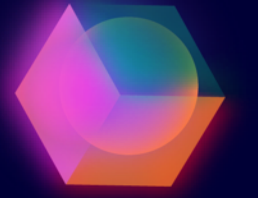
SUMMIT ONLINE

TOKYO| MAY 11-12, 2021



AWS-57

明日から始める!!
Chaos Engineering 始め方ガイド

金森 政雄
アマゾン ウェブサービス ジャパン 株式会社

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

自己紹介

金森 政雄

- 所属/役職：
  DevAx (Developer Acceleration) チーム
  ソリューションアーキテクト

- 好きなサービス

Amazon Elastic Container Service

AWS Step Functions

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

はじめに

対象のお客様

- **既にクラウド上でシステム開発/運用を行なっているお客様を想定します**
  (AWS やAWS のサービス自体の解説は行いません)

- より信頼性の高いシステムの構築のためにChaos Engineering に興味があるが、どのように始めれば良いか悩まれているお客様

注意事項

Chaos Engineering の**具体的な実践方法は対象のシステム毎に異なります**。
そのため、本セッションでは下記については参考情報の提示にとどめます。

- Chaos Engineering のためのツールの使い方

aws

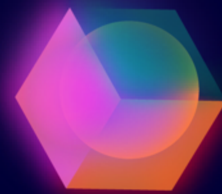
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

アジェンダ

1. なぜChaos Engineering が必要なのか
2. Chaos Engineering を始める前に
3. Chaos Engineering のプロセス
4. 影響を最小限にするために
5. Chaos Engineering の具体例
6. まとめ

aws

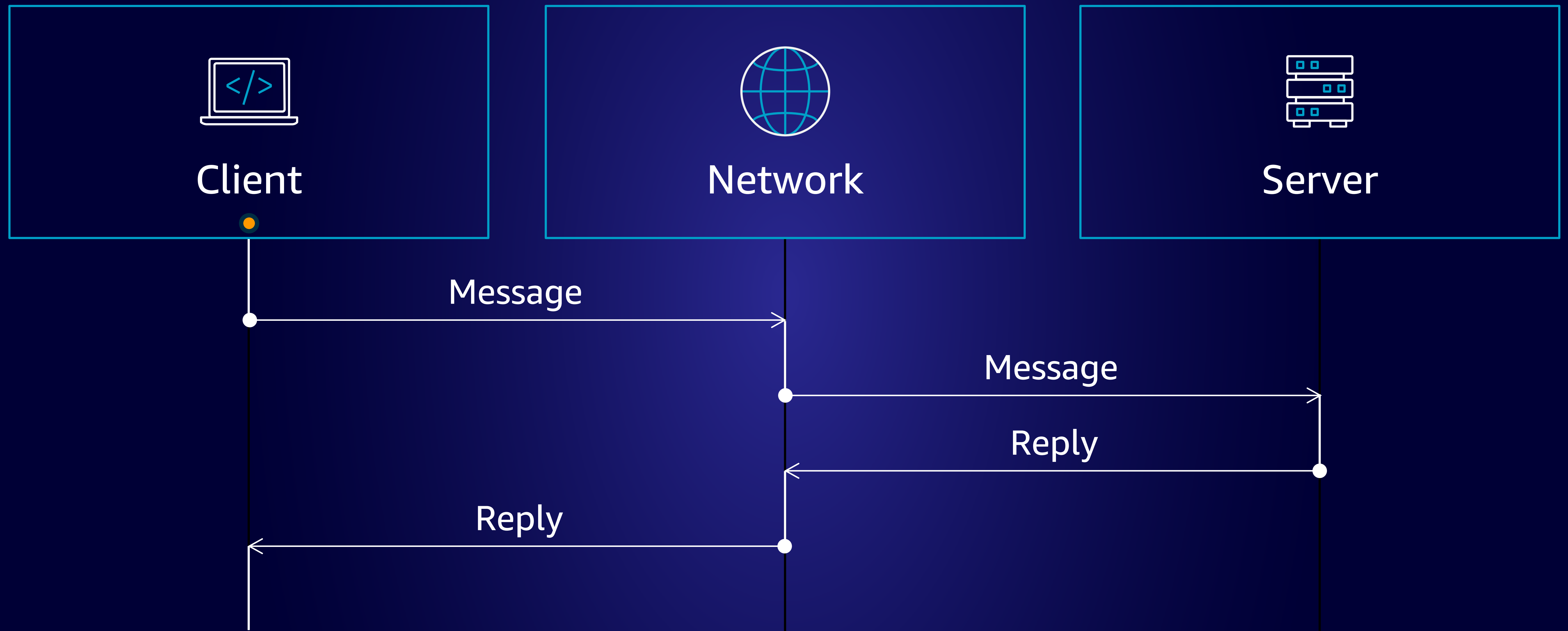
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.



