

# AZ-2008

# DevOps の基礎:

# 中心となる原則と実践

DevOps foundations: The core principles and practices

本講義は以下の **Microsoft Learn** 教材に準拠しています。

<https://learn.microsoft.com/ja-jp/training/paths/devops-foundations-core-principles-practices/>



DevOps の基礎: 中心となる原則と実践

# 本コースの概要



- ・本コースは、**DevOpsの中心となる原則**と、**GitHubでのDevOpsの実践**について1日で学ぶ、**初級者向けのコース**です。



## DevOps の基礎: 中心となる原則と実践



GitHub を使用して DevOps のプラクティスについて説明します。開発チームと運用チームは、アプリケーション ライフサイクルのすべてのフェーズで、コラボレーション、機敏性、継続的インテグレーション、継続的デリバリー、自動化、オペレーションナル エクセレンスの向上を経験します。

# 本コースの学習範囲

- Azureについての話題も少し含まれます。
- 本来、DevOpsという領域は1日ですべてを習得できるものではないため、本コースでカバーされる学習範囲は限定的です。
- 「**1日でサッと学ぶ GitHub での DevOps 入門**」といった内容・レベル感であるとお考えください。

# 関連コースのご紹介

- GitHub・開発・DevOpsについての関連コースもございます。  
ぜひご受講をご検討ください。
- GitHubの基礎: 「GH-900」 難易度: ★
- Azureの基礎: 「AZ-900」 難易度: ★
- Microsoft Azure 向けソリューションの開発 「AZ-204」 難易度: ★★
- Microsoft DevOps ソリューションの設計と実装: 「AZ-400」 難易度: ★★★

# 本コースの前提条件

- GitHubやAzureの利用経験、開発プロジェクトへの参加といった業務経験があることが**望ましい**です。

# 講義



AZ-2008  
DevOpsの基礎:  
中心となる原則と実践

DevOps foundations: The core principles and practices

- ・モジュール1 DevOps の概要
- ・モジュール2 DevOps を使用した計画 (Plan with DevOps)
- ・モジュール3 DevOps を使用した開発 (Develop with DevOps)
- ・モジュール4 DevOps を使用した提供 (Deliver with DevOps)
- ・モジュール5 DevOps を使用した操作 (Operate with DevOps)
- ・まとめ

# 講義



AZ-2008  
DevOpsの基礎:  
中心となる原則と実践

DevOps foundations: The core principles and practices

- モジュール1 DevOps の概要
- モジュール2 DevOps を使用した計画 (Plan with DevOps)
- モジュール3 DevOps を使用した開発 (Develop with DevOps)
- モジュール4 DevOps を使用した提供 (Deliver with DevOps)
- モジュール5 DevOps を使用した操作 (Operate with DevOps)
- まとめ

# モジュール1



DevOpsの必要性、  
メリット、関連技術

## DevOpsの概要

コラボレーション、継続的学習、機敏性、自動化などの DevOps の実践により、アプリケーションのライフサイクル管理を最適化します。組織には、市場投入までの時間の短縮、運用の安定性と信頼性、平均復旧時間の最小化によるメリットがあります。

# モジュール1

- DevOps とは？
- DevOps の普及度
- DevOps のメリット
- スクラム開発やアジャイル開発との関係は？
- ウォーターフォール型開発との関係は？
- DevOps の「∞」の図について
- DevOps のカルチャ（文化）とは？
- DevOps によるアプリケーション開発の最適化
- GitHub とは？
- 「シフトレフト」と「シフトライト」とは？
- よくあるご質問 / まとめ

# DevOpsとは？

- ・ソフトウェア開発手法のひとつ
- ・2009年頃に登場
- ・開発担当チーム（**De**velopers）と運用担当チーム（**Op**erations）が連携・協力して開発を行う
- ・**顧客に継続的に価値を届ける**ための人・プロセス・技術の集まり
  - ・単に DevOps ツールを導入すればOK、というものではない
  - ・組織やチームの意識や働き方の**変革**（**T**ransformation）も必要
  - ・ビジネスの**変革**も必要
  - ・導入には通常、時間がかかる

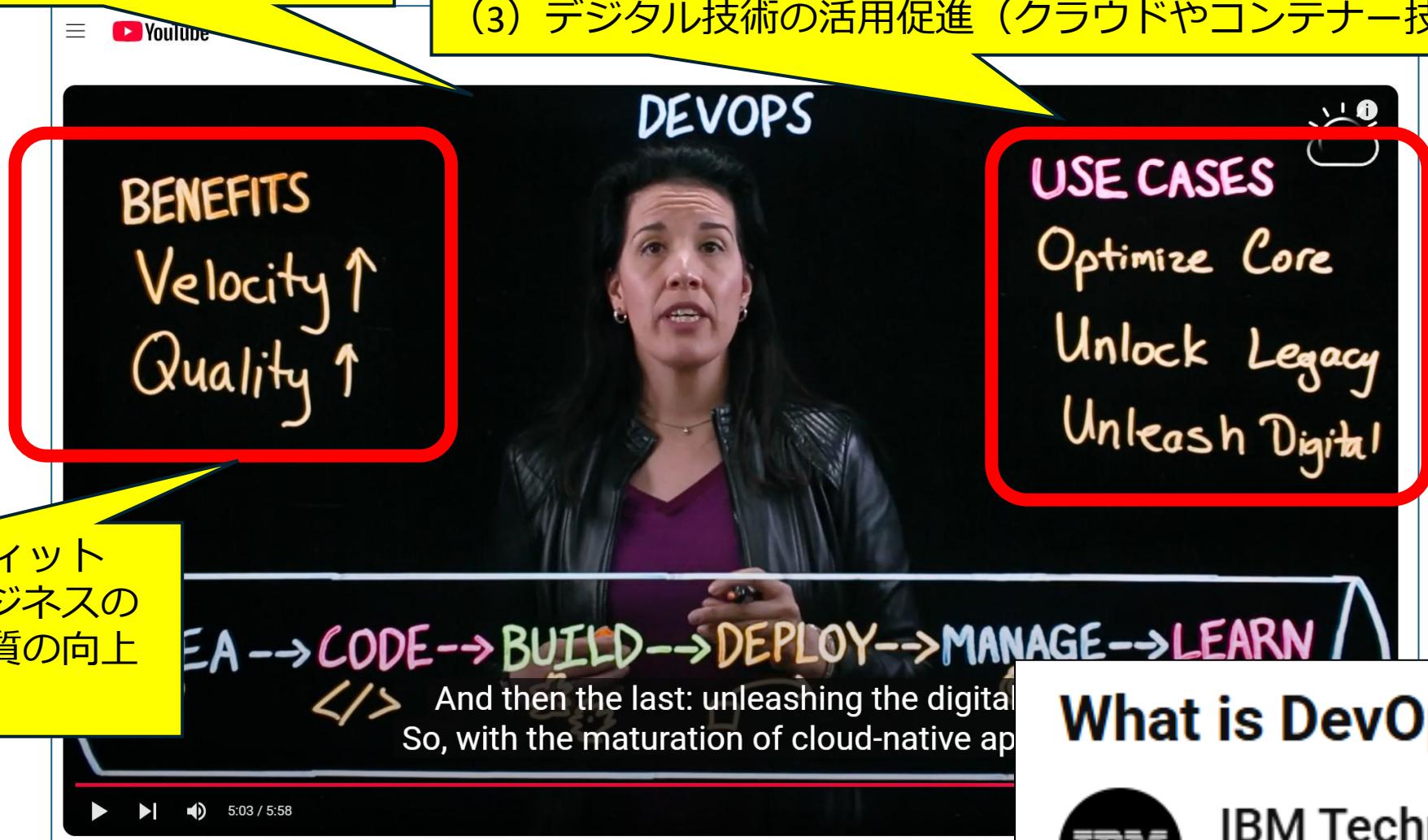
# DevOpsとは何ですか？

DevOpsは、ソフトウェア開発チームとIT運用チームの作業を組み合わせて自動化することで、高品質のアプリケーションとサービスの提供を加速するソフトウェア開発アプローチです。

DevOpsはビジネスの俊敏性（アジャリティ）を高めるための人、プロセス、ツールのあつまりである

DevOpsの3つの主要なユースケース（何に使えるか）：

- (1) 企業のモノリシックな中核システム（System of Record）の最適化
- (2) 伝統的な仕事のスタイルからの開放（文化や働き方を変え、多くの分野にわたるチームを統制の取れた方法で統合する）
- (3) デジタル技術の活用促進（クラウドやコンテナー技術の導入）



## What is DevOps?



IBM Technology ✓

チャンネル登録者数 116万人

<https://youtu.be/UbtB4sMaaNM>

# DevOps モデルとは

DevOps では、従来型のソフトウェア開発と、インフラストラクチャ管理プロセスを使用するよりも速いペースで製品の進歩と向上を達成し、企業がアプリケーションやサービスを高速で配信できるように、文化的な基本方針、プラクティス、ツールが組み合わされています。この高速化により、企業は顧客により良いサービスを提供し、市場競争力を高めることができます。

# DevOps

DevOps は、ソフトウェア デリバリーの速度とサービスの信頼性の向上、ソフトウェアの関係者間の共有オーナーシップの構築を目的とする、組織的で文化的な取り組みです。以下 DevOps について詳しくご紹介していきます。

# DevOpsとは

---

DevOps（デブオプス）とは、「開発（Development）」と「運用（Operations）」を組み合わせた造語です。簡単に言えば、「開発担当と運用担当が緊密に連携して、柔軟かつスピーディーにシステム開発を行う手法」のことです。

DevOpsが生まれた背景には、開発現場におけるチーム間の対立という問題があります。多くのソフトウェアの開発現場では、機能の開発を担当する開発（Dev）チームと、サービスの運用を担当する運用（Ops）チームに分かれて作業を行っています。Devチームの主なミッションは、ソフトウェアに新しい機能を追加していくことです。これに対してOpsチームのミッションは、現在動いている環境を維持し、サービスを安定して継続することです。このように、DevチームとOpsチームには課せられているミッションが大きく異なるわけですが、これが意見の対立を生む原因となってしまします。

DevチームもOpsチームも、価値のあるサービスをユーザーに届けるという最終的な目的は同じはずです。目的が一致しているのであれば、対立して足を引っぱり合うようなことをする必要は、本来ないはずなのです。現代は激しく、急速に変化し続いているビジネスシーンに対応するため、より迅速なソフトウェア開発が求められている時代です。そのためには、内部的な対立でチームが疲弊するようなことがあってはなりません。そこで「ありがちなチーム間の対立を解消し、協力して円滑に開発を進めていこう」という考え方、すなわちDevOpsが注目されているのです。

## ■ DevOpsとは？（Microsoft）

開発 (Dev) と運用 (Ops) の複合語である DevOps とは、継続的に顧客に価値を届けるために人、プロセス、テクノロジをひとつにまとめることです。

DevOps には、チームにとってどのような意味があるのでしょうか。DevOps では、以前はサイロ化されていた役割（開発、IT 運用、品質工学、セキュリティ）を連携させて共同作業させることができ、より優れた信頼性の高い製品を作ることができます。DevOps の文化を DevOps のプラクティスおよびツールとともに導入することで、チームは顧客のニーズにより適切に応え、構築するアプリケーションにおける信頼度を高め、ビジネス目標をより速く達成することができるようになります。

# モジュール1

- DevOps とは？
- DevOps の普及度
- DevOps のメリット
- スクラム開発やアジャイル開発との関係は？
- ウォーターフォール型開発との関係は？
- DevOps の「∞」の図について
- DevOps のカルチャ（文化）とは？
- DevOps によるアプリケーション開発の最適化
- GitHub とは？
- 「シフトレフト」と「シフトライト」とは？
- よくあるご質問 / まとめ

## ■参考: DevOps (DX) の普及度の調査

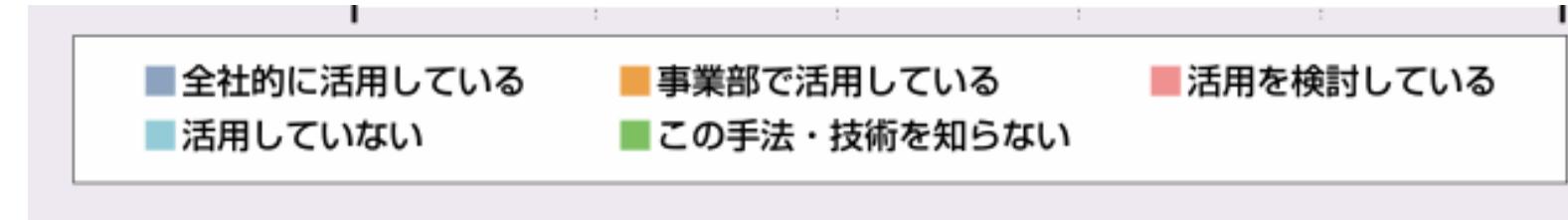
- ・日本の総企業数は約360万社。
- ・経済産業省の所管法人である独立行政法人・情報処理推進機構（IPA）は、2023年、DX（デジタルトランスフォーメーション）への取り組み状況についてのアンケートを実施、543社から回答を得た。
- ・このアンケートの中に、DevOpsなどの開発手法への取り組み状況の質問がある。

日本企業へのアンケートは、「日本企業標準産業分類」の19業種(製造業、非製造業。「公務」を除く)の日本企業の経営層またはICT関連事業部門を対象として実施した。米国企業へのアンケートは日本企業の調査対象範囲に準じた企業のマネージャークラス以上を対象者として実施したものである(図表1-39)。

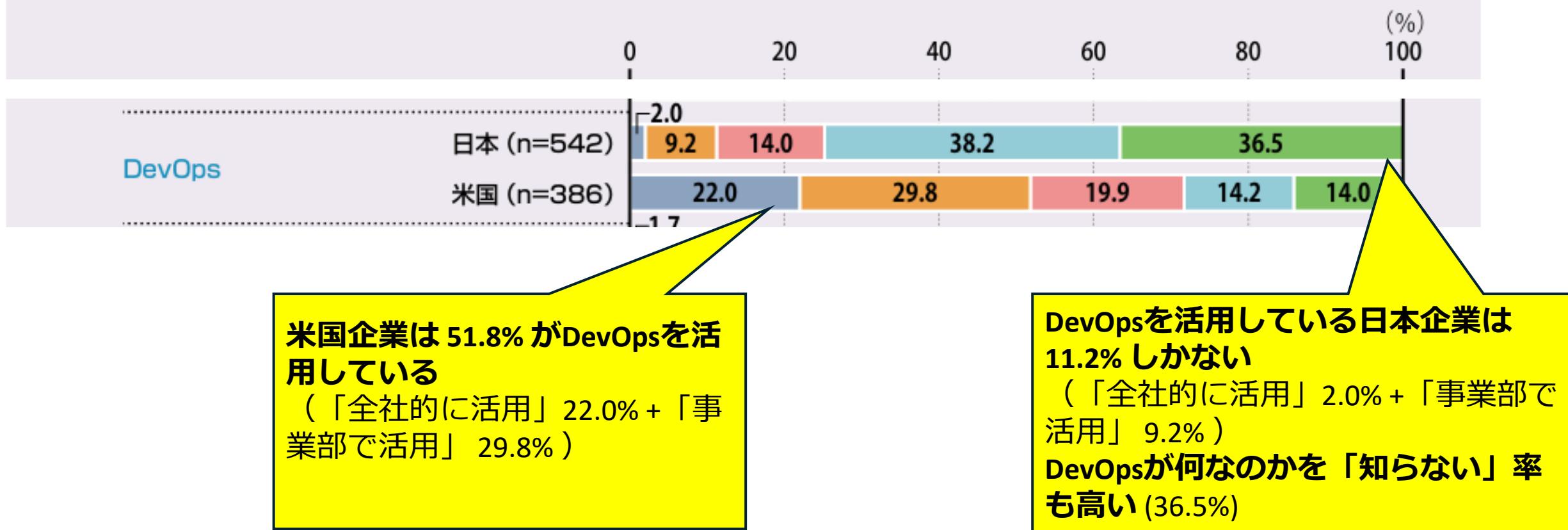
図表1-39 企業を中心としたDX推進に関する調査概要

|              | 日本企業アンケート  | 米国企業アンケート   |
|--------------|--|---|
| 調査対象範囲および対象者 | ・日本標準産業分類(大分類)の19業種(「公務」を除く)の経営層またはICT関連事業部門、DX関連事業部門の責任者もしくは担当者             | ・日本企業の調査先に準じる<br>・所属している企業に対しての責任を持って回答できるマネージャークラス以上 |
| 調査項目         | ・DXの取組状況や企業競争力を高める経営資源の活用<br>・DXの推進やデジタル技術を利活用する人材の把握<br>・デジタル技術の利活用の状況や導入課題 |   |
| 回収数          | 543社   | 386社  |
| 実施期間         | 2022年6月28日～2022年7月28日  | 2022年7月12日～2022年7月26日                                 |

## ■参考: DevOpsの普及度



図表5-22 ➡ ITシステムの開発手法・技術の活用状況



- ・「DXに取り組むような日本企業であっても、その9割はまだDevOpsを導入していない」
- ・前向きに捉えると：
- ・「多数の企業がDevOpsをこれから導入することによってさらに成長できる」
- ・「DevOpsを実践できるエンジニアが活躍できる場所がたくさんある」

# モジュール1

- DevOps とは？
- DevOps の普及度
- DevOps のメリット
- スクラム開発やアジャイル開発との関係は？
- ウォーターフォール型開発との関係は？
- DevOps の「∞」の図について
- DevOps のカルチャ（文化）とは？
- DevOps によるアプリケーション開発の最適化
- GitHub とは？
- 「シフトレフト」と「シフトライト」とは？
- よくあるご質問 / まとめ

# DevOpsのメリット

## DevOps の利点

DevOps の文化、プラクティス、ツールを導入したチームは、パフォーマンスが向上し、より大きな顧客満足度を目指してより優れた製品をより早く構築できるようになります。このようなコラボレーションと生産性の向上は、次のようなビジネス目標を達成するためにも必要不可欠です。



市場投入までの時間を短縮する



市場と競争に適応する



システムの安定性と信頼性を維持する



障害からのすばやい  
修復、  
ダウンタイムの短縮

# DevOpsのメリット

|                      |  |
|----------------------|--|
| 新製品や新機能の市場投入までの時間の短縮 | 継続的インテグレーション (CI) と継続的デリバリー (CD) により、開発・リリースのスピードが向上する。開発と運用の協力が密になり、効率的なプロセスが実現可能となる。 |
| 市場での競争への適応           | 短期間での開発・リリースが可能になり、市場の変化やユーザーニーズに迅速に対応できる。アジャイル手法と組み合わせることで、競争力の維持とイノベーションが促進される。      |
| 安定性と信頼性の維持・向上        | インフラの自動化や監視により、人的ミスを削減し、システムの安定性を確保できる。障害の予兆を検知し、迅速な対応が可能になる。                          |
| 問題の迅速な解決、稼働率の向上      | 開発と運用の密な連携により、システムの問題を素早く特定・修正できる。ログ分析やアラートを活用し、ダウンタイムの最小化とユーザー影響の軽減が可能。               |

# モジュール1

- DevOps とは？
- DevOps の普及度
- DevOps のメリット
- スクラム開発やアジャイル開発との関係は？
- ウォーターフォール型開発との関係は？
- DevOps の「∞」の図について
- DevOps のカルチャ（文化）とは？
- DevOps によるアプリケーション開発の最適化
- GitHub とは？
- 「シフトレフト」と「シフトライト」とは？
- よくあるご質問 / まとめ

