運用現場のリアルから紐解く の現在地

「検知するAI」から「対処するAI」へ

Yoshimichi Ohtahara
JIG-SAW株式会社 Market Development Group / Specialist

JIG-SAW

今日話すこと

01 **AIOps を取り巻く市場動向と現場課題**
なぜ今「自律型AI運用」なのか

02 **自律型AI運用を実現するための技術的挑戦**
AI に任せるために越えるべき5つの壁

03 **実践アーキテクチャ事例**
自律型AI運用基盤「Lorel.ai」の設計

04 **まとめ & メッセージ**

今日考えたいこと

- なぜ、障害の復旧には**時間がかかる**のか？
- AI に、**どこまで運用を任せられる**のか？
- そのとき、**人の仕事はどう変わる**のか？

SECTION 1

AIOps を取り巻く市場動向と現場課題

なぜ今「自律型AI運用」なのか

現場で起きていること

※1

エンジニアの労働時間のうち
障害対応・緊急対応に費やされている
割合

※2

監視・運用の主な懸念として
「アラートやログ、メトリクスなどのデータが多すぎて、重要な情報が埋もれる」
を挙げた割合

※3

2030年に国内で**不足するIT人材**の試算
— 採用では埋まらない規模

異常の検知は自動化できた。しかし、原因の判断と対処は、いまも人の手に委ねられている。
システムは増える一方で、人は増えない。この状況が、運用の自律化を「あれば便利なもの」から「必要不可欠なもの」へと変えた

※1　New Relic「2025 Observability Forecast」（世界のIT・エンジニアリング責任者 1,700名）— エンジニアが障害対応に費やす時間の割合
※2　Grafana Labs「Observability Survey 2025」（世界の監視・オブザーバビリティ実務者 1,255回答）— 最大の懸念として「More noise; signal-to-noise challenge」を選んだ割合
※3　経済産業省「IT人材需給に関する調査」（2019年）— 2030年の需給ギャップ、高位シナリオの試算値（中位 約45万人・低位 約16万人）

SRE の一日は、何に消えているか

トイルの正体

- **アラート疲れ** — 鳴り続けるアラートの大半は対応不要。本当に危険なものを人が選別し続けている
- **情報の収集** — ダッシュボード・ログ・デプロイ履歴・Slack を行き来して「何が変わったのか」を探す。ここに最も時間を取られる
- **定型作業の繰り返し** — 再起動・スケール・ロールバック。やるべきことは分かっているのに、毎回人が手で行っている
- **深夜・休日のオンコール** — 原因が分からないまま調査が2時間続き、翌日の生産性まで失われる

ある深夜のインシデント — よくある光景

DatadogからDBの**CPU使用率95%超**のアラート。担当者が起こされる
メトリクス、APMトレース、ログ、デプロイ履歴を確認し調査開始
APMトレースで遅いクエリは見つかるが、**根本原因がわからない**
DB負荷上昇の影響で**夜間バッチが停止**。業務影響が発生
外部アクセス増加やアプリ変更の影響を疑い、ログやデプロイ履歴を調査
先週追加されたバッチの非効率クエリが原因と特定
インデックス追加とバッチ再実行を実施
復旧を確認。暫定的な対応結果をSlackへ共有
寝不足のまま出社し、対応経緯と原因を整理してレポートを作成

調査から復旧まで2時間以上。そのほとんどが「原因を探す時間」と「疑って外れた時間」だった。

AIOps の三世代 — 何が変わったのか

GEN 1 / 2010〜

ルール・閾値による自動化

- 静的な閾値と、手順書（ランブック）の自動実行
- 想定済みの障害には強い
- 未知の事象や複合的な障害には弱い

GEN 2 / 2016〜

機械学習による検知と相関分析

- 異常検知、アラートの集約、ノイズの削減
- Gartner が「AIOps」と呼び始めた世代
- AI がやるのは**「異常を知らせる」**まで。原因の判断と対処は人が行う

GEN 3 / 2024〜

自律型 — 推論し、行動する

- LLM が状況を読み取り、仮説を立て、ツールを使って動く
- AI が**原因を判断し、対処まで行う**。なぜそうしたかの根拠も示す
- 論点が「精度」から**「どこまで任せるか」**に移った