1. なぜChaos Engineeringが必要なのか

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

分散システムはそれ自体が複雑



https://aws.amazon.com/jp/builders-library/challenges-with-distributed-systems/

aws
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

銀の弾丸などない

ソフトウェア開発から複雑性を取り除くことはできない

偶有的な複雑性

開発者自身が発生させている複雑さ

例) わかりづらい変数名
誤ったIF 設計/コンテキストの分離 etc

本質的な複雑性

解決したい課題自体が持っている複雑さ

例) 複雑化していくデータ構造
入り組んだワークフロー　etc

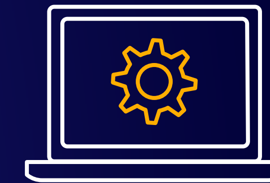
aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

テストだけではカバーできないことがある

コンポーネントに対する
単体テスト

システムを分割し個々の機能が
要件を満たすことを確認する



結合テストによる
機能の確認

それぞれの実行パスが期待された
結果になることを確認する

テスト ＝既知の状態を確認すること

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

予測不可能なものに対処する

Chaos Engineering とは
「対象のシステムが本番環境での不安定な状況を耐えることができる」
という**自信を構築するため**に実施する実験の規律です。

PRINCIPLES OF CHAOS ENGINEERING

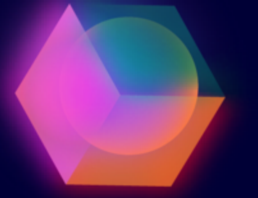
Last Update: 2019 March (changes)

Chaos Engineering is the discipline of experimenting on a system in order to build confidence in the system's capability to withstand turbulent conditions in production.

https://principlesofchaos.org (en), https://principlesofchaos.org/ja/ (ja)

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.



2. Chaos Engineering を始める前に

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Chaos Engineering の前に考えて欲しいこと

①現在のシステムの改善点を確認する

AWS Well-Architected Framework
https://aws.amazon.com/well-architected/

②可観測性(Observability) を確保する

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS Well-Architected(W-A) フレームワーク

AWS のベストプラクティスからシステムの改善点を見つける

（前準備）
セルフチェック

W-A の質問に答えながら、設計中の構成や既に運用しているシステムの現状確認(棚卸し)を実施

AWS Well-Architected Tools へ入力

W-Aレビュー実施

SA とベストプラクティスとのギャップを把握。様々なリスクやクラウドに最適化できるポイントを把握する

2~3時間の集中的な打ち合わせ

クラウド最適化

ビジネス的な判断や優先度づけを実施し、よりクラウドに最適化していく

その後、再度レビュー実施して状況確認

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

可観測性(Observability)を確保する

システムの定常状態と何が起きているかを可視化する

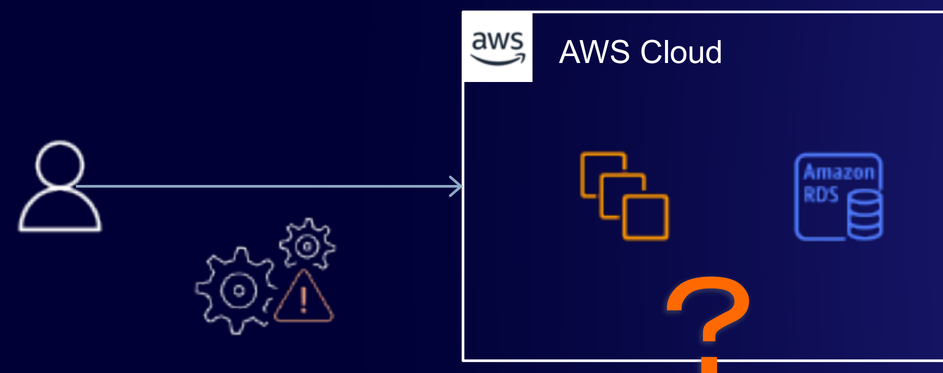
- システムの動作状況を把握できている状態
- システム運用に置いて、判断に必要な情報がきちんと取得できている状態