# スクラム開発やアジャイル開発との関係は？

- DevOpsは「スクラムソフトウェア開発」「アジャイルソフトウェア開発」「リーンスタートアップ」などと相互に関連しながら発展している
  - **変化へのすばやい対応**を重視
  - **顧客に継続的に価値を届ける**ことを重視
  - **顧客からのフィードバック**を重視
- したがってDevOpsはこれらの考え方や開発手法と相性が良い

## ■ DevOpsの歴史(1) DevOps前夜

|      | 開発技術など        | Microsoft              | GitHub |
|------|---------------|------------------------|--------|
| 1950 | PDCAループ       |                        |        |
| 1960 |               |                        |        |
| 1970 | OODAループ       | 創業(1975)               |        |
| 1980 |               |                        |        |
| 1990 |               |                        |        |
| 1991 |               |                        |        |
| 1992 |               |                        |        |
| 1993 | スクラムソフトウェア開発  |                        |        |
| 1994 |               | Visual SourceSafe      |        |
| 1995 |               | Windows 95             |        |
| 1996 |               |                        |        |
| 1997 |               |                        |        |
| 1998 |               |                        |        |
| 1999 |               |                        |        |
| 2000 |               |                        |        |
| 2001 | アジャイルソフトウェア開発 |                        |        |
| 2002 |               |                        |        |
| 2003 |               |                        |        |
| 2004 |               |                        |        |
| 2005 | Git           | Team Foundation Server |        |

## ■ DevOpsの歴史(1) DevOps前夜

|      |               |  |
|------|---------------|--|
|      | 開発技術など        | エドワード・デミングとウォルター・シュハートがPDCAループ (Plan-Do-Check-Act) ループを提唱。プロジェクトの運用や改善における基本的な考え方となる   |
| 1950 | PDCAループ       |  |
| 1960 |               |  |
| 1970 | OODAループ       | ジョン・ボイドが「勝敗論」にて、戦闘機での戦闘におけるOODAループ (Observe-Orient-Decide-Act、観察・状況判断・意思決定・行動) を提唱。ジェフ・ザザーランドがジョン・ボイドの指導を受ける。OODAループの考え方 (PlanではなくObserveからループが始まる) はその後のソフトウェア開発手法に強い影響を与えた |
| 1980 |               |  |
| 1990 |               |  |
| 1991 |               |  |
| 1992 |               |  |
| 1993 | スクラムソフトウェア開発  | ジェフ・ザザーランドとケン・シュエイバーがスクラム (Scrum) ソフトウェア開発手法を共同開発  |
| 1994 |               |  |
| 1995 |               |  |
| 1996 |               |  |
| 1997 |               |  |
| 1998 |               |  |
| 1999 |               |  |
| 2000 |               |  |
| 2001 | アジャイルソフトウェア開発 | 軽量ソフトウェア開発手法分野の12名が集まり、価値観を議論してまとめた「アジャイルソフトウェア開発宣言」を発表  |
| 2002 |               |  |
| 2003 |               |  |
| 2004 |               |  |
| 2005 | Git           | Team Foundation Server   |

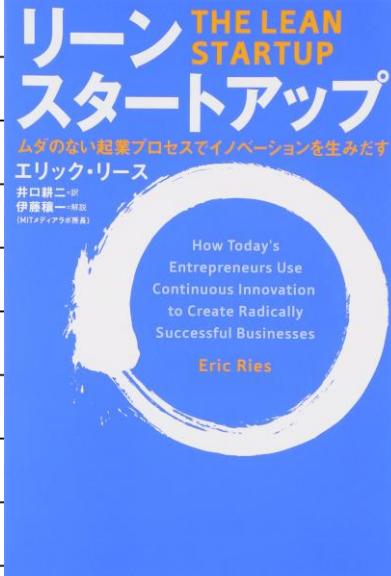
## ■ DevOpsの歴史(1) DevOps前夜

|      | 開発技術など            | Microsoft              | GitHub   |
|------|-------------------|------------------------|--|
| 1950 | PDCAループ           |                        |  |
| 1960 |                   |                        |  |
| 1970 | OODAループ           | 創業(1975)               |  |
| 1980 |                   |                        |  |
| 1990 |                   |                        |  |
| 1991 |                   |                        |  |
| 1992 |                   |                        | ソフトウェアのバージョン管理の製品。   |
| 1993 | スクラムソフトウェア開発      |                        |  |
| 1994 |                   | Visual SourceSafe      |  |
| 1995 |                   | Windows 95             |  |
| 1996 |                   |                        |  |
| 1997 |                   |                        |  |
| 1998 |                   |                        |  |
| 1999 |                   |                        |  |
| 2000 |                   |                        |  |
| 2001 | アジャイルソフトウェア開発     |                        | Visual SourceSafeの後継製品。バージョン管理、プロジェクト管理、ビルド・テスト・リリース管理などの機能を持つ。<br>後のAzure DevOps Server / Azure DevOps Servicesとなる。 |
| 2002 | リーナス・トーバルズがGitを開発 |                        |  |
| 2003 |                   |                        |  |
| 2004 |                   |                        |  |
| 2005 | Git               | Team Foundation Server |  |

## ■ DevOpsの歴史(2) DevOps登場以降

|      | 開発技術など     | Microsoft                    | GitHub          |
|------|------------|------------------------------|-----------------|
| 2006 |            |                              |                 |
| 2007 |            |                              |                 |
| 2008 |            |                              | 創業              |
| 2009 | DevOps     |                              |                 |
| 2010 |            | Windows Azure                |                 |
| 2011 | リーンスタートアップ |                              |                 |
| 2012 |            |                              |                 |
| 2013 |            |                              |                 |
| 2014 |            | Microsoft Azure              |                 |
| 2015 |            |                              |                 |
| 2016 |            |                              | GitHub Projects |
| 2017 |            |                              |                 |
| 2018 |            | GitHub社を買収                   |                 |
| 2019 |            | Azure DevOps Server/Services | GitHub Actions  |
| 2020 |            |                              |                 |
| 2021 |            |                              | GitHub Copilot  |
| 2022 | ChatGPT    |                              |                 |
| 2023 |            |                              |                 |
| 2024 |            |                              |                 |
| 2025 |            |                              |                 |

## ■ DevOpsの歴史(2) DevOps登場以降

|      | 開発技術など  | Microsoft                    | GitHub  |
|------|---|------------------------------|---|
| 2006 |   |                              |   |
| 2007 |   |                              |   |
| 2008 |   |                              | 創業  |
| 2009 | DevOps  |                              | 初のDevOpsイベント「DevOps Days」が開催され、DevOpsという概念が普及し始める   |
| 2010 |   |                              |   |
| 2011 | リーンスタートアップ  |                              |   |
| 2012 |   |                              | エリック・リースが『The Lean Startup』（無駄のないスタートアップ）を出版、ベストセラーに。仮説を立て、MVP（Most Viable Product、試作品）をエリアアダプターに提供し、改善、方向転換（ピボット）を行う。DevOpsとの共通点や関連性がある。 |
| 2013 |   |                              |   |
| 2014 |   |                              |   |
| 2015 |   |                              |   |
| 2016 |   |                              |   |
| 2017 |   |                              |   |
| 2018 |   | GitHub社を買収                   |   |
| 2019 |   | Azure DevOps Server/Services | GitHub Actions  |
| 2020 |   |                              |   |
| 2021 |   |                              | GitHub Copilot  |
| 2022 | ChatGPT   |                              |   |
| 2023 |   |                              |   |
| 2024 |   |                              |   |
| 2025 |   |                              |   |

## ■ DevOpsの歴史(2) DevOps登場以降

|      | 開発技術など                                    | Microsoft                    | GitHub                 |
|------|---|------------------------------|------------------------|
| 2006 |   |                              |                        |
| 2007 |   |                              |                        |
| 2008 |   |                              | 創業                     |
| 2009 | DevOps                                    |                              |                        |
| 2010 |   | Windows Azure                |                        |
| 2011 | リーンスタートアップ                                |                              |                        |
| 2012 | Windows Azure サービス開始、後にMicrosoft Azureに改名 |                              |                        |
| 2013 |   |                              |                        |
| 2014 |   | Microsoft Azure              |                        |
| 2015 |   |                              |                        |
| 2016 |   |                              | GitHub Projects        |
| 2017 | GitHub社を75億ドル(約8,234億円)で買収                |                              |                        |
| 2018 |   | GitHub社を買収                   |                        |
| 2019 | Azure DevOps 提供開始                         | Azure DevOps Server/Services | GitHub Actions CI/CD機能 |
| 2020 |   |                              |                        |
| 2021 |   |                              | GitHub Copilot         |
| 2022 | ChatGPT                                   |                              |                        |
| 2023 |   |                              | AIペアプログラマ              |
| 2024 |   |                              |                        |
| 2025 |   |                              |                        |

# モジュール1

- DevOps とは？
- DevOps の普及度
- DevOps のメリット
- スクラム開発やアジャイル開発との関係は？
- ウォーターフォール型開発との関係は？
- DevOps の「∞」の図について
- DevOps のカルチャ（文化）とは？
- DevOps によるアプリケーション開発の最適化
- GitHub とは？
- 「シフトレフト」と「シフトライト」とは？
- よくあるご質問/まとめ

# ウォーターフォール型開発との関係は？

- ・ウォーターフォール型開発とは？
  - ・要件定義、設計、実装、テスト、運用といった工程を、順番に、後戻りなしで進めていく開発手法
  - ・プロジェクトを計画通りに進行させることを重視
- ・DevOpsは、スクラムやアジャイルに影響を受けているため、どちらかといえば**ウォーターフォール型開発**よりも**アジャイル型開発**との相性がよい
  - ・ドキュメントよりも**動くソフトウェア**を重視
  - ・**顧客に継続的に価値を届ける**ことを重視
  - ・**変化へのすばやい対応**を重視

# ウォーターフォール型開発との関係は？

- ・ちなみに、ウォーターフォール型の開発プロジェクトに、DevOpsのツールを取り入れることは不可能ではないが…
  - ・プロジェクトの進行はウォーターフォール型で管理する
  - ・プロジェクトの中の各作業ではDevOpsツールを活用する
  - ・など

# ウォーターフォール型開発との関係は？

- ・ウォーターフォール型
  - ・「顧客に**計画通りに価値を提供する**」ことを重視
- ・DevOps
  - ・「顧客に**継続的に価値を届ける**」ことを重視
- ・そもそも**価値の提供の考え方やプロジェクト運用方針**が異なる。
- ・ウォーターフォール型開発プロセスにただDevOpsツールを導入しただけでは、DevOpsの本来の価値は得られない。

計画が完了したら  
プロジェクトは終了

需要がある限り  
プロジェクトは継続

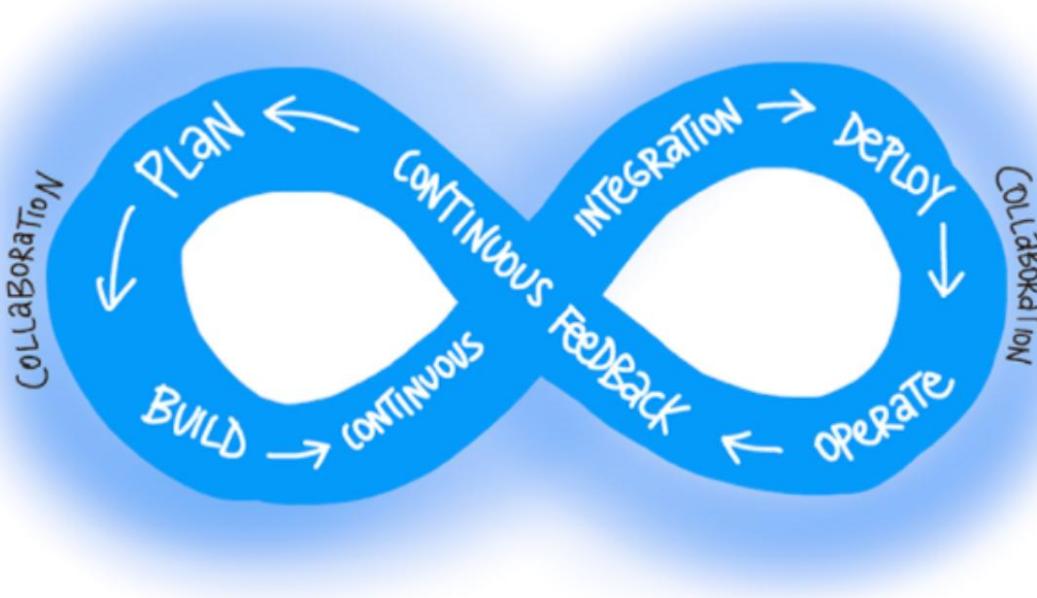
# モジュール1

- DevOps とは？
- DevOps の普及度
- DevOps のメリット
- スクラム開発やアジャイル開発との関係は？
- ウォーターフォール型開発との関係は？
- DevOps の「∞」の図について
- DevOps のカルチャ（文化）とは？
- DevOps によるアプリケーション開発の最適化
- GitHub とは？
- 「シフトレフト」と「シフトライト」とは？
- よくあるご質問 / まとめ

## ■ DevOpsの「8の字ループ」（∞: 無限大）の図

•ループの左側: 開発に必要な  
プロセス、機能、ツールを表現

•ループの右側: 運用に必要な  
プロセス、機能、ツールを表現



DevOpsのプロセスは繰り返される。  
運用からはフィードバック、つまりデータが得られる。  
フィードバックをもとに次の開発の方針を決定する  
(OODAループの考え方)

# DEVOPS



## BENEFITS

Velocity ↑

Quality ↑



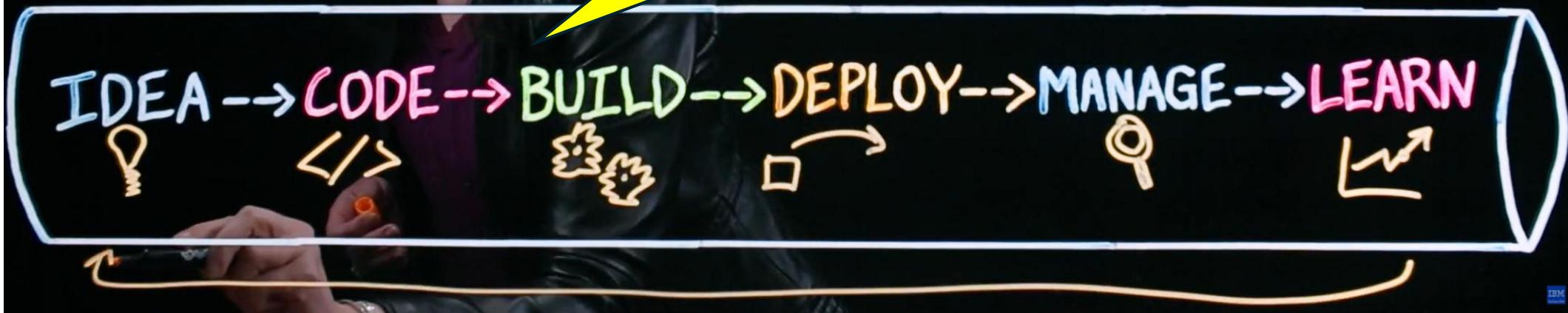
## USE CASES

Optimize Core

Inlock Legacy

Inleash Digital

DevOpsは反復的なプロセスであり、運用から学んだデータを活用して、次のイテレーションで実施すべきことを決定する。



What is DevOps?



IBM Technology ✓  
チャンネル登録者数 116万人

# モジュール1

- DevOps とは？
- DevOps の普及度
- DevOps のメリット
- スクラム開発やアジャイル開発との関係は？
- ウォーターフォール型開発との関係は？
- DevOps の「∞」の図について
- DevOps のカルチャ（文化）とは？
- DevOps によるアプリケーション開発の最適化
- GitHub とは？
- 「シフトレフト」と「シフトライト」とは？
- よくあるご質問 / まとめ

# DevOpsのカルチャー（文化）

- DevとOpsのコラボレーション
  - サイロをなくす

- 測定する
  - KPI
  - メトリック
  - フィードバック

- アジャイル
  - 変化に対応

- 自動化、人的ミス排除
  - CI/CD

- セキュリティの統合
  - DevSecOps

# モジュール1

- DevOps とは？
- DevOps の普及度
- DevOps のメリット
- スクラム開発やアジャイル開発との関係は？
- ウォーターフォール型開発との関係は？
- DevOps の「∞」の図について
- DevOps のカルチャ（文化）とは？
- DevOps によるアプリケーション開発の最適化
- GitHub とは？
- 「シフトレフト」と「シフトライト」とは？
- よくあるご質問 / まとめ

# DevOpsによるアプリケーション開発の最適化 (アプリケーションライフサイクル管理)

- ・アプリケーション開発のすべての工程で、  
DevOpsの手法やツールが活用される。
  - ・計画（要件定義、問題管理、マイルストーン設定）
  - ・開発（プログラミング、テスト、レビュー）
  - ・デリバリー（CI/CD、IaC、リリース）
  - ・運用（保守、監視、フィードバック収集）
- 
- モジュール2で解説
- モジュール3で解説
- モジュール4で解説
- モジュール5で解説

# モジュール1

- DevOps とは？
- DevOps の普及度
- DevOps のメリット
- スクラム開発やアジャイル開発との関係は？
- ウォーターフォール型開発との関係は？
- DevOps の「∞」の図について
- DevOps のカルチャ（文化）とは？
- DevOps によるアプリケーション開発の最適化
- GitHub とは？
- 「シフトレフト」と「シフトライト」とは？
- よくあるご質問 / まとめ

# GitHubとは？

- ・ソフトウェア開発プラットフォーム
- ・世界中の開発者が利用している
- ・オープンソースソフトウェアの開発によく使用される
  - ・が、それに限定されない
- ・個人での利用をサポート（GitHub Free / Pro）
- ・企業での利用もサポート（GitHub Enterprise 等）

# GitHub でできること（一部）

- **GitHub Issues** を使った問題管理
  - (モジュール2で解説)
- **GitHub Projects** を使ったプロジェクト管理
  - (モジュール2で解説)
- **GitHub リポジトリ**を使ったバージョン管理
  - (モジュール3で解説)
- **GitHub Actions**を使ったCI/CD
  - (モジュール4で解説)

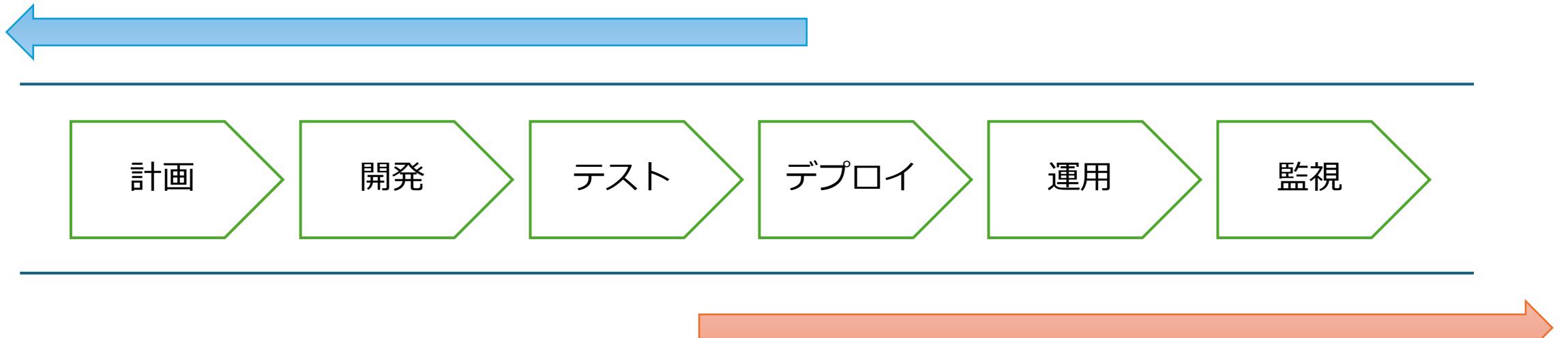
# モジュール1

- DevOps とは？
- DevOps の普及度
- DevOps のメリット
- スクラム開発やアジャイル開発との関係は？
- ウォーターフォール型開発との関係は？
- DevOps の「∞」の図について
- DevOps のカルチャ（文化）とは？
- DevOps によるアプリケーション開発の最適化
- GitHub とは？
- 「シフトレフト」と「シフトライト」とは？
- よくあるご質問 / まとめ

