AWS

SUMMIT

AWS Well-Architected フレームワークによる クラウド ベスト プラクティス

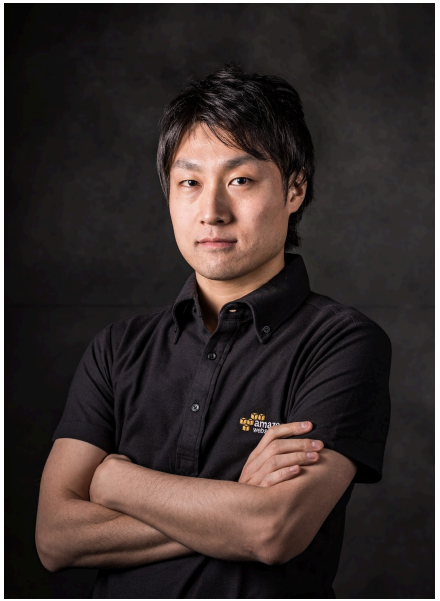
アマゾン ウェブ サービス ジャパン株式会社
ソリューションアーキテクト　畑　史彦

2017/5/31

amazon web services

自己紹介

- **畑　史彦（はた　ふみひこ）**
  - 所属
    - アマゾン ウェブ サービス ジャパン株式会社
      技術統括本部 ソリューション アーキテクト
  - 好きなAWSサービス
    - Amazon WorkDocs



amazon web services | Certified

Solutions Architect - Professional

DevOps Engineer - Professional

アジェンダ

- AWS Well-Architected Framework とは
- クラウドでのアプリケーション設計の考え方
- クラウドでのアプリケーション設計のケーススタディ
- まとめ

※ AWSの個々のサービスの説明は最小限となります

アジェンダ

- AWS Well-Architected Framework とは
- クラウドでのアプリケーション設計の考え方
- クラウドでのアプリケーション設計のケーススタディ
- まとめ

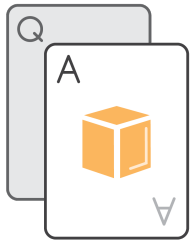
AWSの各種サービスの使い方や疑問

EC2（仮想サーバ）の
状態を簡単に監視したい

RDS（データベースサービス）で
定期的にバックアップを取りたい

様々なリソースやサポートを提供

- ドキュメント、ホワイトペーパー、「よくある質問」
- コミュニティフォーラム、ユーザーコミュニティ
- テクニカルサポート、トレーニング
  etc...



サービスの使い方は分かっても
それらをどう組み合わせて、どう設計すればいい？

クラウドに移行したのに、
以前と変わらぬ手運用、
もっと楽にならないかな…

果たして今の構成で
何かあったとき本当に
大丈夫なんだろうか…

あ、新しいサービスが出てる・・・
このシステムは、適切なサービスや
機能を選択できているんだろうか

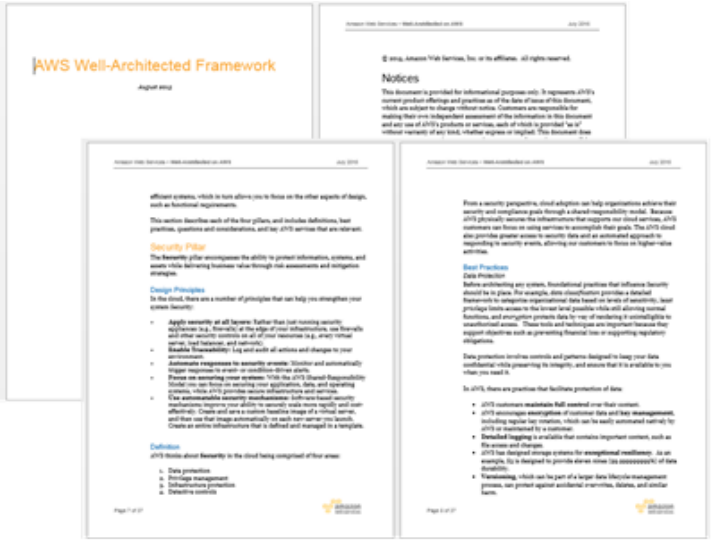
全てを自社で管理してた
オンプレミスと違って
クラウドのセキュリティは特殊？

正確なサーバの容量の
見積もりって周りはみんな
どうしてるんだろう

AWS Well-Architected Framework

- クラウドのアプリケーション設計における考え方とベストプラクティス

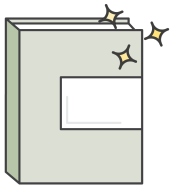
- アーキテクチャを評価するための一貫したアプローチ
  - 質問、評価、そして対処法



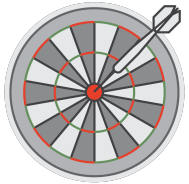

“Are you Well-Architected?"
Werner Vogels

Amazon CTO Werner Vogels @ re:Invent2015

クラウドはITインフラを物理環境の制約から
解き放ち、完全にコントロール可能にする



Well-Architected は、10年以上にわたる
これまでのAWSの経験を投入して作成、
継続的に更新

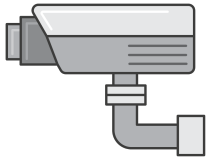
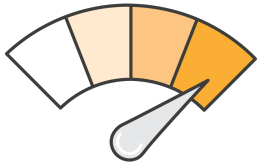


場当たり的な対応ではなく、新たな価値を追加
することにフォーカスできるようになる

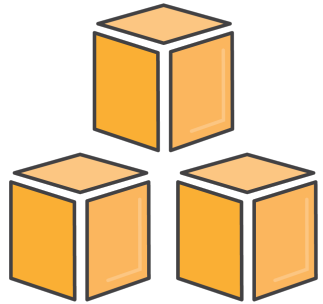
AWS Well-Architected Framework の構成要素

1. 5 つの柱
2. 設計原則
3. 質問事項

1. 5つの柱

セキュリティ	信頼性	パフォーマンス効率	コスト最適化	運用性
				

2. 設計原則



AWS Well-Architected Framework では、
クラウドにおける適切な設計を可能にする
設計原則を明確にしています

- **一般的な設計原則**
    - 「必要な容量の判断を勘に頼らない」
    - 「本番のスケールでテストする」etc...
- **５つの柱ごとの設計原則**
    - **セキュリティの設計原則**：「最小特権の原則の適用」
    - **運用性の設計原則**：「変更は日々、小さく、継続的に」etc...

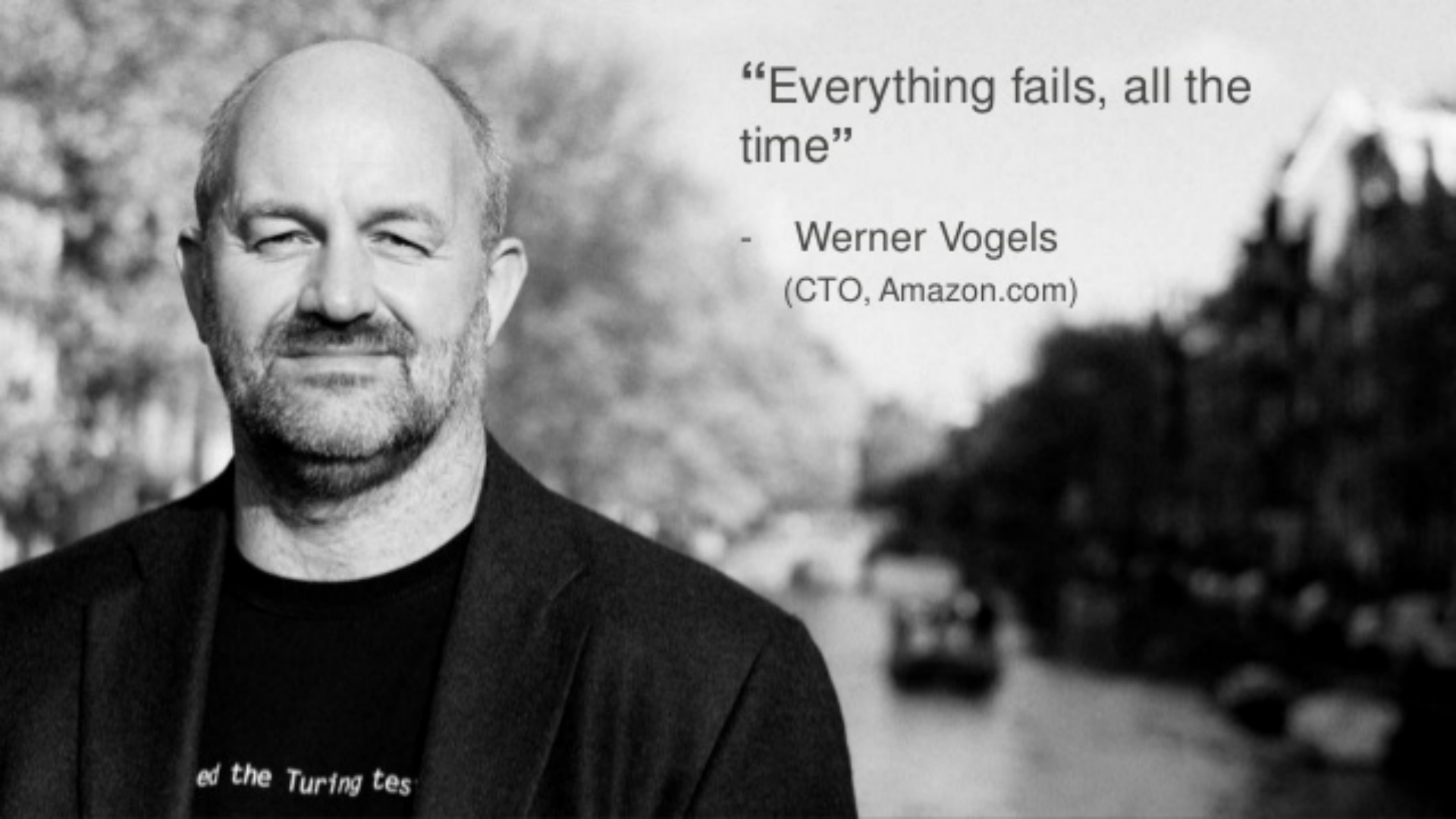
3. 質問事項

- 質問に従って自問することでシステムを
  レビューすることができる
  - 「想定外の運用イベントに、どのように対応していますか？ 」
  - 「パフォーマンスを改善するためにどのようなトレードオフを行っていますか？ 」
    etc...

- 忘れられがちな基本的領域にも対応
  - 「どのようにしてルートアカウントでのログインを保護していますか？ 」
  - 「AWS のコストをどのように管理していますか？ 」
    etc...

アジェンダ

- AWS Well-Architected Framework とは
- クラウドでのアプリケーション設計の考え方
- クラウドでのアプリケーション設計のケーススタディ
- まとめ


“Everything fails, all the time”
- Werner Vogels
(CTO, Amazon.com)
ed the Turing tes

Everything fails all the time

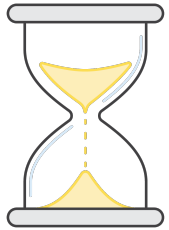
- あらゆるシステムコンポーネントは、ダウンする
- 個々のコンポーネントがダウンしないように
  するのではなく（だけではなく）
- 個々のコンポーネントがダウンしても
  - 自動で検知し、復旧するように
  - アーキテクチャ全体がダウンせずに稼働し続けるように

障害を**未然に防止**しようとするのではなく

障害を**自動で検知**し
障害が**自動で対処**される仕組みを設計する

MTTF and MTTR

$$\text{Availability} := \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$



MTTF: 平均故障時間
MTTR: 平均修復時間

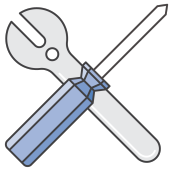
https://www.slideshare.net/ufried/resilience-reloaded-more-resilience-patterns#7 より

- **MTTF（平均故障時間）** をより長くしようとする
  → **障害を未然に防止**しようとするアプローチ
- **MTTR（平均修復時間）** をより短く、あるいは 0 にしようとする
  → **障害を前提に対処**しようとするアプローチ

Not only availability...