視認性　**迅速なトラブル解決**　**顧客体験**

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

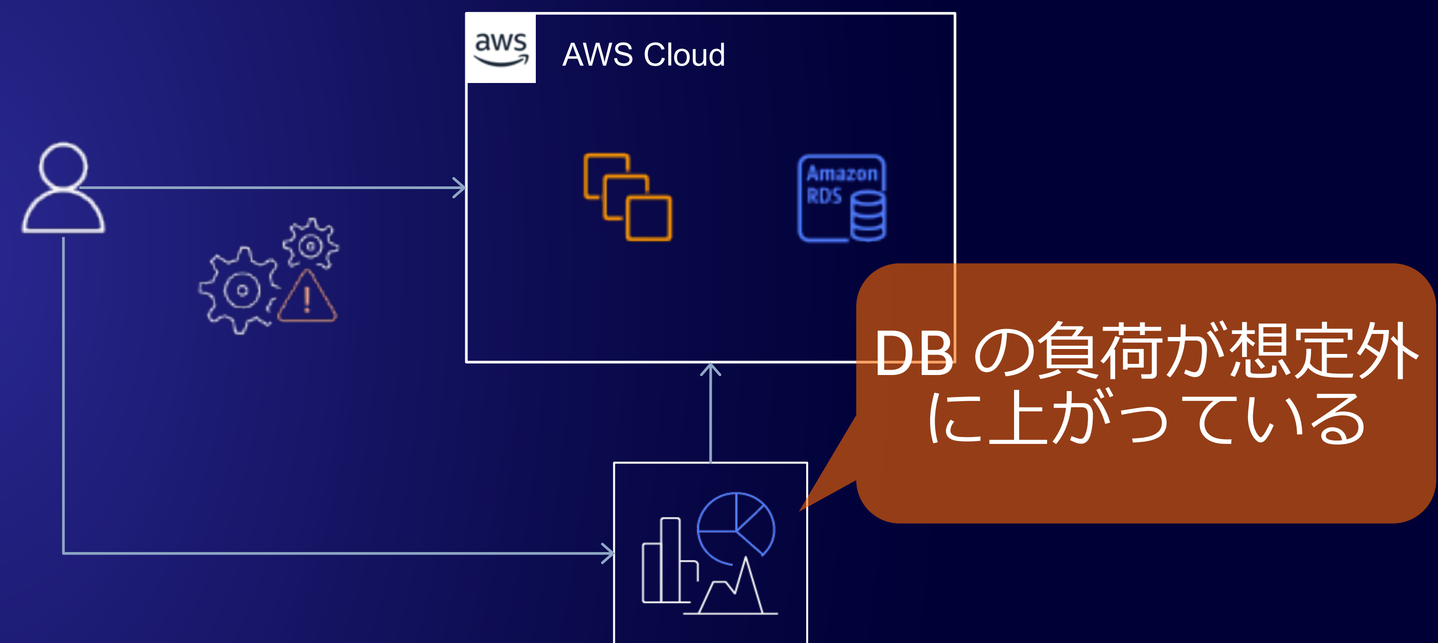
実験から学びを得るために可観測性は必須

<可観測性がない場合>



- Chaos を注入しても何が起きたか不明
- 意図せずにユーザに影響があるリスク

<可観測性を確保すると>



- 振る舞いを確認し学びを得る
- 万が一ユーザ影響がある場合に気付ける

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

実験から学びを得るために可観測性は必須

＜可観測性がない場合＞

AWS Cloud

- Chaos を注入しても何が起きたか不明
- 意図せずにユーザに影響があるリスク

＜可観測性を確保すると＞

AWS Cloud

DB の負荷が想定外に上がっている

- 振る舞いを確認し学びを得る
- 万が一ユーザ影響がある場合に気付ける

Without observability,
you don't have chaos engineering.
You just have chaos.

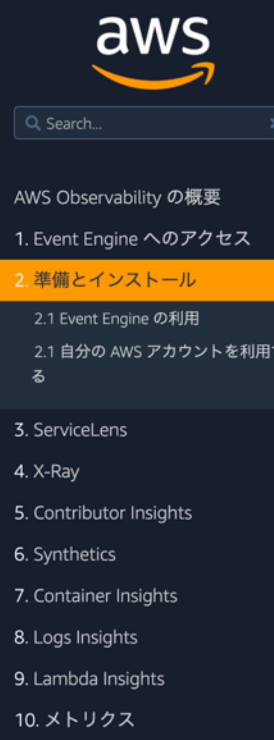
Charity Majors
Cofounder/CTO honeycomb.io

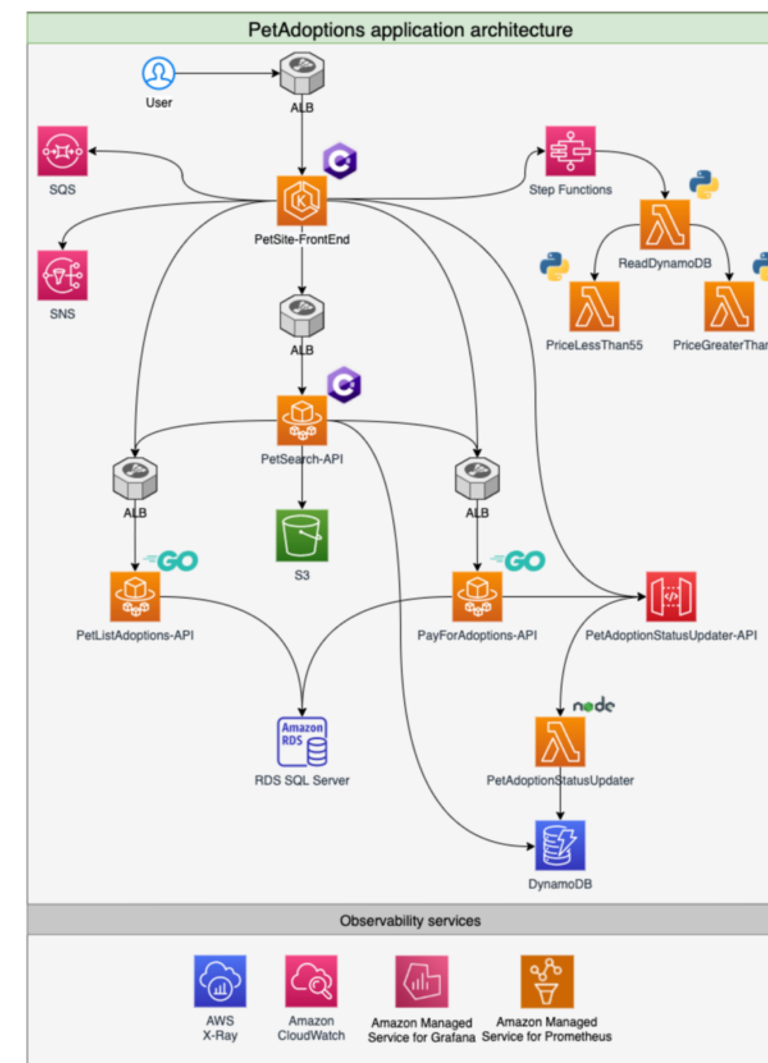
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

One Observability デモワークショップ

https://observability.workshop.aws/ja/

- AWS の監視、可観測性(Observability)に関連するサービスを全体的に学べるワークショップ
- Amazon Managed Service for Prometheus（AMP) など2020年の re:Invent で発表された新サービスにも対応(一部、プレビュー申請が必要)