# 「シフトレフト」と「シフトライト」とは？

- ・ソフトウェア開発プロセス上でテストや品質保証・セキュリティ対策を行うタイミングの戦略を指す。

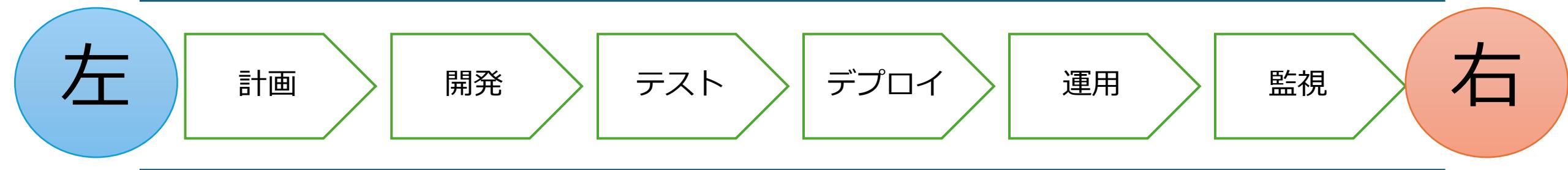
シフトレフト（Shift Left）：従来よりも早い段階での対策の実施



シフトライト（Shift Right）：運用・本番環境での対策の実施

# 「シフトレフト」と「シフトライト」とは？

- ・シフトレフトは各種の対策の「前倒し」実施
- ・シフトライトは各種の対策の「後倒し」実施
- ・「**左で品質を作り込み、右で実運用から学ぶ**」



# シフトレフト shift left

- ・開発工程のより早い段階でセキュリティ検証やテストを実施する手法。
- ・**従来はリリース直前に行われていたテスト・対策を設計・実装など初期フェーズから組み込むことで、欠陥や脆弱性を早期発見・早期修正しする。コストも節約できる場合が多い。**
  - ・例1: 設計段階からの脅威モデリングやコードレビューによるセキュリティ検証を行う。静的解析セキュリティテスト (SAST) や動的解析 (DAST) を開発初期から組み込み、コード内の脆弱性を早期に洗い出す。
  - ・例2: パフォーマンステストを開発段階から行うことで、実際のユーザー・トラフィック発生前にボトルネックを特定し対処できる。

# シフトライト shift right

- ・本番環境においてデプロイ後にテストや検証を行う手法。
- ・**実際の運用環境や実ユーザーデータを用いて、リリース後も継続的にソフトウェアの品質を監視・テストし、必要に応じて不具合修正や性能改善を行う。**
- ・実ユーザーシナリオでの検証や本番監視を重視。
  - ・例1:本番環境でのセキュリティ監視（たとえば脆弱性スキャンの継続実行や侵入検知）や、発見されたインシデントへの迅速な対応を行う。
  - ・例2: カナリアリリースやブルーグリーンデプロイによって、新機能を一部のユーザに提供しながら挙動を確認する

# シフトレフトとシフトライト

| 項目       | シフトレフト(左側で実施)   | シフトライト(右側で実施)   |
|----------|---|---|
| 実施タイミング  | 開発初期・設計～実装フェーズ<br>(テスト工程を前倒し)   | 本番デプロイ後・運用フェーズ<br>(テスト工程を後倒し)   |
| 主な対象作業   | 各種テストの前倒し実行(単体、統合、性能テスト等)<br>セキュリティ対策の組込み(脆弱性スキャン等)   | 本番環境での検証(実ユーザーを用いたテスト、モニタリング)<br>運用段階での品質監視(ユーザー体験の分析、障害対応)   |
| 狙い・アプローチ | 欠陥や脆弱性の早期発見と迅速なフィードバック<br>不具合を大事になる前に対処し開発効率を向上   | 実運用での信頼性検証<br>ユーザー環境で問題を検知し改善(現実の使用状況に適応)   |
| 主なメリット   | 品質向上: 問題を早期に潰し、重大障害を未然防止<br>コスト削減: 後工程での修正費用を大幅削減<br>開発高速化: 後戻り削減でリリースまでの時間短縮<br>開発者メリット: フィードバック早期取得で生産性向上 | 信頼性向上: 本番リリースへの自信増、大規模環境での安定性確保<br>ユーザー重視: 実ユーザーのフィードバックを反映しUXを最適化<br>強靭性向上: 本番障害への積極対処でシステムのレジリエンス強化 |

# Security By Design (Secure By Design)

- ・「シフトレフト」に近い考え方
- ・セキュリティを企画・設計段階から確保するアプローチ
- ・後付けの対策ではなく、開発の初期段階からセキュリティを組み込む
- ・総合的なコストも節約できる場合が多い

[内閣サイバーセキュリティセンター（NISC）](#)

[Provisional\\_Translation\\_JP\\_Principles\\_Approaches\\_for\\_Security-by-Design-Default\\_October.pdf](#)

# GitHubとの関連性(1)

- **GitHub Actions:**

- 各コミットやプルリクエスト時に自動的にビルド・テスト・検証を実行することで、**シフトレフト**を可能とする。
- 新バージョンを限定的にリリースして運用情報を集め、問題がなければ本番環境に大規模に展開する、といったアプローチで**シフトライト**を実現できる。
- **GitHub Actions**についてはモジュール4で解説

# GitHubとの関連性(2)

- **GitHub Advanced Security** (GHAS)
  - コードの脆弱性を発見する「Code Scanning」を全てのプルリクエストに対して実施することで、セキュリティテストのシフトレフトを実現できる。
  - →GHASは本コースでは説明されませんが、ご興味のある方は**関連コース 「GH-500 GitHub Advanced Security」**のご受講をぜひご検討ください。

# よくあるご質問

- DevOpsとクラウドとの関係は？ DevOpsとコンテナー型仮想化技術との関係は？
- →必ずしもDevOpsでクラウドやコンテナー技術を導入しなければならないというわけではありません。
- ただし、DevOpsでは、通常、多くの作業を自動化して、作業効率を向上させ、人的ミスを排除するということを考えます。その際に、クラウドを使ったインフラのプロビジョニング（リソース作成）の自動化、コンテナーとマニフェストによる構成の宣言・自動スケーリングといった技術は**非常に相性がよい**ので、DevOpsではクラウドやコンテナー技術が活用される場合が多いです。
- また多くのDevOpsツールはクラウドアプリ（SaaS型）として提供されます。
  - GitHub、Azure DevOpsなど。

# モジュール1

- DevOps とは？
- DevOps の普及度
- DevOps のメリット
- スクラム開発やアジャイル開発との関係は？
- ウォーターフォール型開発との関係は？
- DevOps の「∞」の図について
- DevOps のカルチャ（文化）とは？
- DevOps によるアプリケーション開発の最適化
- GitHub とは？
- 「シフトレフト」と「シフトライト」とは？
- よくあるご質問 / まとめ

# よくあるご質問

DevOpsは  
一日にして成らず

- DevOpsのカルチャー（文化）を組織に導入することが重要だということはわかりましたが、具体的にはどうすればよいですか？
- →典型的には、以下のような方針の決定と実施を**組織的に行います。**

|                       |  |
|-----------------------|--|
| <b>経営層の理解と支援を得る</b>   | DevOpsの目的やメリットを経営層に説明し、組織全体でDevOpsを推進する環境を整える。                                       |
| <b>継続的な教育を行う</b>      | DevOpsの考え方やプラクティスを社内研修や勉強会を通じて共有する。開発・運用チームのメンバーがDevOpsのメリットを実感できるように、小さな成功体験を積み重ねる。 |
| <b>ツールとプロセスを標準化する</b> | CI/CD、自動化ツール、インフラ管理ツールなどを活用し、DevOpsのベストプラクティスを導入する。標準化されたワークフローを確立し、チーム間の連携を強化する。    |
| <b>フィードバックの文化を育てる</b> | チームの成果を振り返る場（レトロスペクティブ）を定期的に設け、改善点を共有する。透明性を確保し、失敗を責めるのではなく学習の機会と捉える文化を醸成する。         |

# よくあるご質問

- DevOpsやアジャイル開発では**設計書**を作らないのか？
- アジャイル開発ではよく「**ドキュメント**よりも動くソフトウェアを重視する」などと言うが、**ドキュメント**がなければソフトウェアは作れないのでは？
- →DevOpsやアジャイル開発でドキュメントをまったく作らないわけではないです。ただし、ウォーターフォール型の開発でよく使われていたような「**基本設計書**」「**詳細設計書**」「**テスト仕様書**」といったものよりも、より軽量で、よりメンテナンスがしやすいフォーマット、機械可読なフォーマット、専用のツール、オンライン作図ツール、APIやコマンドからコントロールできるツールなどが好まれる傾向があります。
- 専用のツールで効率よく記述・管理・共有されたり、ソースコードと一緒にGitでバージョン管理されたりすることが多いです。

# DevOpsでよく使われる文書フォーマット、 ドキュメント作成ツール、情報管理ツール等

- Markdown
- Wiki
- JSON/YAML/TOML
- reStructuredText
- Mermaid
- Graphviz
- PlantUML
- AsciiDoc
- Sphinx
- Swagger
- API Blueprint / RAML
- JavaDoc
- docstring
- Doxygen
- Cucumber / Gherkin
- draw.io
- Confluence
- Notion
- Jira
- Trello
- RedMine
- Azure DevOps
- GitHub
- GitLab
- Google Spreadsheet

- ・これらのドキュメント形式やツールを試してみて、それでもうまく表現・記録できない設計や仕様などがある場合は、従来型のExcelやWordのドキュメントを使うことになるかと思います。
- ・ただ、やはりDevOpsやアジャイル開発と、従来型の**紙ベースの運用**を前提としたドキュメントは、相性が良くないと思います。ドキュメントの作成、手直し、レビュー、承認、維持にものすごく時間をかける、ドキュメントの通りに下請けに開発させる、といったことは、開発工程の自動化が進む現代において少しづつその意義や必要性が薄れてきていると思います。そこも含めて企業の文化を改革していく必要があるかもしれません。

# エンタープライズ開発でありがちな仕様書の例（いわゆる「神Excel」、印刷前提の仕様書）

用户名

密码

ログイン

- × ファイルのバージョン管理が難しい（誰がいつどこをどう変更したのかがわからない、あるいは変更点がわかるように手作業で改訂履歴を書かなければならぬ）
- × 機械可読ではない（仕様書に基づいて実装作業を人が行う前提）
- × 機械生成可能ではない（あるいはHTMLからツールで仕様書を逆生成している場合、この仕様書はそもそもなぜ必要なのか？検収や将来のメンテのため？テスト仕様書を作るため？）
- × メンテがしづらい
- × 改定の承認作業は形骸化してませんか？
- × 仕様書の印刷はまだ必要ですか？

| UI仕様書  |                          | 株式会社〇Xシステムズ |                        |
|--------|--------------------------|-------------|------------------------|
| 画面ID   | UI00001/pages/login.aspx | 仕様書形式       | UI仕様2012/UI0000def.xls |
| 画面タイトル | 〇Xシステム ログイン画面            |             |                        |
| No     | コントロール種別                 | 名称          | 初期値                    |
| 1      | ラベル                      | lblUserName | ユーザー名                  |
| 2      | テキストボックス                 | userName    | (なし) 入力必須、50文字まで       |
| 3      | ラベル                      | lblPassword | パスワード                  |
| 4      | パスワード                    | password    | (なし) 入力必須、50文字まで       |
| 5      | ボタン                      | btnLogin    | ログイン                   |

詳細仕様

ラベルをクリックすると対応するテキストボックス・パスワードを選択状態とする。

| 改訂履歴        | 担当者名 | 更新内容         | 承認 |
|-------------|------|--------------|----|
| XXXX年XX月XX日 | 山田   | 新規作成         | 斎藤 |
| XXXX年XX月XX日 | 鈴木   | ボタン名称を変更     | 斎藤 |
| XXXX年XX月XX日 | 鈴木   | 最大入力可能文字数を指定 | 斎藤 |
|             |      |              |    |
|             |      |              |    |
|             |      |              |    |

その仕様書の本質的な情報はこのような YAML でも表現できるのでは？

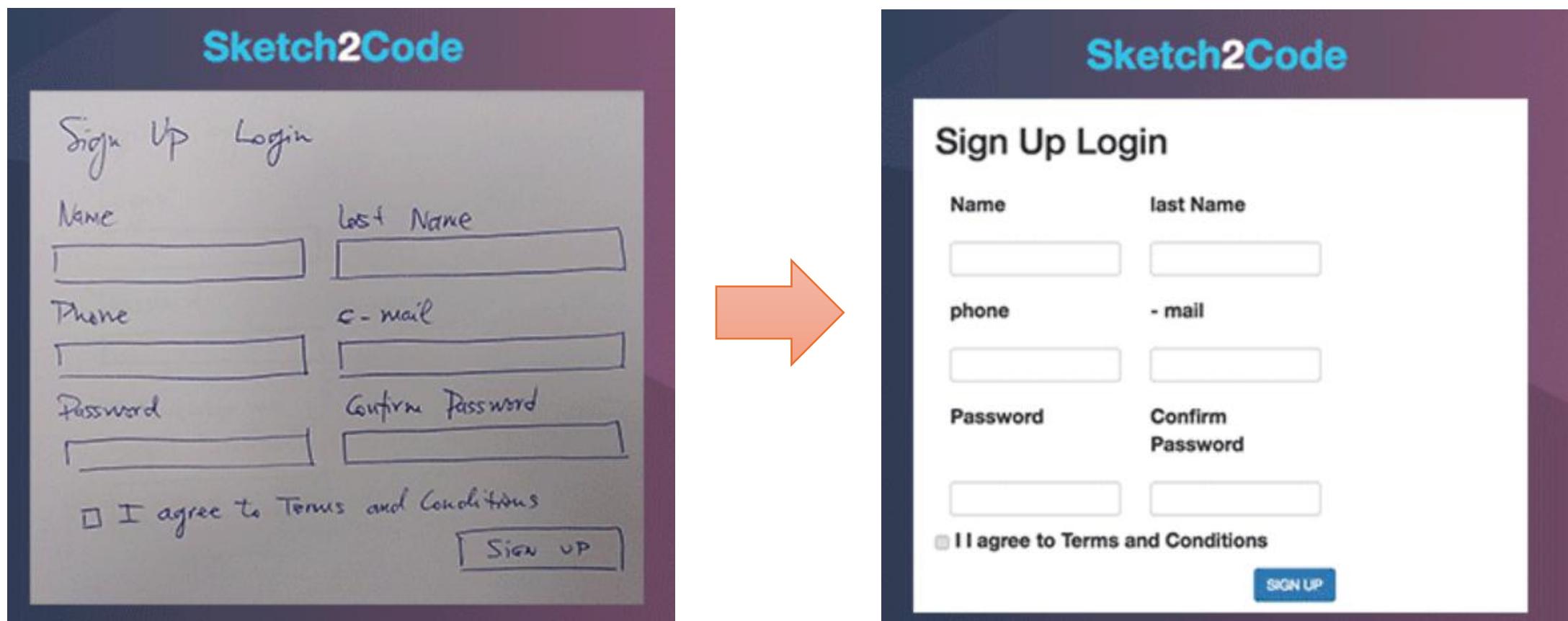
```
controls:
  - control_type: label
    name: lblUserName
    initial_value: ユーザー名
    remarks:
  - control_type: text_box
    name: userName
    initial_value: (なし)
    remarks: 入力必須、50文字まで
  - control_type: label
    name: lblPassword
    initial_value: パスワード
    remarks:
  - control_type: password
    name: password
    initial_value: (なし)
    remarks: 入力必須、50文字まで
  - control_type: button
    name: btnLogin
    initial_value: ログイン
    remarks:
```

- ファイルのバージョン管理が容易（プログラムやHTMLと同様にGitで管理することで、だれがいつどこをどんな理由でどのように変更したのかがわかる）
- 機械可読（このYAMLファイルからHTML等を生成できる、別途バリデータがあれば記述ミスの検出なども可能）
- 機械生成可能（たとえば専用のツールや生成AIを使ってこのデータを作ることが考えられる）
- メンテしやすい（軽微な修正ならばテキストエディタでも可能）
- ドキュメントレベルでの改定の承認作業を省略。このファイルの変更をGitリポジトリでマージする際にレビューを行えばよい。

ご参考: Sketch2Code で、手書きのフォームデザインをHTMLに変換する例。（Azure AI Computer Visionを使用）

※Sketch2Code は現在メンテナンスされていないが、同様のことは現在の生成AI（GPT等）で実現可能。

もはや画面を作るためにUI仕様書が必要というわけではない。手書きのスケッチや、自然言語（日本語など）によるプロンプトでラフなデザインが可能。細かい仕様の定義や検証もAIに任せられるようになる**可能性が高い**。その場合、開発者はUI仕様書やテスト仕様書の記述やメンテ、仕様書に沿った作業に膨大な時間を費やす代わりに、もっと本質的な機能の追加や改善に取り組むことができる。



# よくあるご質問

- 開発チームと運用チームの壁をなくすには？
- 典型的には、以下のような方針の決定と実施を行います。

|   |   |
|---|---|
| <b>共通の目標を持つ</b>   | "開発 vs 運用"ではなく、 <b>共にプロダクトの成功を目指すと<br/>いう認識を持つ</b> 。KPIや成功指標を統一し、開発チームも運用を意識し、運用チームも開発の背景を理解する。 |
| <b>クロスファンクションナルなチーム<br/>(異なる専門分野や役割を持つメンバーが集まり、共通の目標に向かって協力するチーム) を形成する</b> | 開発者と運用担当者が一緒にインフラ管理のコード (IaC) を記述する。セキュリティ担当者が開発の初期段階から関与し、セキュリティを組み込んだ開発を行う (DevSecOps)。       |
| <b>自動化による負担軽減を行う</b>  | インフラのコード化 (IaC) 、CI/CDの導入などで手動の作業を減らす。  |
| <b>コミュニケーションを強化する</b>   | SlackやTeamsなどのチャットツールを活用し、リアルタイムでの情報共有を促進する。定期的な合同ミーティングを設け、課題を共有し、改善策を議論する。                    |

# よくあるご質問

- DevOpsを導入するとどのくらいコスト削減や効率向上が期待できますか？
- →企業あるいはチームによって結果は異なるため一概には言えませんが、一つの調査例によれば、GitHub Enterprise（企業向けのGitHubプラン）を導入することによって、平均して、エンジニアのオンボーディング所要時間が40%削減し、ROIは600%を超え（つまり投資に対して6倍以上の利益が得られ）、1日あたりエンジニアの45分の時間を節約できます。公式Webサイトで詳細を確認できます。