セキュリティも、運用も、開発も考え方は変わらない

- いかなるセキュリティ対策も、予想外の攻撃や
  運用ミスを完全に排除するのは難しい
  → **問題がすぐに検知され、場合によっては自動で対処されるように**



- どんなに整備された運用体制でもヒューマンエラーは発生しうる
  → **手作業をコード化、復旧作業もコード化**
  → **想定外のイベントをテスト**

- あらゆる状況を考慮してテストしたとしてもバグの可能性は消えない
  → **問題があってもすぐに修正改善がリリースできる環境**
  → **テストとテスト作成を自動化**

AWS が提供する基本的なバリュー

障害を自動で検知し自動で対処するシステムを
実現するためのクラウドの基本的なバリュー

1. 俊敏性と弾力性

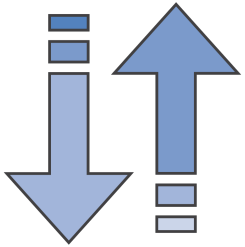
2. グローバルインフラストラクチャ

3. 多様なサービス群

1. クラウドがもたらす俊敏性と弾力性

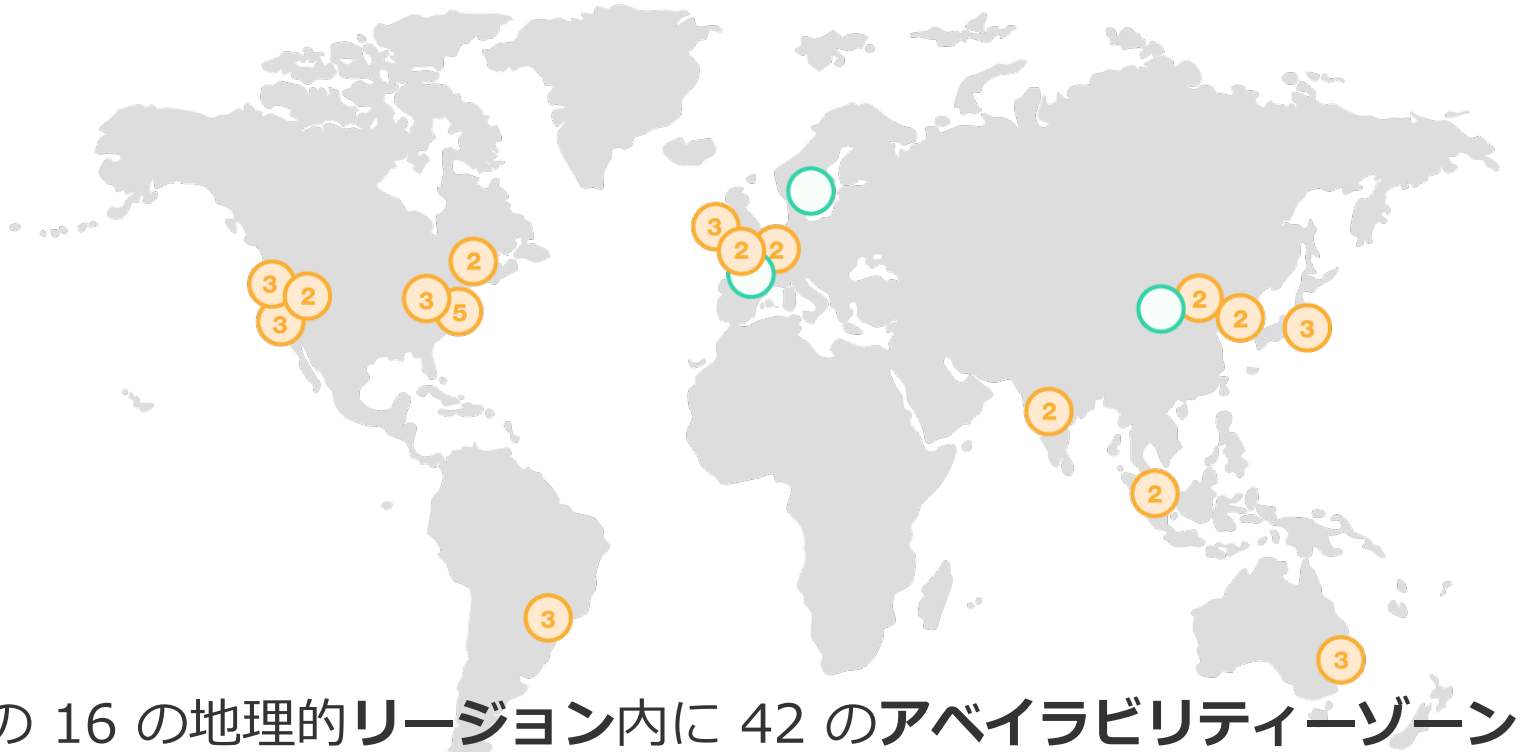
- **俊敏性**
  - わずか数分で数千のコンピューティングリソース

- **弾力性**
  - 必要なときに、必要なだけのリソースを利用
  - 使った分にだけ支払い



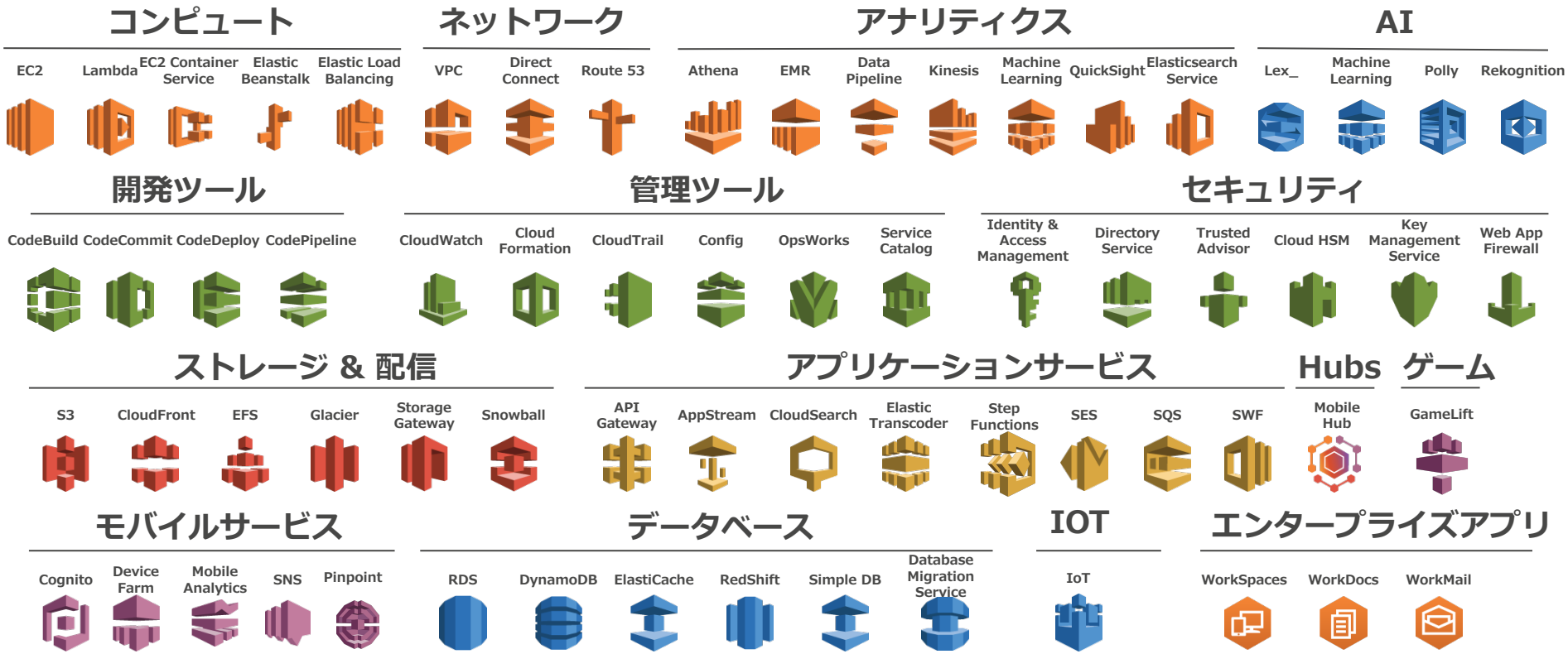
すぐに試せる、作れる、やめられる

2. グローバルインフラストラクチャ



世界中の 16 の地理的**リージョン**内に 42 の**アベイラビリティーゾーン（AZ）**、
今後、さらに 3 つのリージョンと 8 つのAZが追加予定

3. 多様なサービス群


コンピュート
EC2
Lambda
EC2 Container Service
Elastic Beanstalk
Elastic Load Balancing
ネットワーク
VPC
Direct Connect
Route 53
アナリティクス
Athena
EMR
Data Pipeline
Kinesis
Machine Learning
QuickSight
Elasticsearch Service
AI
Lex_
Machine Learning
Polly
Rekognition
開発ツール
CodeBuild
CodeCommit
CodeDeploy
CodePipeline
管理ツール
CloudWatch
Cloud Formation
CloudTrail
Config
OpsWorks
Service Catalog
セキュリティ
Identity & Access Management
Directory Service
Trusted Advisor
Cloud HSM
Key Management Service
Web App Firewall
ストレージ & 配信
S3
CloudFront
EFS
Glacier
Storage Gateway
Snowball
アプリケーションサービス
API Gateway
AppStream
CloudSearch
Elastic Transcoder
Step Functions
SES
SQS
SWF
Hubs
Mobile Hub
ゲーム
GameLift
モバイルサービス
Cognito
Device Farm
Mobile Analytics
SNS
Pinpoint
データベース
RDS
DynamoDB
ElastiCache
RedShift
Simple DB
Database Migration Service
IOT
IoT
エンタープライズアプリ
WorkSpaces
WorkDocs
WorkMail

アジェンダ

- AWS Well-Architected Framework とは
- クラウドでのアプリケーション設計の考え方
- クラウドでのアプリケーション設計のケーススタディ
- まとめ

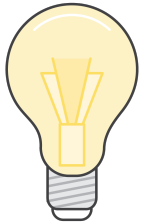
障害を前提に対処する設計例

1. **Auto Scaling** ー 自動スケールアウト
2. **Automatic Feedback Control** ー フィードバックの自動制御
3. **Continuous Delivery & Test Automation** ー 開発サイクルとテストの自動化
4. **Game-day Testing** ー 本番を想定したテスト
5. **Error Injection** ー 障害の注入

1/5 Auto Scaling
自動スケールアウト

Auto Scaling

新しくアプリケーションを構築する際に、
どのようにリソース需要を予測し、最適化
するかを考えます

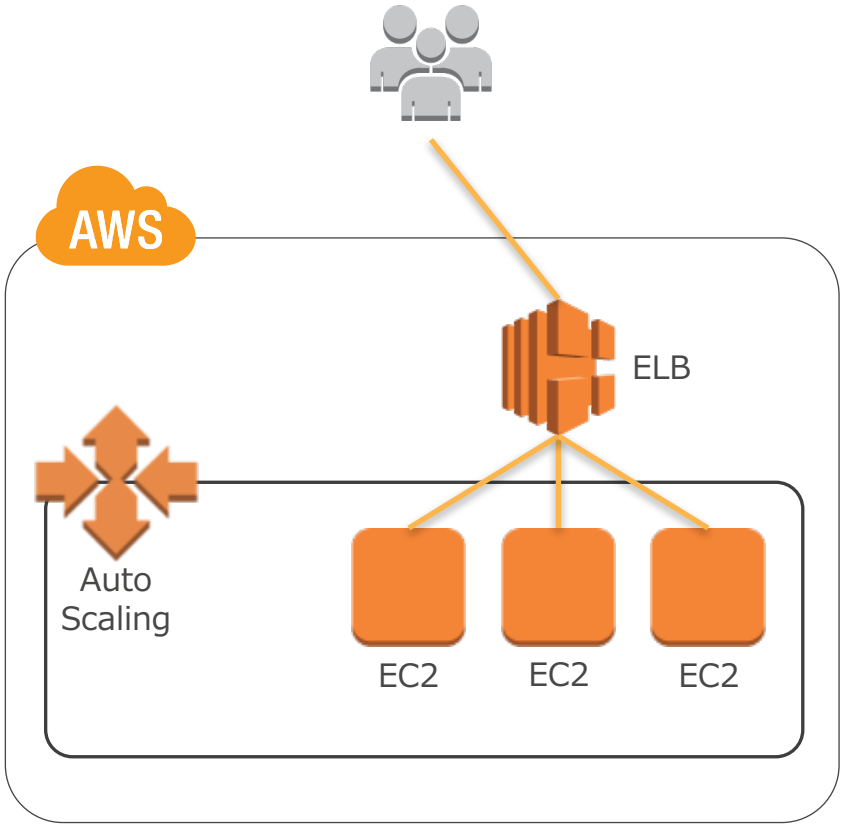


リソース不足を、事前に
防止しようとする考え方

- 必要となるキャパシティを事前に入念に計算し、さらにそれに余裕をもってリソースを確保する
  - → **その時その時のリソース需要に応じて**
    キャパシティを自動的に拡張あるいは縮小させる
  - → **可用性の維持**