Chaos Engineering の前に考えて欲しいこと

①現在のシステムの改善点を確認する

AWS Well-Architected Framework
https://aws.amazon.com/well-architected/

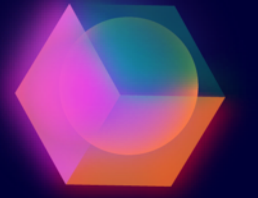
→明示的な改善点がある場合、そちらを解決する方が確実かつコストが低い

②可観測性(Observability) を確保する

→システムの振る舞いを観測できなければ実験から学ぶことはできない

③「誰も責めない」文化

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

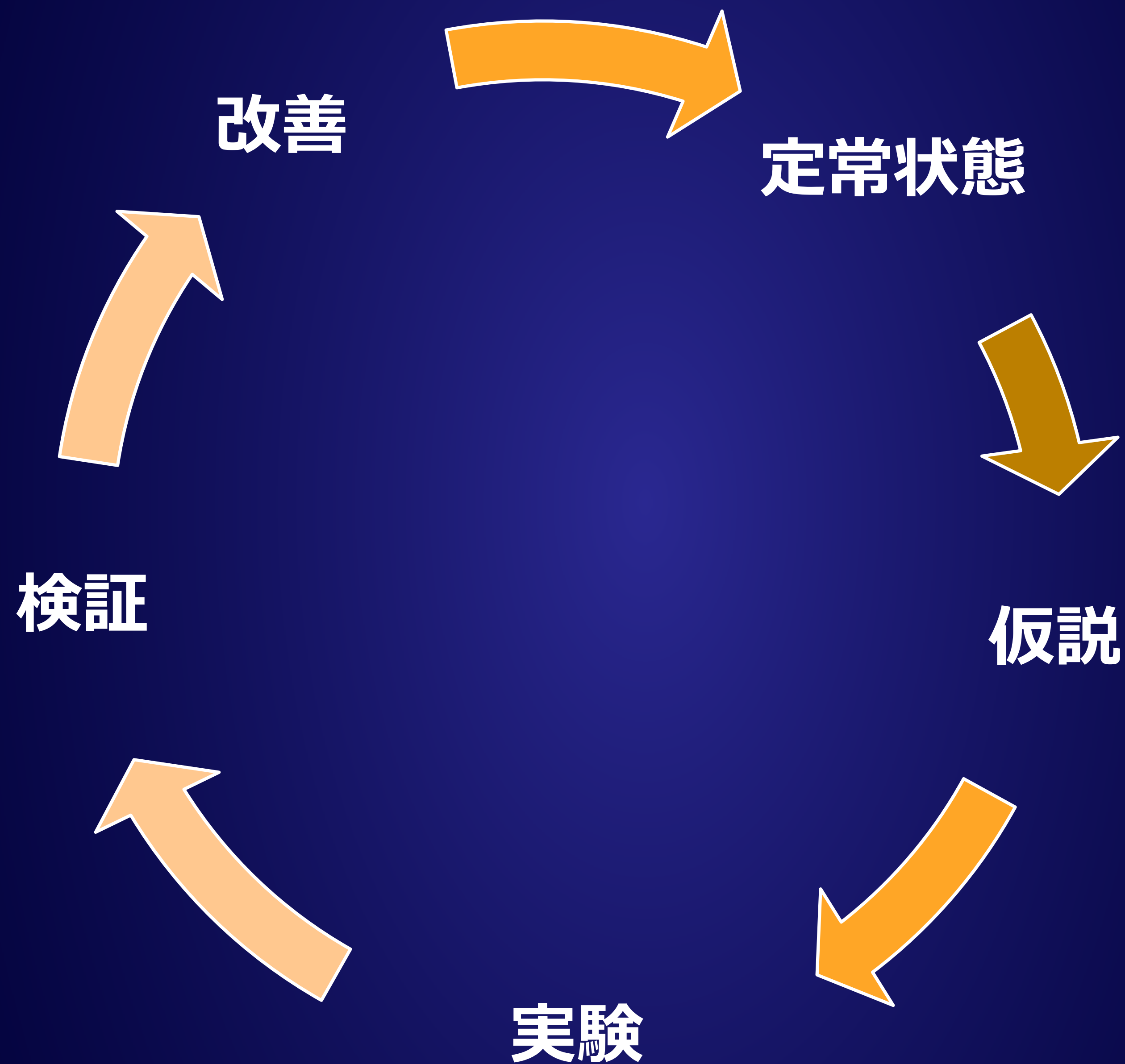


3. **Chaos Engineering** の プロセス

aws

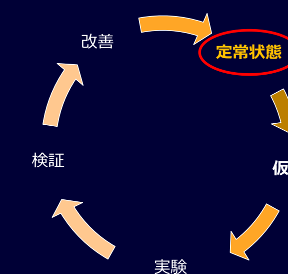
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Chaos Engineering のプロセス



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

定常状態を定義する



通常の動作を示すシステムの測定可能な出力として「定常状態」を定義することから始めます
カオスエンジニアリングの原則 https://principlesofchaos.org/ja/