# よくあるご質問

- 開発やDevOpsでAIは利用できますか？ AIはどのように開発作業を支援しますか？
- → はい、開発・DevOpsの分野でもAIが活用されるようになっています。例：
- AIでコード、ドキュメント、テストを生成する
- AIでコードレビューを行う
- AIで一連のコマンドやスクリプトを生成して実行する
- AIでデバッグ、エラーや障害の分析・対策、セキュリティ対策を行う
- AIでDevOpsパイプラインのワークフロー（YAML）を生成する
- SWE（Software Engineering）エージェントを利用する（バグ修正などの典型的な開発作業を自動的に行う）※2025/5 Project Padawan 発表、2025後半に提供予定
- これらにご興味があればぜひ「**AZ-2007 GitHub Copilot を使用してアプリ開発を高速化する**」コースのご受講をご検討ください

# よくあるご質問

- ・マイクロソフトは現在、 Azure DevOps ServiceとGitHubの2つを提供しています。どちらを採用すべきですか？
- ・→簡単に言えば、手軽で使いやすいDevOpsツールがほしい場合は GitHub、企業向けの豊富な機能を利用したい場合は Azure DevOps Service という選択になります。
- ・どちらを採用しても似たようなDevOps機能（プロジェクト管理、バージョン管理、CI/CDなど）が利用できますが、多くの場合、GitHubのほうがより簡単に使え、 Azure DevOps Serviceのほうはより高度で複雑な機能を利用できる、というパターンが多いように思います。
- ・Azure DevOps Serviceにしかない企業向けの機能があります。たとえばEntra IDを使用したユーザー管理・アクセス制御など。
- ・ぜひ両方を検討してみてください！

# よくあるご質問

- ・マイクロソフトがGitHubを買収したことにより、今後Azure DevOpsやGitHubのどちらかが廃止されたり、2つが統合されたりすることはありますか？
- ・→特にそのような予定はありません。どちらも引き続きご利用いただけます。
- ・特にGitHubに関して、マイクロソフトは買収時に「何も変えない」「独立したオープンなプラットフォームであり続ける」ことを約束しています。

# モジュール1

- DevOps とは？
- DevOps の普及度
- DevOps のメリット
- スクラム開発やアジャイル開発との関係は？
- ウォーターフォール型開発との関係は？
- DevOps の「∞」の図について
- DevOps のカルチャ（文化）とは？
- DevOps によるアプリケーション開発の最適化
- GitHub とは？
- 「シフトレフト」と「シフトライト」とは？
- よくあるご質問 / **まとめ**

# モジュール1まとめ

|             |   |
|-------------|---|
| DevOps      | 顧客に継続的に価値を届けるための人・プロセス・技術の集まり。開発担当チーム（Developers）と運用担当チーム（Operations）が連携・協力して開発を行う。 |
| DevOpsのメリット | 新製品や新機能の市場投入までの時間の短縮、市場競争力の強化、サービスの安定性と信頼性の維持・向上、問題の迅速な解決など                         |
| その他開発手法との関連 | DevOpsは「スクラムソフトウェア開発」「アジャイルソフトウェア開発」「リーンスタートアップ」などと相互に関連しながら発展している。                 |
| GitHub      | ソフトウェア開発プラットフォーム。ソースコード管理、問題管理、Project管理、CI/CDなどの機能を提供。                             |
| シフトレフト      | 各種の対策の「前倒し」実施。開発工程のより早い段階でセキュリティ検証やテストを実施するなど。                                      |
| シフトライト      | 各種の対策の「後倒し」実施。本番環境においてデプロイ後にテストやセキュリティ監視を行うなど。                                      |

# 講義



AZ-2008  
DevOpsの基礎:  
中心となる原則と実践

DevOps foundations: The core principles and practices

- ・モジュール1 DevOps の概要
- ・モジュール2 DevOps を使用した計画 (Plan with DevOps)
- ・モジュール3 DevOps を使用した開発 (Develop with DevOps)
- ・モジュール4 DevOps を使用した提供 (Deliver with DevOps)
- ・モジュール5 DevOps を使用した操作 (Operate with DevOps)
- ・まとめ

## モジュール2



プロジェクト管理

### DevOps を使用した計画 (Plan with DevOps)

GitHub Projects ボードを使用して、ソフトウェア開発プロジェクトをアジャイル方式で計画します。バージョンコントロール、継続的インテグレーションと継続的テスト、コードとしてのインフラストラクチャを使用して、コラボレーション、共同責任、継続的学習、最適化を容易にします。

# モジュール2 プロジェクト管理

- DevOpsにおけるプロジェクト管理の概要
  - バージョン管理とは？
  - 繙続的インテグレーションと継続的デリバリーとは？
  - タスク管理とは？
  - タスク管理の手法 「かんばん」 vs 「スクラム」
- GitHub を利用したタスク / プロジェクト管理
  - GitHub Issues
  - GitHub Projects
- よくあるご質問
- まとめ

# モジュール2 プロジェクト管理

- DevOpsにおけるプロジェクト管理の概要
  - バージョン管理とは？
  - 繙続的インテグレーションと継続的デリバリーとは？
  - タスク管理とは？
  - タスク管理の手法 「かんばん」 vs 「スクラム」
- GitHub を利用したタスク / プロジェクト管理
  - GitHub Issues
  - GitHub Projects
- よくあるご質問
- まとめ

# バージョン管理とは？

- ・ソースコードや文書の変更履歴の管理
- ・**誰が、いつ、どのファイルのどの部分を、なぜ、どのように変更したか**、の情報を管理
- ・バージョン管理ソフトウェア
  - ・昔は Subversion などがよく使用されていた
  - ・現在は Git が主流（デファクトスタンダード）となっている

# バージョン管理ツールの分類

| 分類         | (中央) 集中型バージョン管理               | 分散型バージョン管理                     |
|------------|-------------------------------|--------------------------------|
| 具体例        | Subversionなど                  | Gitなど                          |
| 概要         | 1つの中央リポジトリでコードを管理             | 各開発者が中央リポジトリのコピー（ローカルリポジトリ）を持つ |
| ネットワーク     | 中央リポジトリに対する操作時には常にネットワーク接続が必要 | ローカルリポジトリでの作業時は、ネットワーク接続は不要    |
| 利用の難易度     | 学習しやすい                        | 学習に時間がかかる                      |
| DevOpsでの利用 | 最近はほとんど使われない                  | デファクトスタンダード                    |

Git / GitHub を使ったバージョン管理については **モジュール3** で解説する。

# モジュール2 プロジェクト管理

- DevOpsにおけるプロジェクト管理の概要
  - バージョン管理とは？
  - 繙続的インテグレーションと継続的デリバリーとは？
  - タスク管理とは？
  - タスク管理の手法 「かんばん」 vs 「スクラム」
- GitHub を利用したタスク / プロジェクト管理
  - GitHub Issues
  - GitHub Projects
- よくあるご質問
- まとめ

# 継続的インテグレーションと 継続的デリバリーとは？

- DevOpsでは「継続的インテグレーション」と「継続的デリバリー」を活用して操作を自動化する
- 継続的インテグレーション (Continuous Integration, CI)
  - コードの変更をマージし、ビルド・テストを行う
  - 詳しくは **モジュール3** で解説する
- 継続的デリバリー (Continuous Delivery, CD)
  - ビルドされた成果物をステージング環境や本番環境に配置する
  - 本番環境へのリリースまでを自動化する場合は「継続的デプロイ (Continuous Deploy)」と呼ばれる
  - 詳しくは **モジュール4** で解説する

# モジュール2 プロジェクト管理

- DevOpsにおけるプロジェクト管理の概要
  - バージョン管理とは？
  - 繙続的インテグレーションと継続的デリバリーとは？
  - タスク管理とは？
    - タスク管理の手法 「かんばん」 vs 「スクラム」
- GitHub を利用したタスク / プロジェクト管理
  - GitHub Issues
  - GitHub Projects
- よくあるご質問
- まとめ

# タスク管理とは？

- ・プロジェクトで開発中のソフトウェアのバグ（修正依頼）
- ・プロジェクトで開発中のソフトウェアに対する新機能追加の提案
- ・これらの**タスク（1件の作業）**をプロジェクト管理ツール（GitHub IssueやGitHub Projectなど）に記録し、管理する（**タスク管理**）
  - ・GitHubでは「タスク」ではなく「Issue」や「Item」と呼ぶ
  - ・GitHub Issues: 「Issue」を管理する
  - ・GitHub Projects: 「Item」を管理する
    - ・「Item」はIssueまたはPull Request

# よくある質問

- ・「タスク」、「イシュー」、「アイテム」といった用語を使っていますが、ウチの会社で使っている用語とちょっと違うようです・・・
- ・→はい、プロジェクトの中の「1件の作業」を表す言葉はツールや開発プロセスによって様々なので、職場によっては異なる呼び方をしているかもしれません。

# ご参考: タスク管理の用語 (**ツール**による違い)

- **ツール**によって「1件の作業」を表す用語が異なる
  - Microsoft Project: 「タスク」を管理する
  - Azure DevOps: 「作業項目（Work Item）」を管理する
  - GitHub
    - GitHub Issues: 「Issue」を管理する
    - GitHub Projects: 「Item」を管理する
      - 「Item」はIssueかPull Request
  - RedMine, Jira: 「チケット」を管理する
    - 「チケットを起票しその間にタスクを書く」など
  - Trello: 「カード」（タスクカード）を管理する
    - 「カードをリストに入れる」など

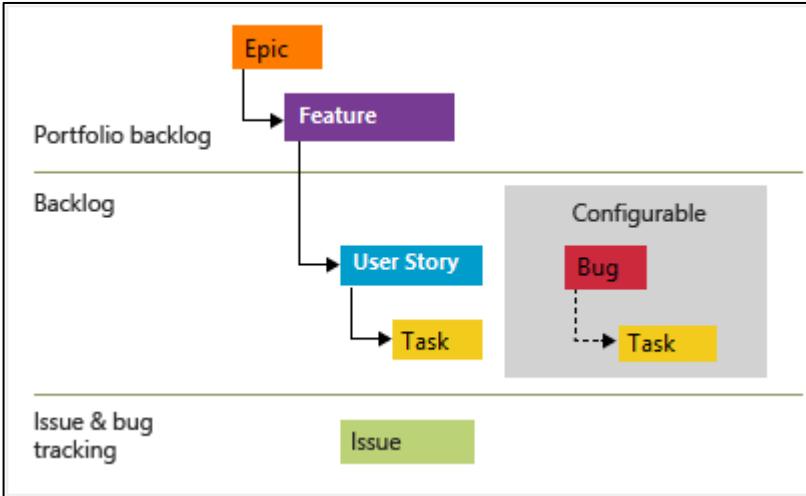
# ご参考: タスク管理の用語（**開発プロセス**による違い）

- **開発プロセス**によって「1件の作業」の**種類（分類方法）**が異なる
  - 「タスク」、「エピック」、「イシュー（Issue）」
    - 「エピックは最上位の作業単位（アイテム）で、大規模なユーザーストーリーを表現するもの」など
    - 「イシューを分解・具体化・期限設定したものがタスク」など
  - 「フィーチャー」（機能要望）、「バグ」など
  - プロジェクトの進行を妨げる障害 = 「インペディメント（impediment）」など

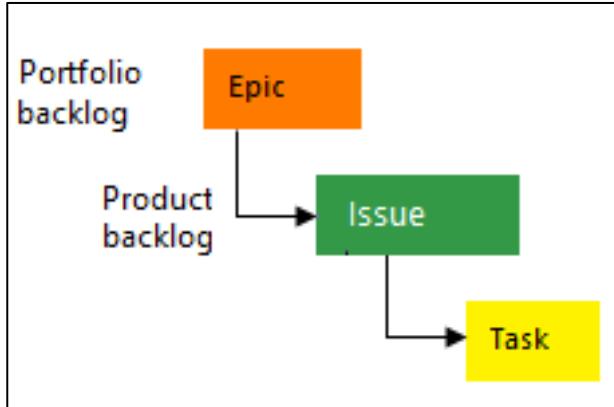
# ご参考: 各開発プロセスにおける作業項目の種類

※本講義ではこれらについて解説しない

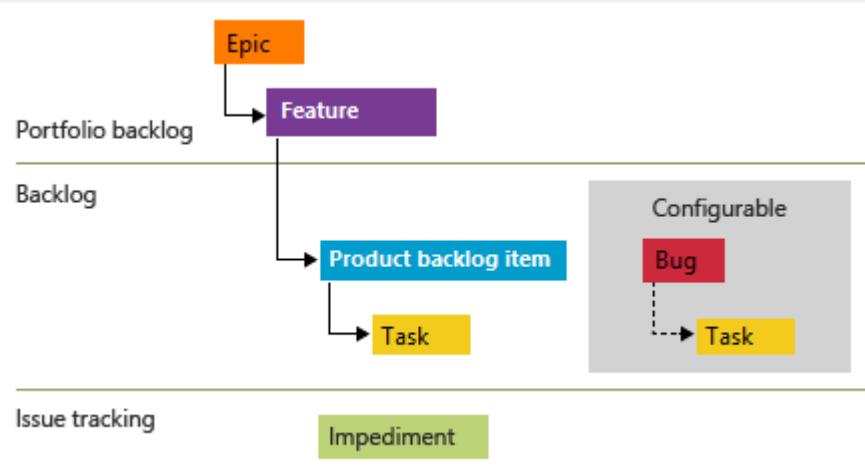
## アジャイル プロセス



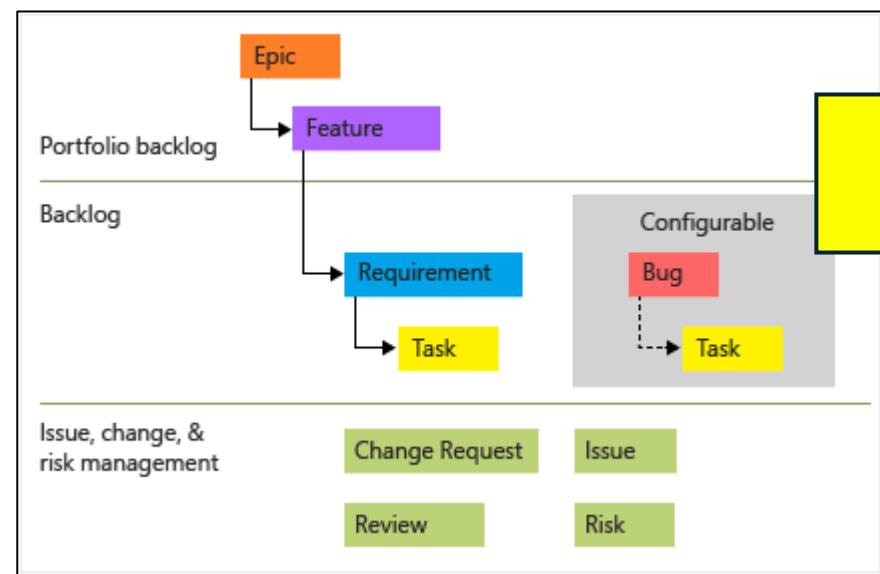
## 基本 プロセス



## スクラム プロセス



## CMMI プロセス



# モジュール2 プロジェクト管理

- DevOpsにおけるプロジェクト管理の概要
  - バージョン管理とは？
  - 繙続的インテグレーションと継続的デリバリーとは？
  - タスク管理とは？
  - タスク管理の手法 「かんばん」 vs 「スクラム」
- GitHub を利用したタスク / プロジェクト管理
  - GitHub Issues
  - GitHub Projects
- よくあるご質問
- まとめ

# タスク管理の手法

|              |  |  |
|--------------|--|--|
| 手法           | かんばん（ソフトウェア開発）   | スクラム（ソフトウェア開発）   |
| 登場年代         | 1940～  | 1980～  |
| 概要           | トヨタ生産方式の一部として誕生。視覚的な管理システムを活用して作業の流れを最適化。  | アジャイル開発手法の一つ。 <b>1～4週間の反復作業（スプリント）</b> を通じて、チームが協力して価値を最大化。  |
| 特徴           | 「かんばん」を使ったタスクの視覚化、WIP制限（Work In Progress制限、チーム全体として同時進行するタスクの数を制限）、継続的作業（スプリントのような固定期間を持たず作業を進め、進行状況を常に監視する）、継続的な改善（隨時、改善） | プロダクトバックログ（TODO）、スプリント計画会議（バックログの中から各スプリントの作業項目を選びメンバーに割り当てる）、デイリースクラム（毎日の進捗報告ミーティング）、スプリントレビュー（スプリントの成果確認）、レトロスペクティブ（スプリントの振り返り、改善） |
| タスク（付箋）管理ツール | かんばんボード  | スクラムボード（タスクボード）  |
| スプリント        | なし。優先順位の高いタスクを順次進行させる。タスク単位で期限を設定。   | あり。どのスプリントでどのタスクを完了させるかを明確にする。   |

# モジュール2 プロジェクト管理

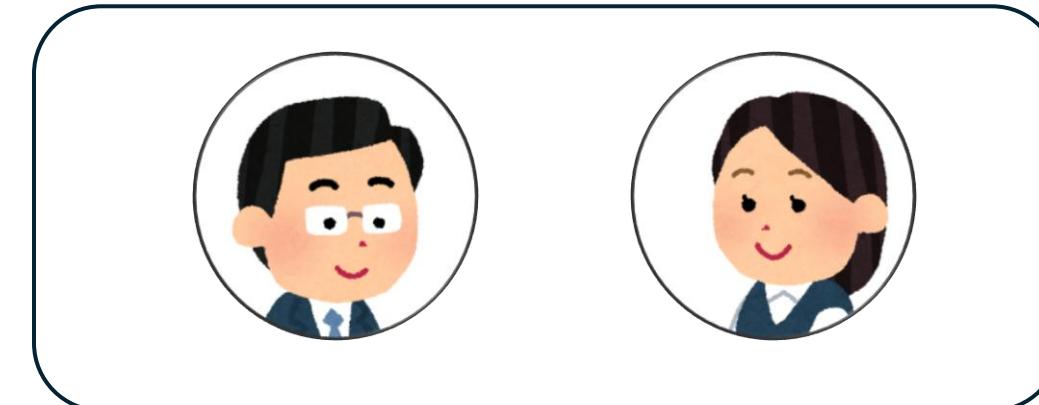
- DevOpsにおけるプロジェクト管理の概要
  - バージョン管理とは？
  - 繙続的インテグレーションと継続的デリバリーとは？
  - タスク管理とは？
  - タスク管理の手法 「かんばん」 vs 「スクラム」
- GitHub を利用したタスク / プロジェクト管理
  - GitHub Issues
  - GitHub Projects
- よくあるご質問
- まとめ

# GitHubでのタスク管理: GitHub Issues

- ・「**GitHub Issues**」では、以下のものを「**issue**（イシュー、問題）」として追跡（トラッキング）できる。
  - ・アイデア (ideas)
  - ・フィードバック (feedback)
  - ・**タスク** (tasks)
  - ・作業 (work)
  - ・バグレポート (bug reports)
  - ・新機能 (new features)
  - ・その他何でも (anything else)

# GitHubを使用したタスク管理の例

- GitHub リポジトリを作る
- リポジトリの中で Issues を管理する
  - Issue の作成
  - Issue へのコメント
  - 担当者 (Assignee) の割り当て
  - Issue に対応する修正作業の実施
  - Issue のコメントでの「コミットID」や「メンション」の使用
  - Issue のクローズ



■GitHub (<https://github.com/>) にサインインし、画面右上の「+」をクリック、「New repository」をクリック

The screenshot shows the GitHub Home page. At the top right, there is a navigation bar with several icons. One icon, a plus sign (+), is highlighted with a red box and a red arrow pointing to it from the top right. A dropdown menu has appeared, listing several options: "New repository" (which is also highlighted with a red box), "Import repository", "New codespace", "New gist", "New organization", and "New project". The "New repository" option is the first item in the list.