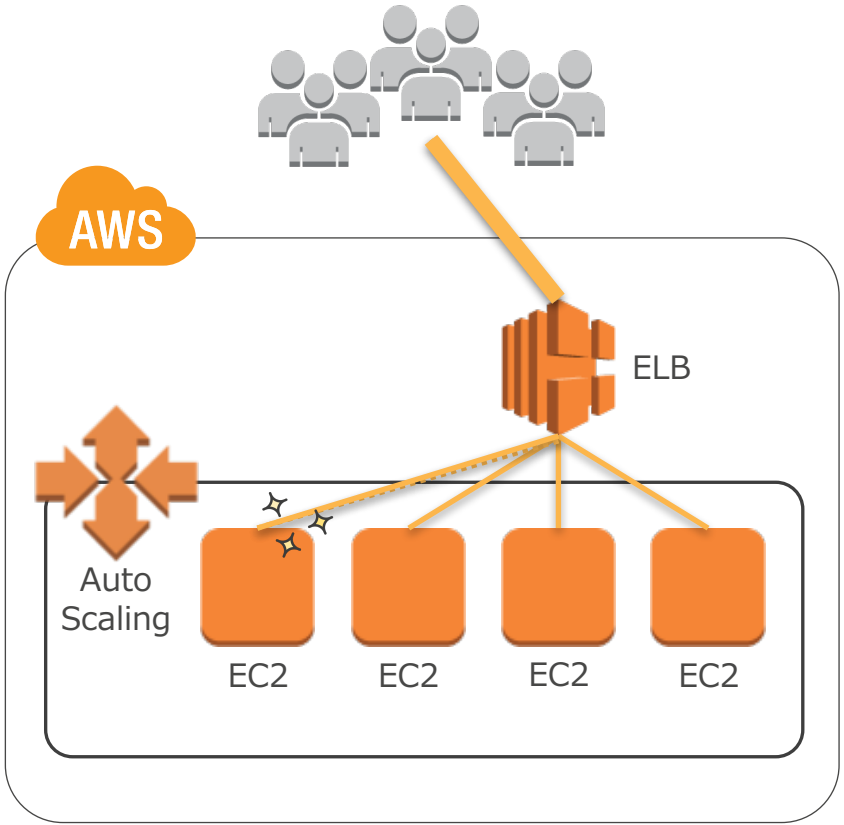
リソースの変動を前提とし
自動で対処しようとする考え方

Auto Scaling



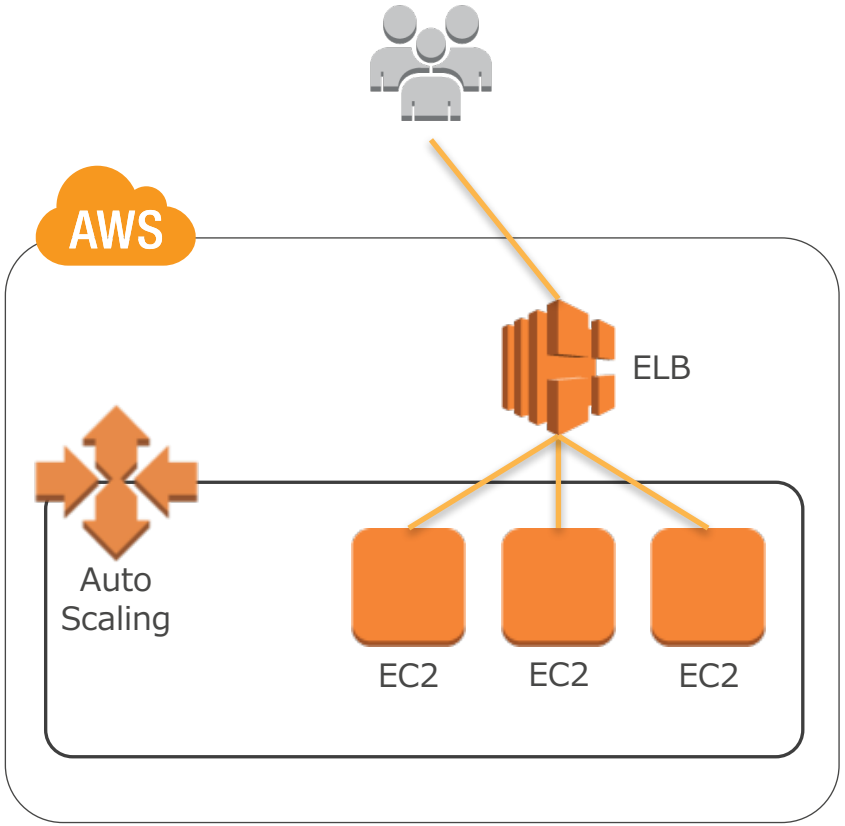
- **Amazon EC2:**
  仮想サーバリソース。多様なスペック・OS をサポートし、1時間単位の従量課金。
- **Auto Scaling:**
  定義された条件に応じて Amazon EC2 のキャパシティを自動的に縮小あるいは拡張する機能。
- **ELB (Elastic Load Balancing):**
  ロードバランサー。高い可用性、自動的なスケーリング、および強固なセキュリティが特徴。

Auto Scaling



- **Amazon EC2:**
  仮想サーバリソース。多様なスペック・OSをサポートし、1時間単位の従量課金。
- **Auto Scaling:**
  定義された条件に応じて Amazon EC2 のキャパシティを自動的に縮小あるいは拡張する機能。
- **ELB (Elastic Load Balancing):**
  ロードバランサー。高い可用性、自動的なスケーリング、および強固なセキュリティが特徴。

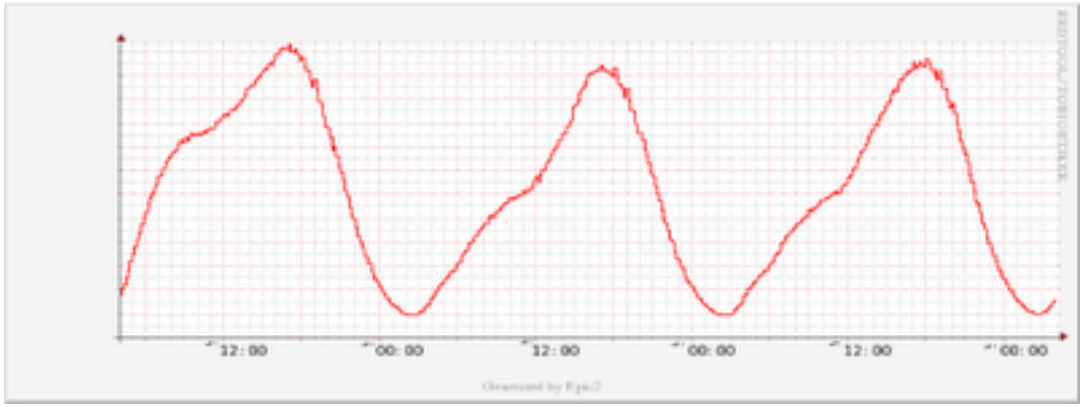
Auto Scaling



- **一般的な設計原則**
  - 必要な容量の判断を勘に頼らない
- **信頼性の設計原則**
  - 容量の推測をやめる
  - 水平スケーリングによる可用性の向上
- **コスト最適化の設計原則**
  - 消費したリソースに対して支払う

Auto Scaling @ NETFLIX

Request



Instances

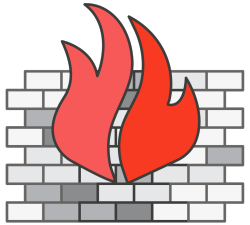
http://techblog.netflix.com/2012/01/auto-scaling-in-amazon-cloud.html

