Engineer

カオスエンジニアリング

稼働中のシステムへ障害を制御下で注入し、耐えられるか実験する取り組み

どのようなものか

カオスエンジニアリングは、稼働中のシステムに現実の障害を意図的かつ制御された形で注入し、対処できるかを調べる取り組みです。多くの実践者が参照する短い文書『Principles of Chaos Engineering』は、本番環境の不安定な状況に耐えられるという「確信を築くために、システムを対象に実験を行う分野」と定義しています。重要なのは、実験することです。カオス実験は機能を仕様と照合するのではなく、システム全体について問いを立て、どのような答えが返ってきても受け止めます。

必要性は、分散システムから説明できます。個々のサービスがすべて正しく動作していても、それらの相互作用から誰も予測しなかった結果が生まれることがあり、サーバーのクラッシュやネットワーク接続の切断といった、まれな現実の出来事が問題を悪化させます。原則の文書は、そこに隠れる弱点として、サービス停止時に正しく働かないフォールバック設定、再試行を増幅する不適切なタイムアウト設定、呼び出し元の負荷に押し潰される依存先、一つのコンポーネントの障害が他の複数のコンポーネントを停止させることを挙げています。どれも、各部分を個別にテストするだけでは現れません。

原則の文書では、実験を四つの手順で示しています。

- スループット、エラー率、レイテンシーのパーセンタイルなど、正常な振る舞いを示す、測定可能なシステムの出力として定常状態を定義します。
- 対照群と実験群の両方で、その定常状態が続くという仮説を立てます。
- サーバーのクラッシュ、ディスク故障、ネットワーク接続の切断など、現実の出来事を反映する変数を導入します。
- 両群の出力の測定値を比較し、予測に反する証拠を探します。
  定常状態を崩しにくいほど、チームはシステムへの確信を強められます。仮説が崩れた場合は、夜中の障害中ではなく、弱点がまだ小さく、人が見守っているうちに発見できたことになります。

最もよく知られるツールはNetflixのChaos Monkeyで、本番の仮想マシンインスタンスやコンテナをランダムに終了させます。READMEでは、原則に従うツールの一例とされ、インスタンス障害に耐えるサービスを技術者が作るようにすることを目的として明示しています。

概念を判断に生かす

「カオスエンジニアリングとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 障害が小さいうちに知らせる監視とアラートを整え、安定した状態をどれだけ維持できているかを数値で示します。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 社内初のオブザーバビリティダッシュボードを構築 — 社内システムの状態を初めて可視化

社内初のオブザーバビリティダッシュボードを構築 お問い合わせ →

なぜ重要なのか

冗長化、フェイルオーバー、フォールバックは、何かが壊れたときの動作についての約束です。試したことのない約束は、仮説にとどまります。一度もトラフィックを受けたことのない待機系、遅い依存先に遭遇したことのない再試行ポリシー、一度も発火していないアラートは、どれも動くかもしれませんが、それを初めての実障害で確かめるのは高くつきます。カオスエンジニアリングは、その発見をチームが選んだ時点へ移します。

また、部分ではなくシステムをテストします。単体テストや結合テストは、誰かがすでに想定した振る舞いを確認します。原則の文書は違いを明示し、カオス実験はシステムがどのように動くかを検証しようとするのではなく、実際に機能することを確認する、としています。そのため、一つのコンポーネントの内部ではなく、エラー率やレイテンシーなど、利用者が気付く出力に注目します。

証拠は本番にあります。システムの振る舞いは環境やトラフィックのパターンによって変わり、原則の文書は、実際のトラフィックからサンプルを取ることだけが、要求の経路を確実に捉える方法だと主張しています。Googleの『Site Reliability Engineering』は、別の方向から同じ結論に達しています。人工的な負荷では、コンポーネントが実際にどこで破綻するか分からない場合があるためです。そのため、過負荷対策を備えたサービスでは、本番サービスの限られた部分に障害を起こして試すことを提案しています。その際、対策が機能せず、人が手動でフェイルオーバーする必要に備え、予備の処理能力を用意しておきます。

