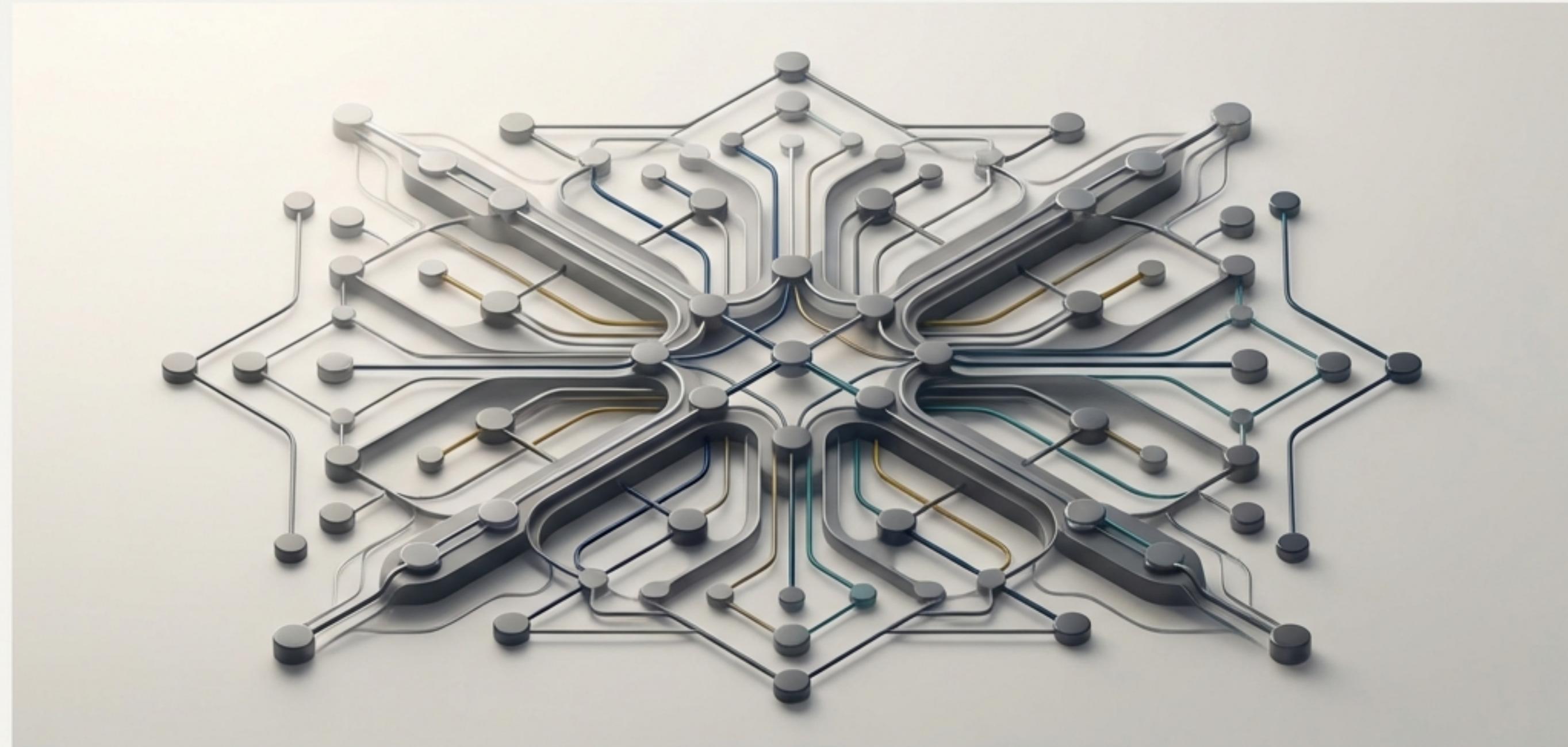


カオスエンジニアリング戦略的概観

複雑化するシステムを『信頼』へと導く、プロアクティブなレジリエンス構築の実践



[Company Logo/Name]

NotebookLM

エグゼクティブサマリー：本資料の3つの要点



1. 新時代の課題 (The Problem)

マイクロサービス化によりシステムの複雑性は爆発的に増大。障害は不可避となり、ダウンタイムの事業インパクトは深刻化しています（例： CrowdStrike社のインシデントではフォーチュン500企業に**54億ドルの損害**）。



2. 戦略的対応 (The Solution)

カオスエンジニアリングは、意図的な障害注入によりシステムの弱点をプロアクティブに発見・修正する規律です。「障害は起きるもの」という前提に立ち、リアクティブな対応から脱却します。これは、システムの「ワクチン」に他なりません。