Dashboard

Top repositories

New

Add a repository...

Home

Ask Copilot

Branch sync patterns

Expla

Feed

Show more

New repository

Import repository

New codespace

New gist

New organization

New project

## ■リポジトリ名を入力して「Create repository」をクリック



New repository

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?  
[Import a repository.](#)

Required fields are marked with an asterisk (\*).

Repository template

No template ▾

Start your repository with a template repository's contents.

Owner \* / Repository name \*

repo4

repo4 is available.

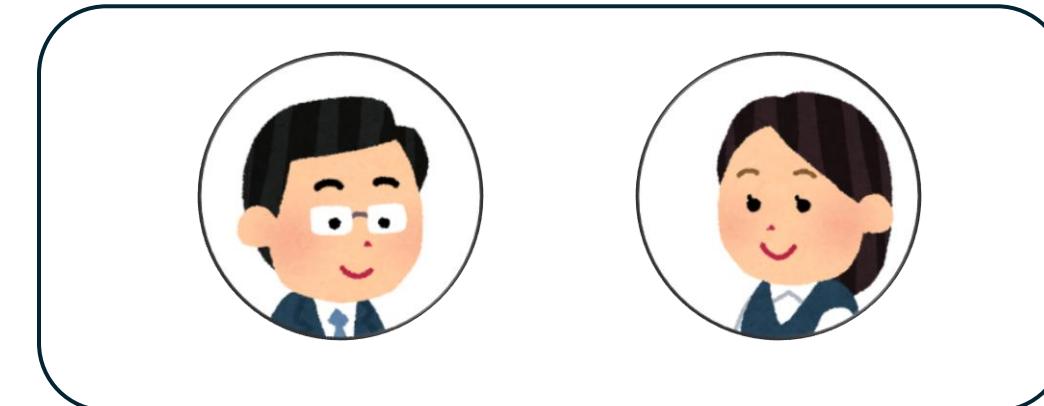
Great repository names are short and memorable. Need inspiration? How about [bookish-doodle](#) ?

Description (optional)

Create repository

# GitHubを使用したタスク管理の例

- GitHub リポジトリを作る
- リポジトリの中で Issues を管理する
  - Issue の作成
  - Issue へのコメント
  - 担当者 (Assignee) の割り当て
  - Issue に対応する修正作業の実施
  - Issue のコメントでの「コミットID」や「メンション」の使用
  - Issue のクローズ



## ■ 「Issues」をクリック

The screenshot shows a GitHub repository page for 'repo4'. The top navigation bar includes links for Code, Issues (which is highlighted with a red box), Pull requests, Actions, Projects, Security, Insights, and more. Below the navigation is a header with the repository name 'repo4' (Private), Unwatch (1), Fork (0), and Star (0) buttons. On the left, there's a profile picture of a man with glasses and a 'Create a codespace' button. Two main callout boxes are present: one for 'Start coding with Codespaces' and another for 'Add collaborators to this repository'. At the bottom, a light blue box contains instructions for quick setup, connection options (Set up in Desktop, HTTPS, SSH), and a copy link icon. A note at the bottom encourages adding files like README, LICENSE, and .gitignore.

/ repo4

Code Issues Pull requests Actions Projects Security Insights

repo4 Private

Unwatch 1 Fork 0 Star 0

Start coding with Codespaces

Add a README file and start coding in a secure, configurable, and dedicated development environment.

Create a codespace

Add collaborators to this repository

Search for people using their GitHub username or email address.

Invite collaborators

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH git@github.com: /repo4.git

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

## ■ 「New issue」をクリック

The screenshot shows the GitHub Issues page interface. At the top, there are navigation links: Code, Issues (which is selected), Pull requests, Actions, Projects, Security, Insights, and a three-dot menu. Below the navigation is a search bar containing the query "is:issue state:open". To the right of the search bar are buttons for Labels, Milestones, and a prominent green "New issue" button, which is highlighted with a red box. Further down, there are filters for Open (0) and Closed (0) issues, and dropdown menus for Author, Labels, Projects, and Milestones, each with a downward arrow indicating they are filterable. On the left side of the main content area, there is a circular profile picture of a person with glasses and a suit. The central message "No results" is displayed above the text "Try adjusting your search filters."

## ■ Issueの「タイトル」 「説明文」を記述し、「Create」をクリック

Create new issue

Add a title \*

プロジェクトに README ファイルがない

Add a description

Write Preview

リポジトリに `README.md` ファイルを作る。

Paste, drop, or click to add files

Create more

Cancel

Create

Assignees  
No one - [Assign yourself](#)

Labels  
No labels

Projects  
No projects

Milestone  
No milestone

### ■新しいIssueが作成された。

Code Issues Pull requests Actions Projects Security Insights Settings

# プロジェクトに README ファイルがない #1

Open

opened now

リポジトリに README.md ファイルを作る。

Create sub-issue

Add a comment

Write Preview

Use Markdown to format your comment

Paste, drop, or click to add files

Close issue  Comment

Assignees  
No one - Assign yourself

Labels  
No labels

Projects  
No projects

Milestone  
No milestone

Relationships  
None yet

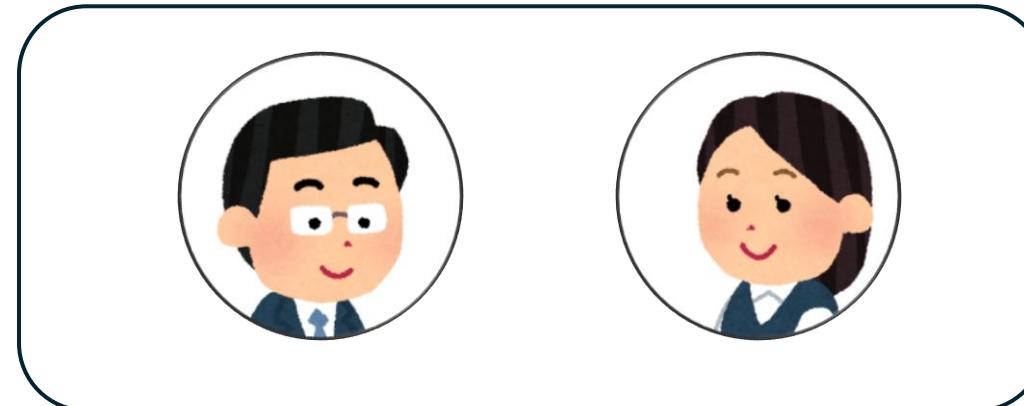
Development

Code with Copilot Agent Mode

Create a branch for this issue or link a pull request.

# GitHubを使用したタスク管理の例

- GitHub リポジトリを作る
- リポジトリの中で Issues を管理する
  - Issue の作成
  - Issue へのコメント
  - 担当者 (Assignee) の割り当て
  - Issue に対応する修正作業の実施
  - Issue のコメントでの「コミットID」や「メンション」の使用
  - Issue のクローズ



■ Issueにコメントを追加することができる。コメント本文を書いて「Comment」ボタンをクリック。

# プロジェクトに README ファイルがない #1

Open



opened 19 minutes ago

リポジトリに README.md ファイルを作る。

Create sub-issue ... 



Add a comment

Write Preview

H B I | 三 <> ⌂ | 三 二三 丶三 | @ ⌂ ← ⌂

この件は私が担当します

Close with comment

Comment

■コメントが追加された。

# プロジェクトに README ファイルがない #1

Open



opened 21 minutes ago

リポジトリに README.md ファイルを作る。

Create sub-issue





now

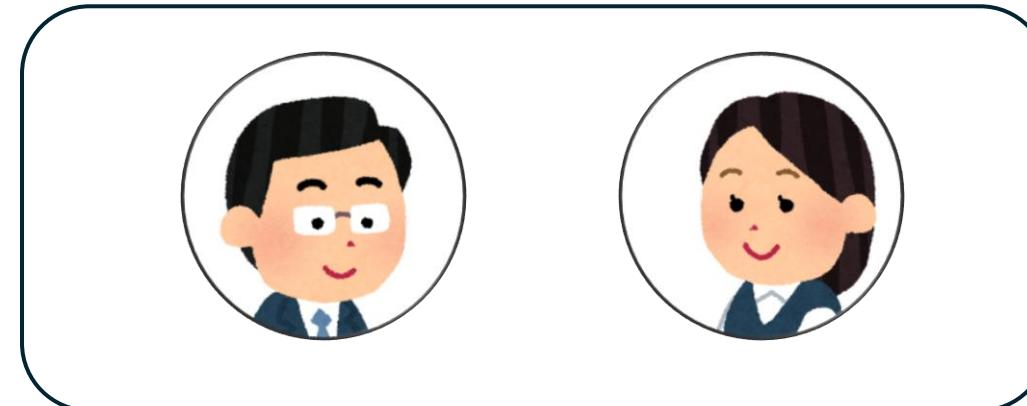
この件は私が担当します





# GitHubを使用したタスク管理の例

- GitHub リポジトリを作る
- リポジトリの中で Issues を管理する
  - Issue の作成
  - Issue へのコメント
  - 担当者 (Assignee) の割り当て
  - Issue に対応する修正作業の実施
  - Issue のコメントでの「コミットID」や「メンション」の使用
  - Issue のクローズ



■ 「Assignees」の歯車をクリックして、このIssueの担当者として自分を選択。

プロジェクトに READMEファイルがない #1

Open

opened 47 minutes ago

リポジトリに README.md ファイルを作る。

Create sub-issue

25 minutes ago

この件は私が担当します

Assignees

Assign up to 10 people to this issue

Filter assignees

No projects

Milestone

No milestone

The screenshot shows a GitHub issue page for a project titled "プロジェクトに READMEファイルがない #1". The issue is currently open. The main content area contains two comments: one from a user named "user" (represented by a male profile icon) suggesting to create a README.md file, and another from a user named "user" (represented by a female profile icon) stating they will handle it. To the right of the comments is an "Assignees" dropdown menu. This menu has a title "Assign up to 10 people to this issue" and a search bar labeled "Filter assignees". Below the search bar is a list of users. The first user in the list, represented by a male profile icon, has a red box drawn around their icon. A red arrow points from this box to the top-right corner of the entire assignees menu, where there is a small gear icon. Another red box highlights the gear icon itself. The menu also includes sections for "No projects" and "Milestone", each with its own gear icon.

■担当者が設定された。

担当者の設定は任意だが、設定しておくと、誰がこのIssueを担当しているのかがわかりやすくなる。

プロジェクトに READMEファイルがない #1

Open

opened 49 minutes ago

リポジトリに README.md ファイルを作る。

Create sub-issue

27 minutes ago

この件は私が担当します

self-assigned this now

Edit New issue

Assigees

Labels

No labels

Projects

No projects

Milestone

No milestone

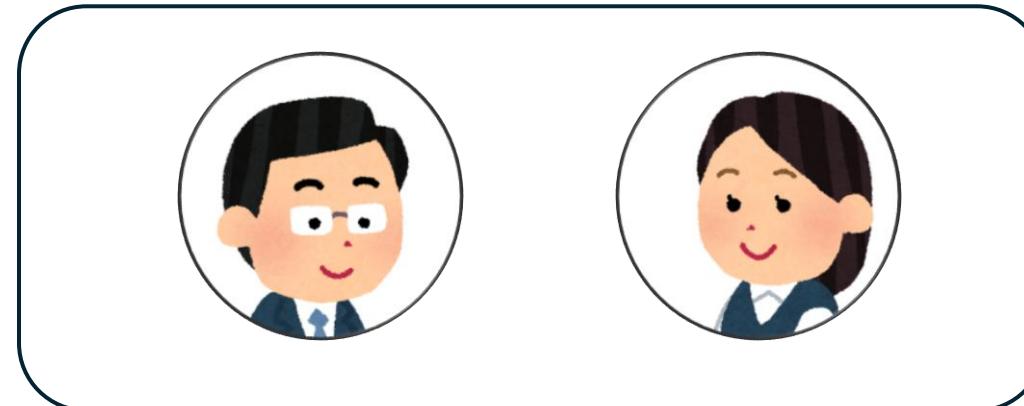
Relationships

None yet

The screenshot shows a GitHub issue page for a project with a README file. The issue is labeled '#1' and is currently open. A comment from a user (represented by a male icon) suggests creating a README.md file. Another user (represented by a female icon) has responded, indicating they will handle the task. This response is highlighted with a red box. On the right side of the page, there are sections for 'Assignees' (listing the female assignee), 'Labels' (none listed), 'Projects' (none listed), 'Milestone' (none listed), and 'Relationships' (none listed). The GitHub interface includes standard navigation buttons like 'Edit' and 'New issue'.

# GitHubを使用したタスク管理の例

- GitHub リポジトリを作る
- リポジトリの中で Issues を管理する
  - Issue の作成
  - Issue へのコメント
  - 担当者 (Assignee) の割り当て
  - Issue に対応する修正作業の実施
  - Issue のコメントでの「コミットID」や「メンション」の使用
  - Issue のクローズ



## ■新しいファイルの作成を行う。リポジトリのトップページに移動し、Creating a new file をクリック

The screenshot shows a GitHub repository page for 'repo4'. The top navigation bar includes links for Code, Issues (1), Pull requests, Actions, Projects, Security, and Insights. The repository name 'repo4' is displayed with a 'Private' badge. A search bar at the top right contains the placeholder 'Type / to search'. On the left, there's a 'Set up GitHub Copilot' section with a button to 'Get started with GitHub Copilot'. To the right, there's a 'Add collaborators to this repository' section with a button to 'Invite collaborators'. Below these, a 'Quick setup' section provides instructions for setting up the repository, mentioning 'Desktop', 'HTTPS', and 'SSH' options, and a git URL: `git@github.com:hiryamada/repo4.git`. It also suggests starting by creating a new file or uploading an existing one, which is highlighted with a red box. A large blue button at the bottom encourages users to '...or create a new repository on the command line' and provides a terminal command example:

```
echo "# repo4" >> README.md  
git init  
git add README.md  
git commit -m "first commit"  
git branch -M main  
git remote add origin git@github.com:hiryamada/repo4.git  
git push -u origin main
```



■ファイル名と、ファイルの内容を入力。Commit changes... をクリック。

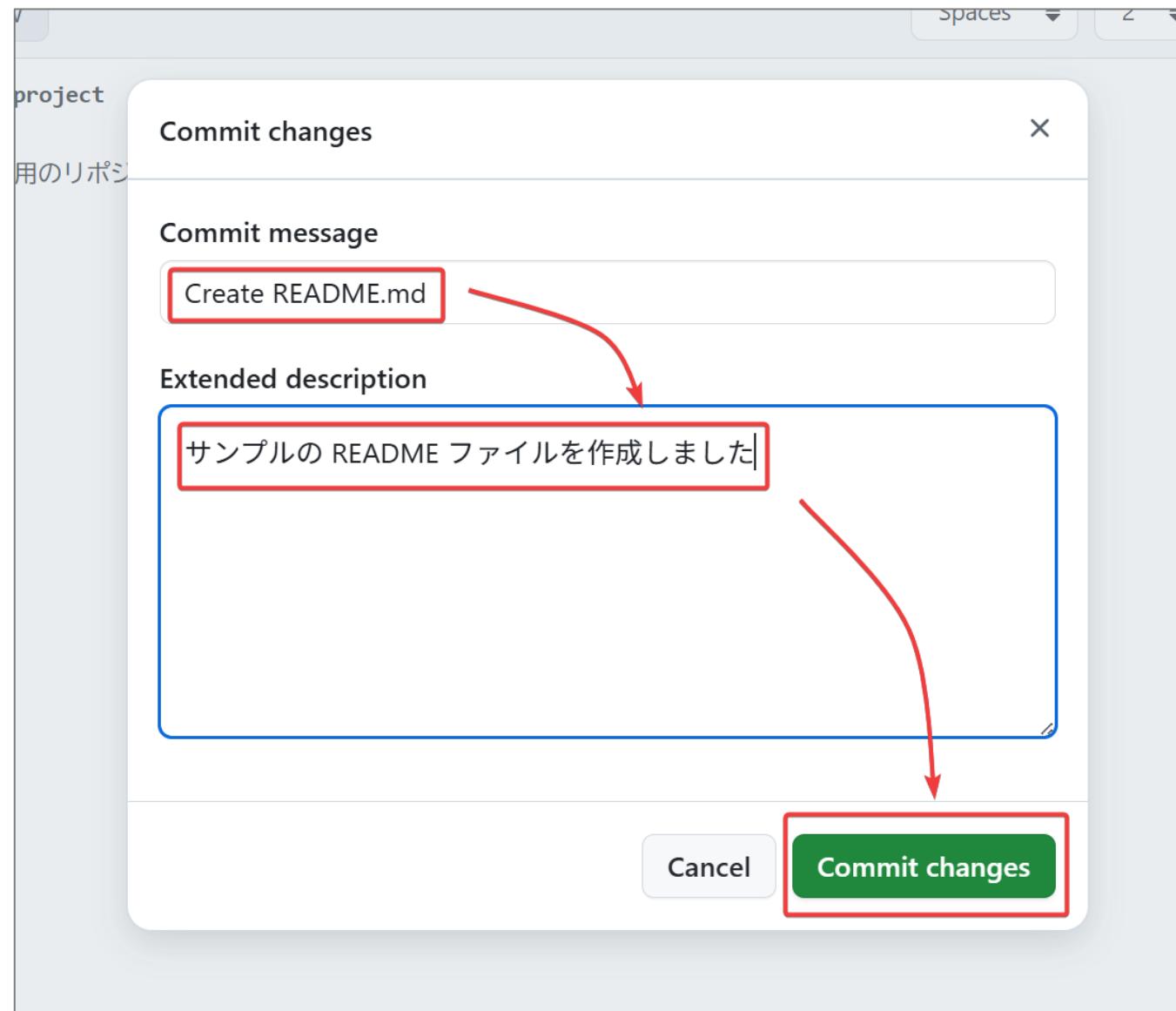
The screenshot shows a GitHub repository interface for a repository named 'repo4'. The 'Code' tab is selected. A file named 'README.md' is open for editing. The content of the file is:

```
1 # Sample project
2
3 これは練習用のリポジトリの `README.md` ファイルです。
4 |
```

The 'README.md' file name is highlighted with a red box. The 'Commit changes...' button in the top right corner of the editor is also highlighted with a red box and has a red arrow pointing to it from the bottom right.

Other visible elements include the GitHub logo, a search bar, and various navigation links like 'Issues', 'Pull requests', 'Actions', 'Projects', 'Security', and 'Insights'.

- 変更をコミットする（変更内容の確定・記録を行う）。  
Commit message（コミットの内容を説明する一文）を入力。  
オプションでExtended descriptionを入力。Commit changesをクリック。



■コミットが実行された（変更内容が確定・記録された）。README.md ファイルが作成された。

repo4 Private

Unwatch 2 Fork 0 Star 0

Help us improve GitHub Codespaces  
Tell us how to make GitHub Codespaces work better for you with three quick questions.

Give feedback ×

main Go to file + <> Code

Merge pull request #3 from hir... · 0c155aa · 1 hour ago

README.md Update README.md 1 hour ago

README

Sample project

これは練習用のリポジトリの README.md ファイルです。

About

No description, website, or topics provided.