最後に、動機付けも変えます。Chaos MonkeyのREADMEは、技術者が障害を経験する機会を増やせば、耐障害性のあるサービスを作る理由になる、と率直に説明しています。どの営業日にもインスタンスが終了され得るチームは、マシンが突然停止した日に不足を見つけるのではなく、初めからそれを前提に設計します。

実際の活用例

カオス実験は、テスト計画と同じように、実行前に書き記します。以下は架空のオンラインショップの例で、数値はすべて説明用に作ったものです。

```
実験:        決済事業者の応答が遅くなる
定常状態:    5分ごとの決済成功率と、決済待ち時間のp99
仮説:        決済事業者への呼び出しに300 msを追加しても、
             実験群の成功率は対照群との差が0.2ポイント以内に収まる
群:          決済トラフィックを各群に2%ずつ、ランダムに割り当てる
変数:        決済クライアントに300 msの遅延を加え、それ以外は変えない
中止条件:    成功率の差が1ポイントを超える状態が2分間続く
実行時間:    営業時間中の30分間、当番エンジニアへ事前に知らせる
結果:        仮説は否定された。再試行で事業者への呼び出しが三倍になり、
             待ち時間のp99がタイムアウトの1 sを超えた
次の対応:    要求ごとに再試行回数の上限を設け、変更後に実験をやり直す
```

ここで役立つ結果は、仮説が否定されたことです。あらかじめ中止条件に合意したうえで、数分間、トラフィックの2%から弱点を見つけられ、次の行動は謝罪ではなく技術的な修正になります。

変数自体の注入には、一般的なツールを使うことも多くあります。Linuxのnetemキューイング規則は、遅延、損失、重複、破損を含む、実際のネットワークの性質を模擬します。たとえば、次のように使います。

```
# 実験1：eth0から出る全パケットに100 msの遅延を加える
tc qdisc add dev eth0 root netem delay 100ms
tc qdisc del dev eth0 root

# 実験2：千パケットに一つをランダムに破棄する
tc qdisc add dev eth0 root netem loss 0.1%
tc qdisc del dev eth0 root
```

重要なのは `del` の行です。実験が成功した場合も、失敗した場合も、途中でクラッシュした場合も実行する、後片付けの手順に入れます。netemは、ランダムな損失を再現するための `seed` も受け取ります。また、マニュアルは、現実的なTCP性能の結果を得るには、受信ホストの入口側にnetemを配置する必要があると注意しています。

マシン全体の単位では、Chaos Monkeyが慎重な既定設定の例になります。初期状態では `enabled` がfalse、 `leashed` がtrueで、終了処理は模擬してログに記録するだけです。有効にすると、アプリ、スタック、クラスターごとのインスタンス群を対象に動作します。終了間隔は群ごとに平均二日、最短一日で、設定されたタイムゾーンの09:00から15:00の間、営業日だけ実行します。アカウント、リージョン、スタックは除外でき、障害が進行中かを伝える障害チェッカーを組み込む設定箇所もあります。ただし、同梱の実装は何もしないため、確認処理は運用者が用意する必要があります。SREの書籍が挙げる本番の障害テストも、同じくらい具体的です。タスク数を急激または徐々に減らす、クラスター一つ分の処理能力を一度に失わせる、バックエンドへの通信をブラックホール化する、といった例です。最後の例には、重要度が低くても、応答しなくなることでフロントエンドを停止させ得るバックエンドも含まれます。

