でカオスエンジニアリングを使用してレジリエンスを高め、カスタマーエクスペリエンスを向上させる AWS

AWS 規範ガイダンス

AWS 規範ガイダンス: でカオスエンジニアリングを使用してレジリエンスを高め、カスタマーエクスペリエンスを向上させる AWS

Table of Contents

でカオスエンジニアリングを使用してレジリエンスを高め、カスタマーエクスペリエンスを向上させる AWS

Amazon Web Services、Federal Financials、最高技術者、Laurent Domb

2025 年 4 月 (ドキュメント履歴)

カオスエンジニアリングは、本番環境の乱れに耐えられる組織とアプリケーションの能力に対する信頼を構築するために、アプリケーションを実験する分野です。これは回復力に対する積極的なアプローチであり、アプリケーションや組織が、人、プロセス、テクノロジーに制御された障害を導入することで、サービスの障害を吸収、適応、最終的に回復できるかどうかを確認することを目的としています。また、本番稼働の停止やその他の中断を引き起こす前に、弱点を特定して排除することも目的です。

Amazon では、障害が存在するにもかかわらず機能することが通常のオペレーションモードである点まで、分散システムでは障害が避けられないことを理解しています。サービス間のやり取りは失敗する可能性があるため、さまざまな障害モード中にサービスがどのように反応するかを理解し、依存関係の障害、再試行ストーム、障害のあるアベイラビリティーゾーン、ホストリソースの枯渇などの主要な脆弱性に強いサービスを構築する必要があります。

再試行ストームの例を見てみましょう。クライアントでのローカライズされた障害は、複数のサービスに大きな影響を与える可能性があります。これは一般的にバタフライ効果と呼ばれます。再試行ストームは、バタフライ効果の兆候であり、依存関係が失敗すると、クライアントとそのクライアントのクライアントが失敗したオペレーションを再試行し、トラフィックが指数関数的に増加します。サービスは、パフォーマンスの低下を処理しながらトラフィックを再試行するだけでなく、通常のトラフィックに応答する必要があるため、過負荷になります。

カオスエンジニアリングは、分散システムの複雑さの高まりへの対応として浮上しています。これは、カオス理論、システム思考、エンジニアリングの原則を組み合わせて、予期しないイベントや動作に強い複雑なシステムを設計および管理するための学際的なアプローチです。カオスエンジニアリングの中核となるのは、不確実性と予測不可能な条件下での複雑なシステムの動作の理解と管理です。結果の予測と制御に依存する従来のエンジニアリングアプローチでは、分散システムの複雑で動的な性質に対処するには不十分であることが多いことを認識しています。これらのシステムが大きくなるにつれて、多くの場合、個々の理解範囲を超過します。

カオスエンジニアリングは、本番環境に現れる前に脆弱性を発見するために、システムに意図的に障害を挿入するための概念、手法、ツールを提供します。このプロアクティブアプローチにより、組

織はストレスの多い条件下でシステムが動作するという信頼を構築できます。カオスエンジニアリングはまだ進化するプラクティスですが、複雑さと相互接続が増す中で回復力のある最新のコンピューティングシステムの設計、管理、運用への根本的な移行を表しています。

このガイドの以下のセクションでは、カオスエンジニアリングの利点について説明し、カオスエンジニアリング実験を行う方法と、組織内でカオスエンジニアリングを大規模に実装するために実行できるアプローチについて説明します。また、カオスエンジニアリング実験のテンプレートとして使用できる実験計画と実験結果のサンプルドキュメントも含まれています。

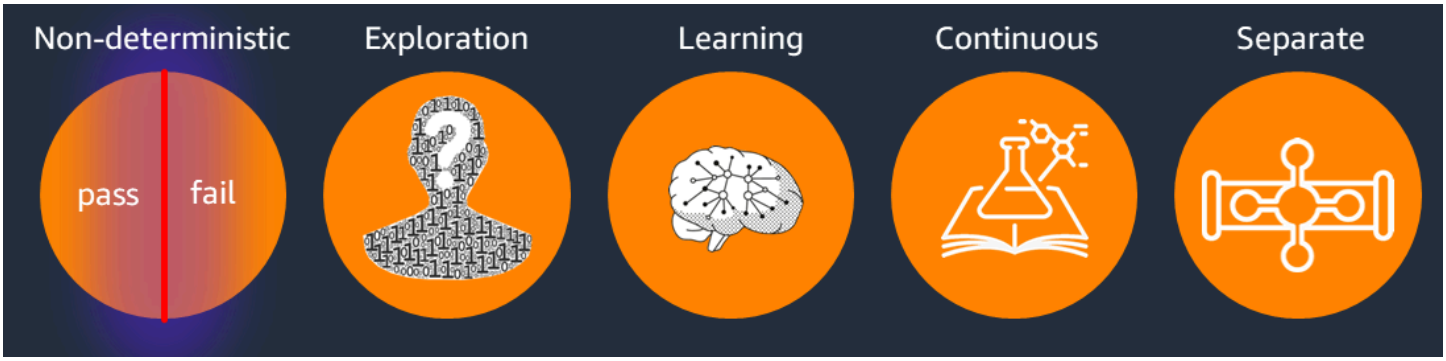
次のセクションでは、カオスエンジニアリングの特性と、単位、煙、統合テストなどの従来の耐障害性テストとの違いについて説明します。

概要

耐障害性テストとカオスエンジニアリングの比較

耐障害性テストは決定論的です。つまり、アプリケーションで実装されているサーキットブレーカー、再試行、フェイルオーバー、フォールバックなど、回復力メカニズムに関する既知の特性を検証します。これらのアプリケーションコンポーネントがユーザーへの影響を最小限に抑えながら、制御された中断をどのように吸収するかを確認します。したがって、耐障害性テストでは、合格/不合格の結果を生成することを目的として、アプリケーションコンポーネントに挿入される既知の障害モードの検証に焦点を当てています。レジリエンステストはパイプラインのステップとして継続的に実行し、レジリエンス体制にリグレッションが発生しないようにする必要があります。耐障害性テストでは、実際のコンポーネントに対してテストを実行するのではなく、特定のコンポーネントをシミュレートするモック APIsを実行することがよくあります。このアプローチにより、制御された環境で障害シナリオを一貫して再現可能なテストできるため、パイプライン統合と回帰の自動テストに適しています。

対照的に、カオスエンジニアリングは非決定的です。つまり、仮説に基づいており、アプリケーションとその依存関係 (人、プロセス、テクノロジー) が予期しない障害モードをどのように吸収、適応、最終的に回復するかについてメンタルモデルを検証します。したがって、カオスエンジニアリングは、未知の障害モードのend-to-endの検証に焦点を当て、欠陥を早期にキャッチし、大規模なイベントになる前にそれらを修復することを目指しています。カオスエンジニアリングは継続的な学習を促進し、デベロッパーのコードデプロイの生産性を妨げることなく、いつでも複数の実験を実行できる別のカオスパイプラインまたはアドホック実験を通じて練習する必要があります。



カオスエンジニアリングプロセスは多くの場合、カオスゲームデーから始まります。カオスゲームデーは、チームが制御された障害や障害を意図的にアプリケーションに注入する専用のイベントです。ゲームデーはプログレッシブです: 低レベルの環境 (開発やテストなど) で始まり、信頼度が高まるにつれて徐々に高レベルの環境 (ステージングや本番稼働前など) に進みます。これらの環境を体系的に移行することで、チームは本番環境に到達する前に、挿入された障害をシステムが適切に許容していることを確認できます。この方法論的な進行により、カオス実験が本番環境で実施されるまでに、チームはシステムの耐障害性能力に大きな信頼を得ています。ゲームデープロセスは、アプリケーションのアーキテクチャと運用プラクティスの弱点と脆弱性を特定する積極的なアプローチであり、予期しない本番稼働停止時の学習のストレスを排除します。

カオスエンジニアリングの価値

複雑なシステムは、今日の世界ではユビキタスです。金融市場から医療まで、私たちの生活のさまざまな側面で重要な役割を果たします。これらのシステムは常に動作することが期待されます。ただし、複雑なシステムは多くの場合、重大な結果をもたらす可能性のある予期しないイベントや動作に対して脆弱です。組織は、中断が発生するかどうかを考えるのではなく、中断を計画する必要があります。そのためには、クリティカルまたはミッションクリティカルなビジネスサービスにシナリオテストを適用します。そこでカオスエンジニアリングが登場します。

カオスエンジニアリングは、リスクを軽減し、レジリエンスを向上させるのに役立つ複雑なシステムを管理するアプローチを提供します。カオス実験に備えるプロセスでは、チームはシステムの動作に関する仮説を立てる必要があります。これにより、システムの構築方法と運用方法の理解が深まります。この準備により、多くの場合、メンタルギャップ、アーキテクチャ上のインサイト、運用上の知識が明らかになり、それ以外の場合は発見されないままになる可能性があります。カオスエンジニアリングは、複雑なシステムが障害にどのように反応するかを理解することで、システム設計と管理における透明性と説明責任の向上を促進します。組織がカオスエンジニアリングを実践する頻度が高いほど、運用上の準備が整います。カオスエンジニアリングは、ユーザーへの影響を最小限に抑えながら、コンポーネントの障害を克服できる回復力のあるアプリケーションを設計するためのベストプラ

クティスを確立するのに役立ちます。これにより、重要なアプリケーションが期待されるサービスレベルと影響耐性内で動作し、チームの独自のシステムに関する知識が継続的に強化されます。

悪影響に備える

を構築するときは AWS、Amazon Elastic Compute Cloud (Amazon EC2) などのゾーンサービス、Amazon Simple Storage Service (Amazon S3) などのリージョンサービス、 AWS Identity and Access Management (IAM) などのグローバルサービス、サードパーティーの Software as a Service (SaaS) サービス、オンプレミスサービスなど、さまざまなタイプのサービスを使用します。サービスのタイプごとに、考慮する必要があるさまざまな障害ドメインが公開されます。自己関連イベントや、組織が制御できないサードパーティーによって引き起こされるイベントに対して、どのように準備すればよいですか?