Q. どちらがよりよい定常状態の定義でしょうか?

A.CPU 使用率が60% を維持する

B.エラー率が1%以下である

「システムが機能しているか」を判断するために、ビジネスKPI やそれに関連する値を設定する

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

仮説を立てる

実験が始まっても定常状態が継続する仮説を立てる

例1. _____ の状況でもお客様は快適にサービスの利用を継続できている

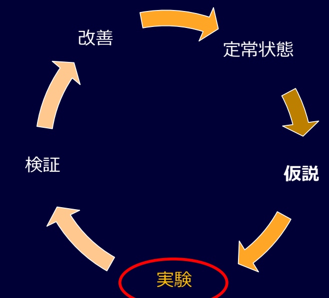
例2. _____ が発生すると、セキュリティチームに通知される

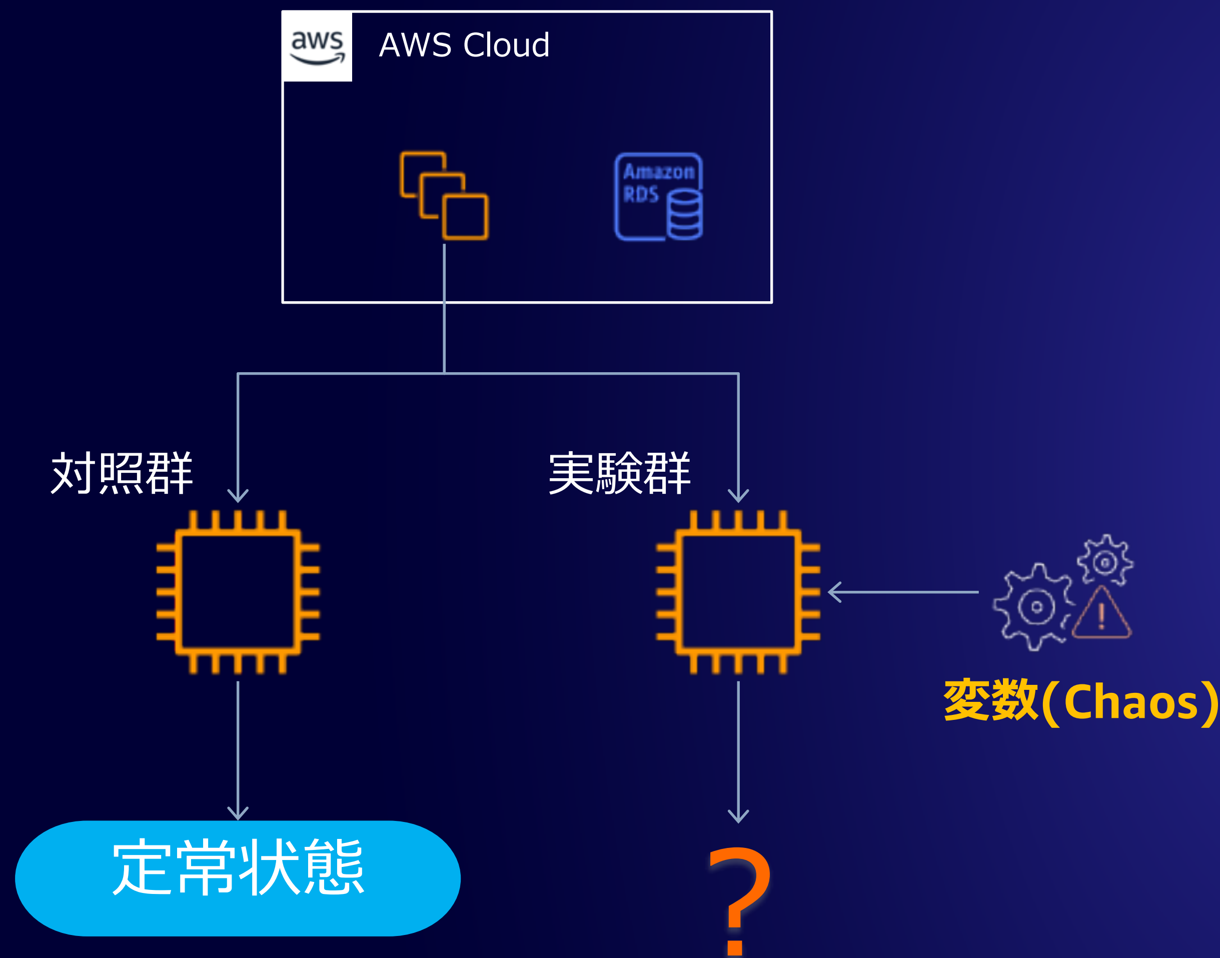
仮説について、関係者と協議することが重要

- 通常のオペレーションで実施しており、何が起きるか明確にわかる
  → 実施不要
- 明らかに問題がある、全く考慮できていなかった
  → 実験をする前に改善する
- 「こうなるはず」だが確証がない
  → **ここがChaos Engineering のターゲット**

aws
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

現実世界を反映する変数を導入し実験する





変数として取り得る値

- サーバダウンなどのHW障害
- 不正な応答などのSW障害
- トラフィックの急増 などのイベント

etc (**実世界の事象は多様である**)

優先度を考える

「発生頻度」と「潜在的な影響範囲」

例) リージョン障害とAZ障害とインスタンスのダウンの頻度と影響範囲
→ワークロードは仮想マシン?コンテナ?