技術は揃った。残る課題は「信頼」

AI に運用を任せるために必要な技術が、この数年で揃った

① AI モデル

LLM がログやメトリクスを読み、**原因の仮説を立てられる**ようになった

② ツールとの接続

MCP（Model Context Protocol）などの標準により、AI が監視ツールやクラウドを**直接操作できる**ようになった

③ 運用データ

OpenTelemetry の普及で、メトリクス・ログ・トレースを**まとめて扱える**ようになった

残る課題 — 信頼

AI はもっともらしく間違えることがある。そんな AI に、本番環境の操作を任せてよいのか。

技術的に「作れる」ことと、安心して「任せられる」ことの間には、まだ大きな差がある

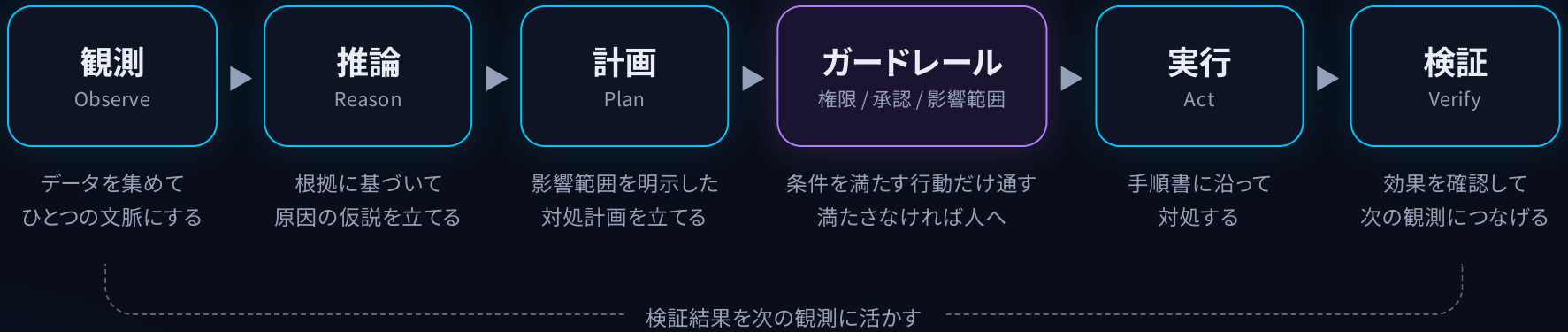
SECTION 2

自律型AI運用を実現するための技術的挑戦

AI に任せるために越えるべき5つの壁

AI による障害対応の流れ

自律型AI運用とは — 観測データから次に打つべき手を自律的に導き出し、安全な範囲で実行し、結果を検証して学習する運用システム



5つの壁

AI モデルが賢くなっても、

権限の制御や実行の記録といった、AI の周りの仕組みづくりが必要になる

文脈の統合
AIは「何が変わったか」を知らない

推論の信頼性
もっともらしい ≠ 正しい

安全な実行
権限を渡すとはどういうことか

耐久性と可観測性
エージェントも運用対象

評価とコスト
「良くなった」をどう証明するか

壁①　文脈の統合 — AIは「何が変わったか」を知らない

何が難しいか	どう乗り越えるか
監視ツールごとにデータの形式が異なる。しかも最も手がかりになる「**直前に誰が何を変えたか**」は、監視データの外にある	**データを共通の形にそろえる** — 監視データとデプロイ履歴を「イベント・対象・変更」という共通の形に変換する。地味だが、ここを後回しにするとAIの判断材料がそろわない
人間のSREは、**5〜6個の画面**を見比べながら、「どのサービスが何に依存しているか」を頭の中で補って判断している	**システムの依存関係を把握させる** — 「このアラートが出たサービスは、どのDBに依存しているか」を構成情報から引けるようにする。これがないと、AIは原因の見当がつかない
すべてを渡すとコンテキストもコストも膨れ上がり、絞り込むと**見落とし**が起きる	**必要なデータをAIに取りに行かせる** — すべてのデータを最初から渡すのではなく、AIが必要に応じてツールで取りに行く形にする。コストを抑えつつ、見落としも防げる

壁②　推論の信頼性 — もっともらしい ≠ 正しい

何が難しいか	どう乗り越えるか
LLM は**自信たっぷりに間違える**。SRE が知りたいのは「なぜそう判断したか」であり、答えだけを出す AI は信用されない	**根拠を必須にする** — 原因の仮説には必ず「どのログの何行目か」「どのメトリクスのどの時刻か」という根拠を添えさせる。根拠を示せない仮説は出させない
相関と因果を混同する — CPU 使用率の上昇と遅いクエリ、どちらが原因でどちらが結果か	**仮説を検証してから進ませる** —「原因は X だ」で終わらせず、「X が原因なら、このメトリクスはこう動いているはずだ」と確認させてから次に進ませる
初めて見る事象や、過去の類似事例と**原因が違う**場合、AI は確かな判断ができない	**分からないときは人に引き継がせる** — 確信が持てないときは「分からない」と言わせて人に渡す。無理に答えを出す AI より、引き際を知っている AI のほうが、安心して任せられる