アプリケーションが悪条件にどのように反応するかを理解するために、 AWS Fault Injection Service (AWS FIS) を使用できます。 AWS FIS は、制御された方法でフォールトインジェクション実験を実行するためのフルマネージドサービスです。このサービスを使用して、アベイラビリティーゾーンの停電やリージョン間の接続の問題など、 AWS提供されたシナリオを挿入したり、サービスによって提供されるさまざまな障害アクションを連鎖させて独自の実験を構築したりできます。 AWS FIS を使用すると、チームは、一般的な障害にアプリケーションがどのように反応し、検出時に欠陥を修正するかを継続的に練習して学習できます。

制御されたカオスエンジニアリングの実践

制御されたカオス実験の主な原則は次のとおりです。

- 本番環境に似た環境から始めます。
- 実験の仮説を確立し、条件を停止します。
- 小さく開始します。
- カオス実験をコントロールします。
- 影響の範囲を設定します。
- サービスベースラインを把握します。
- 実験をスケジュールします。
- 最初に修復し、次に実験します。
- 実験を注意深くモニタリングします。
- 結果から学習します。

- 検出結果の優先順位付け、修正、検証を行います。
- 組織全体に学習内容を伝達します。

カオスエンジニアリングを正常にスケーリングするには、カオス実験を制御された方法で実装する必要があります。を使用すると AWS FIS、Amazon CloudWatch アラームを使用して停止条件を作成できます。これらの条件を実験テンプレートに組み込んで、範囲外の実験が停止し、最後の既知の状態にロールバックされるようにできます。 には、安全レバー AWS FIS も用意されています。これらのレバーをエンゲージすると、 は、マルチアカウント実験を含む AWS リージョン、 のアカウントで実行中のすべての実験 AWS FIS を停止してロールバックし、新しい実験の開始を防ぎます。これにより、取引時間、販売イベント、製品の発売などの特定の期間中、またはアプリケーションのヘルスアラームに応じて、フォールトインジェクションを防ぐことができます。安全レバーは、手動で解除されるまでエンゲージされたままになります。

カオス実験を実行するときは、特に実験が本番環境のアプリケーションに影響を与える可能性がある場合、環境内の望ましくない副作用を防ぐために保護を定義する必要があります。実験を計画するときは、環境内の他のアプリケーションに対する悪影響を予測します。たとえば、他のアプリケーションは、実験の一部であるアプリケーションから誤ったメッセージを受信したり、大量のリクエストボリュームが発生したり、インフラストラクチャを共有している場合にリソースの競合が発生したりする可能性があります。これらのリスクを文書化し、実験を実行する前に、既知の問題や許容できない問題に対処します。

カオスエンジニアリングの開始方法

実験を行う前に、カオスエンジニアリングのプラクティスを最大限に活用するために、いくつかの重要なものを用意することをお勧めします。これらの必須事項には以下が含まれます。

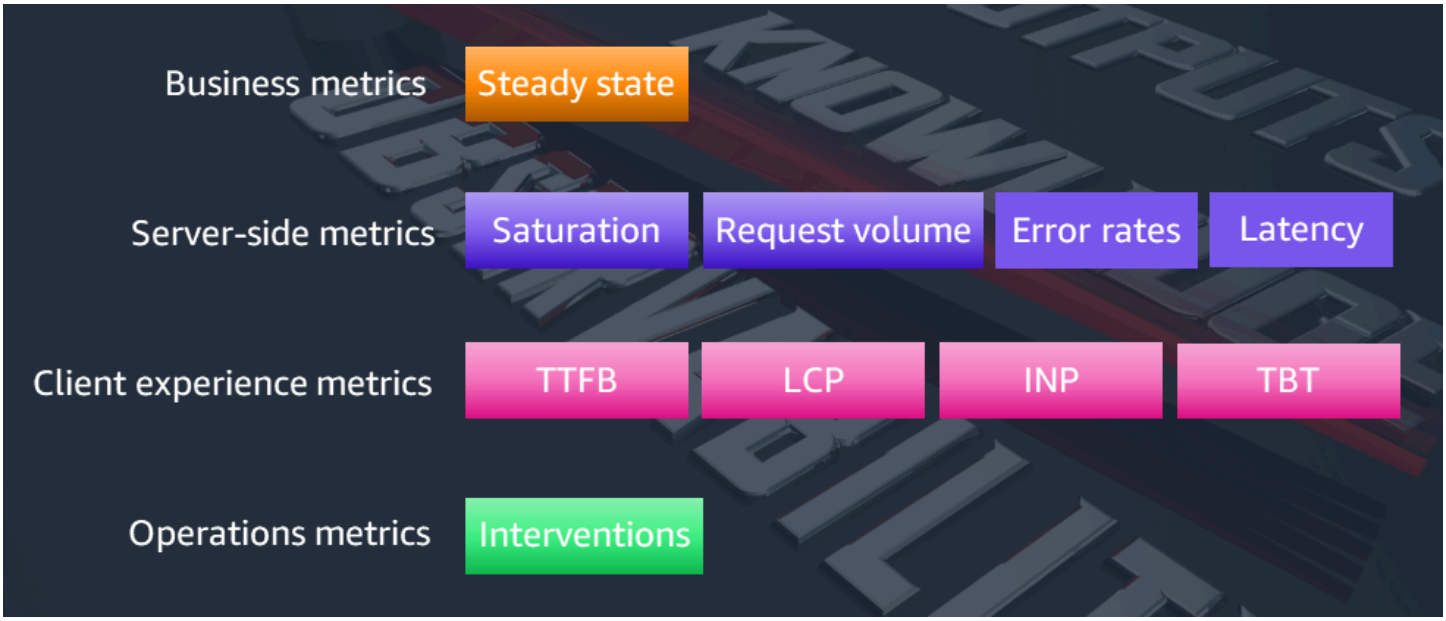
- オブザーバビリティ (メトリクス、ログ記録、リクエストトレース)
- 探索する実際のイベントまたは障害のリスト
- リーダーの賛同による組織のレジリエンススポンサーシップ
- カオス実験の実行時に発見された新機能よりも、潜在的なビジネスへの影響に基づく重要な検出結果の優先順位付け

カオス実験のオブザーバビリティ

メトリクス、ログ記録、リクエストトレースで構成されるオブザーバビリティは、カオスエンジニアリングで重要な役割を果たします。実験を実行するときは、ビジネスメトリクス、サーバー側のメトリクス、クライアントエクスペリエンスメトリクス、およびオペレーションメトリクスを理解する必要があります。オブザーバビリティがないと、定常状態の動作を定義したり、意味のある実験を作成して、アプリケーションに関する仮説が当てはまるかどうかを確認することはできません。

メトリクス

次の図は、さまざまなタイプのアプリケーションのカオス実験で追跡できるメトリクスのタイプを示しています。



- ビジネスメトリクス – 定常状態はシステムの通常のオペレーションを示し、ビジネスメトリクスによって定義されます。1 秒あたりのトランザクション (TPS)、1 秒あたりのクリックストリーム、1 秒あたりの注文、または同様の測定値で表すことができます。アプリケーションが想定どおりに動作している場合、アプリケーションは定常状態を示します。したがって、実験を実行する前に、アプリケーションが正常であることを確認します。障害の割合が許容範囲内にある可能性があるため、障害が発生したときにアプリケーションに影響を与えないとは限りません。定常状態はベースラインです。例えば、支払いシステムの定常状態は、成功率が 99%、往復時間が 500 ミリ秒の 300 TPS の処理として定義できます。視覚的に見ると、定常状態を絵文字 (EKG) と考えることができます。システムの定常状態が突然変動する場合は、サービスに問題があることがわかります。
- サーバー側のメトリクス – 実験中のリソースのパフォーマンスを理解するには、実験前、実験中、実験後のリソースのパフォーマンスに関するインサイトが必要です。リソースが に与える影響を測定するには AWS、Amazon CloudWatch を使用できます。CloudWatch は、アプリケーションをモニタリングし、パフォーマンスの変化に対応し、リソースの使用を最適化し、運用状態に関するインサイトを提供するサービスです。実験中に、飽和、リクエストボリューム、エラー率、レイテンシーなどのサーバー側のメトリクスをキャプチャします。
- カスタマーエクスペリエンスメトリクス – では AWS、CloudWatch RUM を使用して、Locust、Grafana k6、Selenium、Puppeteer などのツールを使用してユーザーリクエストをシミュレートすることで、実際のユーザーメトリクスをキャプチャできます。実際のユーザーメトリクスは、カオスエンジニアリング実験を行う組織にとって不可欠です。実験中に実際のユーザーがどのように影響を受けるかをモニタリングすることで、チームは障害や中断が本番環境の顧客にどのように影響するかを正確に把握できます。主なクライアントエクスペリエンスメトリクスは、Time to First Byte (TTFB)、LCP (Llargest Contentful Paint)、INP (Interaction to Next Paint)、TBT (Total Blocking Time) です。
- オペレーションメトリクス – 介入は、レプリケーションコントローラーや自動スケーリングなどのメカニズムを使用して、ポッド、タスク、EC2 インスタンスの再起動中のクライアントリクエストのレイテンシーの成功など、障害を自動で軽減する方法を測定します。障害発生時に自動的に介入できることは、良好なユーザーエクスペリエンスと直接相関します。これらの緩和メカニズムに経時的なドリフトがあるかどうかを理解することは重要です。成功した自動緩和策と失敗した自動緩和策の両方のメトリクスを定義することで、システム全体で潜在的なリグレッションを特定するのに役立つガイドポストを作成します。