Readme Activity 0 stars 2 watching 0 forks

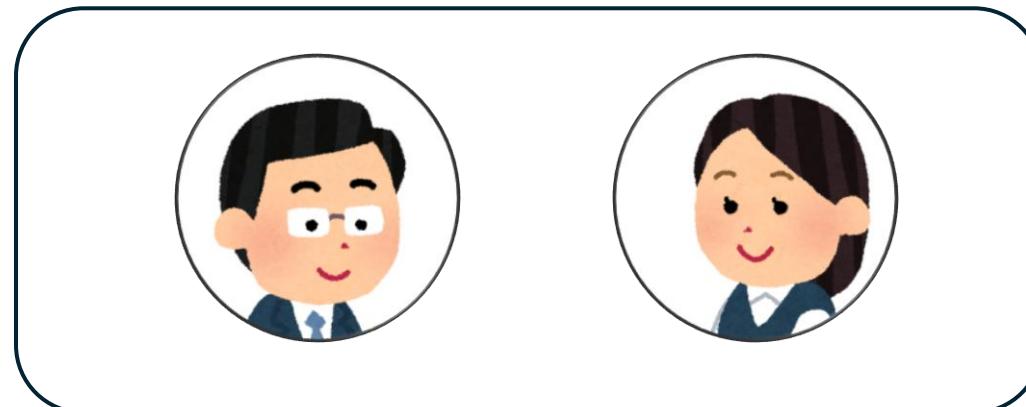
Releases

No releases published Create a new release



# GitHubを使用したタスク管理の例

- GitHub リポジトリを作る
- リポジトリの中で Issues を管理する
  - Issue の作成
  - Issue へのコメント
  - 担当者 (Assignee) の割り当て
  - Issue に対応する修正作業の実施
  - Issue のコメントでの「コミットID」や「メンション」の使用
  - Issue のクローズ



## ■ 時計のアイコン (commits) をクリック

repo4 Private

Unwatch 2 Fork 0 Star 0

Help us improve GitHub Codespaces  
Tell us how to make GitHub Codespaces work better for you with three quick questions.

Give feedback X

main Go to file + Code

Merge pull request #3 from hir... · 0c155aa · 1 hour ago

README.md Update README.md 1 hour ago

README

Sample project

これは練習用のリポジトリの README.md ファイルです。

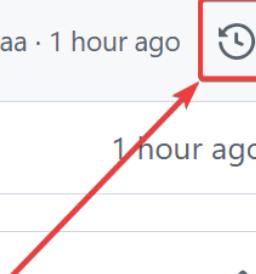
About

No description, website, or topics provided.

Readme Activity 0 stars 2 watching 0 forks

Releases

No releases published Create a new release



- すべてのコミットを確認できる。各コミットには「コミットID」（コミットを識別するチェックサムハッシュ）が付与される。これを使うことで、たとえばIssueのコメントの中で、各コミットを指示示すことができる。コミットIDをコピー。

## Commits

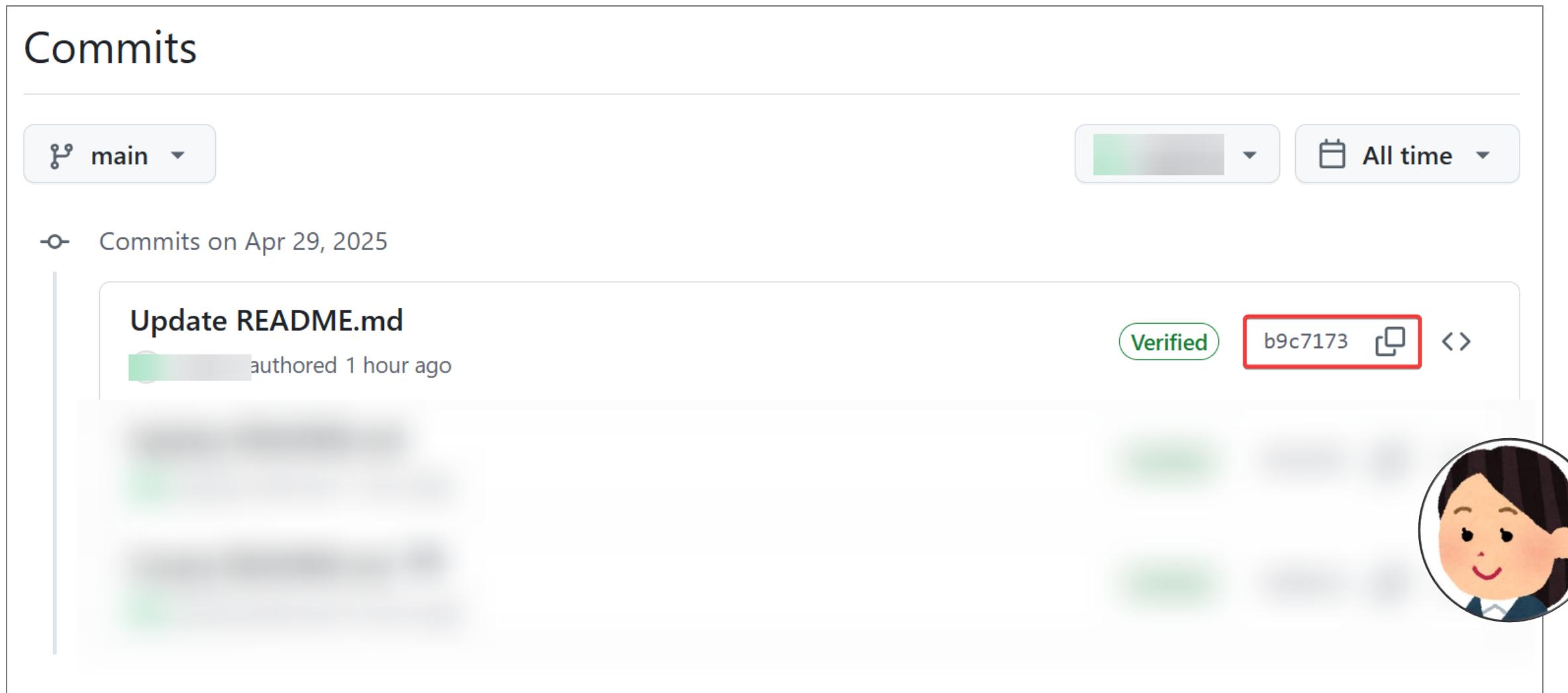
main ▾

All time ▾

-o Commits on Apr 29, 2025

Update README.md  
author 1 hour ago

Verified b9c7173 ↗ <>



- さきほどのIssueに戻り、修正を報告するコメントを書く。コメントにはコミットIDを付与したり、「@ユーザー名」で指定ユーザーに「メンション」（通知）したりできる。

The screenshot shows a GitHub commit comment interface. At the top left is a user icon of a woman with short dark hair. To its right is the title "Add a comment". Below the title is a toolbar with various editing icons: H (Header), B (Bold), I (Italic), a list icon, a code icon, a link icon, a list icon with a fraction, a list icon with a checkmark, and a back arrow. The toolbar also includes tabs for "Write" (selected) and "Preview", and a small edit icon.

A yellow callout box points from the top-left towards the "Write" tab, containing the text "コミットID". Another yellow callout box points from the bottom-left towards the "@[redacted]" placeholder, containing the text "「@ユーザー名」でユーザーを「メンション」できる(そのユーザーに通知される)".

The main comment area contains the following text:

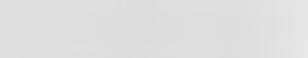
修正しました。  
9d6833a  
@[redacted] 確認をお願いします。|

At the bottom are three buttons: "Add files", a "Close with comment" button with a checked checkbox, and a green "Comment" button.

## ■メンションされたユーザーに送られてきた通知の例（メール）



 <notifications@github.com>  ⤵ ⤶ ⤷ ⤸ | 📄 🌐 ...

宛先:  <repo4@noreply.github.com> 2025/04/30 (水) 0:04

Cc:   他 +1 人

  left a comment ([/repo4#1](#)).

修正しました。  
[9d6833a](#)

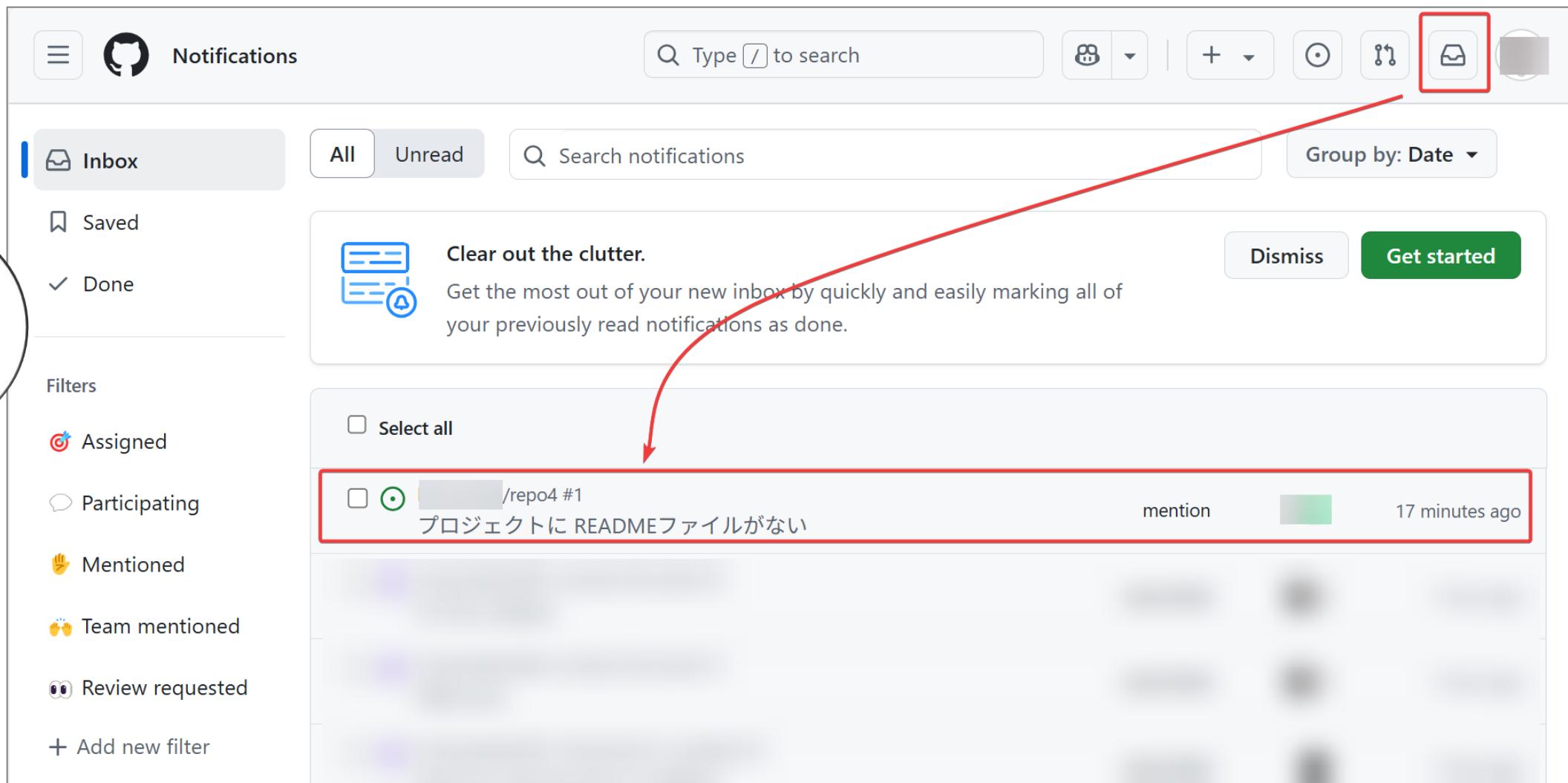
@ 確認をお願いします。

—

Reply to this email directly, [view it on GitHub](#), or [unsubscribe](#).  
You are receiving this because you were mentioned.

 [返信](#)  [全員に返信](#)  [転送](#)

■メンションされたユーザーは、  
画面上部の notifications アイコンをクリックして、一覧形式でもメンションを確認できる



The screenshot shows the GitHub Notifications interface. On the left, there's a sidebar with a user profile picture of a man with glasses and a suit, and a list of filters: Inbox (selected), All, Unread, Saved, Done, Filters, Assigned (highlighted with a red circle), Participating, Mentioned, Team mentioned, Review requested, and Add new filter. At the top right, there are several icons: a search bar, a user dropdown, a plus sign, a circular arrow, a refresh icon, and a mail icon (which is highlighted with a red box). Below the sidebar, there's a promotional message about clearing clutter and marking notifications as done. The main area displays a list of notifications. One notification is highlighted with a red box and a red arrow pointing to it from the 'Mentioned' filter in the sidebar. This notification is from a user named /repo4 #1, mentioning that the project lacks a README file. It includes a checkbox, a green circular icon with a white dot, the repository name, the issue number, the type (mention), a small profile picture, and the timestamp (17 minutes ago).

■コメントのコミットID（リンク）をクリックして、コミットの内容を確認できる。

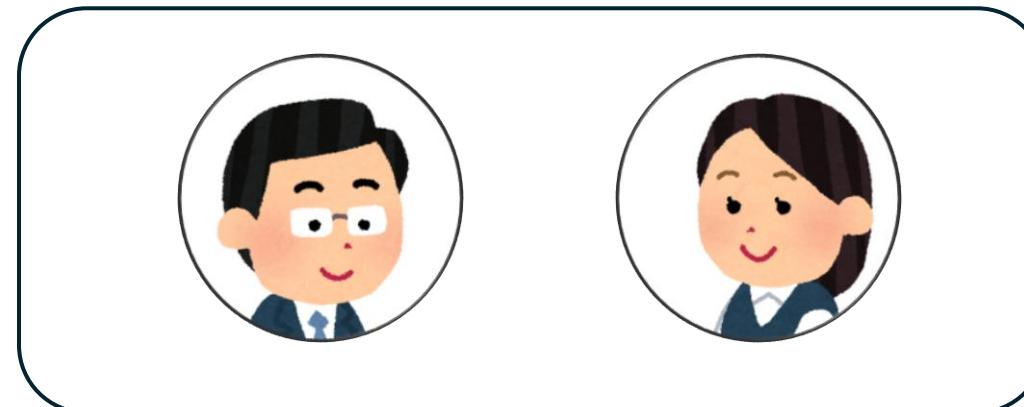
The image shows a GitHub interface. On the left, a pull request is open, and a comment is visible. The comment text is "修正しました。" followed by a link to the commit: [9d6833a](#). Below the link is the text "@ 確認をお願いします。". A large red arrow points from the link in the comment to the commit detail page on the right. The commit detail page shows the commit ID **9d6833a**, which was authored 6 hours ago and is verified. The commit message is "Create README.md" with the note "サンプルの README ファイルを作成しました". The commit has 0 parents and is part of the main branch. The file changed is README.md, showing 1 file changed with +3 -0 lines changed. The diff shows the following changes:

| ... | @@ -0,0 +1,3 @@                     |
|-----|-------------------------------------|
| 1   | + # Sample project                  |
| 2   | +                                   |
| 3   | + これは練習用のリポジトリの `README.md` ファイルです。 |

At the bottom of the commit detail page, there are buttons for "Comments 0" and "Lock conversation".

# GitHubを使用したタスク管理の例

- GitHub リポジトリを作る
- リポジトリの中で Issues を管理する
  - Issue の作成
  - Issue へのコメント
  - 担当者 (Assignee) の割り当て
  - Issue に対応する修正作業の実施
  - Issue のコメントでの「コミットID」や「メンション」の使用
  - Issue のクローズ



■このIssueの作業が完了したので、コメントを書いて、Issueをクローズする。

The screenshot shows a GitHub issue interface. At the top, there is a comment from a user with a green profile picture, posted 21 minutes ago. The comment reads: "修正しました。" followed by a link "9d6833a" and "@ [redacted] 確認をお願いします." Below this is a smiley face emoji.

Below the comments, there is a section titled "Add a comment" with a gray profile picture. The "Write" tab is selected, showing a rich text editor toolbar with buttons for H, B, I, bold, italic, etc. A text area contains the message: "確認しました！ありがとうございます。この Issue をクローズします。".

At the bottom right of the comment input area, there are two buttons: "Close with comment" (with a checked checkbox icon) and "Comment". Both of these buttons are highlighted with a red rectangular border.

# プルリクエスト

- ・ここまでで、Issues の基本的な使い方が説明できた。
- ・ただし、通常、GitHub上で複数のユーザーが協力して作業を行う場合「**プルリクエスト**」というしくみを使う。
- ・また、修正作業は独立した「**ブランチ**」上で実施する。
- ・これらについては **モジュール3** で解説する。

# モジュール2 プロジェクト管理

- DevOpsにおけるプロジェクト管理の概要
  - バージョン管理とは？
  - 繙続的インテグレーションと継続的デリバリーとは？
  - タスク管理とは？
  - タスク管理の手法 「かんばん」 vs 「スクラム」
- GitHub を利用したタスク / プロジェクト管理
  - GitHub Issues
  - GitHub Projects
- よくあるご質問
- まとめ

# GitHub Projects

- GitHubが提供する、プロジェクト管理機能
- GitHubの一部であり、追加費用なしで導入できる
- GitHubの既存の機能（GitHub Issues）とうまく統合されている
- 比較的簡単に利用できる

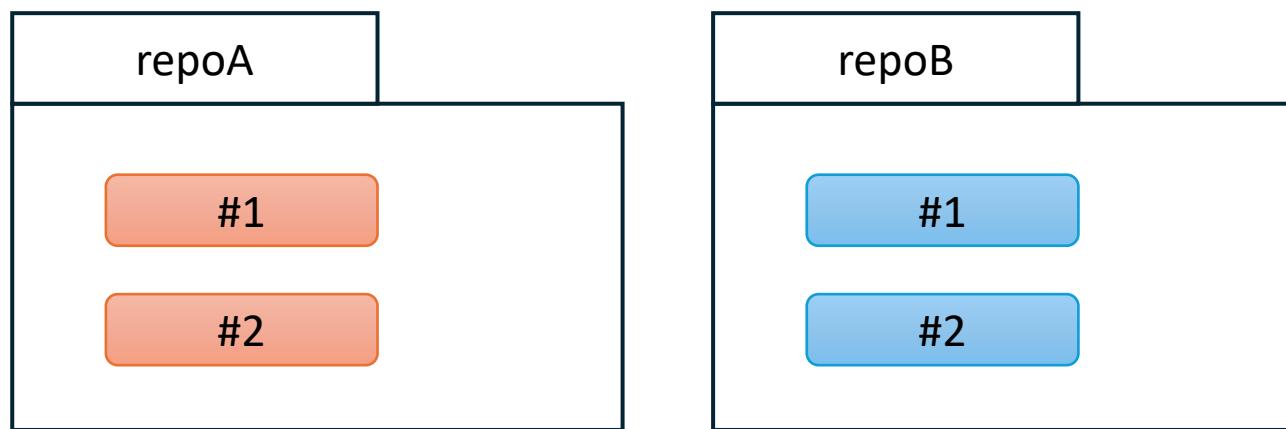
# GitHub Projects

- ・実態としては「複数のリポジトリにまたがって、Issueをわかりやすく表示・編集するツール」といった簡易的なもの
- ・**フル機能のプロジェクト管理ツール・スケジュール管理ツール・リソース管理ツールではない**
- ・一般的なプロジェクト管理ツールにある便利な機能がGitHub Projectsにはない場合が多い。
  - ・×予算・コスト管理
  - ・×人材管理・稼働管理
  - ・×設備管理
  - ・×リスク管理
  - ・×土日祝日などを考慮したスケジューリング
  - ・×WBS作成・ガントチャート作成
  - ・×各ユーザーの細かい操作権限設定
  - ・×PMBOK等に準拠したドキュメント管理