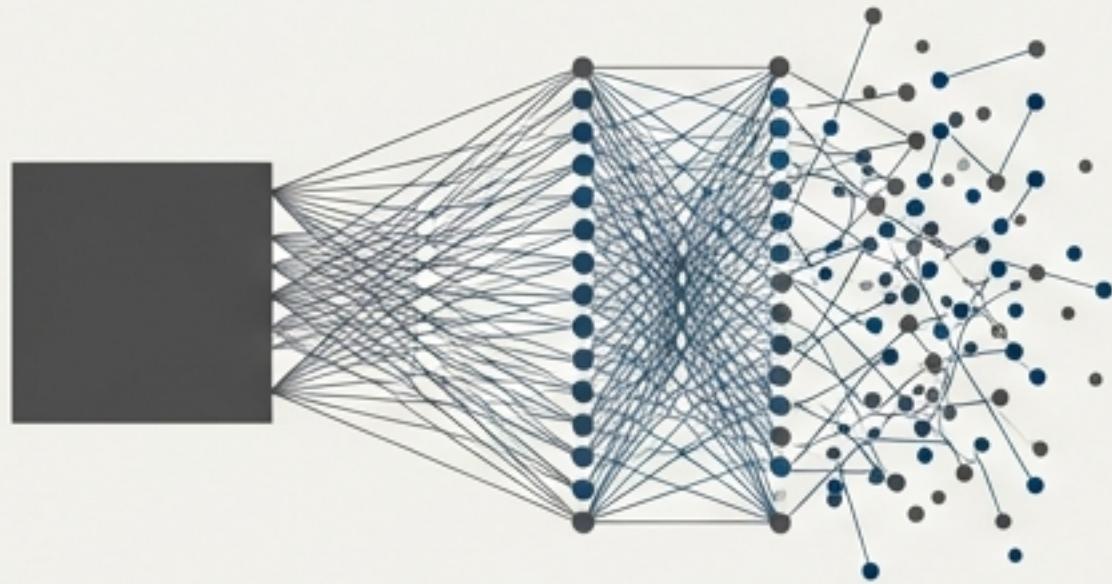


3. 実践への道筋 (The Path Forward)

市場は急成長（2030年までに**35.1億ドル規模**）、特にアジア太平洋地域が牽引（**年平均成長率12.18%**）。本資料では、その原則から実践的プレイブック、ツール選定、DevSecOpsとの連携まで、戦略策定に必要な知見を網羅します。

ソフトウェア開発の「新たな現実」：複雑性とダウンタイムコストの高騰

避けられない複雑性



マイクロサービスと分散システムの普及

モノリスなシステムは解体され、相互に依存する無数のサービス群へと変化。これにより、従来のテスト手法では見逃される、予期せぬ障害経路が指數関数的に増加しています。

Mordor Intelligenceによると、Kubernetes環境がカオスエンジニアリングツール市場の**45.12%**を占めており、このアーキテクチャが主流であることを示唆しています。

許容できないコスト



ダウンタイムがもたらす天文学的損失

システム障害はもはや技術的問題ではなく、経営リスクそのものです。

CrowdStrike社の事例（2024年7月）：一度のソフトウェア障害が Fortune 500企業に**54億ドル**の損害を発生させ、単一ベンダー障害のマクロ経済的影响を露呈させました。

特にヘルスケア業界だけで**19億3,800万ドル**の損失を被り、信頼性の欠如が人々の安全を脅かすことを示しました。

この新しい現実において、障害への「リアクティブな対応」は、もはや戦略的選択肢ではありません。



市場の評決：カオスエンジニアリングはニッチから必須の投資領域へ



世界市場の年平均成長率
(CAGR 2025-2030)

「リアクティブなトラブルシューティングから、
プロアクティブなレジリエンス検証へのシフト
が加速していることを示しています。」

35.1億ドル

2030年の世界市場予測規模
(Source: Mordor Intelligence)