ログ記録

集中ログ記録は、カオス実験の前、最中、および後にアプリケーションコンポーネントの状態を理解する上で重要です。すべてのアプリケーションコンポーネントからログを収集して、実験が挿入され

た時点で各コンポーネントが何をしていたかの統合ビューを構築することをお勧めします。これにより、end-to-endの実験フローを明確に把握できます。

リクエストのトレース

リクエストトレースを使用すると、アプリケーション内のコンポーネント全体で 1 つのリクエストのフローを監視して、挿入された障害がシステムとその依存関係に与える影響を包括的に把握できます。リクエストを追跡することで、障害がさまざまなサービスやコンポーネントにどのように伝播されるかを把握できるため、中断の範囲をより適切に評価できます。でリクエストを追跡するにはAWS、 を使用できますAWS X-Ray。

カオス実験に注入する失敗シナリオ

アプリケーションに一般的な障害を挿入する目標は、アプリケーションがこれらの予期しないイベントにどのように反応するかを理解することです。これにより、緩和メカニズムを作成し、システムがそのような障害に対して回復力を持たせることができます。さらに、カオスエンジニアリングを使用して過去の障害シナリオを再生し、緩和メカニズムが期待どおりに機能し、時間の経過とともにドリフトしていないことを確認する必要があります。

カオスエンジニアリング実験を計画するときは、次のイベントを考慮してください。

失敗モード	説明
サーバーの障害	EC2 インスタンスを再起動し、Kubernetes ポッドまたは Amazon Elastic Container Service (Amazon ECS) タスクを削除して、アプリケーションがこのようなクラッシュにどのように反応するかを理解します。
API エラー	AWS および独自のサービス APIsして、アプリケーションの動作を理解します。
ネットワークの問題	レイテンシーや輻輳を導入するか、接続をブロックして実際のネットワーク問題を模倣します。
AWS アベイラビリティーゾーンの障害	アベイラビリティーゾーン全体の障害を再生して、ゾーン間の復旧を確認します。

失敗モード	説明
AWS リージョン 接続の障害	全体でネットワーク障害を再生 AWS リージョン して、セカンダリリージョンのリソースがそのようなイベントにどのように反応するかを検証します。
データベース障害	データベースレプリカをフェイルオーバーしたり、データが破損したり、データベースインスタンスにアクセスできなくなったりして、アプリケーションと復旧戦略への影響を理解します。
データベースと Amazon S3 レプリケーションの一時停止	データベースまたは Amazon Simple Storage Service (Amazon S3) レプリケーション AWS リージョン をアベイラビリティーゾーン間で一時停止するか、ダウンストリームアプリケーションの影響を理解します。
ストレージの劣化	I/O を一時停止するか、Amazon Elastic Block Store (Amazon EBS) ボリュームを削除するか、ファイルを削除してデータの耐久性と復旧を確認します。
依存関係の障害	サードパーティーのサービスなど、依存するダウンストリームまたはアップストリームサービスのパフォーマンスを低下または低下させ、end-to-endのフローと顧客への影響を理解します。
トラフィックの急増	ユーザートラフィックの急増を生成して自動スケーリング機能をテストし、コールドブート時間がアプリケーション全体の状態にどのように影響するかを確認します。
リソースの枯渇	CPU、メモリ、ディスク容量を最大にして、アプリケーションの正常な低下を確認します。

失敗モード	説明
カスケード失敗	ダウンストリームアプリケーションとコンポーネントにカスケードするプライマリ障害を開始します。
不正なデプロイ	ロールバックメカニズムを検証するために、問題のある変更または設定をロールアウトします。

組織のレジリエンススポンサーシップ

カオスエンジニアリングは、組織全体に適用されると最大の価値を提供します。組織全体のレジリエンス目標の設定を支援し、ドメインに対する懸念、不確実性、疑念を排除し、変革プロセスを開始して、全員の責任のレジリエンスを実現できるエグゼクティブスポンサーと協力することをお勧めします。

カオスエンジニアリングプラクティスを構築するビジネスケースをサポートするには、カオスエンジニアリングの取り組みを重要なビジネスプロジェクトにアタッチします。レジリエンスをアセットにし、アクセラレーションを促進することで、時間の経過とともに成功を追跡するのに役立ちます。まず、1 か月または 1 四半期あたりの重大なインシデントの数、平均復旧時間、およびこれらのインシデントが顧客や組織に与えた影響から始めます。カオスエンジニアリング実験に応じてアプリケーションスタック全体で改善が行われるため、6 か月から 12 か月の期間にわたってインシデントの数を減らすことをチームと目標を設定します。

インシデントの反復性が高いかどうかを測定します。たとえば、クライアントが信頼できる接続を確立できないために、期限切れの TLS 証明書がダウンタイムにつながるとします。複数の TLS 証明書の有効期限が切れたために 1 年に複数のインシデントが発生した場合は、TLS 証明書の有効期限の実験を実行し、チームがアラートを受け取るか、そのような問題を自動的に軽減できることを確認できます。これにより、このような障害に対する耐障害性を確保できます。

カオスエンジニアリングの進行状況を経時的に追跡するには、次のメトリクスをキャプチャして、アプリケーションのライフサイクル全体でカオスエンジニアリングの価値を強調します。

- インシデント率の低下 – 本番環境のインシデントの数を経時的に追跡し、この数をカオスエンジニアリングの導入と関連付けます。重大なインシデントの発生率が低下することが予想されます。
- 平均解決時間 (MTTR) の改善 – インシデントの解決にかかる平均時間を計算し、このデータを追跡して、時間の経過とともにカオスエンジニアリングが改善されるかどうかを確認します。

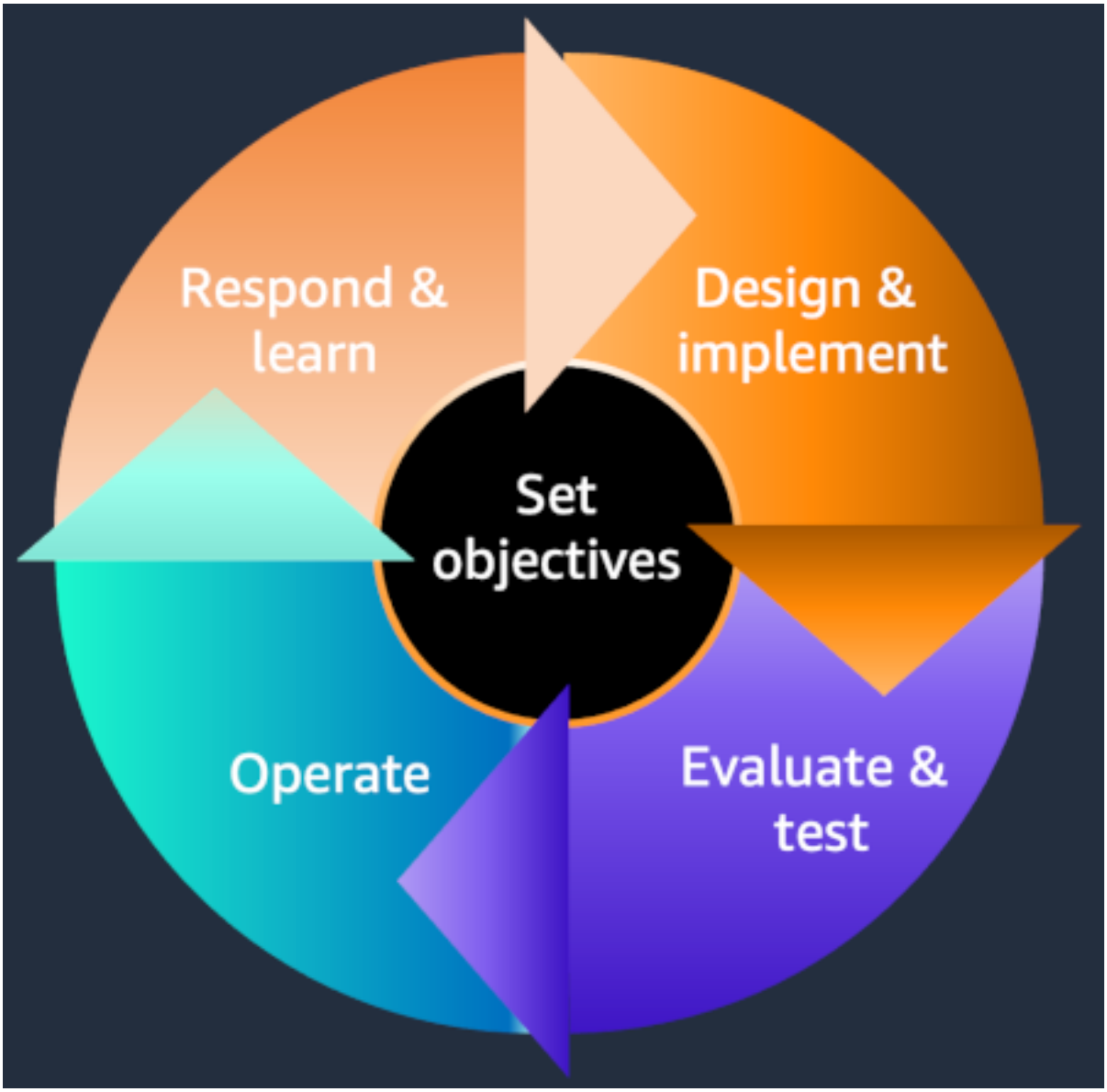
- アプリケーションの可用性の向上 – カオス実験によるアプリケーションの耐障害性の向上に応じて、サービスレベルのメトリクスを使用して可用性の向上を示します。
- 市場投入までの時間の短縮 – カオスエンジニアリングは、アプリケーションの耐障害性がわかっているため、革新的な製品をより迅速に起動できるという自信を与えることができます。製品のリリース速度の増加を追跡します。
- 運用コストの削減 – カオスプラクティスの導入により、アラートノイズ、運用負荷、アプリケーションを管理するための手動作業などの指標が減少するかどうかを定量化します。
- 信頼を高める – 開発者、サイト信頼性エンジニア (SREs)、その他の技術スタッフにアンケートを行い、カオスエンジニアリングがアプリケーションの耐障害性に対する信頼を高めたかどうかを判断します。認識は重要です。
- カスタマーエクスペリエンスの向上 – カオスエンジニアリングを、サービスの中断、ロールバック、停止の軽減など、お客様の良い成果につなげます。