このサイトの、2026年9月の侵入防御の遮断が毎回機能することを確かめた事例は、完全な意味でのカオス実験ではありませんが、同じ構造を持っています。強化設定を実行するたびに、標準が文書用に予約する範囲のアドレスを遮断し、サービス自身のステータスを信じるのではなく、生成されたルールをカーネルから読み戻します。そして、確認が成功しても失敗しても実行する後片付けのブロックで、遮断を解除します。テスト用アドレスは誰のものでもないため、失敗した実行の後に遮断が残っても、実在のネットワークを締め出しません。事例は代償も率直に記しています。毎回、本番へルールを書き込むことは、稼働中のシステムへの実際の介入です。それでも受け入れたのは、別の方法のほうが悪いとすでに分かっていたためです。バックアップと復元テストの実装は、設計上、この考え方を復旧へ適用しています。月次の処理で既知のファイルを作業用の場所へ復元し、バイト列を比較します。2026-10-27の実装記録では、リポジトリが設定されていない状態ではタイマーが無効だったとされ、実際の復元を完了した記録もありません。処理はコードとして存在しますが、復旧訓練の成功は別途実現すべき成果として残っています。

問題が起きるところ

仮説を立てずに壊します。 ランダムなサーバーを停止し、何が起こるかを見るのは、実演であって実験ではありません。先に定常状態、仮説、中止条件を書き記します。定常状態をダッシュボードでリアルタイムに確認できないなら、まず直すべきはそこです。実験は後にできます。

出力ではなく内部を見ています。 一台のホストのCPUや一つのキューの長さでは、利用者が異常に気付いたかはほとんど分かりません。原則の文書がスループット、エラー率、レイテンシーのパーセンタイルなど、システム全体の測定可能な出力を求めるのは、仮説がそれらについてのものだからです。

影響範囲に制限がありません。 原則の文書は短期的な悪影響をある程度認めますが、その波及を抑えることは技術者の責任としています。一つのインスタンスか、トラフィックのごく一部から始め、障害を起こしたツールに依存せずに動く中止スイッチを用意します。進行中のインシデントに重ねて実験してはいけません。

検証環境から先へ進みません。 検証環境は最初の一歩としては適切ですが、そこで終えるのは不十分です。トラフィックもデータも、周辺のシステムも、本番とは違うからです。本番前の環境で問題なく実験できたら、その結果を答えと見なすのではなく、実際のトラフィックの小さな部分へ進めます。

一回だけで終えます。 三月の実験に耐えたシステムも、その後は何度も変更されています。原則の文書は、手動の実験は労力が大きく、最終的には持続できないとして、継続的に動くよう自動化することを求めています。そうすれば、耐障害性の後退が入り込んだ週に分かります。

マシンを停止させる実験しか行いません。 明確なクラッシュは、何かが検知するため、比較的単純なケースです。原則の文書は、不正な形式の応答などのソフトウェア障害や、トラフィック急増、スケーリングといった障害以外の出来事も挙げています。SREの書籍は、要求を受け付けても決して応答しないバックエンドを追加しています。要求のタイムアウトが長く設定されている場合、それでもフロントエンドに悪影響を与え得ます。起こりやすさや、想定される影響の大きさで、変数に優先順位を付けます。

発見したことが、その先につながりません。 否定された仮説は、名前が付いた弱点です。他の作業と同じ課題管理システムで、担当者と優先度を割り当てます。修正をリリースしたら実験を再実行し、今度は定常状態が保たれることを確認します。

参考資料

- カオスエンジニアリングの原則：定義、実験の四つの手順、五つの発展的な原則を説明します。最終更新は2019年3月です。

- Chaos Monkey：Spinnakerを必要とする、Netflixのインスタンス終了ツールの文書です。
- Chaos Monkey：Spinnakerによる動作設定：終了頻度、グループ分け、例外と、その既定値を説明します。
- Chaos Monkey：設定ファイルの形式：leashedモードと稼働時間帯を含む、すべての設定と既定値を説明します。
- 連鎖的な障害への対処：Googleの『Site Reliability Engineering』第22章です。障害が起きるまでのテストや、本番での障害テストを扱います。
- tc-netem(8)：遅延、損失、破損の注入に使うネットワークエミュレーターのLinuxマニュアルです。

用語集

実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「カオスエンジニアリングとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/chaos-engineering/