アジア太平洋 (APAC) 市場
の年平均成長率 (CAGR
2025-2030) - 全地域で最速

「急速なデジタルトランスフォーメーションと
サイバーインシデントへの露出増加が、この地
域の成長を強力に牽引しています。」

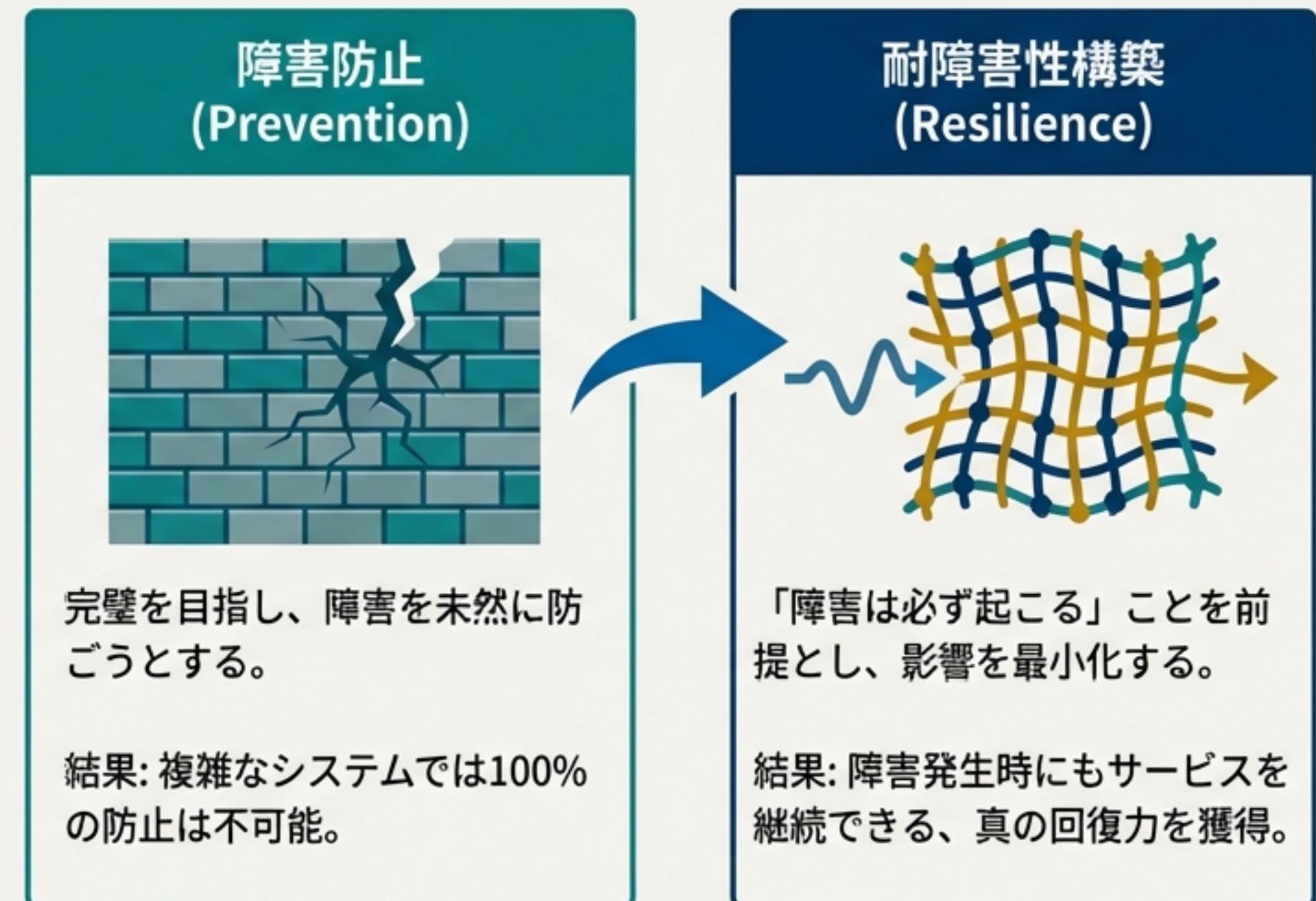
戦略的対応としてのカオスエンジニアリング

コンセプト: システムのための「ワクチン」

カオスエンジニアリングとは、システムに意図的に制御された小規模な「偽の障害」を経験させることで、将来起こりうる大規模で未知の障害に対する「免疫」を獲得させるアプローチです。

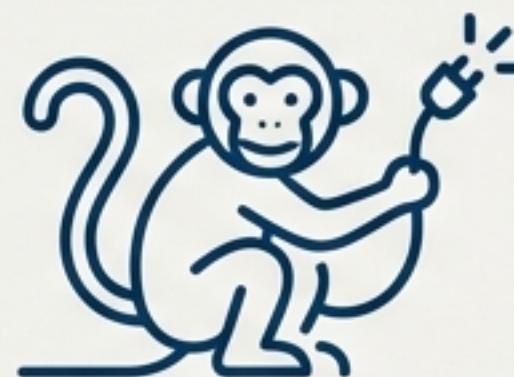
目的はシステムを破壊することではなく、本番環境で発生しうる様々な混乱状態に耐えうるという自信を構築するために、弱点をプロアクティブに発見し、修正することにあります。

Paradigm Shift



カオスエンジニアリングの歩み：Netflixの黎明期から日本での実践へ

2010年 - Netflix



「Chaos Monkey」の誕生

背景: 2008年の大規模データベース障害を教訓に、AWSへの移行を推進。クラウド環境の予測不能な障害に備えるため、意図的にサーバーインスタンスを停止させるツール「Chaos Monkey」を開発。

