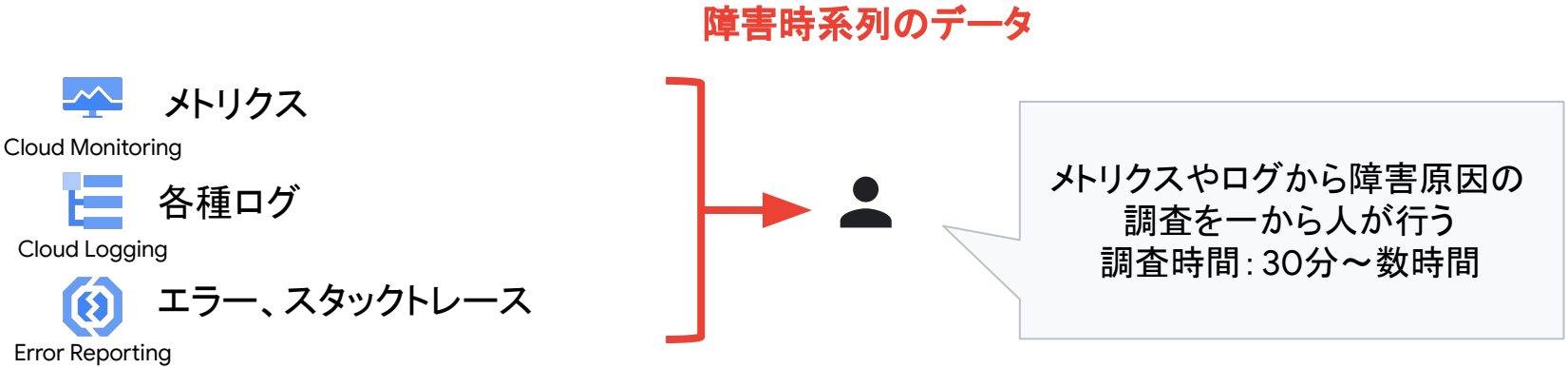
※ Google Cloudで実現するオブザーバビリティのサービス

	Cloud Monitoring: 収集された指標を使用して、正常性とパフォーマンスのモニタリング、傾向や問題の特定、動作変更の通知を行います。
	Cloud Logging: 収集されたログを使用して、アプリケーションのデバッグとトラブルシューティングを行い、アプリケーションに関する分析情報を取得します。
	Error Reporting: 実行中のクラウド サービスで発生したエラーを表示して分析します。
	Cloud Profiler: アプリケーションの CPU とメモリの使用状況を分析して、パフォーマンスを向上させる機会を特定できます。
	Cloud Trace: デバッグとトラブルシューティングの際に、アプリケーションリクエストのフローとレイテンシを表示して分析します。

AI活用で解決できる課題

課題：障害原因調査に時間がかかる

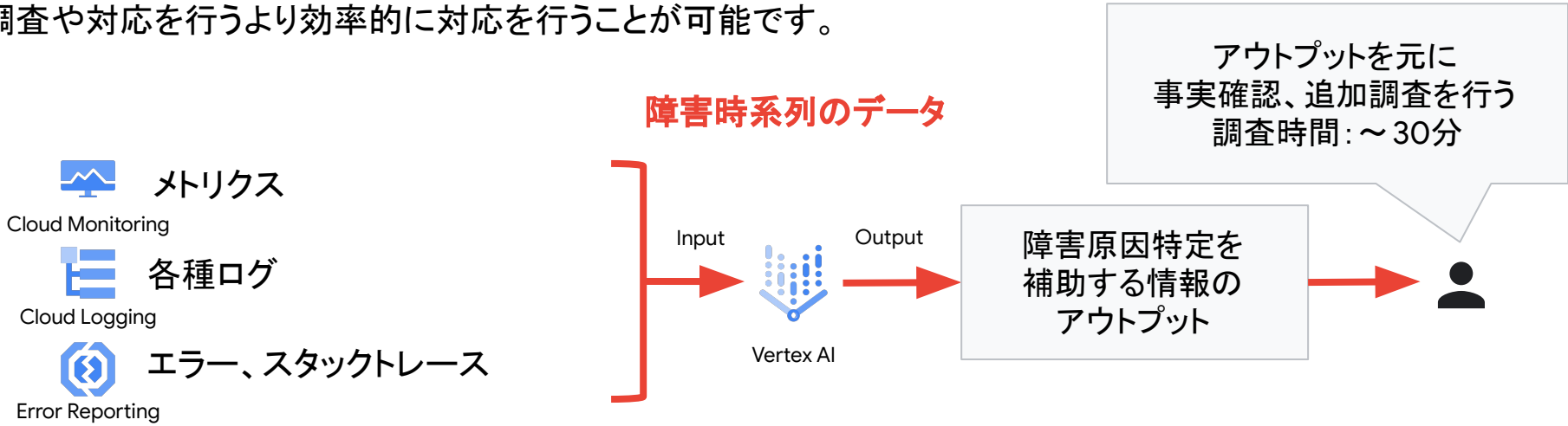
従来の方法では障害発生時に障害時系列のメトリクス、ログやイベントの内容を元に　人が一から調査を実施しています。