壁③　安全な実行 — 権限を渡すとはどういうことか

自律化の段階 — 「任せる・任せない」の二択ではない

L0 観察	L1 提案	L2 承認付き実行	L3 条件付き自律実行	L4 自律実行
データを読んで分析し、結果を通知するだけ	対処案を提示する。実行するのは人	人が承認した対処を実行する	事前に定めた範囲内なら自動で実行し、事後に報告する	未知の事象にも自ら対処し、例外だけ人に引き継ぐ

何が難しいか	どう乗り越えるか
AI に広い権限を渡すと、削除のような**想定外の操作**まで実行できてしまう	**最小権限 × 操作の分類** — 「スケールアップは許可、削除は禁止」のように操作を分類し、種類ごとに権限を与える。AI に何でも実行できるシェルは渡さない
同じ操作でも、対象や規模によって**影響の大きさ**が違う	**影響範囲の事前見積もり** — 実行前に「何に、どれだけ影響するか」を見積もり、閾値を超える場合は自動的に L2（承認付き）に切り替える
AI が誤作動したとき、その**後始末**を AI 自身に任せられない	**フェイルセーフと巻き戻し** — 想定外の動きをしたときに安全な状態へ戻す仕組みは、AI ではなくシステム側に持たせる

壁④　耐久性と可観測性 — エージェントも運用対象

何が難しいか	どう乗り越えるか
インシデント対応は**数分から数時間**続く。途中で止まったときに安易に再実行すると、スケールアップが二重にかかる	**途中から再開でき、二重に実行しない** — 各ステップを記録し、止まっても中断した地点から続けられるようにする。副作用のある操作は、二重に実行されない仕組みにしておく
「**AI が何をしたのか分からない**」状態は、監査やポストモーテムで致命的になる	**AI の行動をすべて記録する** — 何を見て、どう判断し、どのツールを呼び、何が返ってきたか。すべてを記録して後から追えるようにする。AI 自身も、監視の対象にする
障害のときこそ使いたいのに、その障害の影響で**AI の基盤まで不調**になる	**監視対象のシステムと切り離す** — 監視対象の障害が AI の基盤に波及しないよう、配置と依存関係を分けておく

壁⑤　評価とコスト ― 「良くなった」をどう証明するか

何が難しいか	どう乗り越えるか
本番の障害は**再現できない**ため、テストデータが作れない	**過去インシデントの再生** ― 実際のアラート・ログ・対応記録を評価セットにして、「人と同じ結論に至るか」をオフラインで測る
同じ入力でも出力が変わるため、従来の回帰テストが成立しない	**シャドーモードでの比較** ― L0（観察）で人と並走させ、判断を突き合わせる。これが L1、L2 へ自律レベルを上げる根拠になる
「復旧時間が短くなった」のは AI の成果なのか、たまたまその月の障害が簡単だっただけなのか、区別できない	**比較対象の確保** ― 導入前後の期間比較ではなく、同じ種類のインシデントで人と AI の所要時間を比べる
推論のたびに課金される。アラートが殺到すると、**コストも同じように膨らむ**	**コストの階層化** ― まずルールでふるいにかけ、次に軽量モデルで分類し、高性能なモデルは本当に必要なときだけ使う

第2章のまとめ — 5つの壁と、どう乗り越えるか

壁	何が難しいか	どう乗り越えるか
① 文脈の統合	AI は「何が変わったか」を知らない	データを共通の形にそろえ、システムの依存関係を把握させ、必要なデータは AI に取りに行かせる
② 推論の信頼性	もっともらしい ≠ 正しい	根拠を必須にし、仮説を検証してから進ませ、分からないときは人に引き継がせる
③ 安全な実行	権限をどこまで渡すか	操作を分類して権限を与え、影響範囲を事前に見積もり、元に戻す仕組みはシステム側に持たせる
④ 耐久性と可観測性	AI も運用の対象になる	途中から再開でき二重に実行しない仕組みにし、AI の行動をすべて記録し、監視対象のシステムと切り離す
⑤ 評価とコスト	「良くなった」の証明	過去のインシデントで試し、人と並走させて比べ、高性能なモデルは必要時だけ使う