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

変数の投入方法の選択肢

インフラレイヤー

- 導入しやすい
- 影響範囲のコントロールが難しい

例) EC2 インスタンスを停止する
DB をフェイルオーバーする

アプリレイヤー

- 導入コストがかかる
(アプリ改修が必要なことが多い)
- 影響範囲を細かく制御できる

例)特定のリクエストへのレスポンスを
一定時間遅延させる

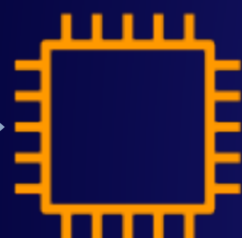
aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

同じ結果になる非効率な実験に注意

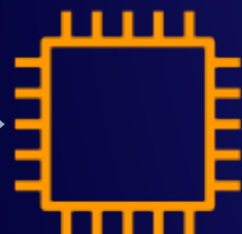
異常の多くはレイテンシの増大
として現れる傾向がある

実験①：レイテンシを注入



unhealthy

実験②：CPU使用率を100%に



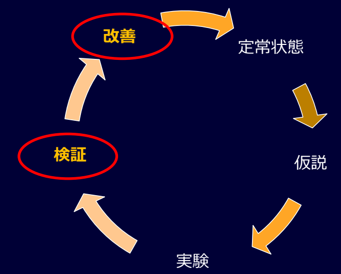
unhealthy

Application Load
Balancer

システム全体の
振る舞いは
どちらも同じ結果

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

検証し、必要なら改善する



起こったことを記録する

- 書記役(記録係)をアサインし、実施したことを記録する
  - タイムスタンプも重要
  - チャットツールなどに随時書き込んでいく方法も有効
  - 特に、「仮説通りにいかなかったこと」は記録する

仮説の反証を試みる

- 反証された場合：「学び」をもとにシステムの改善を計画する
- 反証されない場合：システムの安定性への「自信」に
  - オペレーションの観点などで改善の余地があればそれも取り入れる

「誰も責めない」文化

- 参加者が全員が率直に意見を出し合えるように
- 「障害」は原因に関わらず誰か個人の責任ではない

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

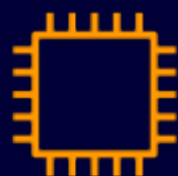
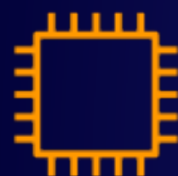
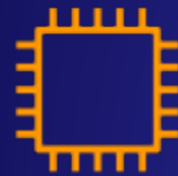
4.影響を最小限にするために

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

影響を最小化する

よくある誤解

Chaos Engineering
= **本番環境**で**ランダム**にインスタンスを**停止**

Stop

Chaos Engineering の目的

予測困難な分散システムの動作を
制御された検証によって学ぶこと

- 仮説に基づいて実験する
- 意図しないユーザへの影響は可能な限り防ぐ

Minimize Blast Radius

aws

本番環境での実施は必須なのか

本番環境で検証を実行する
システムは環境やトラフィックパターンによって異なる動作をします。利用状況は常に変わるので、実際のトラフィックを抽出することが、確実に要求経路を捕捉する唯一の方法になります。システムの実行方法の信頼性と現在の導入済みシステムとの関連性の両方を保証するため、カオスは本番環境のトラフィック上で直接検証することを強くお勧めします。
カオスエンジニアリングの原則 https://principlesofchaos.org/ja/

本番環境で実施する意義

本番環境で実施する際の課題
- 意図しないユーザ影響
- 再現実験の難しさ
- 心理的障壁

STG/検証環境からはじめる
- 妥当なコストで再現できる構成
- (特に不慣れな場合) 安全な環境でツールや実験内容に慣れる
- 本番環境での実験に臨むための自信を構築する

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

社内イベントとしてはじめて徐々に自動化する

まずは社内のイベントとして実施する

- 何を/何のためにするのか理解を得る
- 想定外の事態に十分対応できる用意をする

社内イベントとして実施する際の考慮点

- **関係者を広く招集する**
  開発者、運用者、SRE、
  ビジネスオーナー、各領域の専門家 etc
- **開始前/後にアナウンスを行う**
- **実際のトラブルが発生した場合直ちに停止する**
- **徐々に自動化していく**

タイムテーブルのサンプル

時間	実施内容
13:00	集合、録画開始
13:05	実施内容の確認、修正
13:30	開発環境での開始をアナウンス
13:35	開発環境に変数を導入
13:45	エラー通知を取得、対応開始
14:00	開発環境の実験を終了、アナウンス
14:05	本番環境での実施可否の判断
14:10	本番環境での開始をアナウンス
14:15	本番環境に変数を導入
14:25	エラー通知を取得、対応開始
14:40	本番環境の実験を終了、アナウンス
14:45	振り返り