インパクト: 「いつサーバーが落ちても大丈夫な設計」をエンジニアに意識させ、障害に強い文化を醸成。カオスエンジニアリングの原型となる。

2018年頃 - 日本のパイオニアたち



cookpad

国内企業としてはいち早くカオスエンジニアリングを導入。ステージング環境での小規模な実験から始め、「本番環境で障害が起きたときもユーザーへ影響を与えないこと」を重視し、システムとチームの耐障害性を高める。

「参照: Chaos Engineeringやっていく宣言（クックパッド開発者ブログ）」



システムのモダナイゼーションに伴い、プロダクション環境でのカオスエンジニアリングを積極的に実践。CaaSプラットフォームの信頼性担保に活用し、全社的な安定性向上を目指す。

「参照: ヤフーが実践するプロダクション環境でのカオスエンジニアリング（Yahoo! JAPAN Tech Blog）」

カオスエンジニアリングは、グローバルなテックジャイアントから日本の先進企業まで、複雑なシステムを運用するための標準的なプラクティスへと進化しています。

カオスエンジニアリングの実践サイクル：4つの基本ステップ

ステップ4: 結果の検証と学習 (Verify & Learn)

実験結果を分析し、仮説が正しかったか、覆されたかを検証します。もし弱点が発見されれば、それは最大の学びです。チームで原因を特定し、恒久的な改善へと繋げます。

ステップ1: 定常状態の定義 (Define Steady State)

システムが「正常」である状態を、客観的なビジネス指標（スループット、エラーレート、レイテンシなど）で定義します。これが実験の成否を判断するペースラインとなります。

ステップ3: 実験の実行 (Run Experiment)

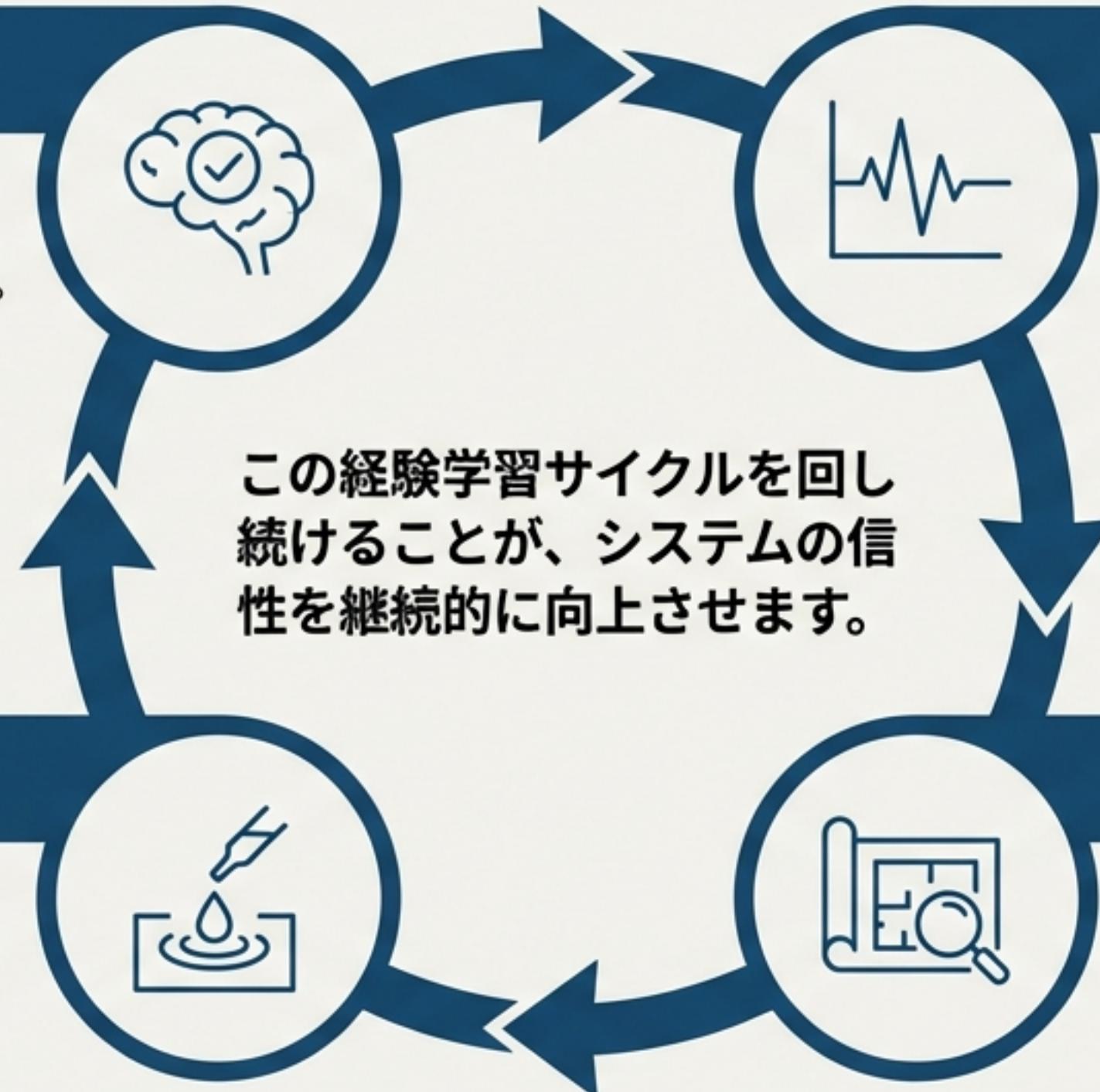
仮説を検証するため、サーバークラッシュ、ネットワーク遅延など、現実に起こりうる事象を模した障害を意図的に注入します。影響範囲を最小限に留めることが最重要です。