2/5 Automatic Feedback Control

フィードバックの自動制御

Automatic Feedback Control

WAF (Web Application Firewall) によって、
アプリケーションレイヤーの保護を設定する
ユースケースを考えます



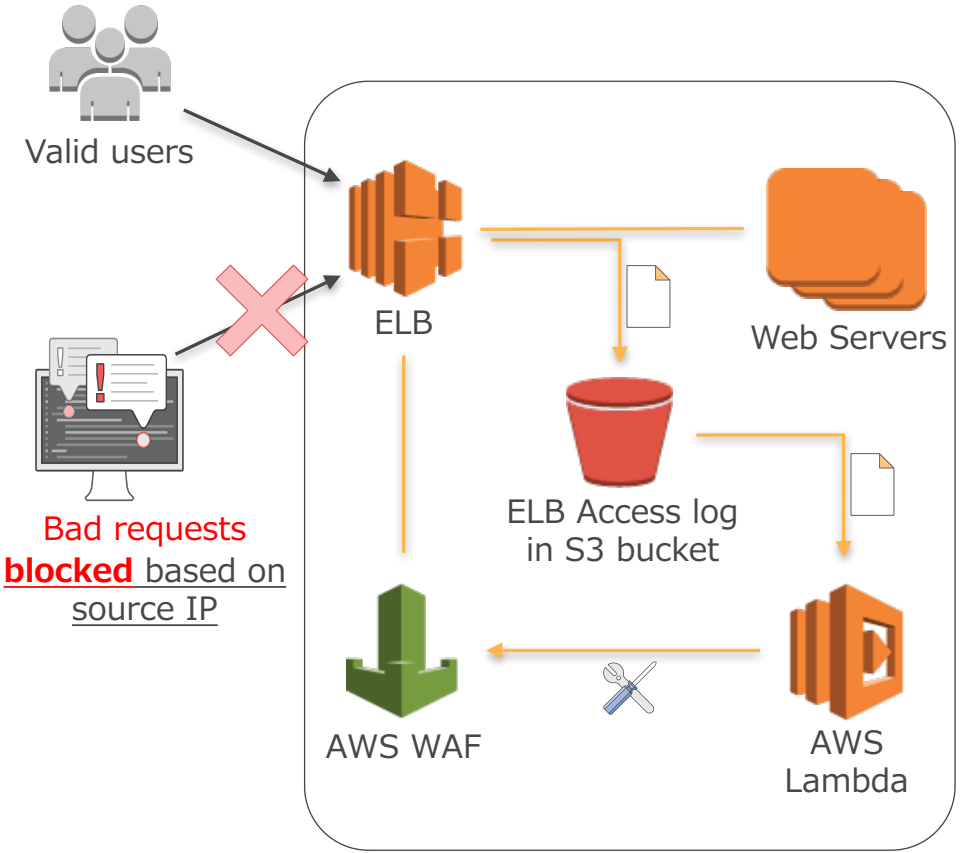
攻撃を未然に防止
しようとする考え方

- ブロックすべき対象の条件を入念に準備する

  → **アクセスログを継続的に解析、集計し、不審な挙動のIPアドレスをブロック対象として動的にWAF の設定を更新する**

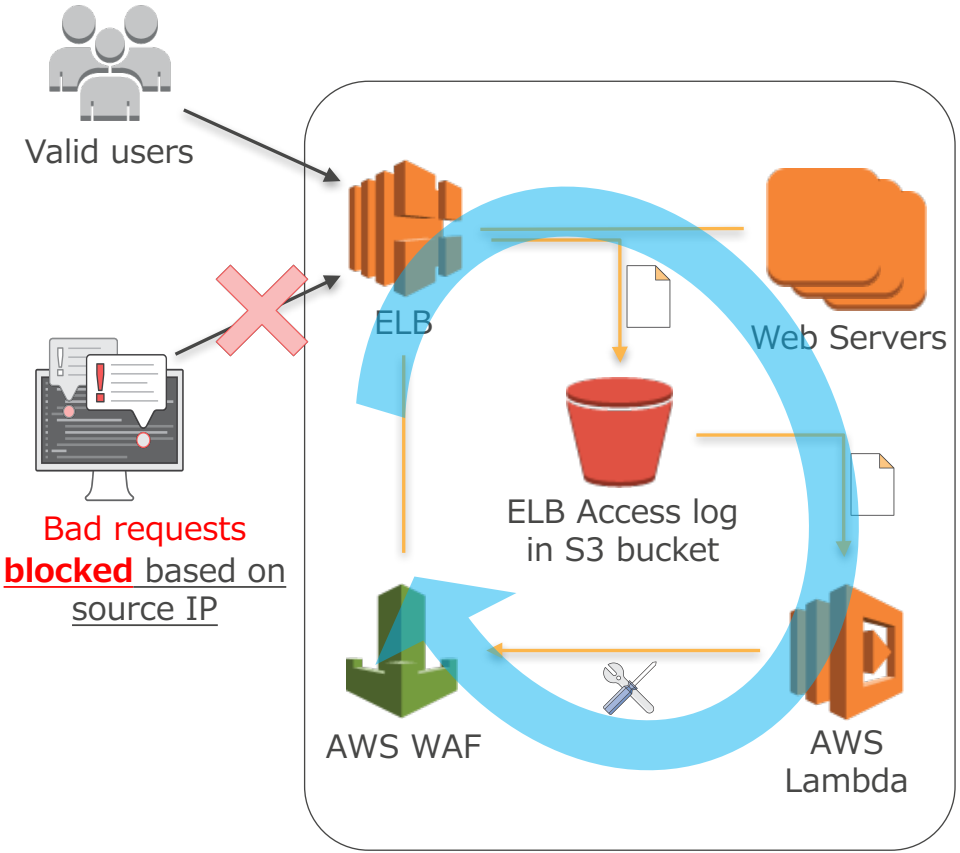
想定外の攻撃の存在を前提とし
自動で対処しようとする考え方

Automatic Feedback Control



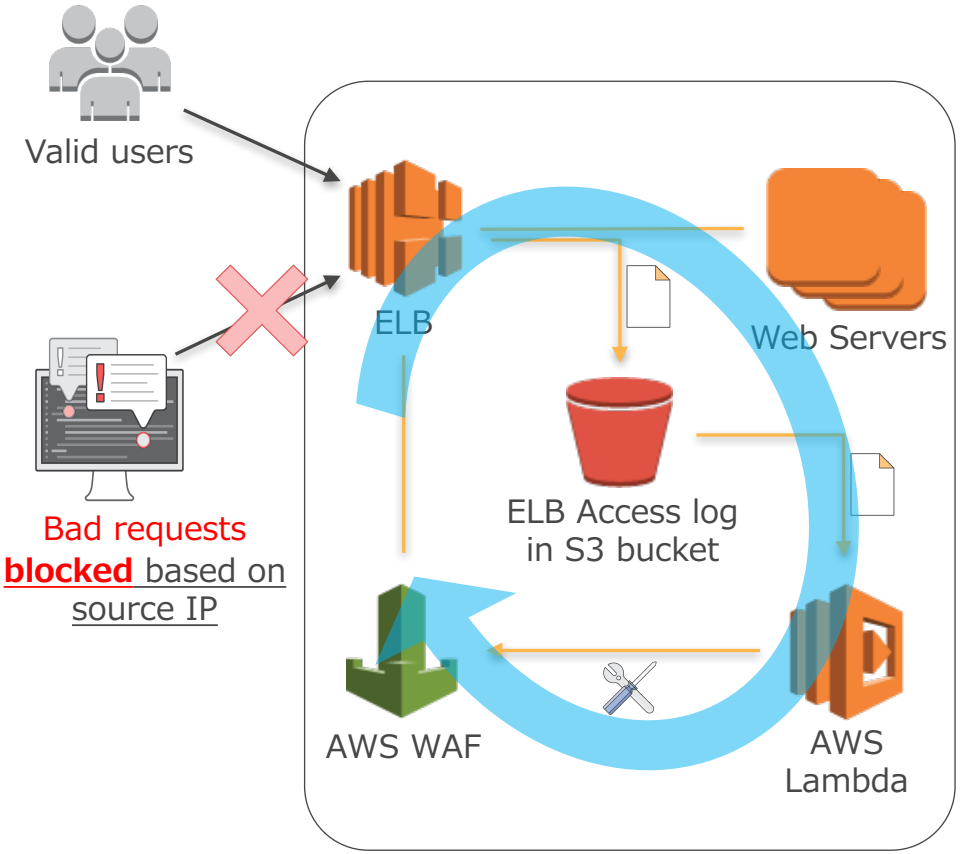
- **AWS WAF:**
  ウェブアプリケーションファイアウォール。
  API を使用して完全に管理可能。
- **Amazon S3:**
  マネージドのクラウドストレージ
  サービス。容量無制限、高可用、そして高い
  データ耐久性。
- **AWS Lambda:**
  サーバレスコンピューティングリソース。
  サーバーのプロビジョニングや管理なしで
  コードを実行。

Automatic Feedback Control



- **AWS WAF:**
  ウェブアプリケーションファイアウォール。
  API を使用して完全に管理可能。
- **Amazon S3:**
  マネージドのクラウドストレージ
  サービス。容量無制限、高可用、そして高い
  データ耐久性。
- **AWS Lambda:**
  サーバレスコンピューティングリソース。
  サーバーのプロビジョニングや管理なしで
  コードを実行。

Automatic Feedback Control



- **セキュリティの設計原則**
  - トレーサビリティの有効化
- **パフォーマンス効率の設計原則**
  - サーバーレスアーキテクチャ

3/5 Continuous Delivery & Test Automation

開発サイクルとテストの自動化

Continuous Delivery & Test Automation

アプリケーションの品質を向上させるための
開発手法について考えます

できるだけテスト漏れを
防ごうとする考え方

1. 問題が起きないように時間をかけて慎重に
   開発、テスト、デプロイを実施する
   → **問題が発生しても、修正、テスト、デプロイが**
   即座に可能な開発プロセスを整備する

失敗を前提に対処
する考え方

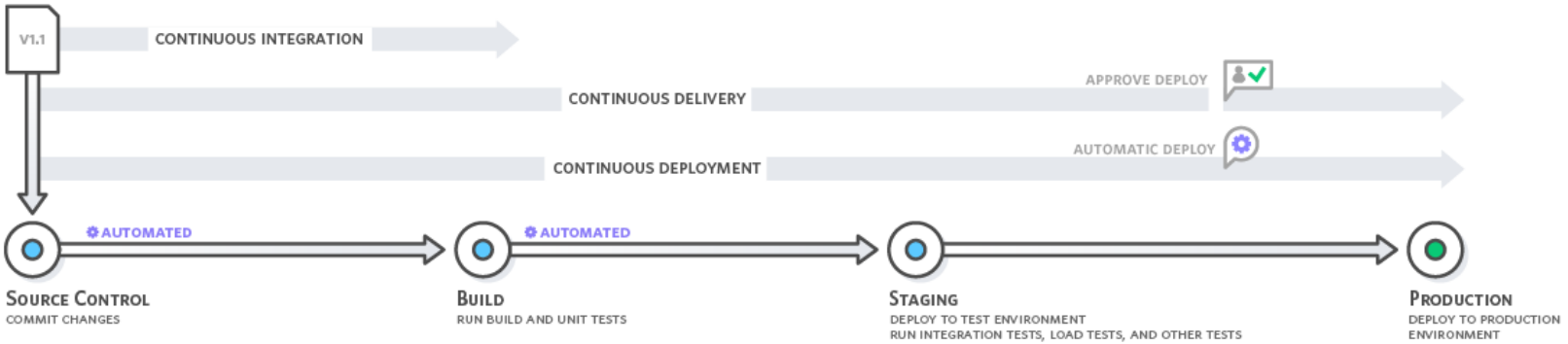
2. バグを出さないように大量の単体テストを書く
   → **テストケースの生成を自動化する**

失敗を未然に防止
する考え方

抜け漏れを前提に対策
をしようとする考え方

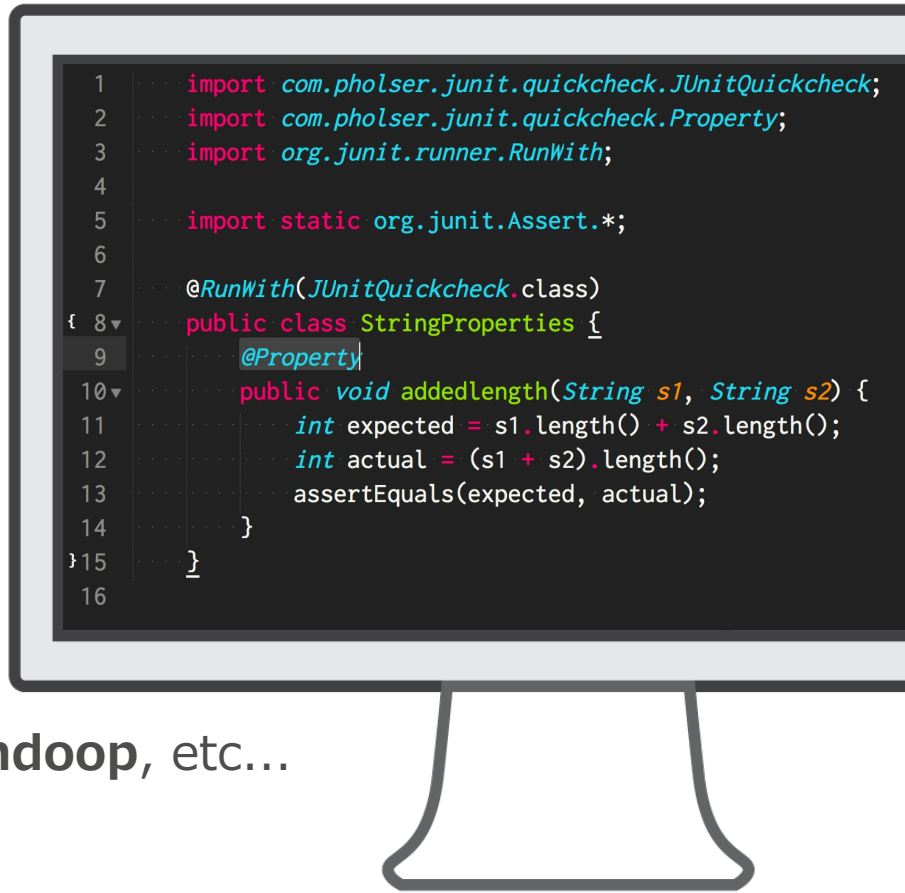
1. Continuous Delivery (CD)

- コード変更が発生すると、自動的にビルド、テスト、および本番へのデプロイ準備が実行される開発手法
- デプロイ準備の整ったビルドの成果物を常に手元に保持
- 迅速な開発サイクル



2. Property-based testing

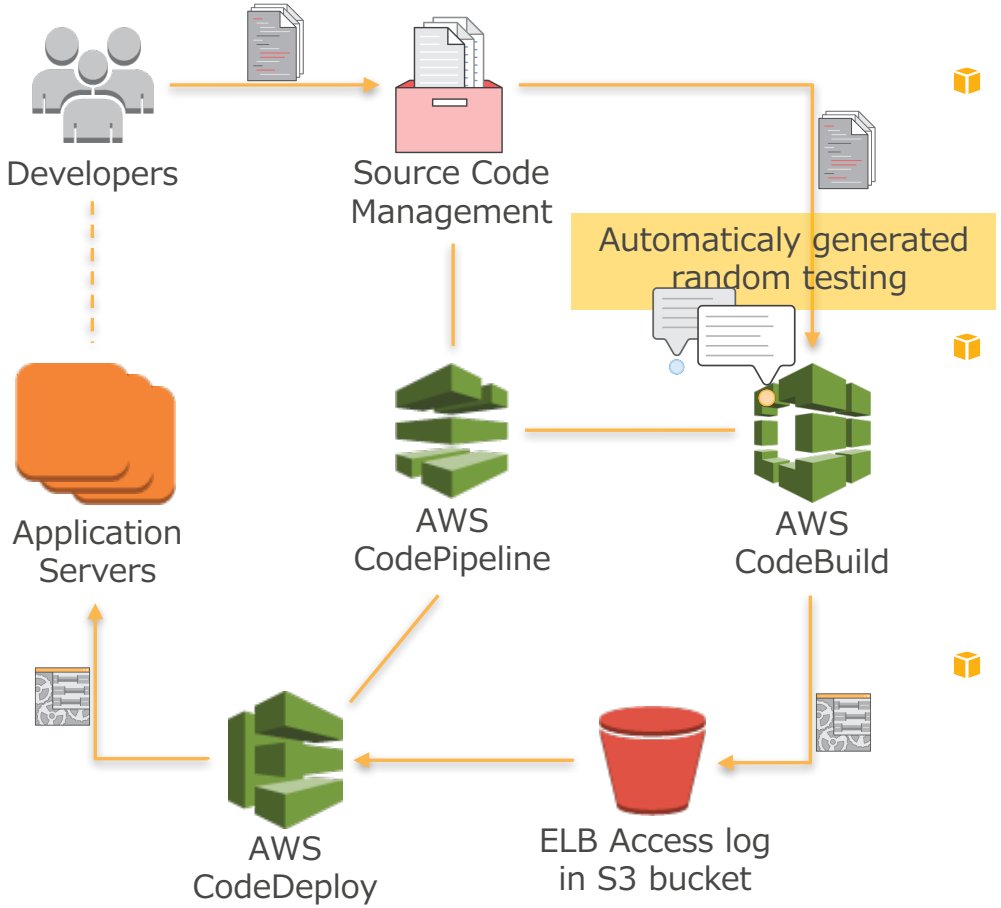
- 定義された性質(property)を満たすかを検証するために、テストケースをランダムに自動生成し実行する
- 人手では書けない大量かつ多様なテストケース
- Tools
  - Haskell **QuickCheck** (1999~)
  - Java　**junit-quickcheck**, **Randoop**, etc…



```
1  import com.pholser.junit.quickcheck.JUnitQuickcheck;
2  import com.pholser.junit.quickcheck.Property;
3  import org.junit.runner.RunWith;
4
5  import static org.junit.Assert.*;
6
7  @RunWith(JUnitQuickcheck.class)
8  public class StringProperties {
9      @Property
10     public void addedlength(String s1, String s2) {
11         int expected = s1.length() + s2.length();
12         int actual = (s1 + s2).length();
13         assertEquals(expected, actual);
14     }
15 }
16
```