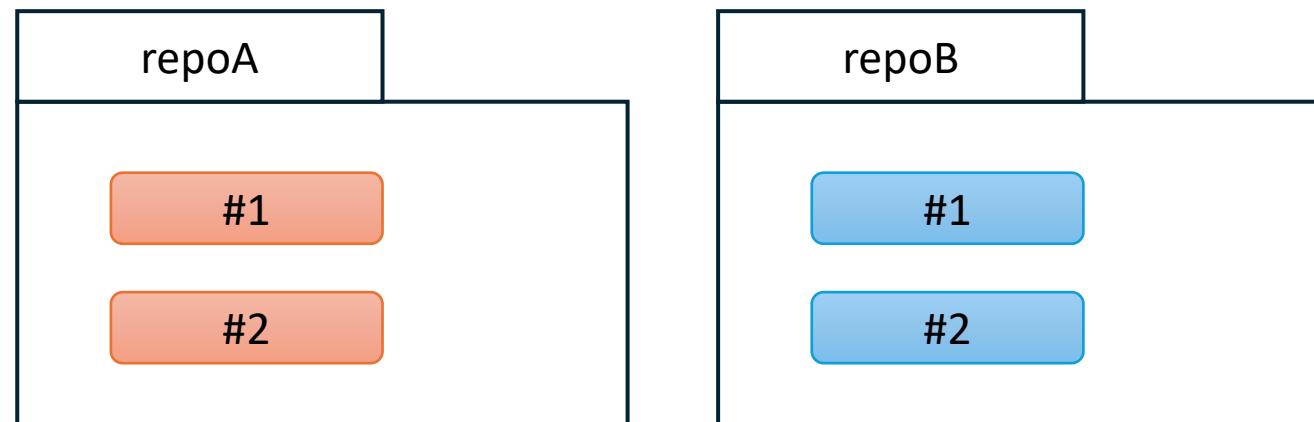
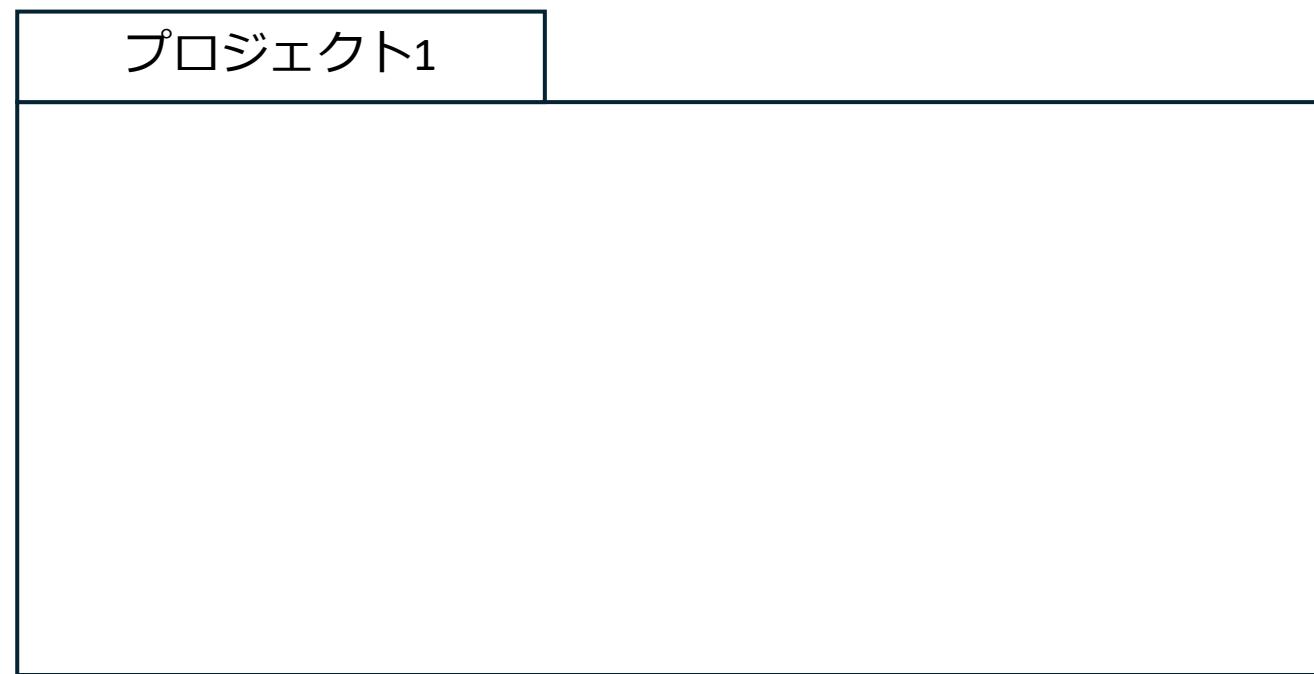
# GitHubを使用したプロジェクト管理

- GitHub Projects でのプロジェクト管理
  - どのような場合にプロジェクトを作るのか？
  - プロジェクトを作る
  - Item (IssueとDraft) を作る
  - Boardビュー
  - Tableビュー
  - Roadmapビュー

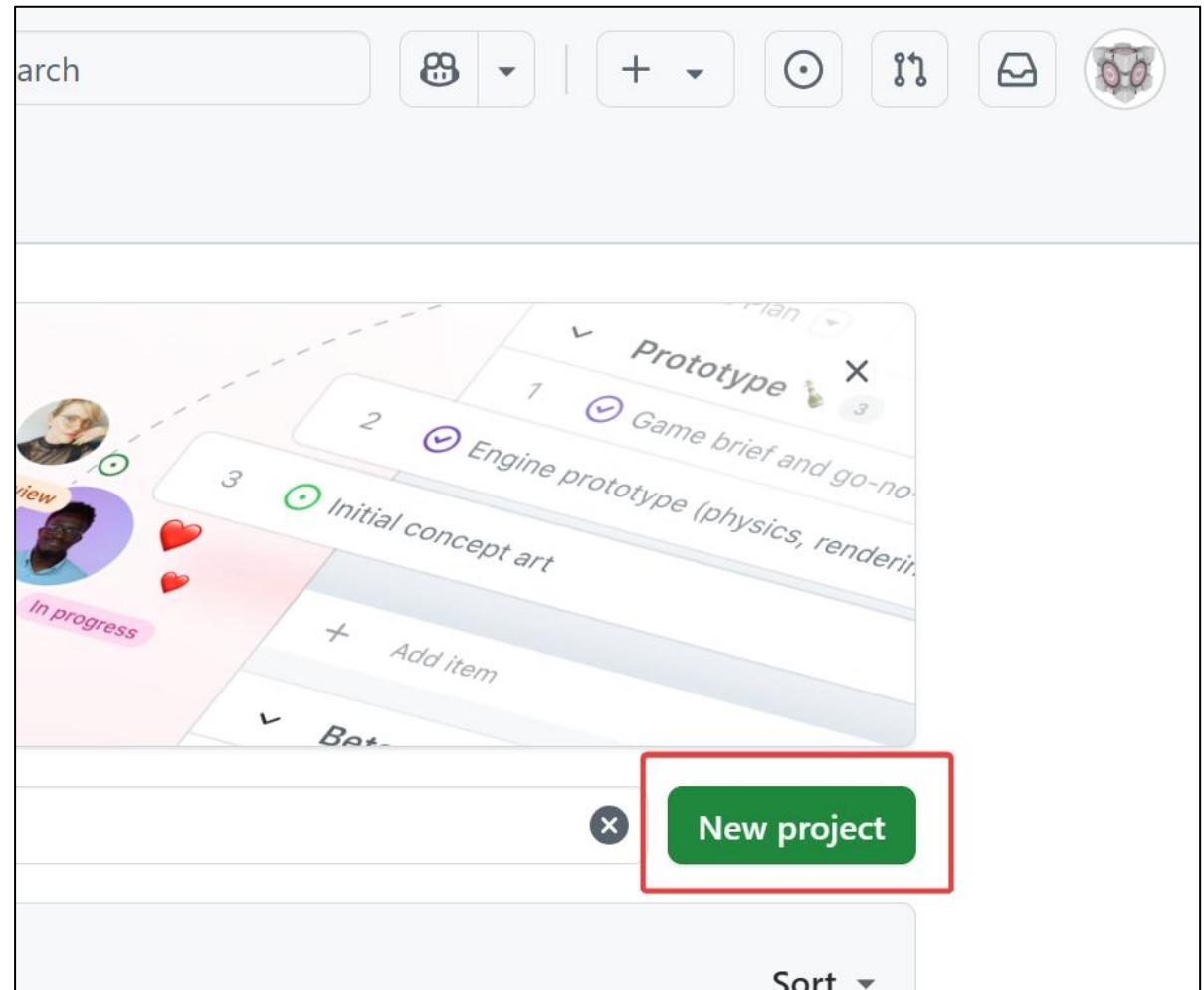
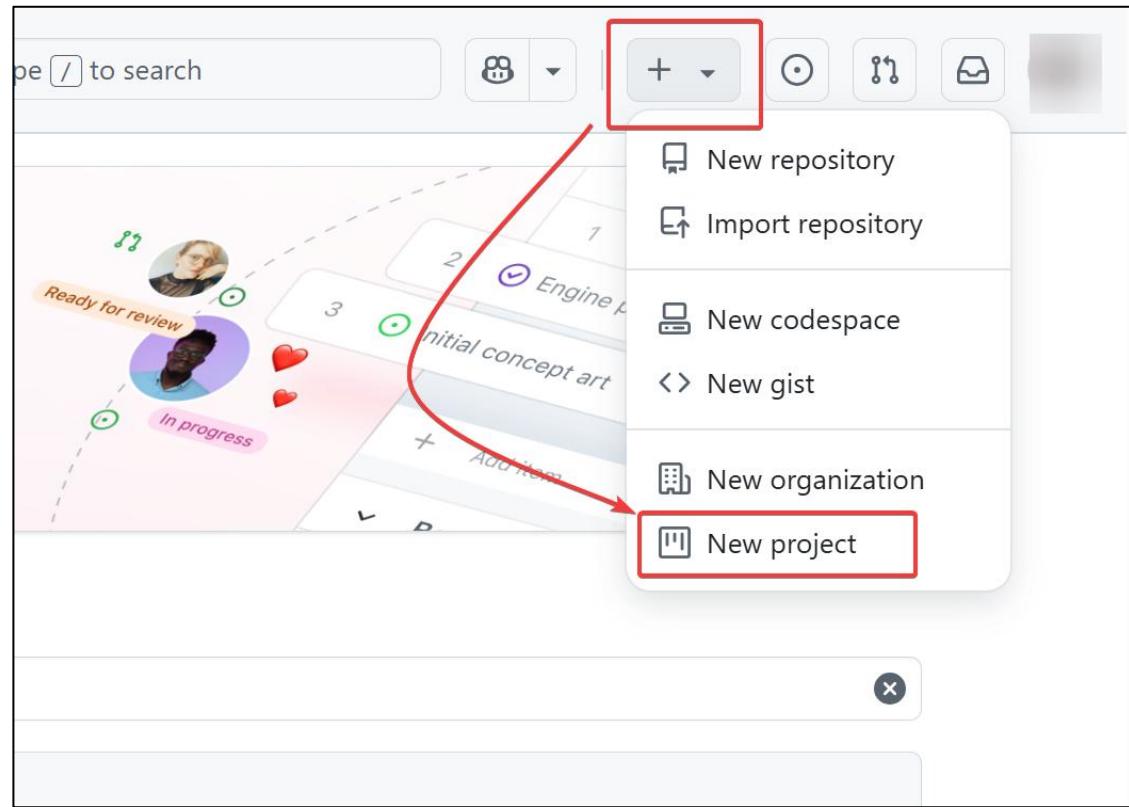
■2つのGitHubリポジトリ「repoA」と「repoB」があり、それぞれにIssueが作られている。



■ GitHub Projects の「プロジェクト」は、複数のリポジトリをまとめて管理するためのしくみ。  
タスクを表す付箋を貼り付ける「ホワイトボード」のようなもの、と考えるとよいかもしれない。



■GitHubの画面上部の「+」をクリックし、New projectをクリック。緑色の「New Project」をクリック。

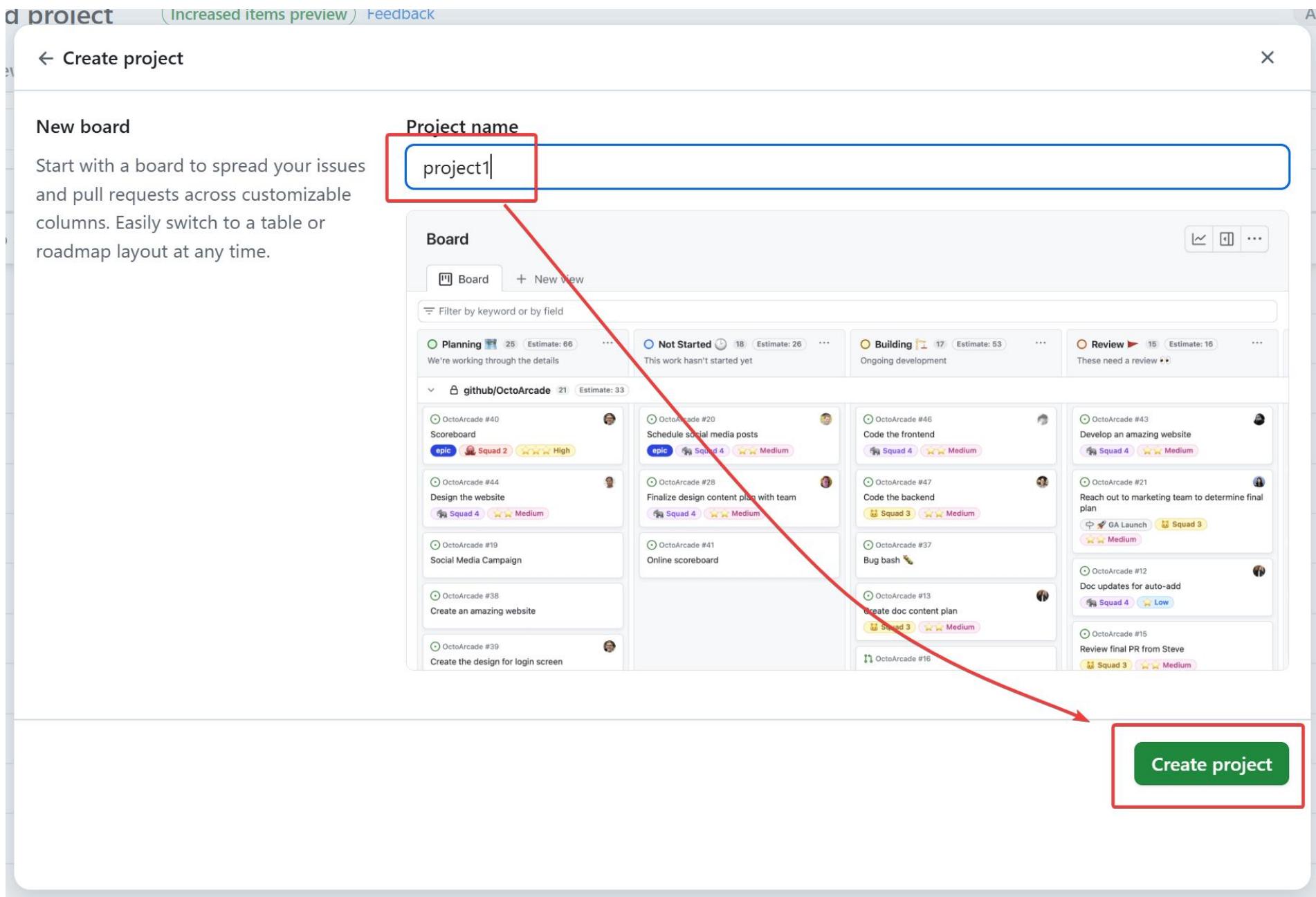


# ■いくつかの「テンプレート」が用意されているが、ここではまずシンプルな「Board」で練習してみよう

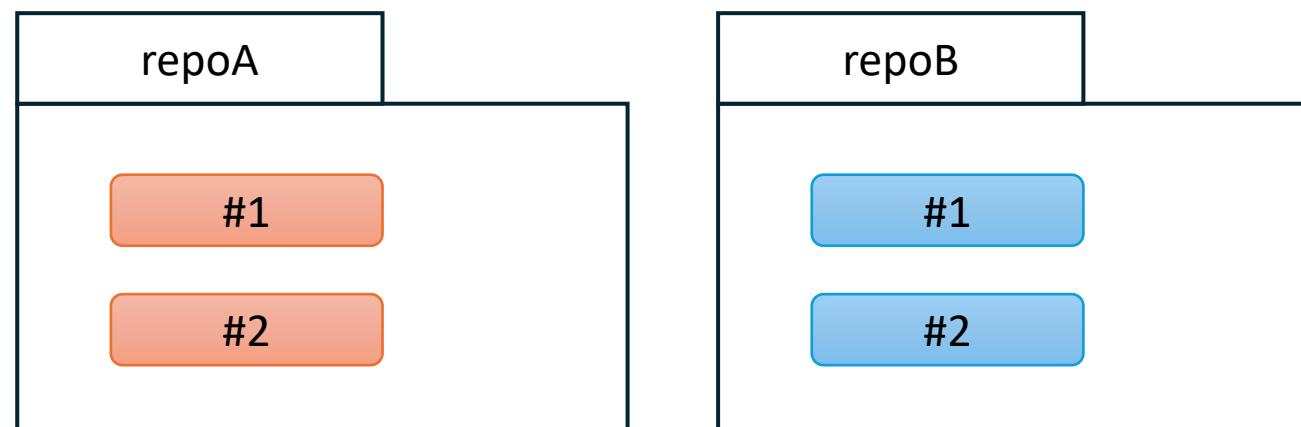
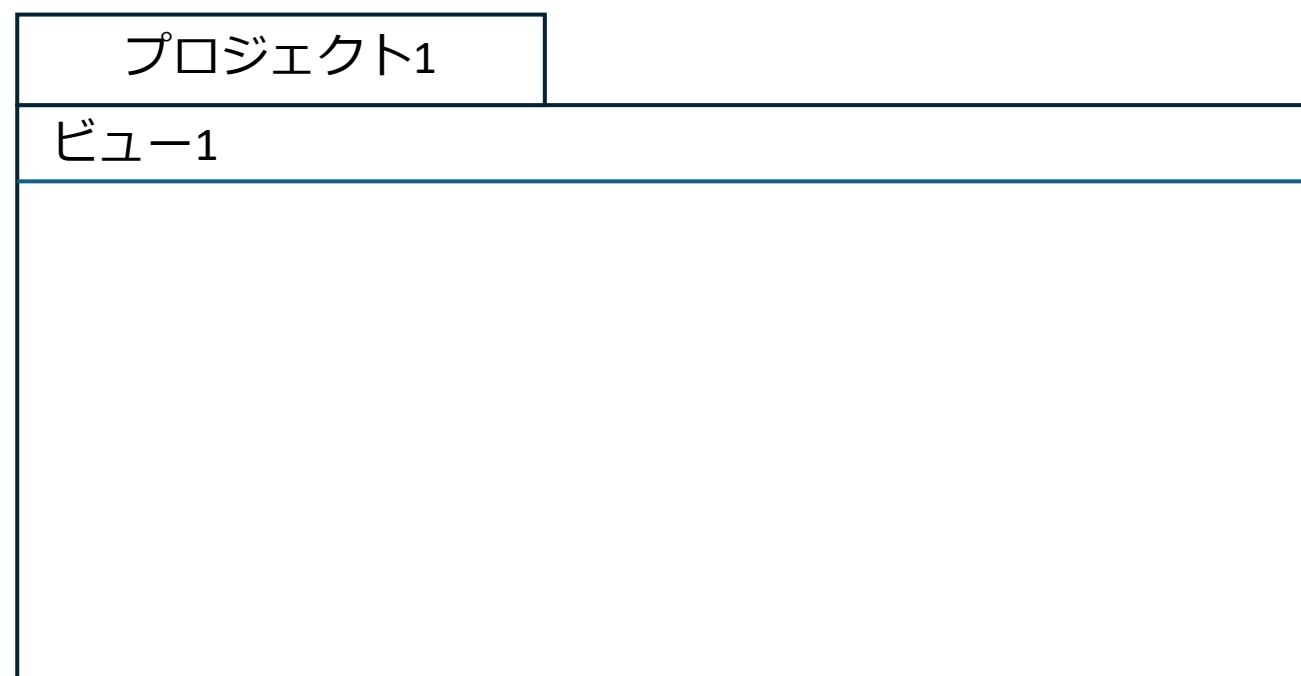
The screenshot shows the GitHub Project Templates interface. On the left, there's a sidebar with options: 'Create project' (disabled), 'Project templates', 'Featured' (selected), 'Start from scratch', 'Table', 'Board' (highlighted with a red box), and 'Roadmap'. Below the sidebar are four cards, each representing a different board template:

- Team planning • GitHub**: Manage your team's work items, plan upcoming cycles, and understand team capacity. It shows a Kanban board with columns: Backlog, In Progress, Done, and In review. Items include 'planning-tracking-item #1234' and 'Integrate with Commerce Service'.
- Feature release • GitHub**: Manage your team's prioritized work items when planning for a feature release. It shows a Kanban board with columns: Prioritized backlog, In progress, Done, and In review. Items include 'Updates to velocity of the ship and alien movements' and 'Create user account and subscription'.
- Kanban • GitHub**: Visualize the status of your project and limit work in progress. It shows a Kanban board with columns: Backlog, Ready, In progress, In review, and Done. Items include 'Explore a bug' and 'planning-tracking-item #1234'.
- Bug tracker • GitHub**: Track and triage your bugs. It shows a Kanban board with columns: Prioritized bugs, To triage, In progress, In review, Done, and My items. Items include 'planning-tracking-item #1234' and 'Investigate error message'.