ステップ2: 仮説の立案 (Formulate a Hypothesis)

「もし〇〇という障害が発生しても、システムの定常状態は維持されるだろう」という仮説を立てます。

例: 「データベースサーバーの1台が停止しても、冗長化されているためユーザーの検索機能には影響が出ないはずだ」

この経験学習サイクルを回し続けることが、システムの信性を継続的に向上させます。



実践プレイブック：計画から実行、そして改善へ

準備フェーズ (Preparation)

目的: 安全で効果的な実験を計画し、関係者間の認識を揃える。



アクション1: 定常状態の定義とSLI/SLOの確認
実験の成功/失敗を判断するための客観的指標を明確にする。



アクション2: シナリオドキュメントの作成
障害の想定*: 何が、どのように壊れるかを具体的に記述
(例: CaaSプラットフォームの特定のハイパーバイザーがダウン)。
影響の仮説: SLIがどう変化するか、アラートは何分で通知されるか、自動復旧は何分で完了するか等を具体的に予測・記述する。
ポイント: 最初から完璧を目指す必要はありません。「結果が不明」な項目も仮説として記述し、実行フェーズで検証することが重要です。



アクション3: 中止条件とロールバック計画の策定
実験を即座に停止する条件 (例: エラーレートが5%超) と、システムを正常な状態に戻す手順を明確にする。

実行・改善フェーズ (Execution & Improvement)

目的: 仮説を検証し、システムの未知の弱点を発見する。



アクション1: 影響範囲を限定して実験開始 (Start Small)
最初はユーザー影響のない環境や、ごく一部のトラフィック (1-5%) から開始する。

例: APIを1つだけ落とす → Podを落とす → VMを落とす → AZごと落とす、のように段階的に範囲を拡大。



アクション2: 結果の分析と認識ギャップの解消
仮説と実際の結果を比較。もしギャップがあれば、それが改善点です。
システムの修正だけでなく、SLAの値やアラート閾値の適正化も改善に含まれます。

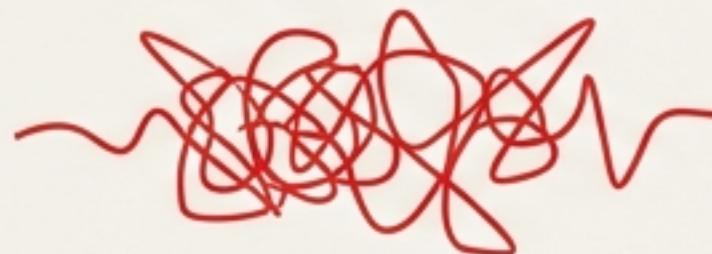


アクション3: 自動化と継続的実行
仮説が正しかった場合、そのシナリオを自動化し、CI/CDパイプラインに組み込むなどして継続的に実行する体制を構築する。

本番環境での実験をどう始めるか？リスクを管理し、自信を構築するアプローチ

「開発環境で起きなかつたことが、なぜか本番環境で起こる」
——この不都合な真実に対し、どうプロアクティブに向き合うか？

A) 意図しない大規模障害の後に改善する
(リアクティブ)



ビジネスインパクト大、エンジニアの心理的負荷大。

B) 意図的な小規模障害から日々改善を積み重ねる
(プロアクティブ)



ビジネスインパクト小、計画的な改善、チームの学習促進。

1. 影響範囲を最小化する



最初はごく一部のトラフィック（カナリア）や、特定のAPI、単一のPodなど、影響を極小に限定したスコープで開始します。

Yahoo! JAPANのCaaSプラットフォーム事例：
まずは単一のワーカーノードダウンを週次で自動実行することから始め、信頼性を確認した上で、半年に一度の関係者が集う「GameDay」でAZダウンという大規模シナリオに挑みました。

2. 自動停止メカニズムを設ける
(Implement a Kill Switch)



エラーレートやレイテンシなどの主要メトリクスを監視し、事前に定義した閾値を超えた場合に実験を即時自動停止する仕組みを導入します。AWS FISやGremlinなどの主要ツールにはこの機能が組み込まれています。