修復の優先順位付け

カオス実験を実行すると、アプリケーションが意図したとおりに動作しない改善領域を特定する可能性が高くなります。このような項目の修復はバックログで機能するため、機能開発などの他の作業とともに優先順位を付ける必要があります。将来の障害を避けるため、これらの機能強化に時間をかけることをお勧めします。これらの学習と修復タスクが引き起こす可能性のある影響のレベルに基づいて、優先順位を付けることを検討してください。アプリケーションのレジリエンスやセキュリティに直接影響する検出結果は、顧客への影響を避けるために、新機能よりも優先する必要があります。チームが機能開発よりも修復作業の優先順位付けに苦労する場合は、エグゼクティブスポンサーに連絡して、ビジネス上のリスク許容度に基づいて優先順位が設定されていることを確認することを検討してください。

でのカオスエンジニアリングの実装 AWS

カオスエンジニアリングは、次の図に示すように、AWS 耐障害性ライフサイクルの評価およびテスト段階の一部です。分散アプリケーションは他のアプリケーションやクライアントとは別に動作しないため、耐障害性ライフサイクル全体を確認することをお勧めします。分散アプリケーションでは、ネットワークが進化し、アップストリームアプリケーションとダウンストリームアプリケーションがシフトし、クライアントの使用状況が時間の経過とともに変化するため、変化は一定です。



アプリケーションに対するこれらの変更がレジリエンスにどのように影響するかを理解するには、カオスエンジニアリングをday-to-dayの一部にします。カオス実験はさまざまな方法で実装できます。

- アドホック – カオス実験を 1 回限りの実験として実行して、特定の問題や質問に対処できます。
- カオスゲームデー – これらは、アプリケーションの信頼性と耐障害性を検証するように設計された構造化された定期的なイベントです。カオスゲームデーの目的は、人、プロセス、テクノロジー全体で潜在的なレジリエンスの問題や欠陥を特定し、インシデントを特定、軽減、対応するためのプロセスと手順を実践することです。
- カオスパイプライン – 継続的インテグレーションと継続的デリバリー (CI/CD) は、新機能を構築し、環境全体に安全にデプロイすることです。カオスエンジニアリング実験を実装するには、CI/CD パイプラインとは別のカオスパイプラインを作成します。理由を理解するために、ダウンストリームコンポーネントのパケット損失が増加する CI/CD パイプラインに 1 つのカオスエンジニアリング実験を追加すると仮定します。この実験は 3 回実行され、毎回完了するまでに 5 分かかります。パケット損失は実行ごとに 10% から 20% から 30% に増加し、実験が完了するまでに全体で 15 分かかります。並列デプロイが 100 回ある場合は、1 回の実験が完了するまで 1500 分待つ必要があります。実行する実験が 10 件ある場合、開発者への影響は耐え難いものになります。大規模なカオスエンジニアリングには、ソフトウェア開発ライフサイクル (SDLC) プロセスと並行して実験を実行できるようにする独自のパイプラインが必要です。
- Canary デプロイ – Canary はカオス実験用のテスト環境を提供します。トラフィックのごく一部を Canary サービスに誘導するか、トラフィックミラーリングやリプレイなどの方法を使用することで、安定した本稼働システムに影響を与えずに新しいインフラストラクチャやコードの変更を検証できます。Canary に対して実験を実行し、必要に応じて障害を挿入できます。エンドユーザーへの影響の範囲を制限できるためです。
- スケジュールされた実験 – アプリケーションの予測可能な復旧メカニズムを検証するための実験をスケジュールできます。スケジュールされた実験を使用して一般的なイベントを再生し、自動スケーリンググループの背後にある EC2 インスタンスの終了、Kubernetes ポッドの削除、Amazon ECS タスクの削除などのイベントからシステムが回復する方法をキャプチャします。

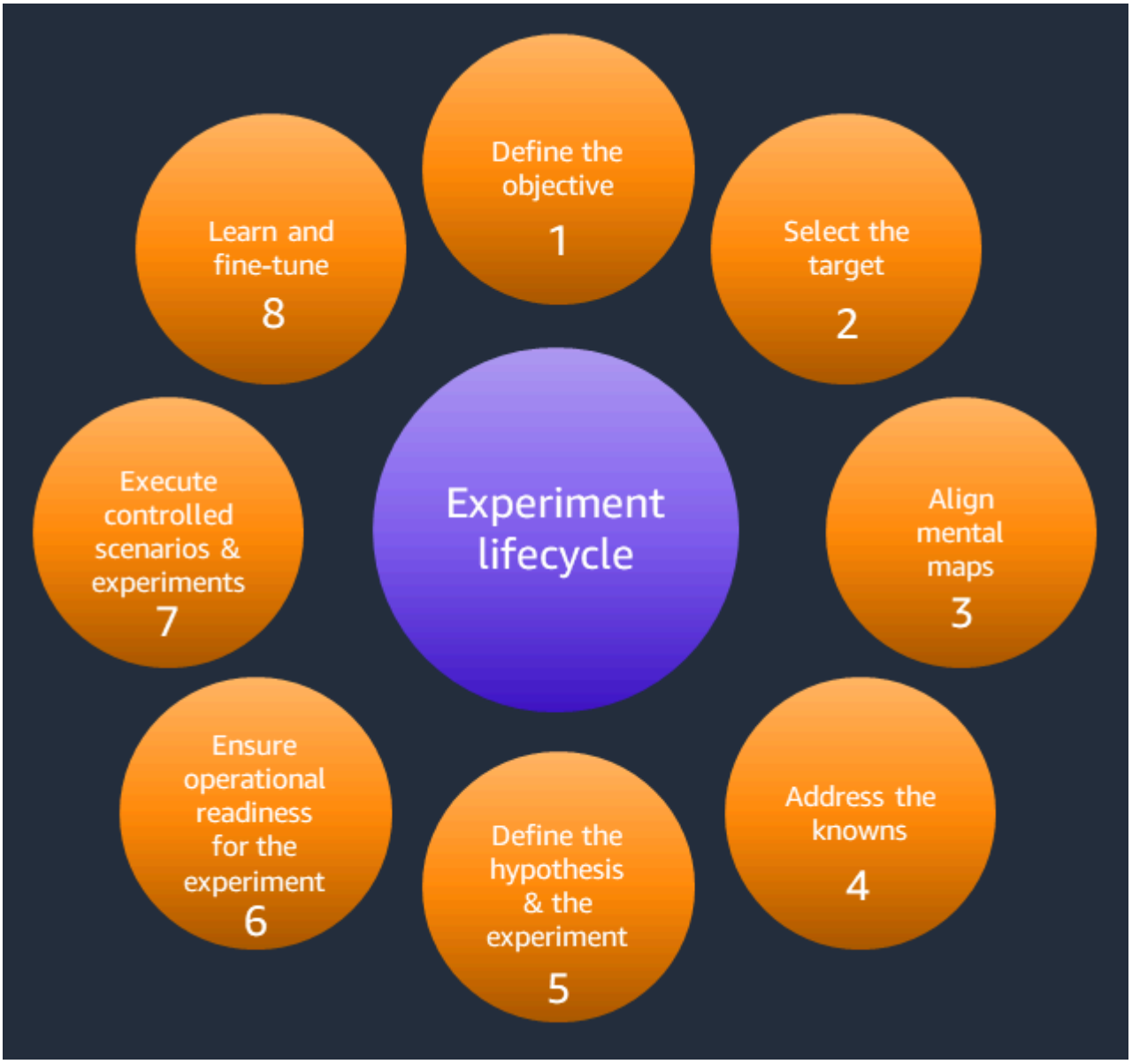
継続的なカオスエンジニアリング実験のライフサイクル

前のセクションで説明したように、さまざまな方法でカオスエンジニアリング実験を実装できます。いずれの場合も、有意義なカオス実験を構築するための鍵は、アプリケーション、過去のインシデント、実装された修復を理解し、レジリエンスやセキュリティなどの調査すべき領域を明確に理解することです。アプリケーションに関する知識は、アプリケーションの潜在的な弱点に関する仮説を立て、障害が挿入されたときに検出、修復、復旧する方法を理解するのに役立ちます。

カオス実験のライフサイクルには、以下のステップが含まれます。

1. 実験の目的を定義します。
2. ターゲットアプリケーションを選択します。
3. メンタルマップを調整します。
4. アプリケーションに関する既知の問題に対処します。
5. 仮説と実験を定義します。
6. 実験の運用準備が整っていることを確認します。
7. 制御されたシナリオと実験を実行します。
8. 実験から学び、微調整します。