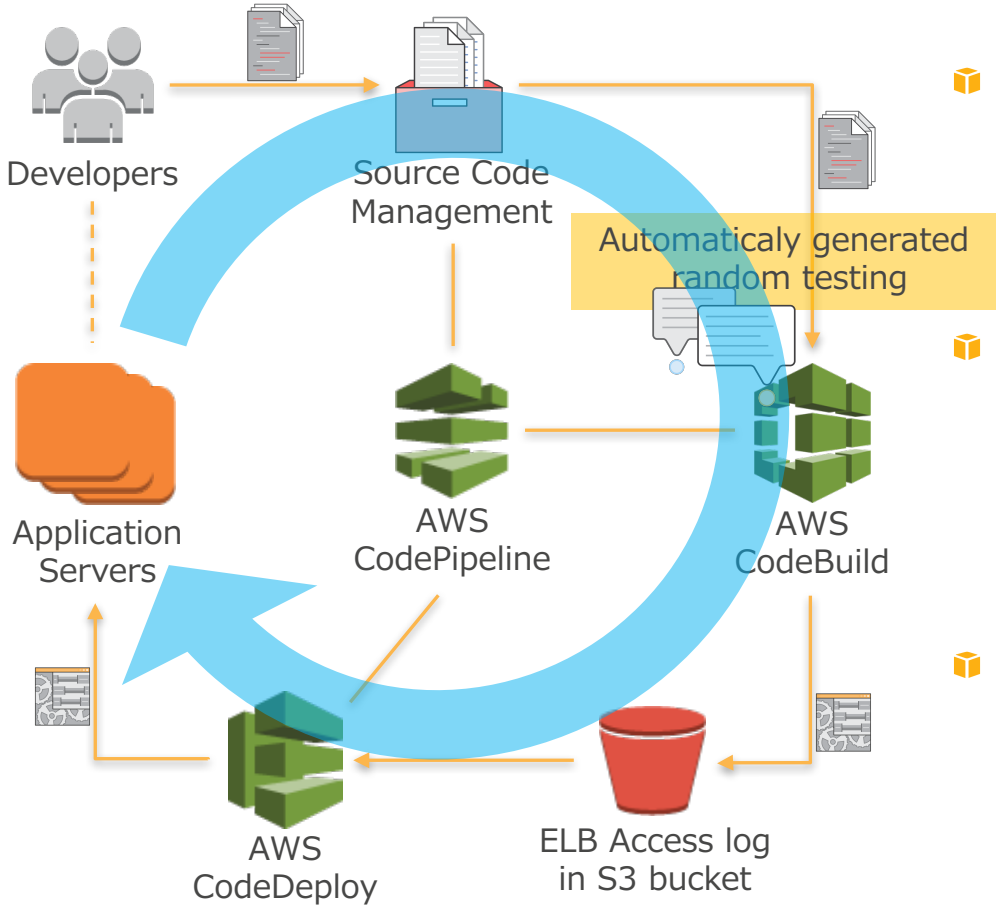
http://www.cse.chalmers.se/~rjmh/QuickCheck/
https://github.com/pholser/junit-quickcheck/
https://github.com/randoop/randoop

Continuous Delivery & Test Automation



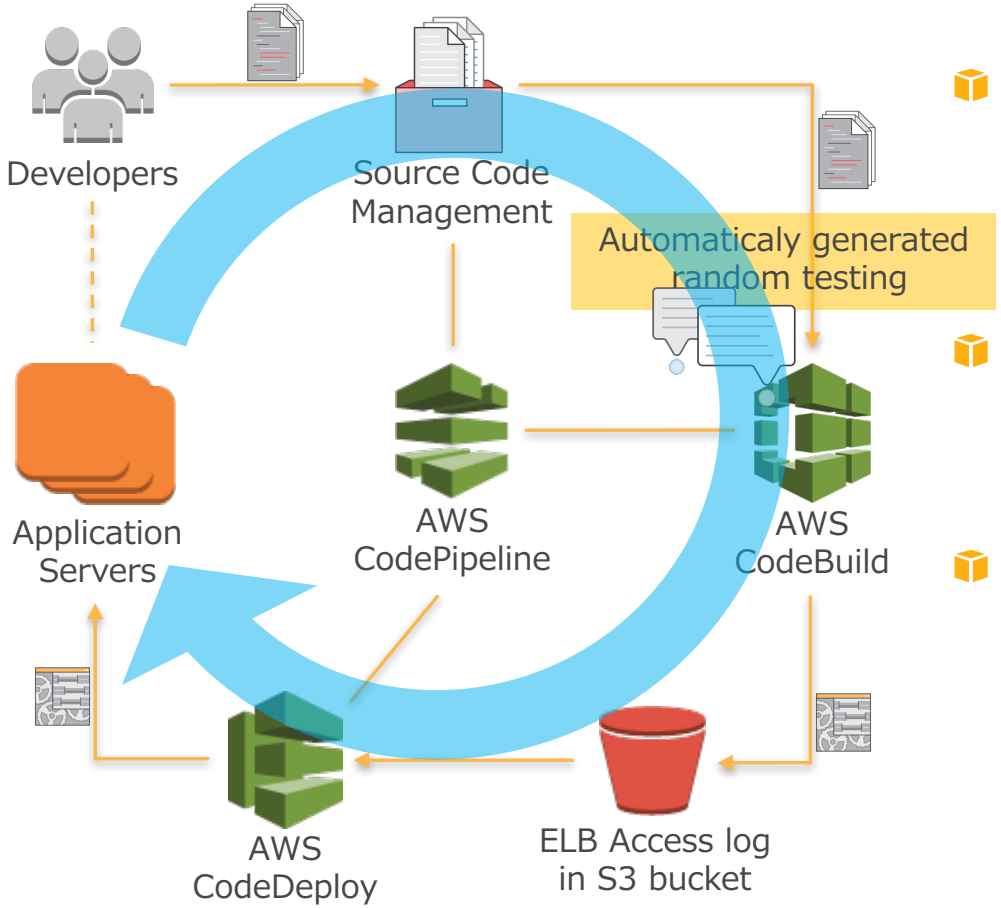
- **AWS CodePipeline:**
  継続的デリバリーのサービス。コードの変更のたびに速やかに、ビルド、テスト、デプロイなど一連のプロセスを実行。
- **AWS CodeBuild:**
  ソフトウェアパッケージのビルドサービス。コードをコンパイルし、テストを実行し、デプロイできるパッケージを作成、という一連のステップをスケーラブルに自動化。
- **AWS CodeDeploy:**
  アプリケーションのデプロイのサービス。コンソール、CLI、SDK、API からデプロイを一元管理するとともに、完全に自動化。

Continuous Delivery & Test Automation



- **AWS CodePipeline:**
  継続的デリバリーのサービス。コードの変更のたびに速やかに、ビルド、テスト、デプロイなど一連のプロセスを実行。
- **AWS CodeBuild:**
  ソフトウェアパッケージのビルドサービス。コードをコンパイルし、テストを実行し、デプロイできるパッケージを作成、という一連のステップをスケーラブルに自動化。
- **AWS CodeDeploy:**
  アプリケーションのデプロイのサービス。コンソール、CLI、SDK、API からデプロイを一元管理するとともに、完全に自動化。

Continuous Delivery & Test Automation



- **一般的な設計原則**
  - アーキテクチャの実験を容易にするために自動化を取り入れる
- **運用性の設計原則**
  - 変更は日々、小さく、継続的に
- **コスト最適化の設計原則**
  - マネージドサービスを使用して所有コストを低減

4/5 Game-day Testing

本番を想定したテスト

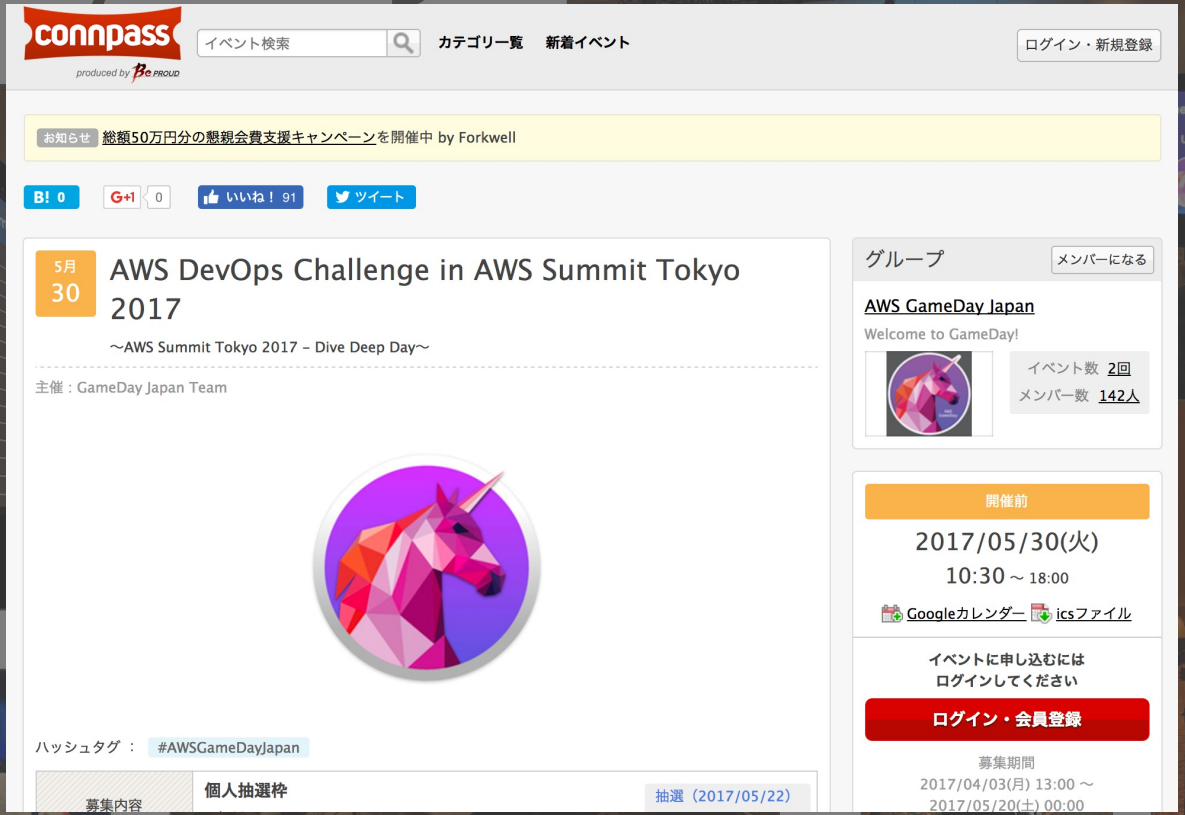
Game-day Testing

game-day: 【形】試合がある日の

https://eow.alc.co.jp/search?q=game-day

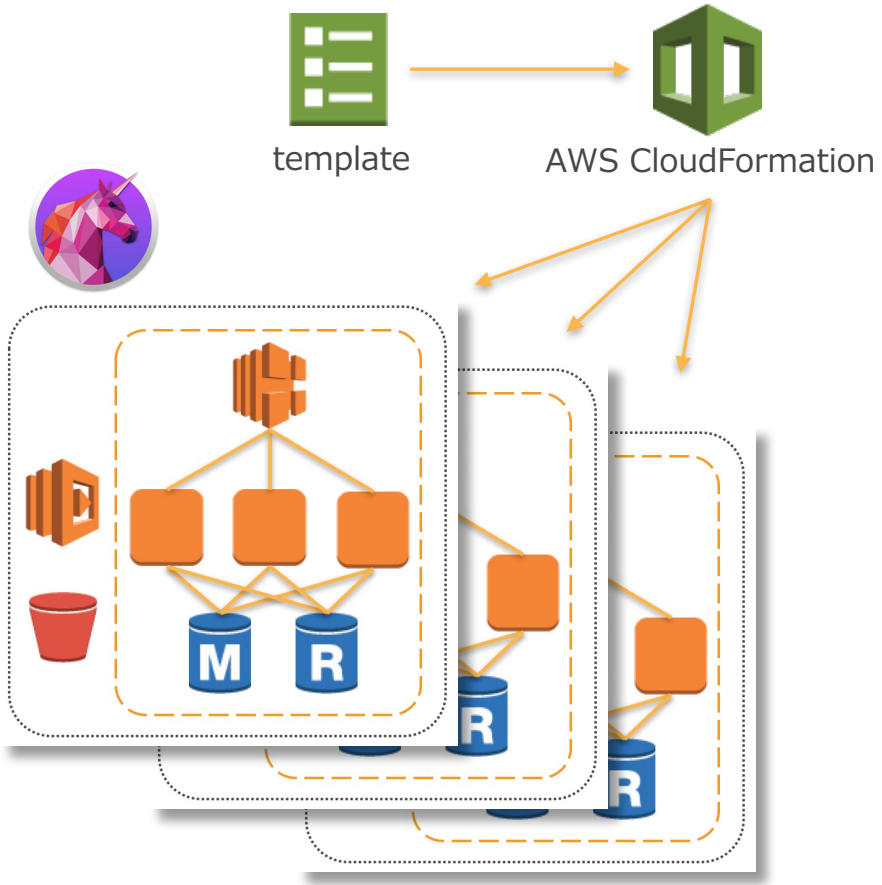
- トラブルが起きたときしかトラブルシュートしなくて、いざトラブルが起きた時にスムーズに対応できますか？
- 実際の本番環境で、異常事態の対応を訓練
- 数人のチームに分かれて、AWS 上に払い出された実際の環境をトラブルシュートしていくコンテスト形式など