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

いつでも止められるようにする

本番環境での別のイベント発生

仮説とは違う挙動

想定外のユーザ影響

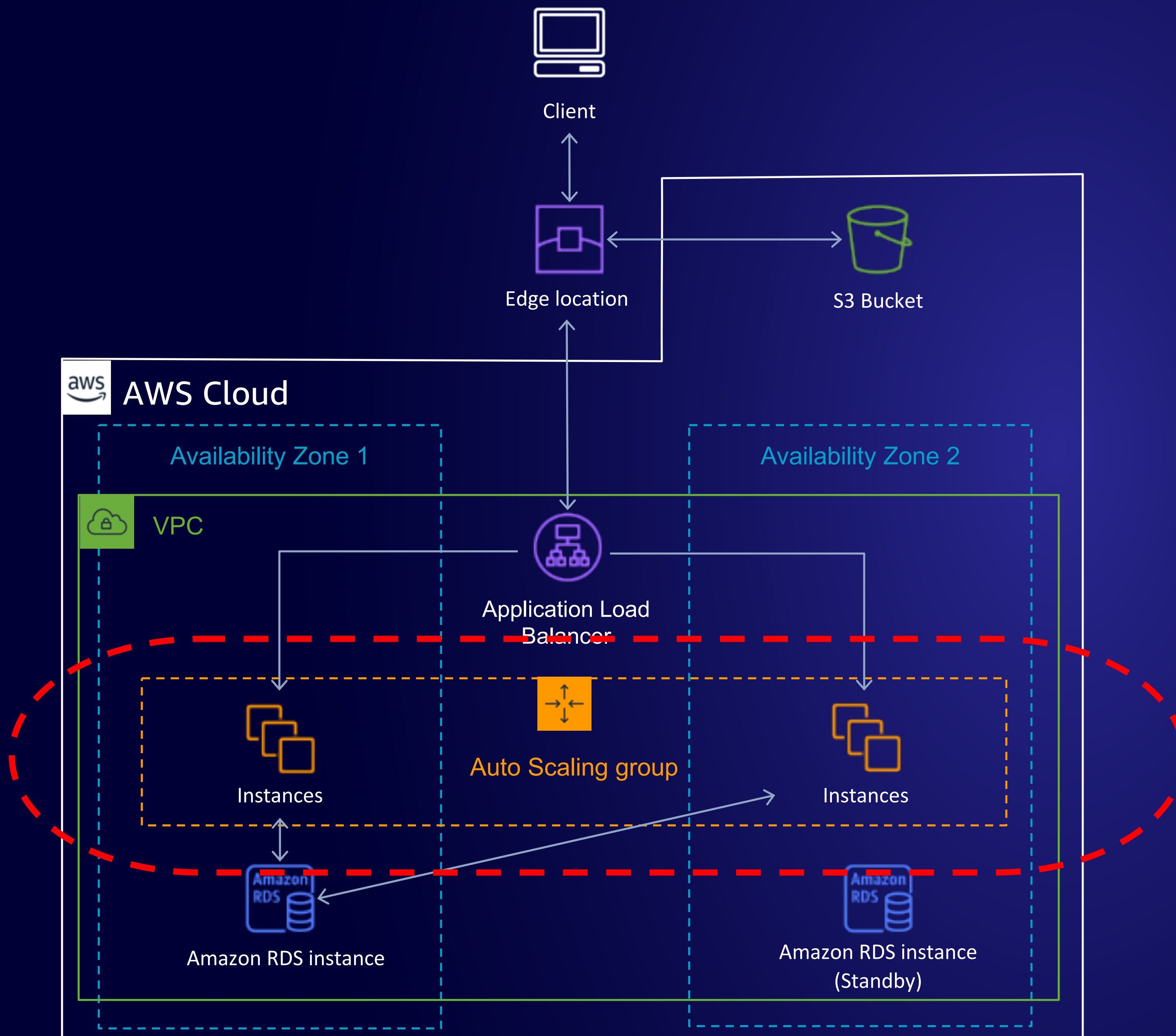
aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

5. Chaos Engineering の具体例

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

サンプル構成



一般的なWeb 三層のアーキテクチャ

CDN としてAmazon CloudFront を利用

静的なコンテンツはAmazon S3 に配置

Amazon EC2 Auto Scaling を使い、インスタンスの数を一定数に維持(スケールイン/アウトは不使用)

DBはAmazon RDS をマルチAZ で利用

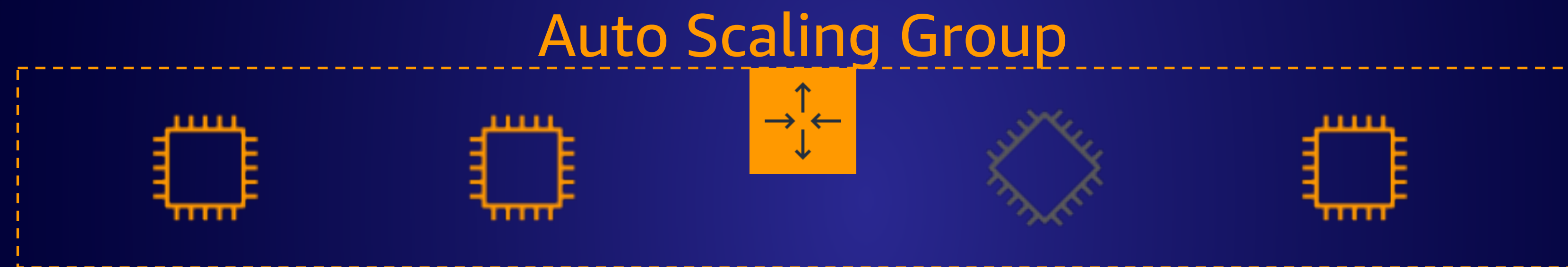
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

サンプル構成への実験の例

Amazon EC2 Auto Scaling に任せておけば大丈夫?