3. チームで訓練する
(Train as a Team)



大規模な実験（GameDay）では、関係者全員（プラットフォーム、インフラ、アプリ開発者）が集まり、インシデント対応のコミュニケーションや役割分担をシミュレーションします。これにより、技術的な耐性だけでなく、組織的な耐性も向上します。

カオスエンジニアリングツールランドスケープ

目的と環境に応じて最適なツールを選択する

マネージドSaaS (Managed SaaS)

特徴:

GUIベースで直感的な操作、豊富な障害シナリオ、エンタープライズ向けの安全機能（自動停止、RBAC）を提供。導入が容易で、すぐに始めたいチームに最適。

Gremlin

Gremlin

業界のリーダー的存在。マルチクラウド、Kubernetes、VMなど幅広い環境をサポート。



Steadybit

Resilience policies（耐障害性ポリシー）を定義し、システムがそれに準拠しているかを実験を通じて検証するアプローチが特徴。

クラウドプロバイダーネイティブ (Cloud Provider Native)

特徴:

各クラウドプラットフォームに深く統合されており、そのサービスのコントロールプレーンに直接障害を注入できる。AWS/Azure/GCP中心の環境で効果を發揮。



AWS Fault Injection Simulator (FIS)

EC2、EKS、RDSなどAWSサービスに特化した障害を注入可能。



Azure Chaos Studio

Azureリソースに対する障害実験プラットフォーム。Chaos Meshとの連携もサポート。

オープンソース / CNCF

特徴:

Kubernetesネイティブなツールが多く、高いカスタマイズ性と拡張性を持つ。コミュニティ主導で開発が進んでおり、コストを抑えて導入可能。



Litmus (CNCF Incubating)

Kubernetesネイティブ。豊富な実験ライブラリ(ChaosHub)とWeb UI(ChaosCenter)を提供。



Chaos Mesh (CNCF Incubating)

PingCAP社が開発。KubernetesのCRDとして実験を定義でき、GitOpsとの親和性が高い。

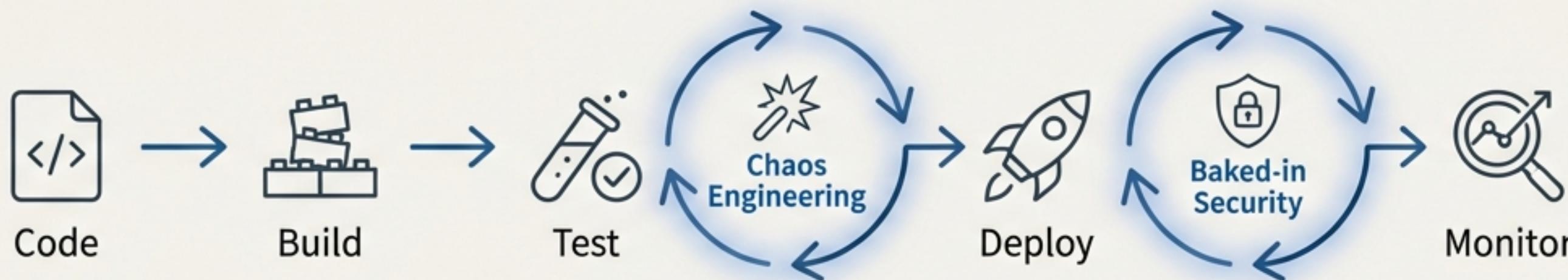
主要カオスエンジニアリングツール 機能比較マトリクス

ツール名 (Tool)	ライセンス (License)	主要プラットフォーム (Platforms)	GUI	CLI	REST API	共有ライブラリ (Shared Fault Library)	実験の自動停止 (Test Halting)	スケジューリング (Scheduling)	ヘルスチェック (Health Checks)
Gremlin	Commercial	K8s, Containers, Linux/Win	✓	✓	✓	✓	✓	✓	✓
Litmus	Open Source (Managed optionあり)	K8s	✓	✓	✓	✓ (ChaosHub)	✓	✓	✓
AWS FIS	Commercial	AWS Cloud	✓	✓	✓	—	✓ (CloudWatch Alarms)	✓	—
Azure Chaos Studio	Commercial	Azure Cloud	✓	✓ (REST API)	✓	—	✓	✓	—
Chaos Mesh	Open Source	K8s	✓	✓	✓	—	✓	✓ (via YAML)	—
Chaos Toolkit	Open Source	K8s, Docker, Cloud	—	✓	✓	✓	—	—	✓

注釈: 本表は代表的な機能の比較です。詳細は各ツールのドキュメントをご参照ください。出典: Gremlin社資料、各プロジェクト公式サイトの情報を基に作成。

レジリエンスを超えて：DevSecOpsと継続的ATO(C-ATO)の中核としてのカオスエンジニアリング

「DevSecOpsは、アジャイルでセキュアな開発、デプロイ、運用を可能にするパラダイムである。」(NIST SP 800-204C)