共通する考え方：**AI の力を引き出すのは、周りの仕組み。仕組みで支えれば、AI に安心して運用を任せられる**

SECTION 3

実践アーキテクチャ事例

自律型AI運用基盤「Lorel.ai」の設計

事例：Lorel.ai の全体像

既存の監視ツールにつなぐだけで、検知から対処までを AI エージェントが回し、人がそのループを改善していく。追加のインフラは不要

01 接続
データの取り込み
Datadog・New Relic・Grafana などの監視ツール、クラウド、デプロイ履歴に接続。既存の仕組みはそのまま使う

▶

02 正規化
文脈の統合
形式の異なるデータを共通モデルに変換し、「どのサービスが何に依存しているか」と結びつける

▶

03 推論
予兆検知と原因特定
ログ・メトリクス・変更履歴・過去の類似事例から、根拠付きで原因を特定する

▶

04 実行
手順書の自律実行
原因に合う手順書（Runbook）をエージェントが選び、ポリシーの範囲内で自動実行する

▶

05 記録・報告
説明可能性
何を・いつ・なぜ実行したかをすべて記録し、Slack などへ報告する

▶

06 改善 人が担う
自律度を上げる
対応の記録をもとに、検知ルールのチューニングや Runbook の整理・追加を行う

改善した結果を、次の検知と対処に活かす

段階的な自律化 — まずは「提案のみ」から始め、実績に応じて自動実行の範囲を広げていく。自律レベルは操作の種類ごとに設定する

自動フェイルセーフ — 想定外の処理になった場合は、安全な状態へ自動で戻す。この仕組みはエージェントではなく基盤側が持つ

同じ深夜 2:15 — AI エージェントならどう動くか

BEFORE

人が対応した夜（第1章の光景）

02:15	DatadogからDBのCPU95%超アラートを受信。担当者が起こされる
02:20	メトリクス、APMトレース、ログ、デプロイ履歴を確認し調査開始
02:40	APMトレースを確認。遅いクエリは見つかるが、根本原因がわからない
02:50	DB負荷上昇の影響で夜間バッチが停止。業務影響が発生
03:10	外部アクセス増加による負荷やアプリケーション変更の影響を疑い、ログやデプロイ履歴を調査
03:40	先週追加されたバッチの非効率クエリが原因と特定
04:00	インデックス追加とバッチ再実行を実施
04:30	復旧を確認。暫定的な対応結果をSlackへ共有
09:00	出社後、対応経緯と原因を整理してレポートを作成

復旧まで約2時間15分。深夜対応に加え、翌日の生産性も低下

AFTER

Lorel.ai が対応した夜

DatadogからDBのCPU 95%超アラートを受信。同時に調査を開始

【人は起こされない】

メトリクス、ログ、トレース、デプロイ履歴を収集・相関分析

先週追加されたバッチによる非効率クエリが原因と特定。根拠となるログとメトリクスを整理

Runbookに従いDBスケールアップを実施。バッチ停止を回避

復旧を検証。CPU使用率・クエリ性能の正常化を確認

Slackへ対応経緯・根拠・実施内容を報告。あわせてインデックス追加の推奨SQLを提示

担当者は出社後にレポートを確認。推奨事項を採用するか判断し、恒久対策を実施

復旧まで 5〜10分。人は朝、報告を確認して、改善に取りかかる

人の仕事は、なくなるのではなく「AI のマネジメント」に移る

これまで人がやっていたこと

- 深夜のアラート対応
- 複数ツールを行き来する原因調査
- 再起動・スケール・ロールバックの手作業
- 対応後の報告書作成

→ 繰り返しが多く、深夜・休日に集中し、人を消耗させる仕事

これから人が注力すること

- **Runbook の整備** — AI が実行できる手順を増やし、質を上げる
- **エージェントのチューニング** — 判断の精度を上げ、任せる範囲を広げる
- **検知ルールの最適化** — ノイズを減らし、本当に重要な予兆を捉える
- **自律度を上げる設計そのもの** — 「次は何を任せるか」を決める

→ 創造的で、運用の品質に直結し、積み上がる仕事

AI が自律的に対処するほど、人は「対処する人」から「自律度を設計する人」へと役割が変わる。

Runbook を整えるほど、AI に任せられる範囲が広がる。人の仕事が、積み上がる資産になる

SECTION 4

まとめ & メッセージ

今日の3つの問いへの答え

Q1 なぜ、障害の復旧には時間がかかるのか？

復旧を遅らせるのは「原因探し」

復旧時間の大半は、直す作業ではなく調査に費やされている。深夜の負担の正体もここにある

Q2 AI に、どこまで運用を任せられるのか？

原因探しと対処は、AI に任せる

根拠を示し、定めた範囲の中で自律的に動く。だから人は深夜に起こされずに済む

Q3 そのとき、人の仕事はどう変わるのか？

人は「自律度を設計する側」へ

Runbook を整えるほど、AI に任せられる範囲が広がる。人の仕事が、積み上がる資産になる

MESSAGE TO SRES

AI に運用を「置き換えられる」側ではなく、AI に何を任せるかを　　に回ろう

手順書を書く、判断の根拠を残す、権限を整理する —
SRE が積み上げてきたプラクティスこそが、エージェントのガードレールになる

Thank you

JIG-SAW

JIG-SAW コーポレートサイト

https://www.jig-saw.com/

Lorel.ai

Lorel.ai サービスサイト

https://www.jig-saw.com/service/lorelai/