Game-day Testing



https://gameday-japan.connpass.com/event/52437/

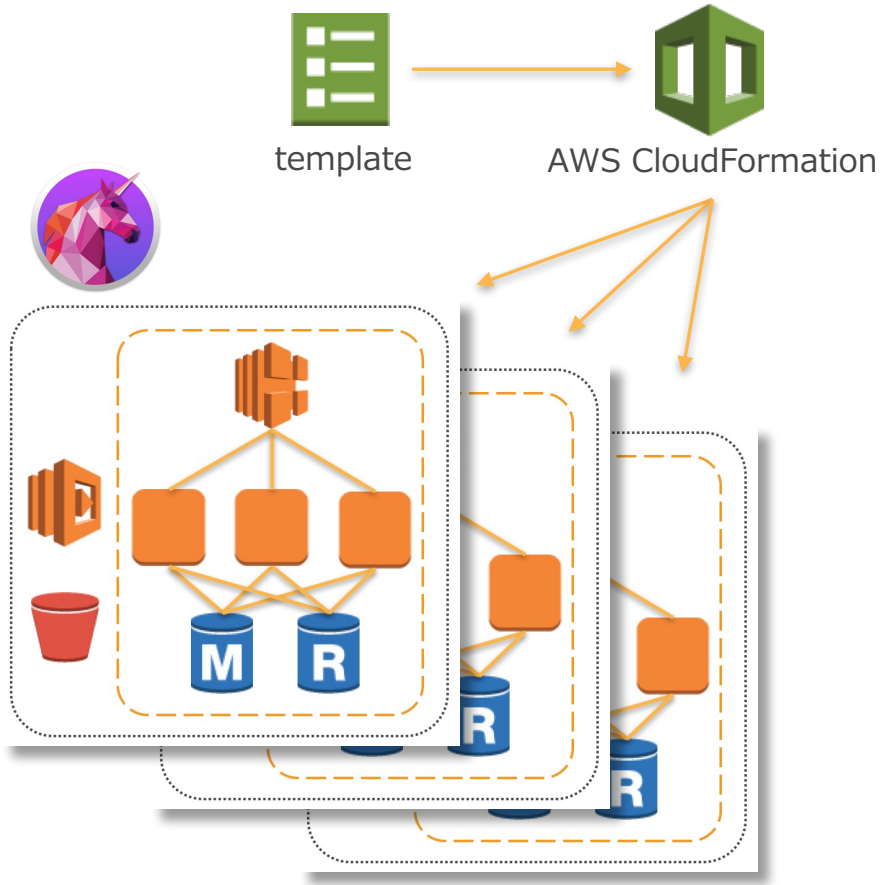
Game-day Testing



AWS では、本番と同様の構成を素早く
プロビジョニングし使い終わったら即座に
破棄する、といった運用が容易になる。
そのため、お客様がお客様自身の環境で
Game-day Testing を行うことも容易に。

- **AWS CloudFormation:**
  テンプレートを元に、関連する一連の AWS
  リソースのプロビジョニングや更新を自動化。
  テンプレートは JSON フォーマットで、
  様々なサンプルがあるとともに独自に作成
  することが可能。

Game-day Testing



- **一般的な設計原則**
  - 本番で想定される事態を
    あらかじめテストする
- **運用性の設計原則**
  - コードによる運用
- **信頼性の設計原則**
  - インフラストラクチャの変更を
    自動化する

5/5 Error Injection

障害の注入

Error Injection

障害が起きてもシステムが想定通りに振る舞うことを
確認するために、継続的かつ意図的に障害を引き起こす

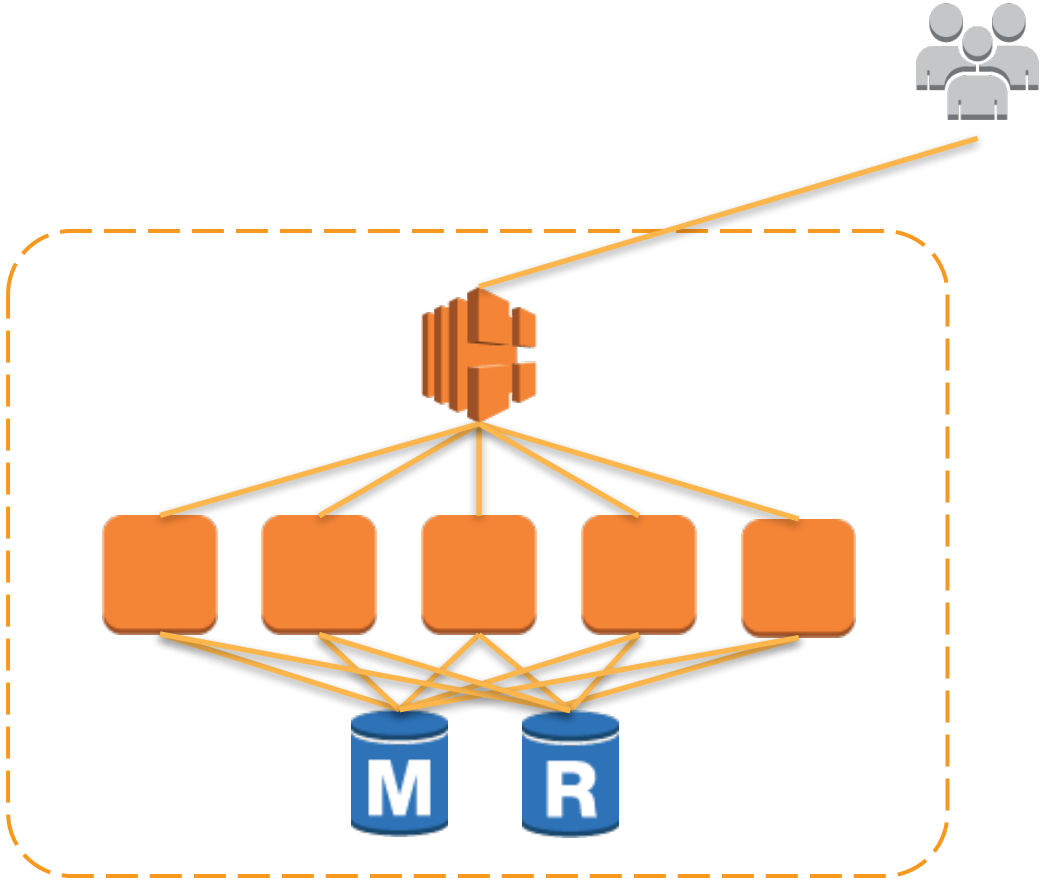
Netflix Siman Army

- **Chaos Monkey**
  個々のリソースレベルの障害を注入
- **Chaos Gorilla, Chaos Kong**
  AZやリージョンのレベルで障害を注入