DevSecOpsへの統合

"Shift-Left"から"Shift-Everywhere"へ

カオスエンジニアリングは、単なる本番環境でのテストではありません。CI/CDパイプラインに組み込むことで、開発の初期段階からシステムのレジリエンスを継続的に検証できます (Shift-Left Chaos)。

NISTの視点: 米国国防総省(DoD)のDevSecOpsイニシアチブでは、「Chaos engineering」は、コンテナの動的な再起動やMoving Target Defenseを実現する「最先端の実践要素(state-of-practice ingredients)」の一つとして挙げられています。

継続的運用認可 (Continuous Authority to Operate - C-ATO) の実現

信頼性の「証明」

DevSecOpsパイプライン内でカオス実験を自動実行し、その結果（ログ、メトリクス）を監査証跡として活用することで、システムのセキュリティと信頼性が継続的に維持されていることを証明できます。

これにより、従来の時間のかかる手動の認可プロセスから、データに基づいた継続的な運用認可 (C-ATO) への移行が可能となり、開発スピードとセキュリティガバナンスの両立を実現します。

戦略的考察：カオスエンジニアリング導入を巡る追い風と課題

市場の追い風 (Market Drivers)



1. マイクロサービス/クラウドネイティブの普及:
相互依存するサービスが増加し、障害経路が複雑化。プロアクティブな検証が不可欠に。



2. ダウンタイムコストの高騰: サービス停止が直接的な収益損失とブランド毀損に繋がるため、信頼性への投資対効果が向上。



3. DevSecOpsとShift-Leftの義務化: セキュリティと信頼性を開発ライフサイクルの早期に組み込む動きが加速。



4. 規制要件の強化 (例: EU-DORA): 欧州の金融機関では、デジタル運用の耐障害性テストが法的要件に。この流れはグローバルに波及する可能性。



5. AIによる導入障壁の低下: AIが実験シナリオの設計を支援し、高度な専門知識がなくとも導入が容易に。

導入の障壁 (Market Restraints & Challenges)



1. 文化的な抵抗とスキルギャップ: 「本番環境を意図的に劣化させる」ことへの心理的抵抗。体系的な思考を持つSRE人材の不足。



2. セキュリティ/プライバシー懸念: 障害注入がデータ整合性や機密性に与える影響への懸念。特に規制の厳しい業界で顕著。



3. 経営層の理解とスポンサーシップ: トップダウンの支援がなければ、文化変革と継続的な投資は困難。



4. スモールスタートの重要性: 最初から大規模な実験を目指するのではなく、信頼を醸成しながら反復的に進めるアプローチが必要。

国内事例に学ぶ：Yahoo! JAPANのCaaSプラットフォームにおける体系的アプローチ

背景: 全社的なシステム刷新の中で、新たにリリースするCaaS (Container as a Service) プラットフォームの信頼性を初期段階から担保する必要があった。