これらのステップを図に示し、以下のセクションで説明します。



目標を定義し、期待を設定する

各実験の前に、目標と期待が具体的、測定可能、達成可能、関連性があり、期限があることを確認してください。以下を明確に定義します。

- システムやサービスの潜在的な障害や弱点を特定し、それらがユーザーにどのように影響するかを理解します。これには、サーバーのクラッシュ、ネットワーク障害、ソフトウェアのバグなどの潜在的な障害モードを特定し、システム全体のパフォーマンスと信頼性に対する潜在的な影響を評価することが含まれます。

- システムおよびサービスに対する主要リスク指標 (KRIs) を定義することで、障害の影響を定量化します。これには、レイテンシー、スループット、エラー率などのメトリクスが定常状態から逸脱した場合の障害の影響の測定が含まれます。このような逸脱の影響を理解することで、ビジネスリスクに基づいて障害を軽減するための取り組みを優先できます。
- 障害を軽減または防止するための戦略を開発および検証します。これには、冗長性、エラー修正、フォールバック戦略などの潜在的なソリューションの特定、制御された環境での有効性のテストが含まれます。これらの戦略を検証することで、障害の防止または軽減に効果的であることを確認し、自信を持って本稼働システムにデプロイできます。
- インシデント対応とディザスタリカバリのプロセスを改善します。制御された環境で障害を再生することで、インシデント対応プロセスをテストし、潜在的なボトルネックやギャップを特定し、ディザスタリカバリ手順を改善できます。これにより、予期しない障害が発生した場合に迅速かつ効果的に対応できるようになります。

ターゲットアプリケーションを選択する

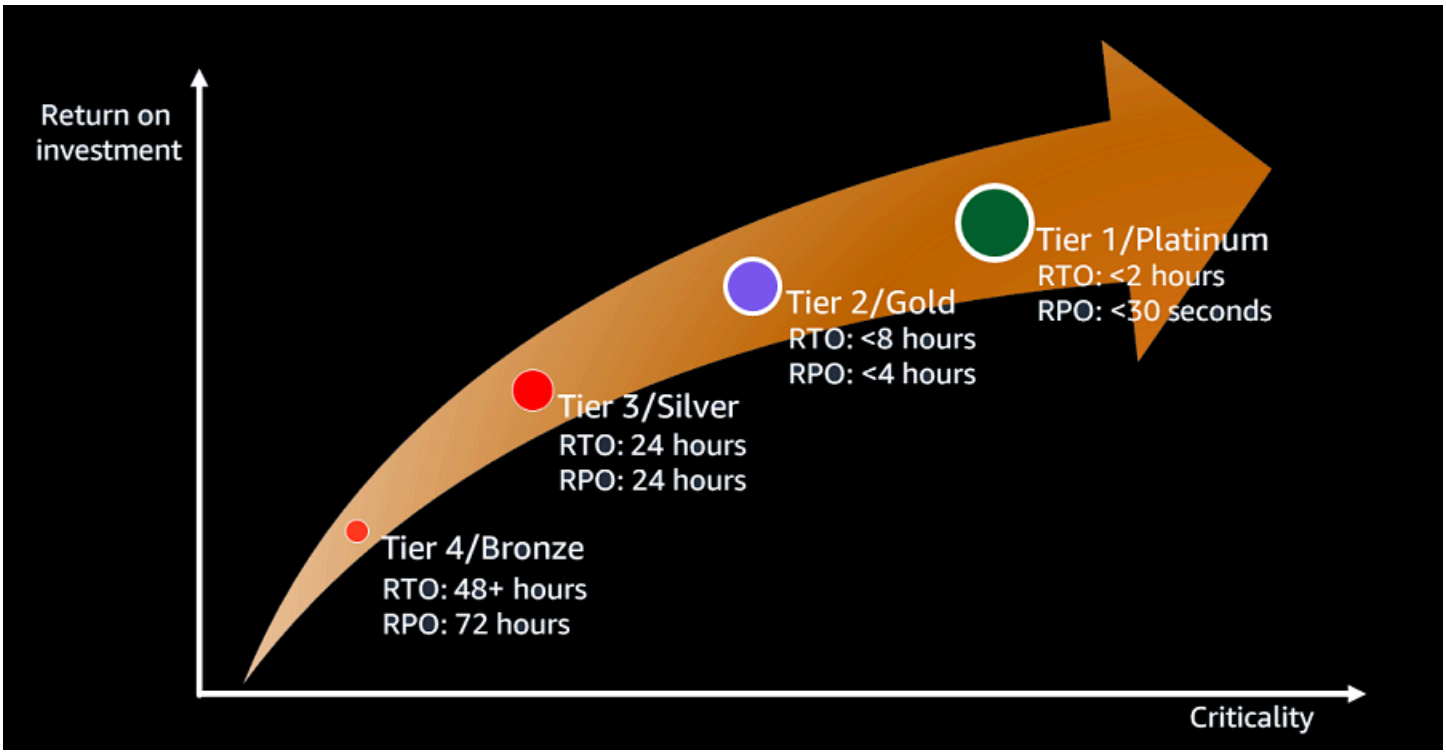
カオスエンジニアリングは強力な手法ですが、価値を最大化するには慎重に優先順位を付ける必要があります。カオスエンジニアリングの取り組みの焦点を決めるときは、まずビジネスの重要なサービスを考慮してください。ソフトウェア開発ライフサイクルステージを繰り返して、まずテスト環境で障害を挿入するようチームに依頼します。ビジネスクリティカルなアプリケーションは、収益、カスタマーエクスペリエンス、コアオペレーションに直接関連しています。これらのサービスのカオス実験では、対処されていない場合、組織、および潜在的に市場全体に大きな影響を与える可能性のある脆弱性を発見できます。例えば、まず取引システムや注文システムなどの顧客向けサービスに焦点を当てます。これらの中央サービスに優先順位を付けると、時間の投資ごとに最も保護されます。

重要なサービスの後は、データベース、メッセージキュー、ネットワーク、共有サービス APIs などの基本的なコンポーネントを確認します。これらは組織全体の共有コンポーネントまたはサービスとして使用される可能性があるため、障害が発生すると広範な問題が発生する可能性があります。インフラストラクチャサービスのレジリエンスを確認することで、依存するアプリケーションがその上位にとどまらないという確信が得られます。例えば、Kafka クラスターを破壊するカオスエンジニアリング実験は、ダウンストリームアプリケーションの耐障害性について多くのことを明らかにしています。システムインフラストラクチャは直接顧客向けではありませんが、カオスエンジニアリングの主なターゲットです。

人、プロセス、施設情報、サードパーティーの依存関係のメンタルギャップをマッピングすることを忘れないでください。これらは、組織の影響許容度目標と一致しない場合、大きな中断を引き起こす可能性があります。カオスエンジニアリングの ROI の測定の詳細については、戦略ドキュメント

「戦略的必要性としてのカオスエンジニアリングへの投資」の「カオスエンジニアリングの ROI の定量化」を参照してください。

次の図は、さまざまなサービスの階層でカオス実験を実行するための投資収益率を示しています。



メンタルマップの整列 (アプリケーション検出)

アドホック実験やゲームデーを実行するときは、アプリケーションの詳細のマッピングに焦点を当てたホワイトボードセッションを開催して、アプリケーション検出プロセスを開始します。(カオスパイプラインで実験を実行する場合、ターゲットアプリケーションを定義して、そのメンタルマップを既に調整しています)。メンタルギャップを理解するための適切なアプローチは、最も経験の浅いチームメンバーがまずアプリケーションの図を描き、その後、より上級のスタッフメンバーに図に徐々に追加するよう依頼することです。これにより、経験レベル全体で理解のギャップが明らかになります。

この図は、アプリケーションの直接のアップストリームとダウンストリームの両方の依存関係と、重要なサードパーティーの統合を示しています。アプリケーションを介したリクエストの予想されるフローに整合性があることを確認します。主要なワークフローとユーザージャーニーをマッピングして、顧客がアプリケーションをどのように使用しているかを明確にします。シーケンス図を使用してこの情報をキャプチャすることを検討してください。

この共同セッションの後、チームはアプリケーション、その重要な依存関係、モニタリング能力の共有メンタルモデルを持ち、潜在的なカオス実験を進めるかキャンセルするかを情報に基づいて決定するためのリスクを理解する必要があります。

アプリケーションの既知の問題に対処する

カオスエンジニアリング実験は、アプリケーションの欠陥をプロアクティブに表面化するように設計されています。レイテンシーの増加、サーバーの再起動、アベイラビリティーゾーンの電源障害などの障害を発生させることで、アプリケーションの現実的な中断を許容する能力を検証できます。ただし、このプロセスでは、ターゲットアプリケーションの基盤となるレベルの安定性と正常性を前提としています。すでに問題のあるアプリケーションでカオス実験を実行すると、より深い問題を隠すリスクがあります。

カオスエンジニアリングを行う前に、チームはアプリケーションの既知の欠陥、バグ、パフォーマンスの問題を解決する必要があります。

仮説と実験を定義する

アプリケーションまたは組織内の他のアプリケーションに中断を引き起こした過去のインシデントは、カオス実験のアイデアの優れたソースとして機能する可能性があります。例えば、以前の停止は設定エラーや回復力パターンの欠落によってトリガーされましたか? インシデント履歴を確認し、カオス実験を通じて現実世界の障害の根本原因を再生することは、将来同様の問題に対する回復力を開発する効果的な方法です。

実験概念のもう 1 つの貴重なソースは、アプリケーションに最も精通しているエンジニア、アーキテクト、オペレーターから直接取得できます。チームメンバーがアプリケーションを大幅に中断する可能性のある架空の障害シナリオを送信できるようにすると、インサイダーの知識に基づいてアイデアを収集できます。その後、アプリケーションチームは、提案されたシナリオのうち、潜在的な影響が最も大きいシナリオや、未知のリスクが最も大きいシナリオを評価できます。このような高リスクであまり理解されていないシナリオのカオス実験をターゲットにすると、重要な学習が生成され、将来の問題を防ぐことができます。