<仮説> EC2 インスタンスが停止してもAuto Scaling によりオートヒーリングされる

<実験>

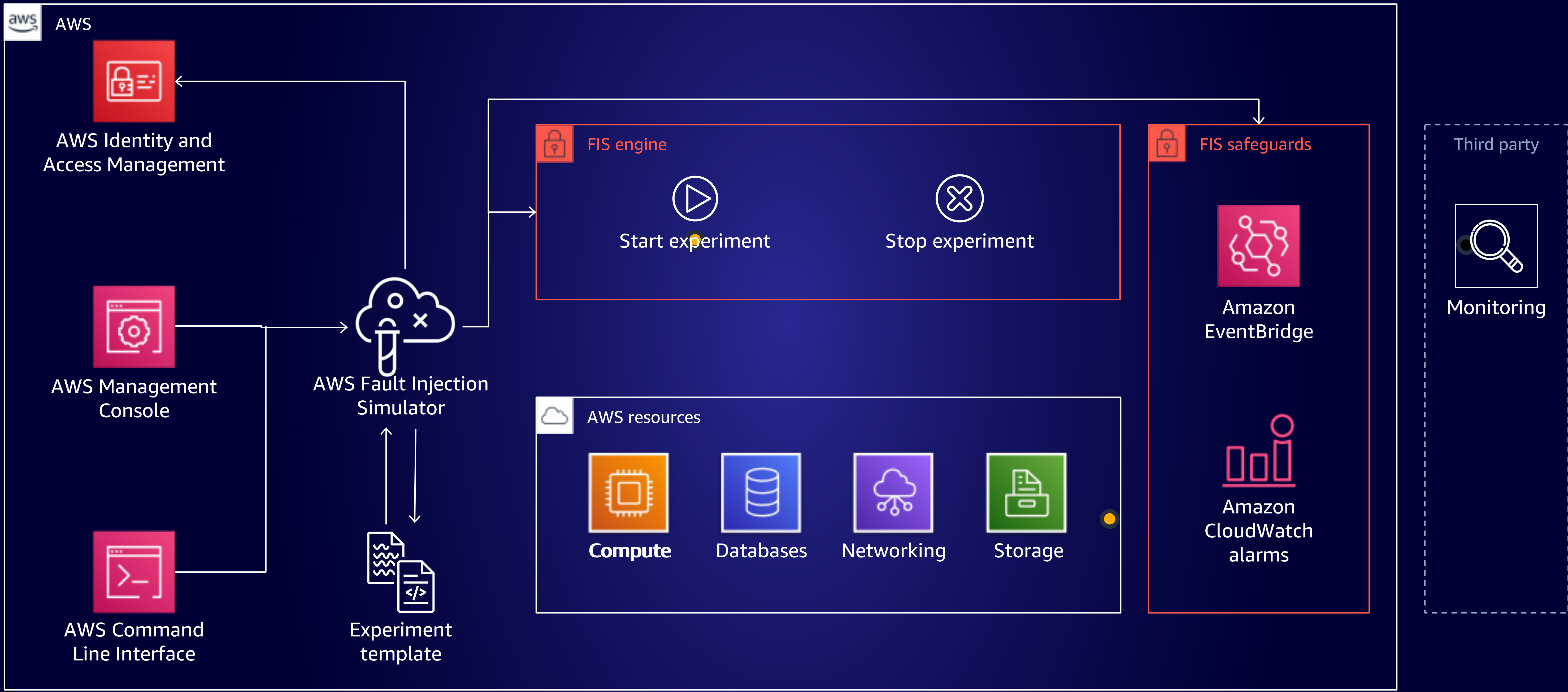


<発展的な例>
- EC2インスタンスは停止させず、アプリケーションのプロセスを停止
- 特定のインスタンスのネットワークを遅延させる
- AWS Systems Manager を使って自動的に実施する

<考えれる原因>
- ユーザーデータやライフライクルフックが陳腐化している
- ヘルスチェックの設定に問題がある etc

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS Fault Injection Simulator



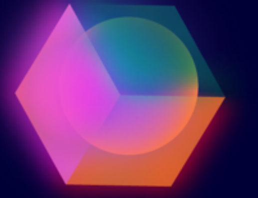
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

サポートする Fault Injection

※2021/03/23時点の情報です

- Server error (EC2)
- Stop, reboot, and terminate instance(s) (EC2)
- API throttling
- Increased memory or CPU load (EC2)
- Kill process (EC2)
- Latency injection (EC2)
- Container instance termination (ECS)
- Increase memory or CPU consumption per task (ECS)
- Terminate nodes (EKS)
- Database stop, reboot, and failover (RDS)
- And more to come in 2021

aws
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.



5. まとめ

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

システムの複雑性に立ち向かうために

- 見つけられる課題(参考:AWS Well-Architected フレームワーク)の解決と、振る舞いの可視化(可観測性)からまずは取り組む
- 定常状態と仮説を定義し、制御した実験を行う
- 小さく初めて広げていく
  - 開発/STG環境での実施
  - 社内イベントとして初めて自動化していく
  - いつでも止められる仕組み

複雑なシステムへの理解を広げることで
システムの信頼性、可用性への自信を構築しましょう

aws

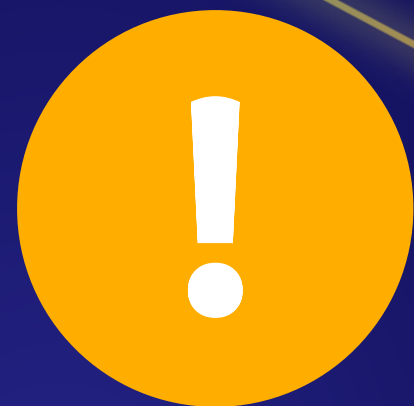
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Thank you!

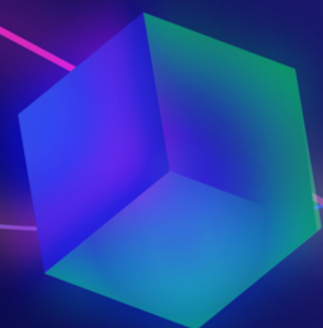
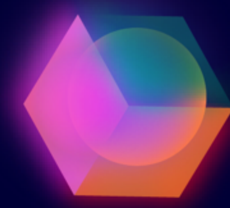
金森政雄

アマゾン ウェブサービス ジャパン 株式会社
ソリューションアーキテクト

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Please complete the session survey

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.