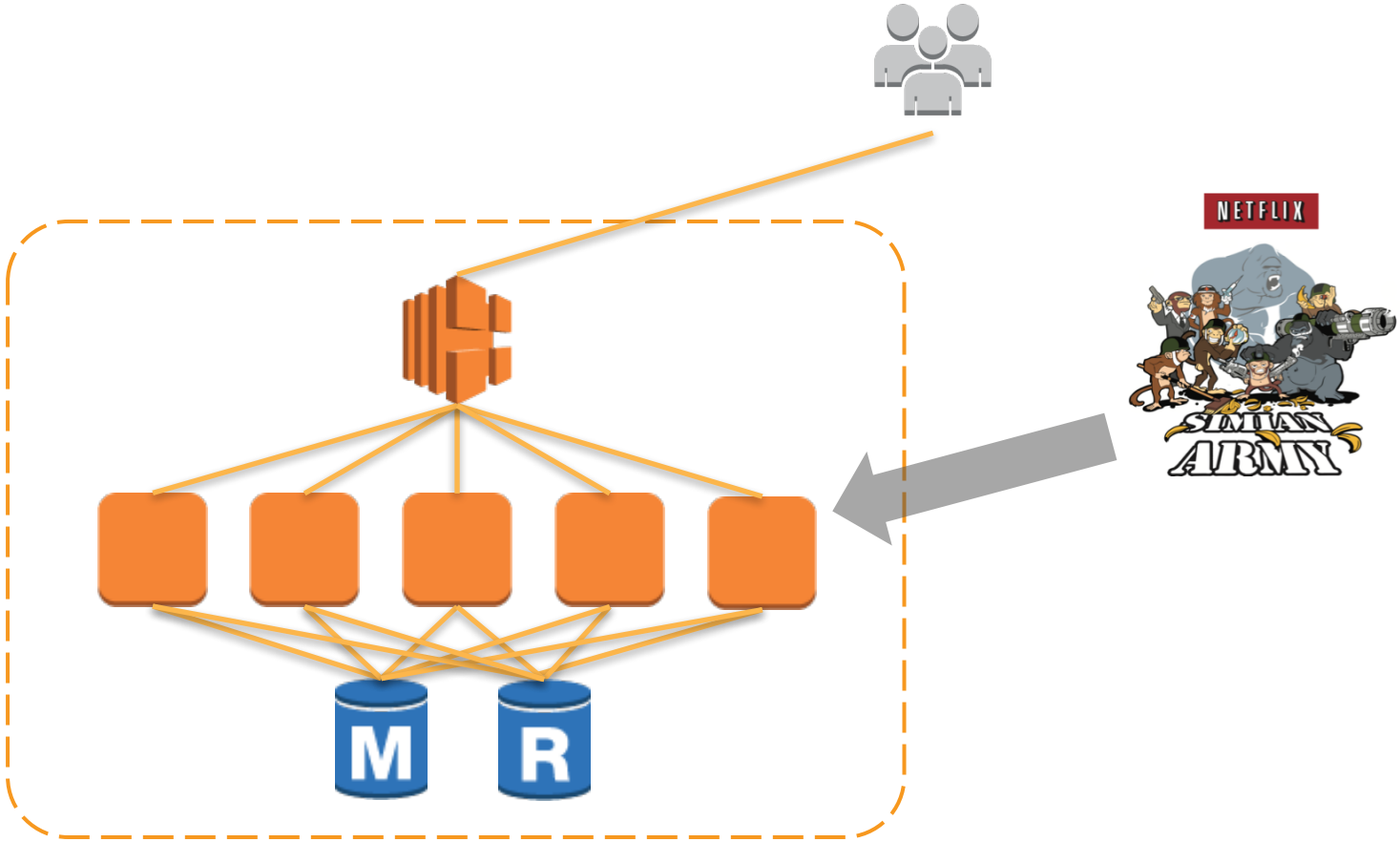
and more unique monkeys…

https://github.com/Netflix/SimianArmy

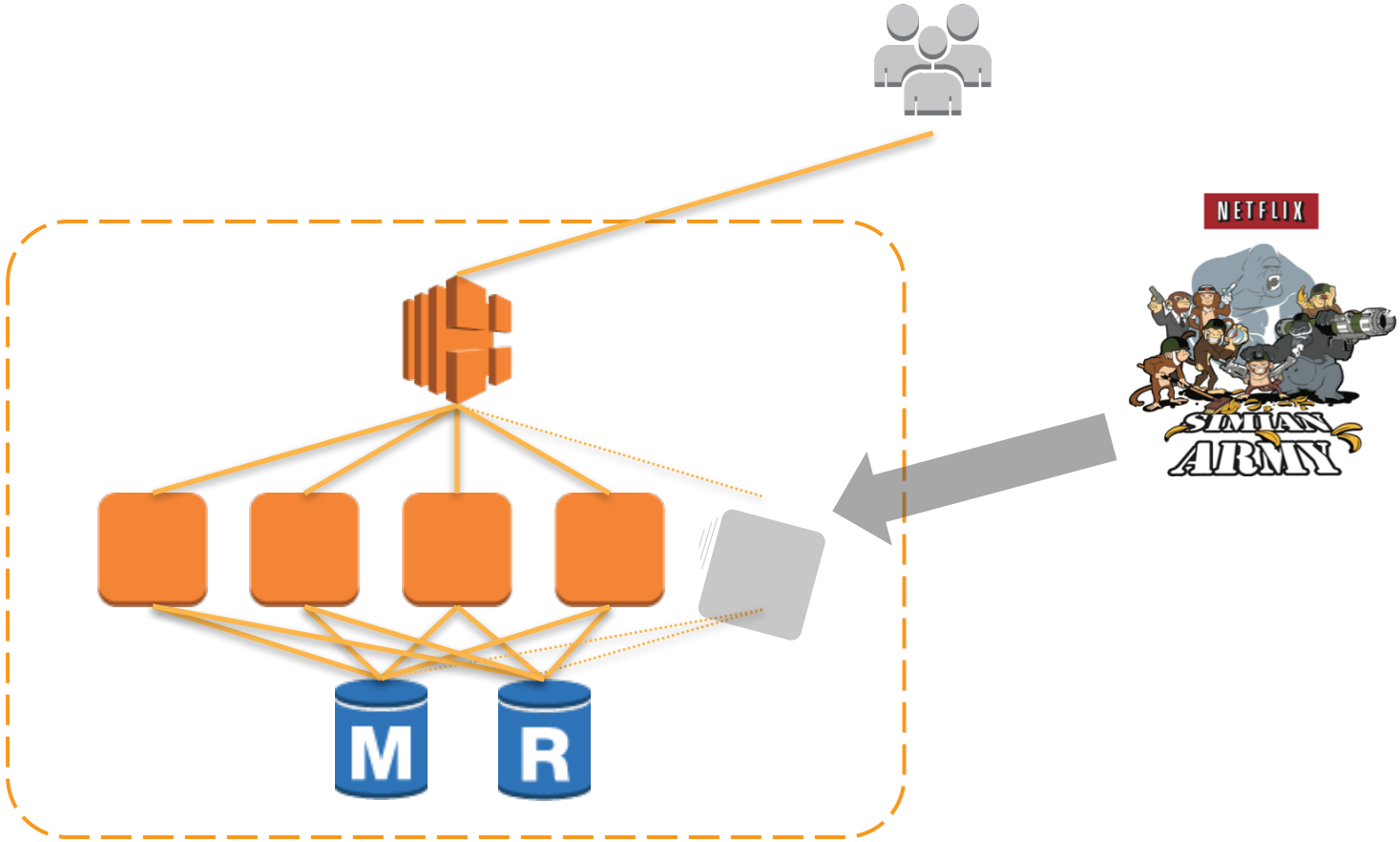
個々のリソースレベルの障害の注入



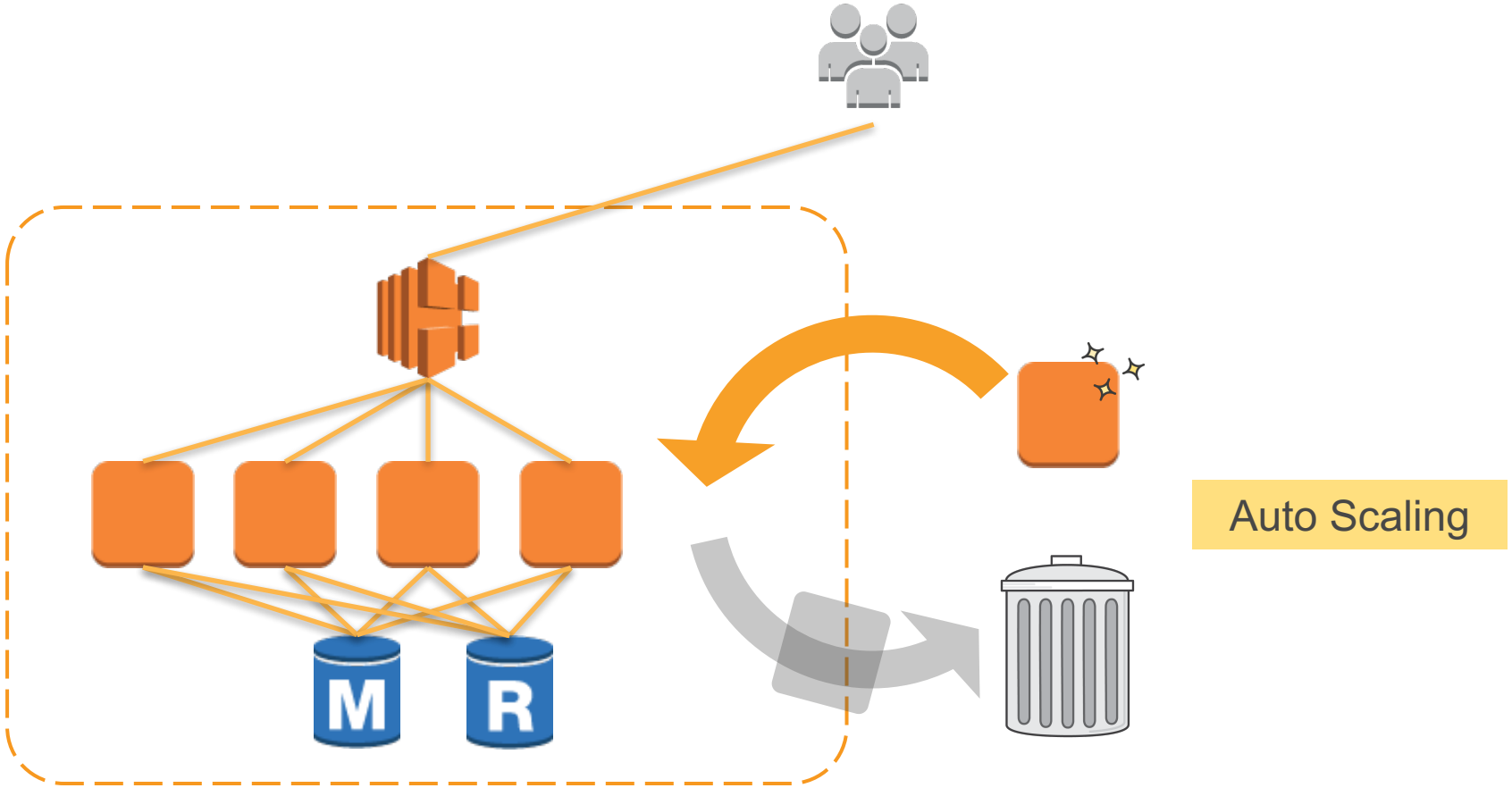
個々のリソースレベルの障害の注入



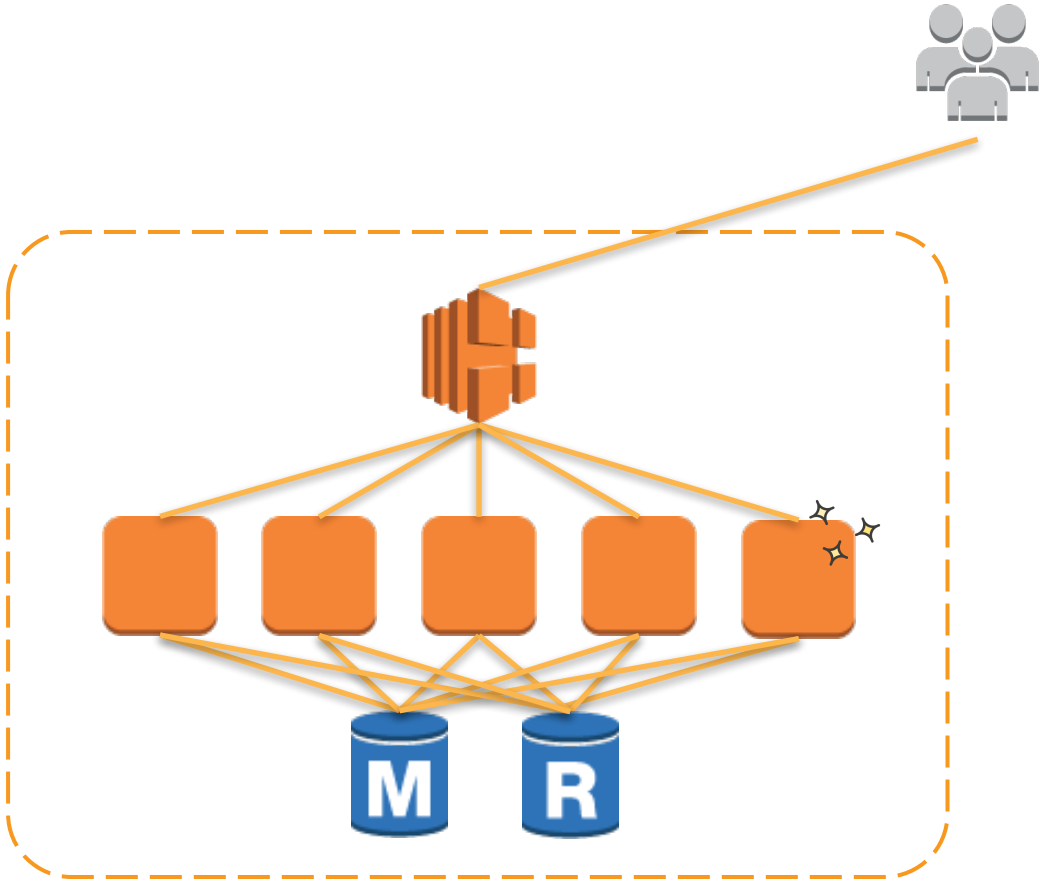
個々のリソースレベルの障害の注入



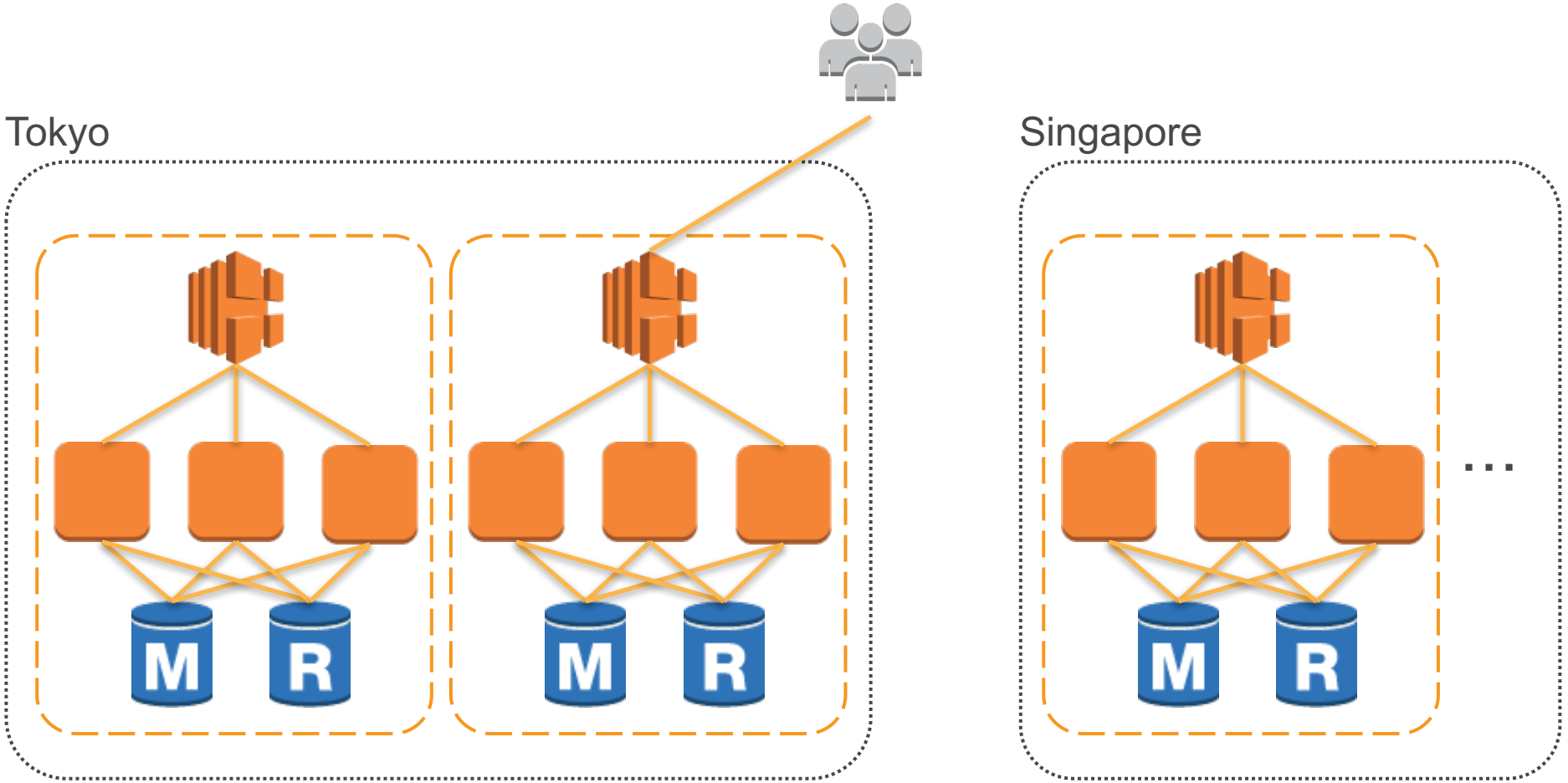
個々のリソースレベルの障害の注入



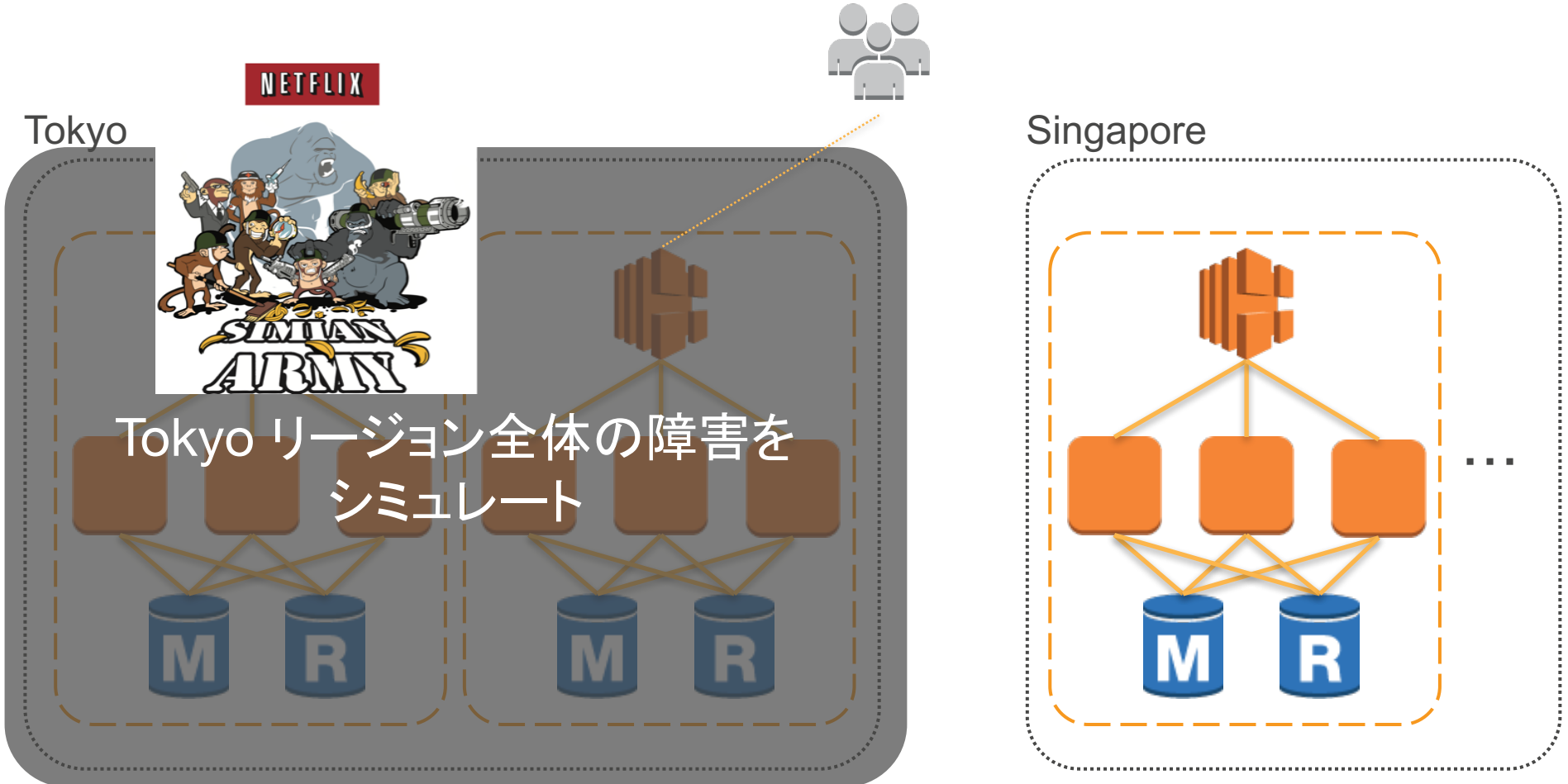
個々のリソースレベルの障害の注入



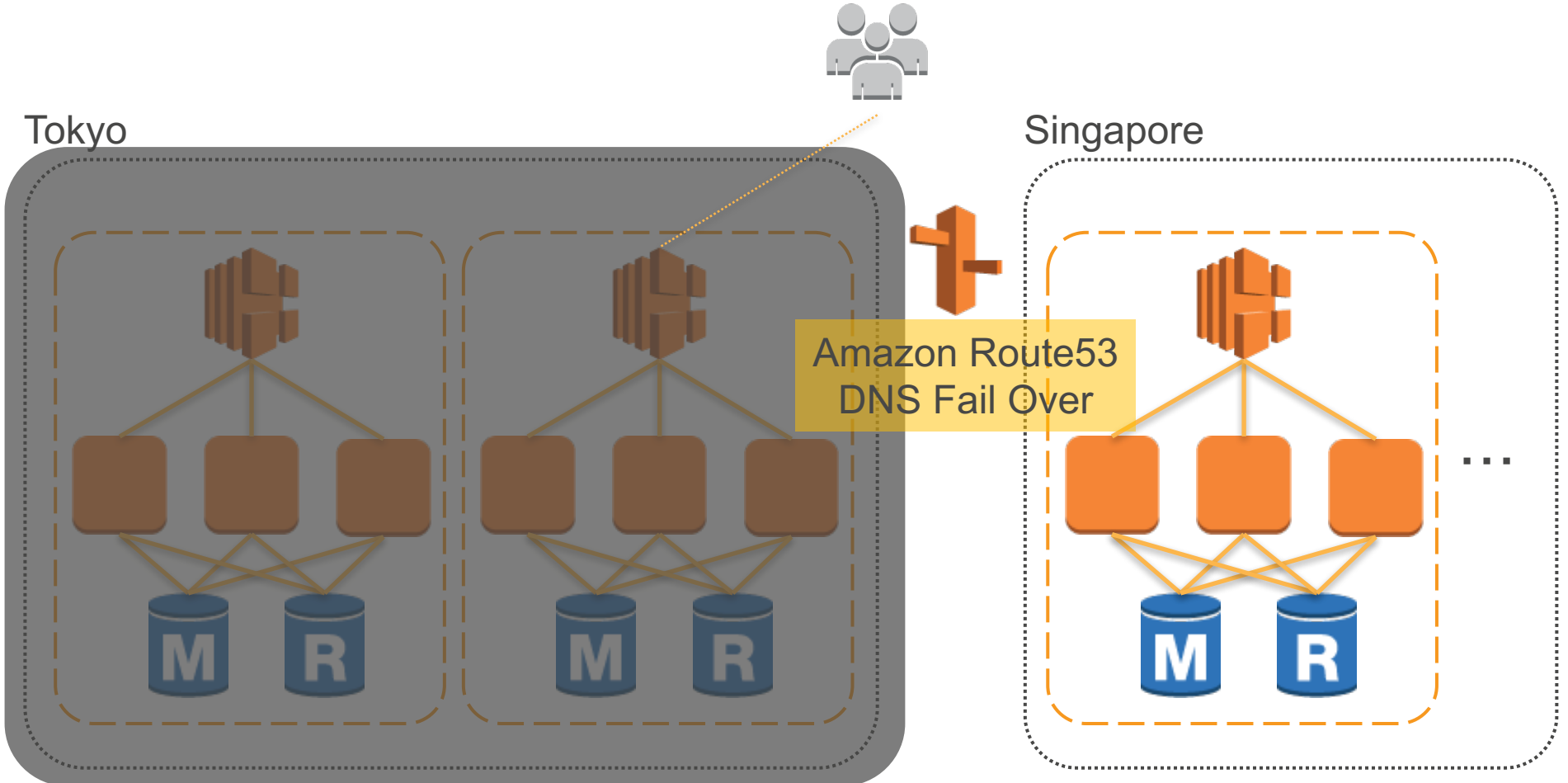
データセンターや地域レベルの障害の注入



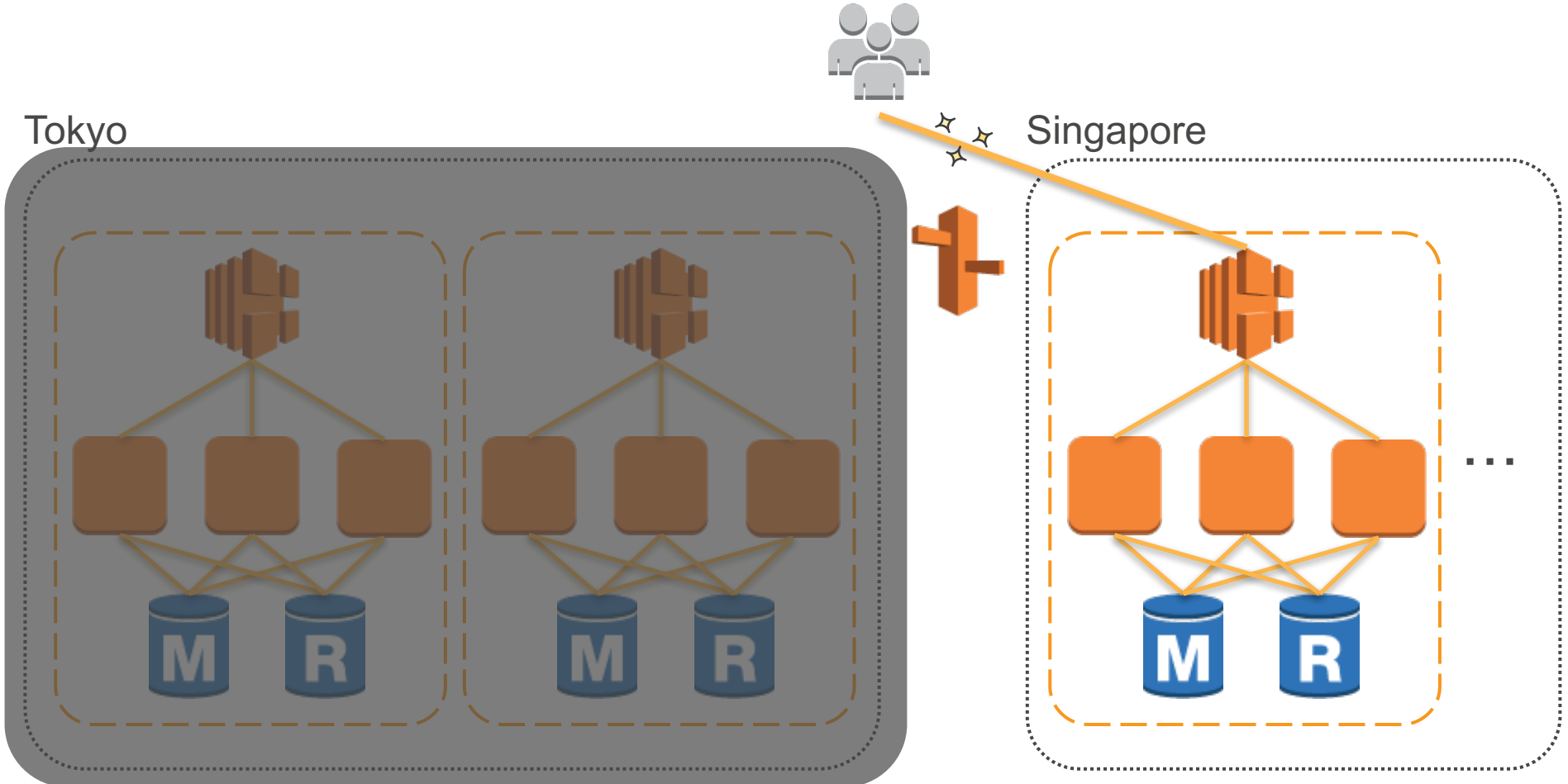
データセンターや地域レベルの障害の注入



データセンターや地域レベルの障害の注入



データセンターや地域レベルの障害の注入



アジェンダ

- AWS Well-Architected Framework とは
- クラウドでのアプリケーション設計の考え方
- クラウドでのアプリケーション設計のケーススタディ
- まとめ

まとめ

- クラウドでは、障害や失敗を未然に防止するだけではなく、**障害の迅速な検知と自動的な対処**を設計することが大切