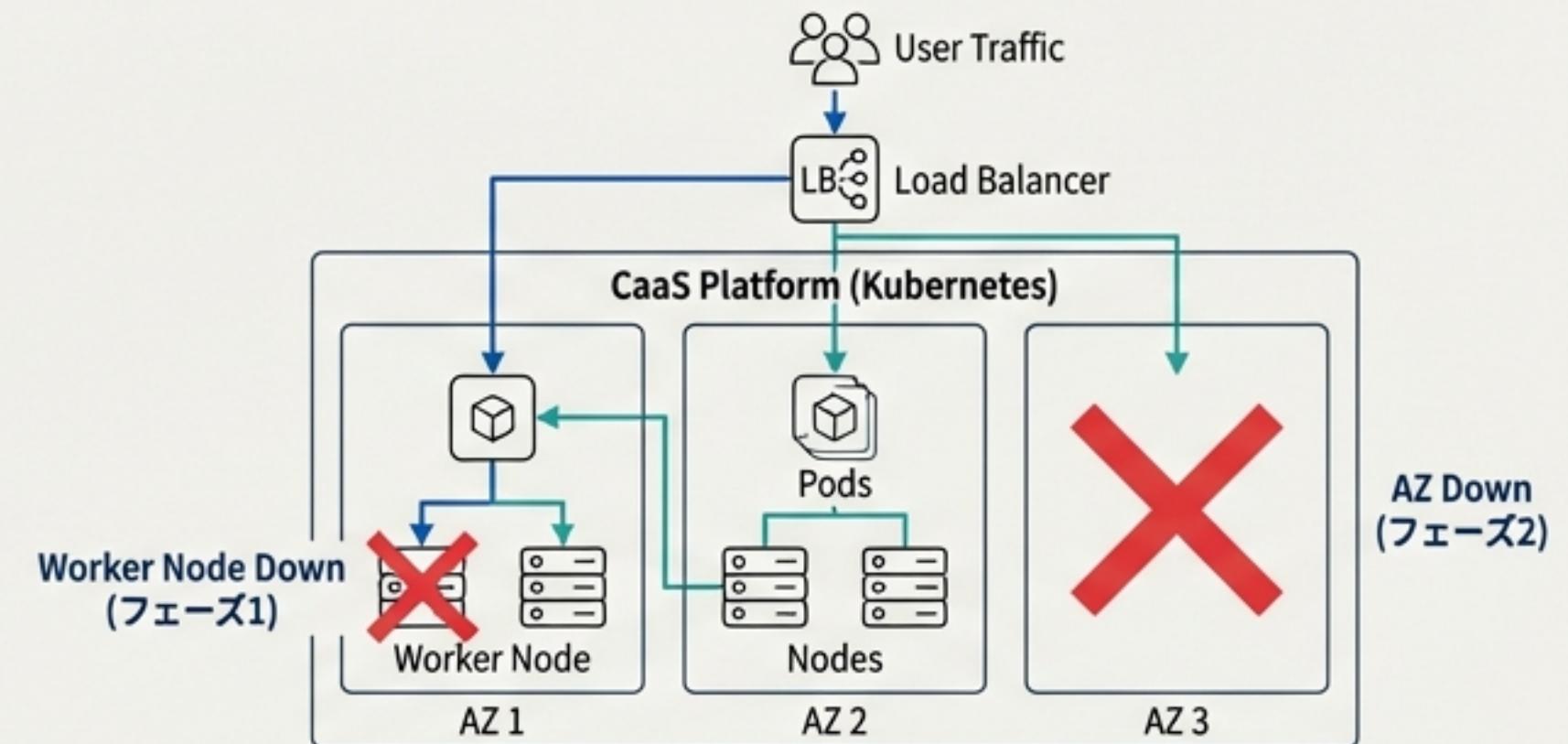
アプローチ:

1. 段階的なシナリオ実行:

- フェーズ1 (週次) : 同一ハイパーバイザー上のワーカーノードダウン。影響範囲が限定的で、自動復旧が正しく機能することを継続的に担保。
- フェーズ2 (半期ごと - "GameDay") : AZダウン。関係者全員が参加し、ロードバランサーのトラフィック迂回やPodの偏りなど、より広範な影響を検証・改善。

2. 開発者との文化醸成:

「このプラットフォームは定期的にカオスエンジニアリングを実施する」と宣言することで、プラットフォーム上でアプリを開発するチームにも冗長化対策を促し、結果的にアプリ層の堅牢性向上にも貢献。



得られた成果:

- システムの安定性向上: 管理系Podの冗長化不備などを早期に発見・修正。AZダウン後もアプリのリクエスト成功率100%を維持できることを確認。
- 組織の学習促進: インシデント対応訓練を通じて、特に若手エンジニアの複雑なアーキテクチャへの理解が深まった。
- 自信の構築: 「この程度の障害で人間が対応する必要はない」という状態をデータで証明し、運用負荷を削減。

結論: 新規プラットフォームのリリース時にカオスエンジニアリングを導入することで、技術的負債を抱える前に高い信頼性を確立し、利用者（社内開発者）にとっても魅力的な基盤を構築することに成功した。

信頼性向上の旅を始めるための、あなたの最初のステップ

1 

スモールスタートで始める (Start Small)

アクション: まずは影響範囲の少ない非本番環境（ステージングなど）や、内部向けツールから実験を開始します。目的は、ツールに慣れ、チーム内に「**安全に実験できる**」という信頼を醸成することです。

最初の実験例: CPU使用率を意図的に上昇させる、特定のPodを一つだけ停止させる。

2 

パイロットプロジェクトを定義する (Define a Pilot Project)

アクション: ビジネスインパクトが比較的小さく、しかし**改善効果が測定しやすい**システムを一つ選び、正式なパイロットプロジェクトとして位置づけます。シナリオドキュメントを作成したし、定常状態の定義から仮説検証までのサイクルを一度回してみましょう。

3 

OSSツールで検証する (Experiment with OSS Tools)

アクション: 本格的な商用ツール導入の前に、まずはCNCFの「[Litmus](#)」や「[Chaos Mesh](#)」といったオープンソースツールをKubernetes環境で試し、カオスエンジニアリングの概念と効果を体感します。これにより、**自社のニーズ**に合ったツール要件を明確にできます。

プロアクティブなレジリエンス構築は、一度きりのプロジェクトではありません。
今日始める小さな一歩が、将来の深刻な障害を防ぎ、持続的な事業成長を支える基盤となります。