AI活用で解決できる課題

解決：障害原因特定を補助し調査時間を短縮

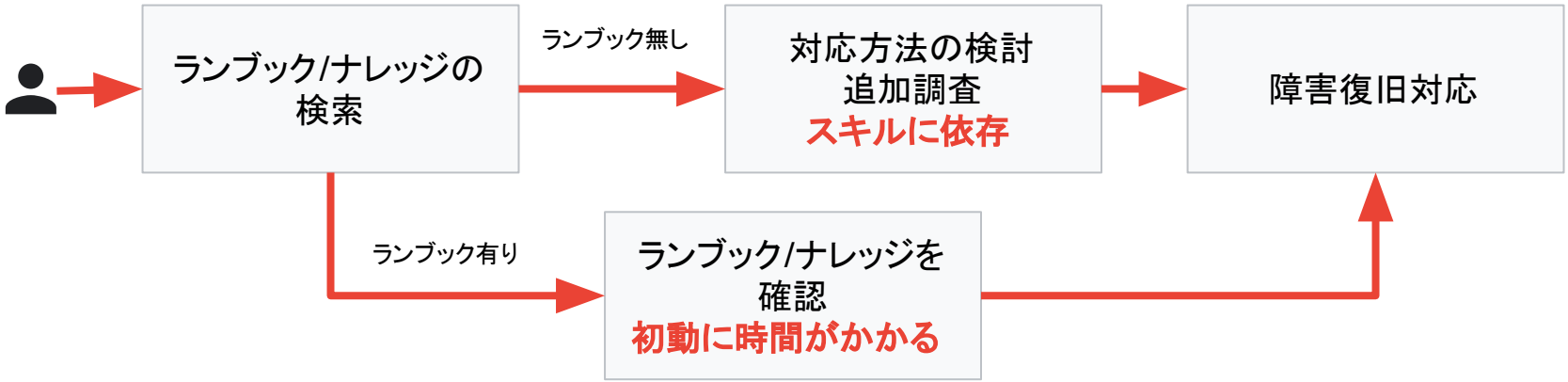
ログや取得可能なメトリクスより、障害の発生原因特定や考えられる事象をAIにインプットし
AIからのアウトプットによる情報を障害原因特定の補助的な情報として活用することで、　　一から人が
調査や対応を行うより効率的に対応を行うことが可能です。



AI活用で解決できる課題

課題:ランブック /ナレッジの結び付けに時間がかかる
マニュアル化できないスキル依存の対応

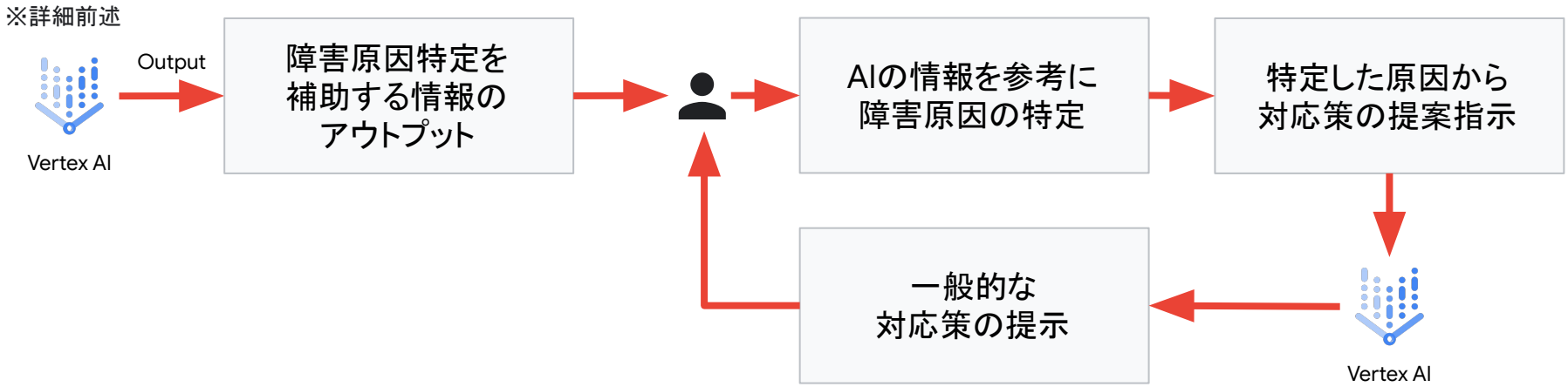
従来ではランブックやナレッジを検索しオペレーションを開始するのが一般的ですが、ランブックやナレッジを結び付けるのに時間がかかったり、ランブックやナレッジがない場合は対応者のスキルに依存した対応が必要となる場合があります。