3 つ目のアイデアは、レジリエンスモデリングを実行して、特定されたビジネス上の損失につながる状況を予測することにあります。レジリエンスモデリングの演習には、レジリエンスモデルを構築するためのコンポーネントベースのアプローチと、システムベースのアプローチがあります。コンポーネントベースのアプローチでは、「コンポーネント x が極端に負荷がかかっている、または障害が発生したときにどうなるか」という質問をします。次に、レジリエンスモデルを開発するチームは、

このようなシナリオが広範なアプリケーションに与える影響を推測し、シナリオの影響を検出して軽減するために現在実施されているモニタリングと予防的コントロールを特定します。または、システムベースのアプローチはトップダウンプロセスに従って、「ウェブストアフロントが古いインベントリレベルを表示している」など、アプリケーションの望ましくない状態を強調し、アプリケーションチームに、アプリケーションがこの方法で動作する原因となる条件を予測するよう依頼します。

実験の運用準備を確保する

オブザーバビリティに関する セクションで前述したように、アプリケーションとその動作に対する悪条件の影響を測定するには、定量化可能な指標が必要です。アプリケーションの動作を測定できるため、悪影響がアプリケーションに影響を与えたかどうか、どの程度影響したかを判断できます。

アプリケーションに影響を与えるかどうかを理解する最善の方法は、その定常状態を測定することです。定常状態は、通常のオペレーションがどのように見えるかを測定し、通常、特定のアプリケーションのビジネスおよびクライアントエクスペリエンス指標と一致します。次のステップに進む前に、影響を理解するためのオブザーバビリティが設定されていることを確認し、実験が期待どおりに実行されない場合に備えてメカニズムをロールバックする準備を整えてください。

制御された実験とシナリオを実行する

では AWS、本番環境のアプリケーションで最初のカオス実験を行うことはお勧めしません。カオス実験の目的は、アプリケーションがストレスの多い条件下でどのように動作するかについて新しいことを学ぶことです。実験中にアプリケーションの動作が予測できない可能性があるため、本番環境で初めて実験を実行すると、顧客に影響を与える可能性があります。したがって、実際のユーザーに影響を与える可能性を最小限に抑えた低レベルの環境で最初のカオス実験を常に実行し、アプリケーションが挿入されたアクションを吸収、適応、回復できることを確認してから、環境を繰り返し実行する必要があります。

付録に記載されている実験計画ドキュメントと同様に、主要な詳細をキャプチャしたドキュメントを使用して、各実験を徹底的に計画します。含める重要なフィールドには、定常状態の定義、仮説、障害挿入の方法などがあります。カオス実験の計画、実行、分析は、単一のアーティファクトでカバーできます。

実験の書面による計画を確定したら、ドキュメントに概説されている計画された中断を挿入するために必要なコードを用意します。

実験中に潜在的な影響を把握するには、オブザーバビリティメカニズムが設定されていることを確認します。実験レポートなどの AWS FIS 実験結果を自動的にキャプチャする方法がまだない場合は、

実行時にメモを取るチームメンバーを特定し、ダッシュボードのスクリーンショットをキャプチャして、実験を通じてチームを主導します。

学習と微調整

各実験が終わったら、チームとしてカオス実験を確認して検討します。責任のない考え方を維持するために意識的に取り組みます。目標は、責任を割り当てるのではなく、最大限の学習を引き出すことに集中したオープンで建設的な対話を持つことです。

まず、実験の定常状態の定義と仮説を確認します。アプリケーションは期待どおりに動作しましたか? 仮定を無効にした驚きはありましたか? 実験中にアプリケーションがどのように反応したか、良い点と悪い点の両方について観察します。メトリクス、ログ、スクリーンショットなど、収集されたデータは、何が起こったのかを正確に説明する必要があります。

このデータレビューには、判断ではなく関心を持ってアプローチし、学習に基づいてアプリケーション設計、ドキュメント、モニタリング、またはその他の機能を改善できる領域を特定します。これらのアクション項目は、アプリケーションの耐障害性を高めるために、フォローアッププロジェクトとしてキャプチャされます。

この責任のないアプローチにより、何がうまくいかず、どのように修正できるかについて率直に話し合うことができます。関与するすべての人から肯定的なインテントを引き受け、良い成果に向けて取り組んでいたと信頼します。共有目標は、継続的な学習と適応による組織の成長と進歩です。建設的で非難のない方法で実施されるカオス実験レビューは、チームが貴重なインサイトを得るための安全なスペースを提供し、アプリケーションと組織の信頼性と回復性を長期的に高めます。焦点は、人々ではなく学習に留まります。チーム全体に学習内容を分散するには、実験結果レポートを一元的に公開し、他のユーザーがそこから学習できるように結果をアドバタイズします。

組織全体でのカオスエンジニアリングのスケーリング

組織がカオスエンジニアリングを採用するにつれて、標準化と実装には課題が伴います。成熟の初期段階では、さまざまなチームがさまざまなツールを使用し、前のセクションで説明したカオスエンジニアリングプロセスのバリエーションを使用する可能性があります。同時に、潜在的な利点にもかかわらず、一部のチームはカオスエンジニアリングを優先または採用しない場合があります。以下のセクションでは、これらの課題を克服する方法に関するガイダンスを提供します。

全体として、カオスエンジニアリングへのアプローチは、一元的なリーダーシップと分散的な参加のバランスを取るように設計する必要があります。このバランスにより、カオスエンジニアリングが開発プロセスに統合され、学習結果が組織全体で共有されます。

カオスエンジニアリングプラクティスの確立

カオスエンジニアリングの実践を標準化することで、カオスエンジニアリングの導入を加速できます。実験から学んだことをチーム間で共有することで、カオスエンジニアリングへの投資に対するリターンを高めることができます。

カオスエンジニアリングプラクティスの一環として、一元化されたセンターオブエクセレンスを構築するか、対象分野のエキスパートのグループを構築します。小規模で一元化された機能として、このチームはソフトウェア開発、インフラストラクチャ、セキュリティ、ビジネスチーム全体で機能し、それらのチームが使用する標準を維持できます。わかりやすくするために、 このガイドの残りの部分では、センターオブエクセレンスを一元化された練習チームと呼び、カオスエンジニアリングを適用するグループを練習チームと呼びます。

一元化されたプラクティスチームの役割

一元化されたプラクティスチームは、組織全体でカオスエンジニアリングプラクティスを開発および実装する責任があります。彼らは練習チームと密接に連携して、実験の設計と実施をガイドし、実験がビジネスにとって価値があることを確認します。また、一元化されたプラクティスチームは、開発、インフラストラクチャ、セキュリティチームにガイダンスとサポートを提供し、カオスエンジニアリングを開発プロセスに統合するのに役立ちます。

一元化されたカオスエンジニアリングプラクティスチームの主な責任は次のとおりです。

- 有効化 – 一元的なカオスエンジニアリング機能は、ゲームデーやワークショップを通じてカオスエンジニアリングの実践を紹介するファシリテーターとして機能します。これらは、障害シナリオ

の選択、仮説の定義、より広範な組織と共有されるレポートの作成など、カオスエンジニアリングの過程でチームをガイドします。一元化された練習チームはトレーニング資料を所有し、カオスエンジニアリングの使用について練習チームにスキル向上に取り組む必要があります。

- アドバイザリ — 集中型練習チームは、練習チームによって実施される実験を監督するアドバイザリの役割を担うこともできます。彼らの経験と知識は、実験がビジネスに価値をもたらし、安全な方法で実施されることを保証します。同様に、チームは実験の実行と報告を監督して、カオスエンジニアリングを初めて利用する人々をガイドできます。
- マーケティングと価値の追跡 – カオスエンジニアリングのビジネス価値を伝えることは、そのようなプログラムを成功させるための鍵です。カオスエンジニアリング実験に参加する各チームは、ビジネス全体の実験からデータを収集し、カオスエンジニアリングへの組織の投資の価値を実証する必要があります。これには、各実験中に回避されたインシデントの数、実験が失敗した場合に発生したはずのダウンタイム、および障害シナリオが本番環境で発生した場合のビジネスへの全体的な影響を定量化して祝うことが含まれます。このようなデータをチーム間で収集して一元化し、組織全体で利用できるようにすることで、一元化されたプラクティスチームは組織全体のカオスエンジニアリングの導入から得られる価値を追跡し、影響を与えることができます。
- 標準 – 一元化されたプラクティスチームは、カオス実験を実施するためのプロセス、実験の計画と報告のためのテンプレート、実験の実行に使用されるツールを所有し、維持する必要があります。

  中央チームは、実験計画テンプレート、実験レポートテンプレート、プロセスドキュメント、有効化マテリアルを所有および管理する必要があります。ベストプラクティスのドキュメントと有効化資料は、実験の影響を制限するために使用できるガードレール、本番環境で実験を実行するタイミング、時間の経過とともにカオスエンジニアリングの使用を進化させる方法などのトピックについてチームを実践するためのガイダンスを提供します。テンプレートと出力の例については、付録を参照してください。

  一元化されたプラクティスチームは、コミュニケーションやエスカレーション、実験前または実験中に組織内の他のチームといつどのようにコミュニケーションを取るかなど、実験を実行するプロセスも所有する必要があります。このプロセスでは、ガードレールが必要なタイミングについても概説する必要があります。

  一元化されたプラクティスチームは、カオス実験を実施するためのコアツール (などのツール) も選択して所有する必要があります AWS FIS。負荷生成ツールなどの補足ツールの選択と実装は、練習チームに任せる必要があります。練習チームは、ニーズに最適なプロセスとツール全体を調整できる必要があります。