- 設計を実現するための道具となる**AWS の多様なサービスと機能**
- 設計のベストプラクティスを提供する**AWS Well-Architected Framework**

次のステップ

1. 5 つの柱
2. 設計原則
3. 質問事項

次のステップ

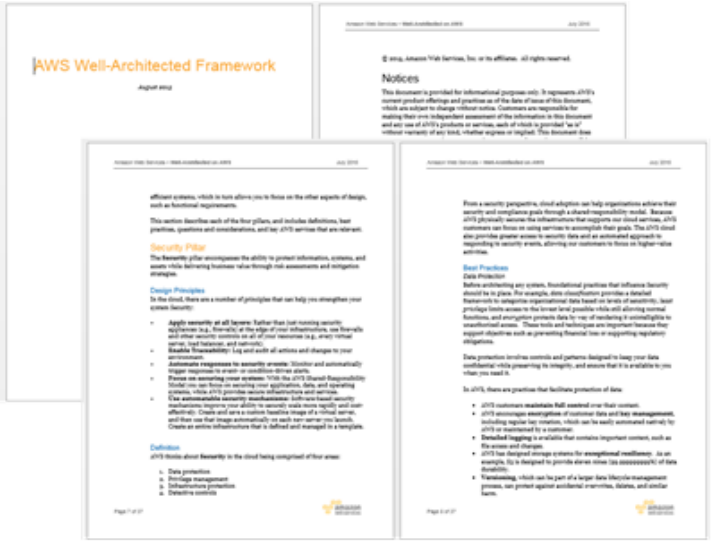
1. ５つの柱
2. 設計原則
3. 質問事項

質問事項の抜粋エクセル版

https://s3-ap-northeast-1.amazonaws.com/aws-jp-blackbelt/public/Well-Architected-JA-20170314.xlsx

次のステップ

ホワイトペーパー
http://d0.awsstatic.com/whitepapers/architecture/AWS_Well-Architected_Framework.pdf



次のステップ

ウェブサイト
https://aws.amazon.com/jp/architecture/well-architected/

本セッションのFeedbackをお願いします

受付でお配りしたアンケートに本セッションの満足度やご感想などをご記入ください
アンケートをご提出いただきました方には、もれなく素敵なAWSオリジナルグッズをプレゼントさせていただきます

アンケートは受付、パミール3FのEXPO展示会場内にて回収させて頂きます

AWS

SUMMIT

ご清聴ありがとうございました。

amazon web services