AI活用で解決できる課題

解決：対応策を提案しランブックの自動提案と対応品質の均一化

障害原因の取り除きや修復対応の為の対応策をAIに提案を行わせることで
俗人化しない形でスキルの平坦化とインシデント対応の品質を向上します。



AI導入事例

AI導入事例

Hallelujahの運用での活用事例

JIG-SAWでは自社のノウハウでRAGを実行したGenAI(Hallelujah)で
障害原因特定と対応策提示を実際のオペレーションで活用しています。

解決した課題

- 障害原因特定までにかかっていた時間の短縮
- スキル依存と属人化の軽減
- 障害復旧までの対応時間の短縮

活用に向けたポイント

- AIからのアウトプットはあくまで補足の情報として取り扱う
- アウトプットされた内容は精査を行い更なるノウハウとしてナレッジを蓄積
- 精度に課題があってもRAGを行いAI自体を強化していく

AI導入事例

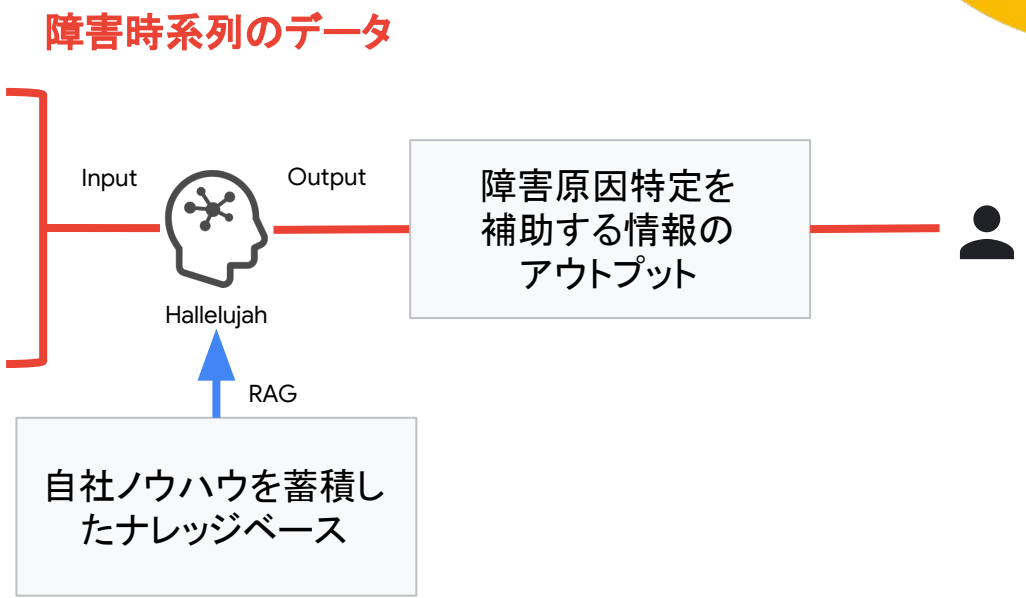
Hallelujahの運用での活用事例　-　運用イメージ

自社ナレッジベースの情報で拡張したGenAIで障害対応をサポートしています。

メトリクス
Cloud Monitoring

各種ログ
Cloud Logging

エラー、スタックトレース
Error Reporting



AI導入事例

Hallelujahの運用での活用事例　-　運用イメージ

Tomcatのプロセスダウンを検知しHallelujahで障害原因特定を実施した例

Tomcatのプロセスがダウンした理由は、メモリ不足（ Out of Memory: OOM）によるものと考えられます。以下のログからその根拠を説明します：

```
Aug  1 01:00:00 localhost kernel: Out of memory: Kill process 19534 (java) score 468 or sacrifice child

Aug  1 01:00:00 localhost kernel: Killed process 19534 (java), UID 1001, total-vm:7790724kB, anon-rss:4108910kB, file-rss:6822kB, shmem-rss:0kB
```