練習チームの役割

集中型チームは全体的なカオスエンジニアリング戦略を推進しますが、実践チームはプロセスに参加し、実験の開発と実行を所有しています。これにより、実験が各特定の製品やサービスに関連し、学習が実行可能であり、製品の信頼性と耐障害性を向上させるために適用できます。一元化されたプラクティスチームは、組織のカオスエンジニアリング標準とプロセスの指導者および所有者として機能します。ただし、一元化されたチームがボトルネックにならないようにするには、個々の練習チームが一元的なプラクティスから学び、カオス実験を実行する必要があります。

実践コミュニティの確立

一元化されたチームを作成するだけでなく、カオスエンジニアリングに関心のある実践者の非公式なコミュニティを確立することをお勧めします。このコミュニティは、実践的なチームや組織全体で知識、ベストプラクティス、経験を共有するためのプラットフォームを提供します。

実践コミュニティは一元的なカオスエンジニアリング実践チームによって運営できますが、組織内の誰でもコミュニティのメンバーになることができます。一元化されたチームは、実践コミュニティを活用して、更新とソース学習をブロードキャストし、一元化されたチームが管理する標準とプロセスを使用している実践チームからフィードバックを収集できます。コミュニティはフィードバックループとして機能し、実践チーム全体のカオスエンジニアリングプラクティスの有効性を一元化されたチームに通知します。その後、一元化されたプラクティスチームは、製品チームをサポートするためにドキュメントとサポートアーティファクトを調整できます。

カオスエンジニアリングを運用レジリエンスに組み込む

カオス実験は、本番環境でのインシデントを防ぐための企業による投資です。この投資に対する最大のリターンを企業がどこで実現できるかを判断する必要があります。組織は、一元化されたカオスエンジニアリングプラクティスチームと協力して標準を更新し、カオス実験を必要とするほど重要な製品を決定できます。

システム開発プロセス

カオスエンジニアリングとカオス実験は、アプリケーションのライフサイクルの一部として繰り返し実行する必要があります。チームがディザスタリカバリテストを定期的に実行する方法と同様に、カオス実験やゲームデーを年間を通じて継続的かつ定期的に実施する必要があります。このアプローチにより、組織がインシデントを予測、観察し、対応する方法が向上します。

結論

カオスエンジニアリングの規律は、過去 10 年間に大きく進歩しました。これはさまざまな業界で採用されており、組織が回復力のあるサービスを作成し、顧客満足度を高めるのに役立ちました。（組織がこれらのプラクティスを実装した例については、「Chaos Engineering Stories」を参照してください。）カオスエンジニアリングにより、組織はクラウドプロバイダーサービスを含むアプリケーションスタックのすべてのレベルで制御された障害を導入することで、ミッションクリティカルなアプリケーションのリスクを軽減できます。アプリケーションスタック全体に制御された方法で影響を与える能力があれば、本番稼働停止のストレスなしに、継続的な耐障害性、運用上の優秀性、オブザーバビリティ、復旧指向のアーキテクチャの向上が可能になります。このアプローチは、組織全体のレジリエンステストプラクティスの向上にもつながります。カオスエンジニアリングの採用を開始するには、カオスゲームデーまたはワークショップを実行して、カオスエンジニアリングが組織に提供できる価値を紹介します。

リソース

AWS FIS 障害モデルの実装：

- AWS Fault Injection Service を使用して、マルチリージョンおよびマルチ AZ アプリケーションの耐障害性を示す (AWS ブログ記事)
- AWS Fault Injection Simulator が Amazon EKS Pods でのカオスエンジニアリング実験をサポート (AWS ブログ記事)
- Amazon ECS ワークロードの AWS Fault Injection Simulator 新機能の発表 (AWS ブログ記事)
- Amazon EBS ボリュームメトリクスと AWS Fault Injection Simulator によるアプリケーションの耐障害性の向上 (AWS ブログ記事)
- マルチアカウント AWS 環境で AWS Fault Injection Simulator を活用するカオスエンジニアリング (AWS ブログ記事)
- Fault Injection Simulator を使用した Amazon RDS AWS でのカオス実験 (AWS ブログ記事)
- カオスエンジニアリングを使用した回復力のあるサーバーレスアプリケーションの構築 (AWS ブログ記事)
- FIS を使用してスポットインスタンスを中断する (AWS ワークショップ)
- 配信パイプラインでのカオスエンジニアリングの自動化 (AWS コミュニティ投稿)

その他のリソース

- 戦略的必要性としてのカオスエンジニアリングへの投資
- カオスエンジニアリングストーリー
- 「Amazon CloudWatch ドキュメント」
- Amazon CloudWatch RUM ドキュメント
- AWS X-Ray ドキュメント
- AWS FIS ドキュメント

付録: サンプルドキュメント

このセクションで提供されるサンプルドキュメントでは、カオス実験のターゲットアプリケーションとして架空のペット導入サイト (PetSite) を使用しています。

- 実験計画ドキュメント
- 実験結果ドキュメント

実験計画ドキュメント

定常状態

プロセス名	ペット導入サイト
物理アーキテクチャ	(アーキテクチャ図へのリンク)
論理アーキテクチャ	(論理図へのリンク)
定常状態を定義する	ペット導入サイトの最大コンテンツペイント (LCP) を使用して測定される平均ページロード時間は 2.5 秒以下、99 パーセンタイルレイテンシー (P99) は 4.0 秒以下、ベースラインは 5000 人の同時ユーザーです。
定常状態メトリクス	ユーザーベース全体でキャプチャされた LCP メトリクスとゴールデンメトリクス (レイテンシー、スループット、エラー率、飽和度) 。
定常状態オブザーバビリティ	LCP はユーザーのブラウザによってキャプチャされ、Amazon CloudWatch に送信され、CloudWatch RUM で検査されます。60 秒間で、その期間内のすべてのリクエストの平均と P99 LCP 時間が集計されます。最上位のゴールデンメトリクスは、CloudWatch を使用してキャプチャされます。

プロセス名	ペット導入サイト
定常状態を実現するプロセス	Grafana K6 は、約 5K00 人の同時ユーザーの通常の本稼働トラフィックレベルをシミュレートする負荷を作成するために使用されます。

オブザーバビリティ要件

チームは以下を表示できる必要があります。

- 定常状態: アプリケーションが通常の状態であることを確認するために何が観察されますか?
- 失敗条件: ダッシュボードに失敗条件はどのように表示されますか? 例えば、次のようになります。
  - トリガーする必要があるアラーム
  - 生成する必要がある ログ
- 障害の影響: 影響を受けることが予想されるコンポーネント (影響の範囲) を表示するには、何を監視する必要がありますか?
- 復旧: MTTR をキャプチャするために復旧を表示および測定する方法
- デバッグ: 実験失敗の詳細をトラブルシューティングします。

次の表は、オブザーバビリティ要件グラフの提案と例を示しています。特定の実験に基づいて、観察すべき内容を定義する必要があります。

監視する必要があるもの	オブザーバビリティツールへのリンク	観察対象
入力のソース	Grafana K6 ダッシュボード	• 実行中のコンテナ数 • 1 秒あたりのリクエスト
全体的なアプリケーションの状態	• ペット導入 CloudWatch ダッシュボード • ペット導入ユーザーエクスペリエンスダッシュボード (RUM)	• Amazon EKS の正常なノード数

監視する必要があるもの	オブザーバビリティツールへのリンク	観察対象
		• Amazon EKS ノードの CPU 使用率
ワークフローの状態	ペット導入 CloudWatch ダッシュボード	LCP 時間、ゴールデンメトリクス
トレース	ペット導入 X-Ray ダッシュボード	• リクエストのレイテンシー • Request count • 失敗数
ログ	ペットの採用 CloudWatch Logs	ポッドで発生したエラーは CloudWatch Logs に発行されます。

実験定義

実験名	Amazon EKS PetSite ポッド CPU ストレス
実験ソースコード	(実験ソースリポジトリへのリンク)
実験の説明	この実験では、PetSite アプリケーションポッドの CPU 使用率の増加がカスタマーエクスペリエンス全体にどのような影響を与えるかを調べます。実行中の各 PetSite ポッドに CPU ストレスを挿入 することで、顧客への影響とその影響の程度を理解できます。
実験の要件またはパラメータ	アプリケーション負荷: 本番稼働平均 ポッドラベルセレクタ: `app=petsite`
実験期間	10 分

実験名	Amazon EKS PetSite ポッド CPU ストレス
環境	Alpha テスト環境
実験ターゲットリソース	PetSite アプリケーションポッド
負荷生成ツールを通じて導入される実験ベースライン	• リクエストの 54% の LCP は <2.5 秒です。 • リクエストの 46% の LCP は <4 秒です。 • エラーは観察されません。
バックオフ条件	なし

仮説

次の場合	Impact	復旧
PetSite アプリケーションポッドが通常の本番稼働レベルのトラフィックで 10 分間 60% を超える CPU 使用率を経験または引き起こした場合、定常状態はどうなりますか?	P99 が 4.0 秒以下のユーザーの P50 では、LCP 時間は 2.5 秒未満のままになります。コンシューマーは PetSite ランディングページをロードできる必要があります。	検出: • CPU ストレスは、CloudWatch で設定されたアラームによって検出されます。 • LCP メトリクスは、ユーザーエクスペリエンスの低下に関するアラームも生成します。 自己修復: • マイクロサービスアーキテクチャの分散性は、ポッドの多くのインスタンスが複数のアベイラビリティー

次の場合	Impact	復旧
		ゾーンで実行されていることを意味します。 • EKS クラスターコントロールプレーンは、影響を受けたポッドからトラフィックを遠ざけ、ワーカーノードで新しいポッドを起動します。 復旧: CPU 使用率が正常に戻ると、LCP は自動的に回復します。

実験プロセス

以下のサンプルstep-by-stepします。

1. すべての Amazon CloudWatch、CloudWatch RUM、 AWS X-Ray ダッシュボードへのアクセスと機能を検証します。
2. アプリケーション環境の状態を検証します。
   a. CloudWatch ダッシュボードを使用して、EKS クラスターが正常であることを確認します。
   b. サンプル URL のテストペット導入サイトアプリケーションのデプロイにアクセスしてください。
3. 定常状態を実現するために負荷を開始します。
   a. ロードジェネレーターが実行されていて、1 秒あたり 5000 リクエストを送信していることを確認します。
   b. アプリケーションが定常状態になるまで 5 分待ちます。
   c. CloudWatch RUM ダッシュボードを使用してアプリケーションの定常状態を確認します。
4. 障害を開始する (実験):

a. AWS FIS コンソールを開きます。

b. pet-adoption-pod-stress 実験を実行します。

c. コンソールで実験が実行されていることを確認します。

5. アプリケーションに対する障害の影響を確認します。

a. CloudWatch RUM および CloudWatch ダッシュボードからスクリーンショットをキャプチャし、異常なデータポイントを書き留めます。

b. 実験が完了したら AWS FIS、追加のスクリーンショットをキャプチャして、アプリケーションがストレスなしで定常状態に戻ったかどうかを記録し、データポイントに異常をメモします。

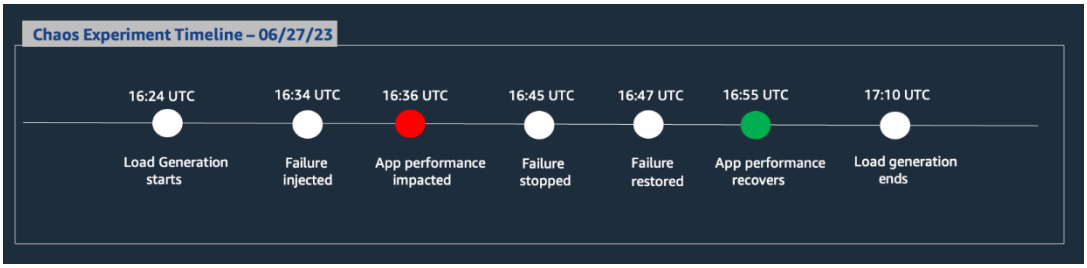
c. 定常状態が再開しない場合は、アプリケーションを復旧する手順を実行し、実行した手順を記録します。

6. 環境が正常に戻ったことを確認します。

- すべてのビジネス、ユーザーエクスペリエンス、アプリケーション、インフラストラクチャのメトリクスを確認して、システムが既知の状態に戻ったことを確認します。役に立つ場合は、ダッシュボードのスクリーンショットをキャプチャします。

実験タイムライン

負荷生成、障害の注入、アプリケーションの影響と回復の観察から、負荷生成を停止したときに終了するまで、end-to-endの実験のタイムラインを必ずキャプチャしてください。これについて、次の例で説明します。



実験結果

実験実行 ID	実験結果
PET-ADOPT-EXP-23	(実験結果へのリンク)

特定された欠陥

- Kubernetes クラスターは PetSite ポッドの CPU 障害を検出しなかったため、追加のデプロイをスケジュールしませんでした。
- この実験の結果、4XX または 5XX エラー率の増加はありませんでした。
- リソースの制約がある場合、LCP への影響を考慮してポッドのヘルスチェックを調整する必要があります。

実験結果ドキュメント

設定

実験の特定の設定を文書化します。例えば、次のようになります。

- 1 秒あたり合計 85 件のリクエストを発行する 5K ユーザーをシミュレートするように設定されたロード生成。

前提条件

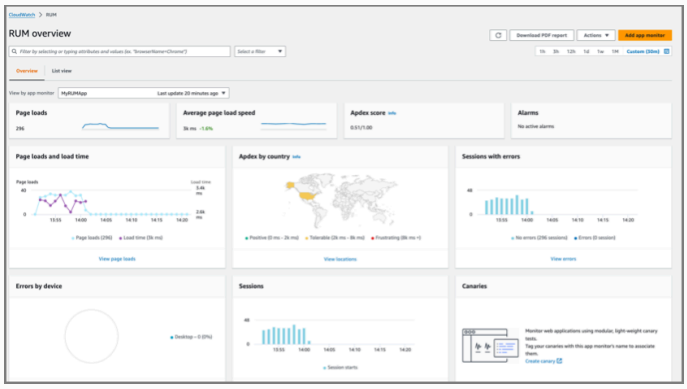
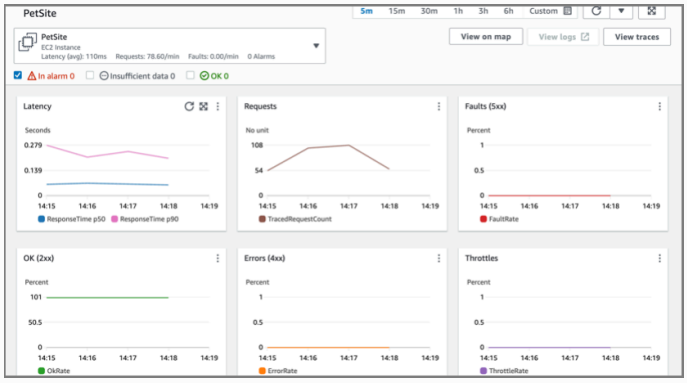
- ペット導入サイトがアルファテスト環境で実行されていることを確認しました。
- 実験テンプレートが、EKS クラスターで実行されている PetSite アプリケーションポッドに CPU ストレスを適用するように設定されていることを確認します。 アプリケーションポッドは Kubernetes ラベル で識別されましたapp=petsite。
- ロードが実行されていることが確認され、1 秒あたり 85 件のリクエストが生成されました。

定常状態

定常状態を達成するために実行されたステップと、それをどのように検証したかを文書化します。例えば、次のようになります。

ペット導入サイトのテストデプロイでは、定常状態をシミュレートするために 85 RPS の負荷が生成されています。CloudWatch RUM と CloudWatch ダッシュボードがレビューされ、すべてのビジネスメトリクスとアプリケーションメトリクスが実験の実行前に正常範囲内にあったことが確認されました。

オブザーバビリティデータ:

予想	観測値
• P99 リクエストの LCP は 4 秒未満です。 • 応答レイテンシーは 500 ミリ秒未満です。 • 4XX または 5XX エラーはありません。	 

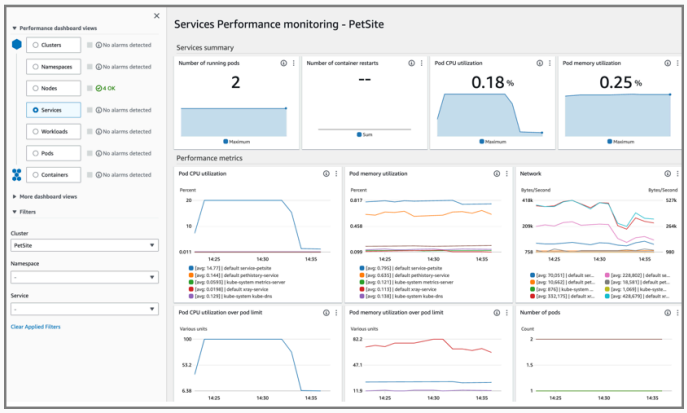
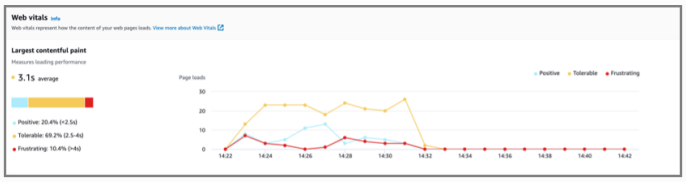
フォールトインジェクション

AWS FIS は、実験テンプレート (リンクを提供) を使用して障害を挿入するために使用されます。実験は 10 分間実行するように設定され、ワーカーノードで CPU ストレスが 60% を超えた場合のロールバックが設定されました。

障害観測

CloudWatch RUM ダッシュボードと CloudWatch ダッシュボードは、アプリケーションの定常状態 (LCP メトリクスを使用して定義) を追跡するためにレビューされました。 スクリーンショットを次の表に示します。

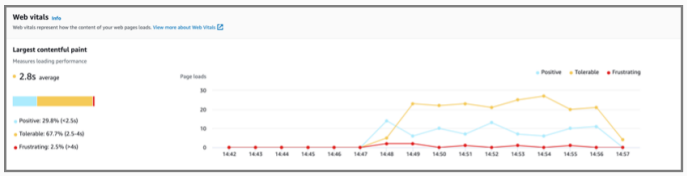
オブザーバビリティデータ:

予想	観測値
• P99 の場合、LCP は 4 秒未満のままにする必要があります。 • 応答時間は 500 ミリ秒未満にする必要があります。 • 4XX または 5XX エラーは発生しません。	 

復旧

ストレスが削除された後 (AWS FIS 実験が完了し、ポッドから CPU ストレスが削除された後) 、アプリケーションは通常の定常状態を再開する必要があります。 手動による介入は必要ありません。

オブザーバビリティデータ:

予想	観測値 (スクリーンショット)
LCP P99 は 4 秒未満で、平均は 2.5 秒未満である必要があります。	

ドキュメント履歴

以下の表は、本ガイドの重要な変更点について説明したものです。今後の更新に関する通知を受け取る場合は、RSS フィード をサブスクライブできます。

変更	説明	日付
初版発行	—	2025 年 4 月 4 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。