詳細な説明

1. **メモリ不足の発生** ：
   - ログに「Out of memory」と記載されていることから、システム全体のメモリが不足していたことがわかります。
2. **プロセスの強制終了** ：
   - 「Kill process 17143 (java)」とあるように、システムがメモリを確保するために Javaプロセス（Tomcat）を強制終了しました。
3. **メモリ使用量** ：
   - total-vm:7790724kB 、anon-rss:4108910kB などの情報から、Javaプロセスが大量のメモリを使用していたことがわかります。

仮説

Tomcatが動作している Javaプロセスが大量のメモリを消費し、システム全体のメモリが不足したため、カーネルがこのプロセスを強制終了したと考えられます。これにより、Tomcatのプロセスがダウンしました。

対策

- **メモリの増設** ：サーバーの物理メモリを増やす。
- **Javaヒープサイズの調整** ：Javaのヒープサイズを適切に設定し、メモリ使用量を管理する。
- **メモリリークのチェック** ：アプリケーションコードにメモリリークがないか確認する。

AI導入事例

puzzleでのAI導入事例

自社開発のモニタリングツール「puzzle」で回帰分析や動的時間伸縮法を使ったAI分析でリソース枯渇の予測や予兆、異常の検知を実現。

解決した課題

- 障害発生前に予兆の検知が可能に
- 事前に将来的なリソース使用状況を把握可能に

活用に向けたポイント

- 予兆や予測を活用し障害発生前から対策を行うことで障害を未然に防ぐ
- 将来的な予測から計画的なメンテナンスや拡張を実施する

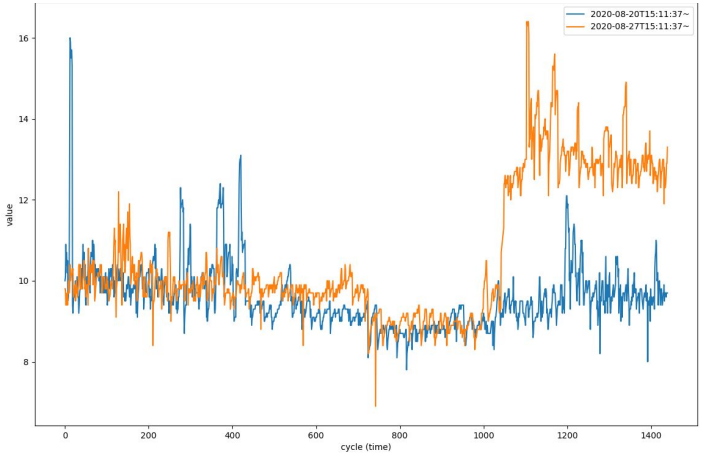
AI導入事例

puzzleでのAI導入事例

リソース枯渇予測やリソースの消費傾向から外れた外れ値を検出する異常検知を実装しています。従来ではリソース枯渇に近い状態から検知したり、過去の実績と違う動きの検知は不可で有ったが事前の予兆と予測が可能となりました。

回帰分析を用いたリソース枯渇予測

動的時間伸縮法を用いたリソース消費傾向における異常検知



AIを運用で活用する際の注意点

AIを運用で活用する際の注意点1

”人が遂行可能”であるオペレーションを整備する

AIがなくても人が対応可能なオペレーションとしてフローやルールを整備し
その上でフロー、ルールに則ってAIが機能するように導入を進めます。

- 最新化された構成情報を整備します。
- 想定されるインシデントを可能な限り洗い出します。
- 想定されるインシデントの対応マニュアルを作成します。

AIを運用で活用する際の注意点2

オブザーバビリティ (可観測性)を高める

障害原因を特定するのにあたって、特定が可能な情報を
正しく取得できていることが重要になります。

- 各メトリクスが正しく取得できているか整備します。
- 各ログが正しく取得できているかを整備します。
- 各イベントが正しく取得できているかを整備します。

まとめ

まとめ

AI活用で解決できる運用課題

- ✓ 複雑なシステムの障害原因特定の補助を効率的に実現
- ✓ 修復や障害対応の対応策の提案によりスキルの均一化や品質の向上

AIを運用で活用する際の注意点

- ✓ 人がオペレーションを実現できないものはAIでも実現できないのでちゃんと整備する必要がある
- ✓ 参照できるデータ(ログやメトリクス)がちゃんと出力されていないと障害原因特定ができない

まとめ

コンテナ技術やサーバレスが主流となり

クラウドネイティブな運用が求められる昨今、

「 高いオブザーバビリティ 」×「 AI 」で

より効率的で品質の高い運用を実現できます。


Thank you