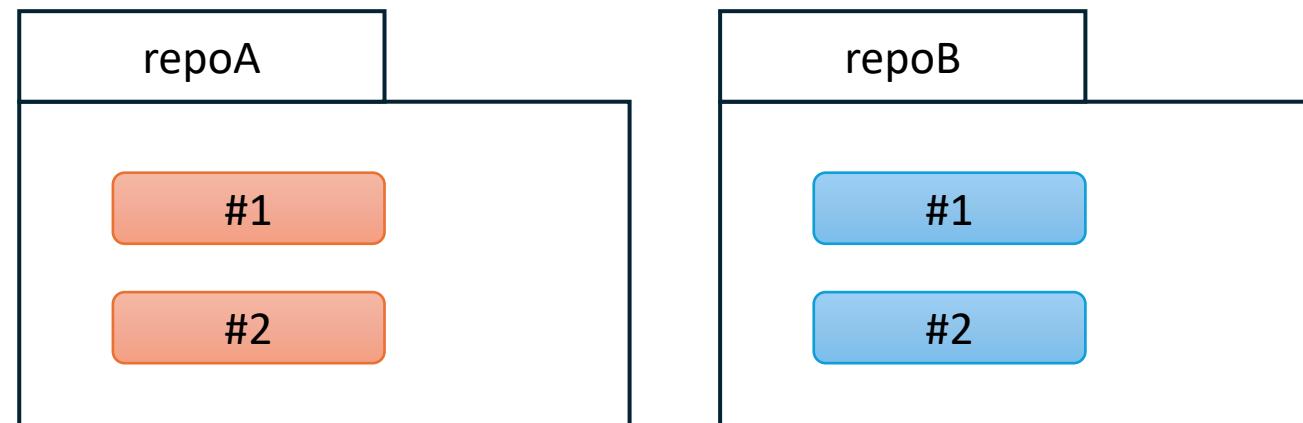
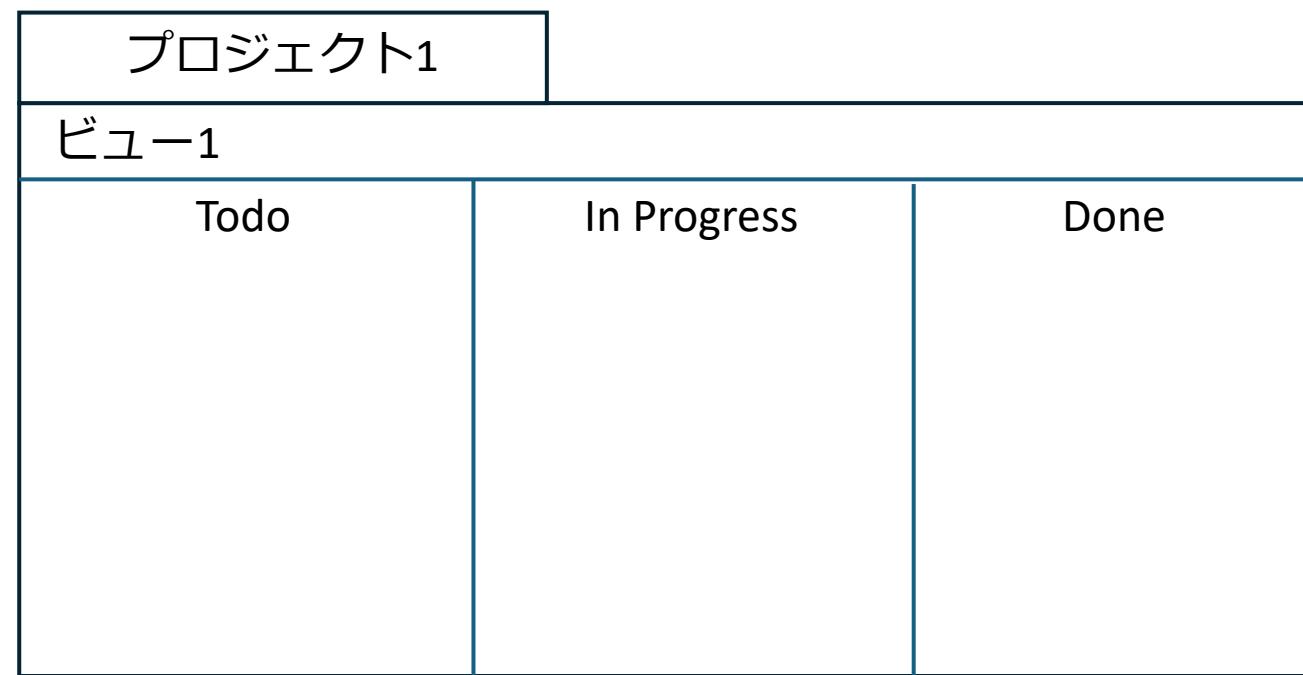
## ■プロジェクトの名前を適当に入力して、「Create project」をクリック



■プロジェクトの中には「ビュー」というものが作られる。これは「表示モード」的なものである。ビューはプロジェクトの中に複数作ることができる。ここではまず「ビュー1」が作られている。

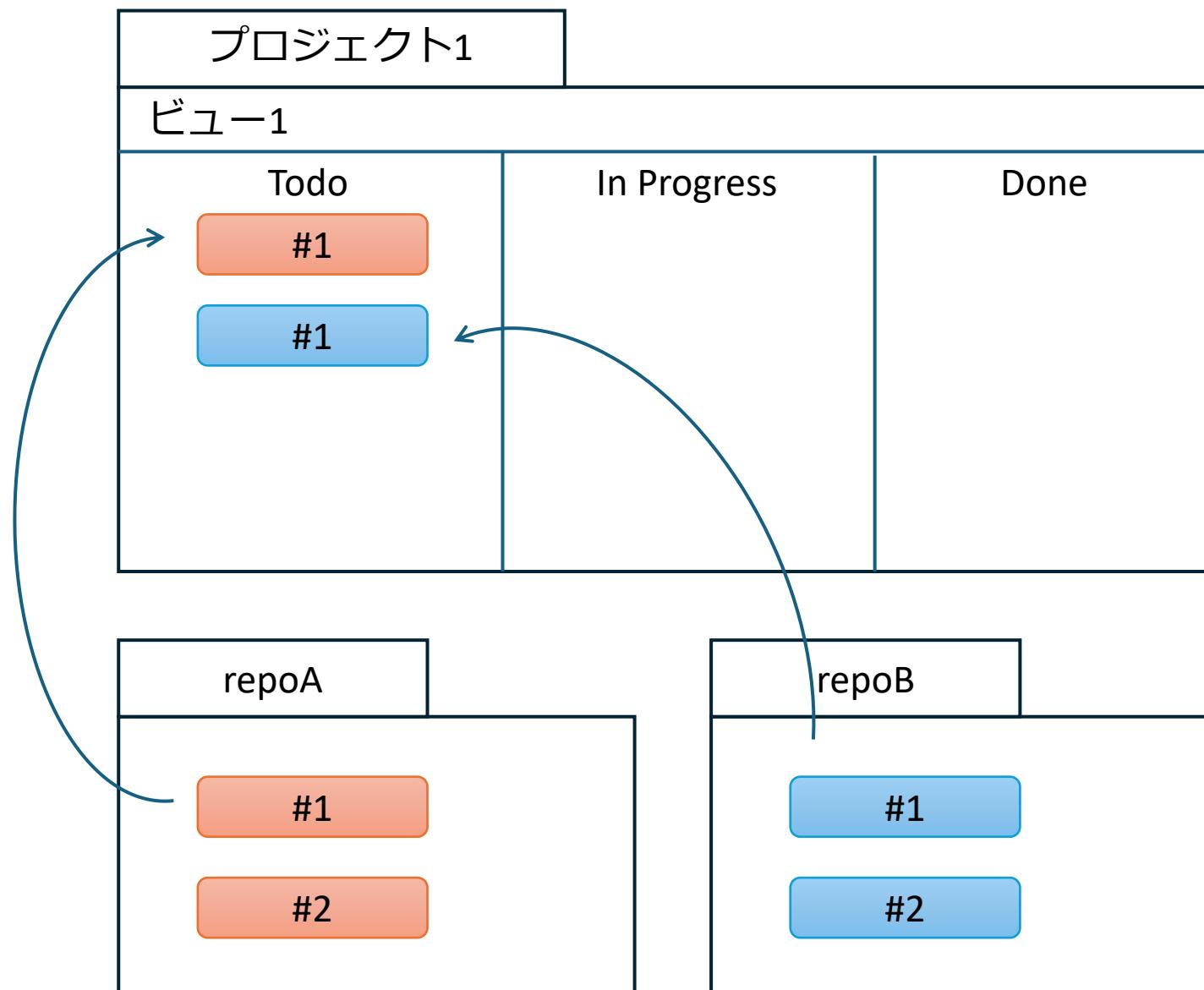


- ここで、「ビュー1」では「**ボード**」（Board）という「**レイアウト**」が選択されている。  
「**ボード**」は Issue の「Todo」「In Progress」「Done」の3つの状態（Status）を視覚化するために使用される。

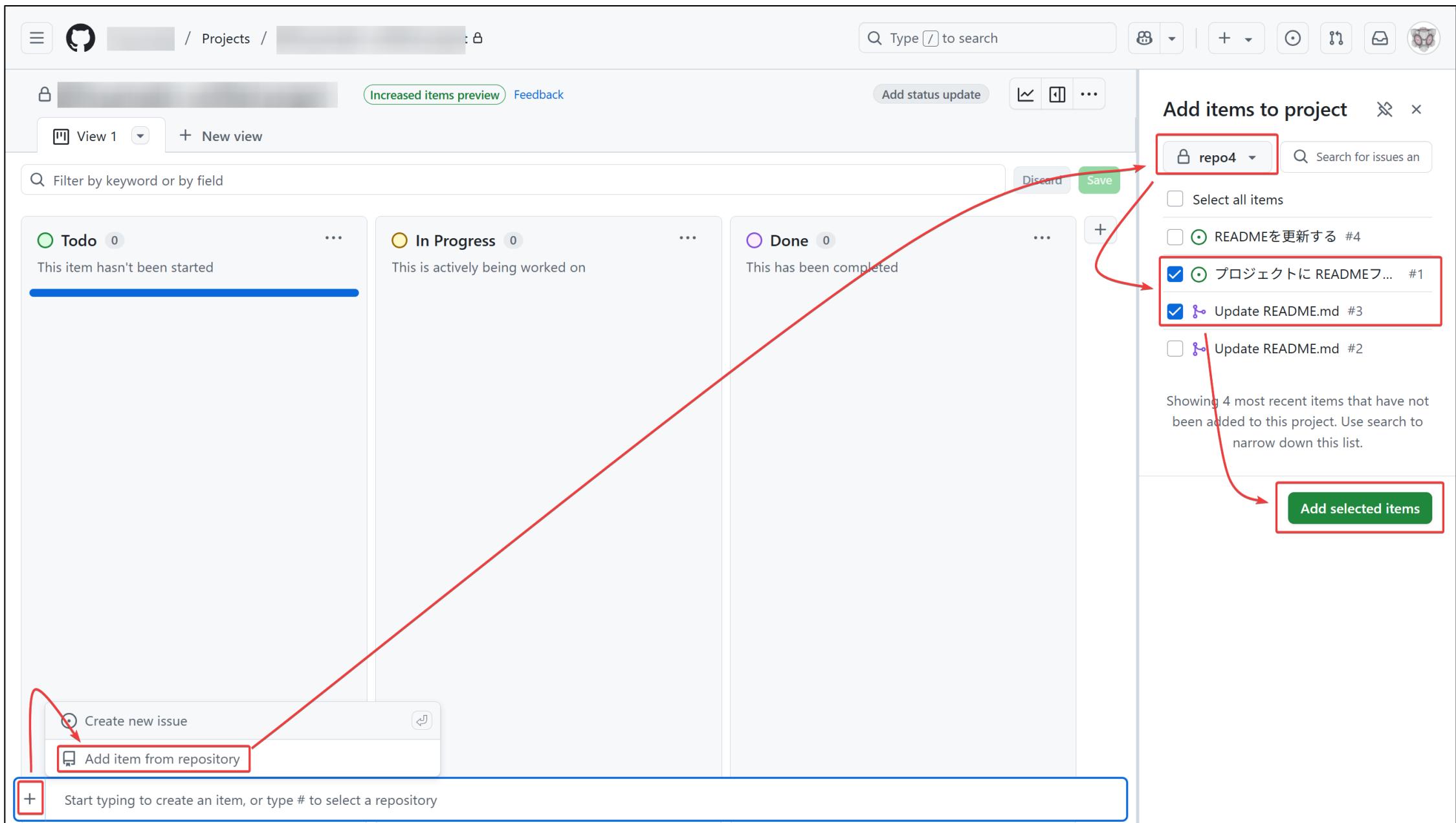


■プロジェクトに、各リポジトリの Issue を追加できる。

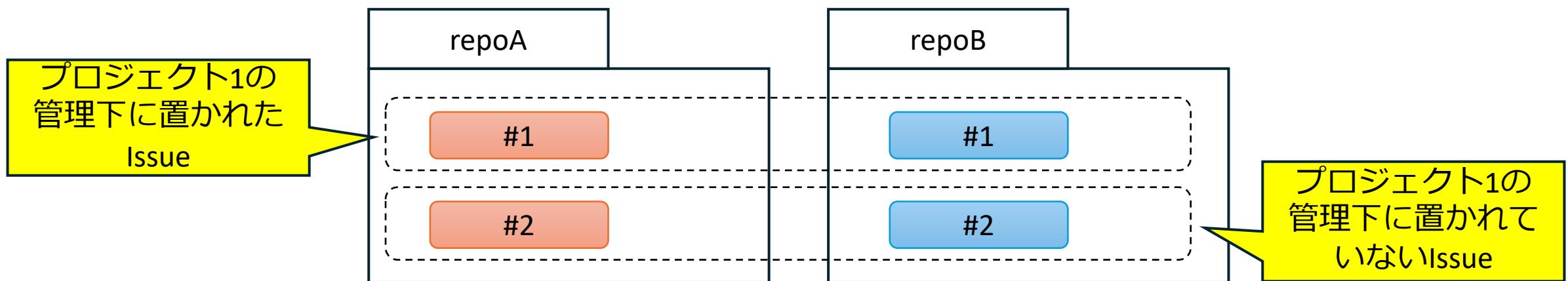
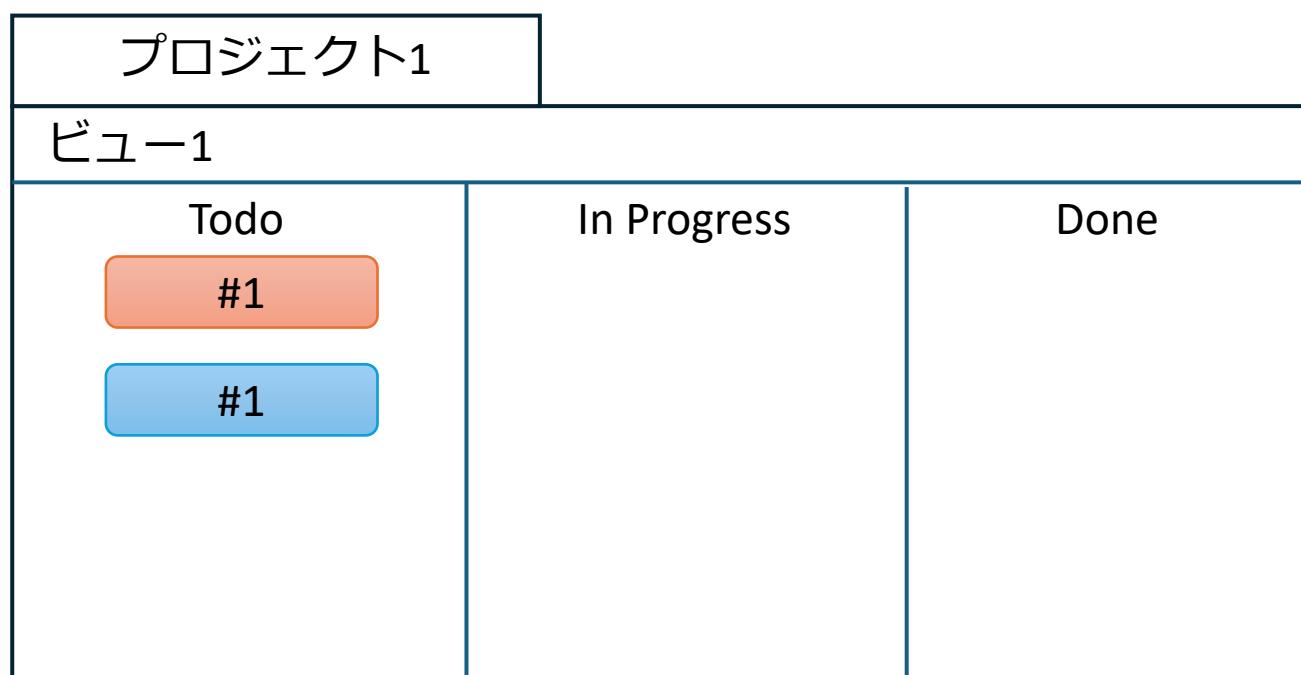
※Issueのコピーができるわけではない。リポジトリに作成された Issue が、  
プロジェクトにも**表示されている**（プロジェクトでも**管理されている**）という状態。



■プロジェクトの下部で「+」をクリックし、「Add item from repository」をクリック。  
画面右側でリポジトリを選び、リポジトリ内のIssueなどを選択し、「Add selected items」をクリック。



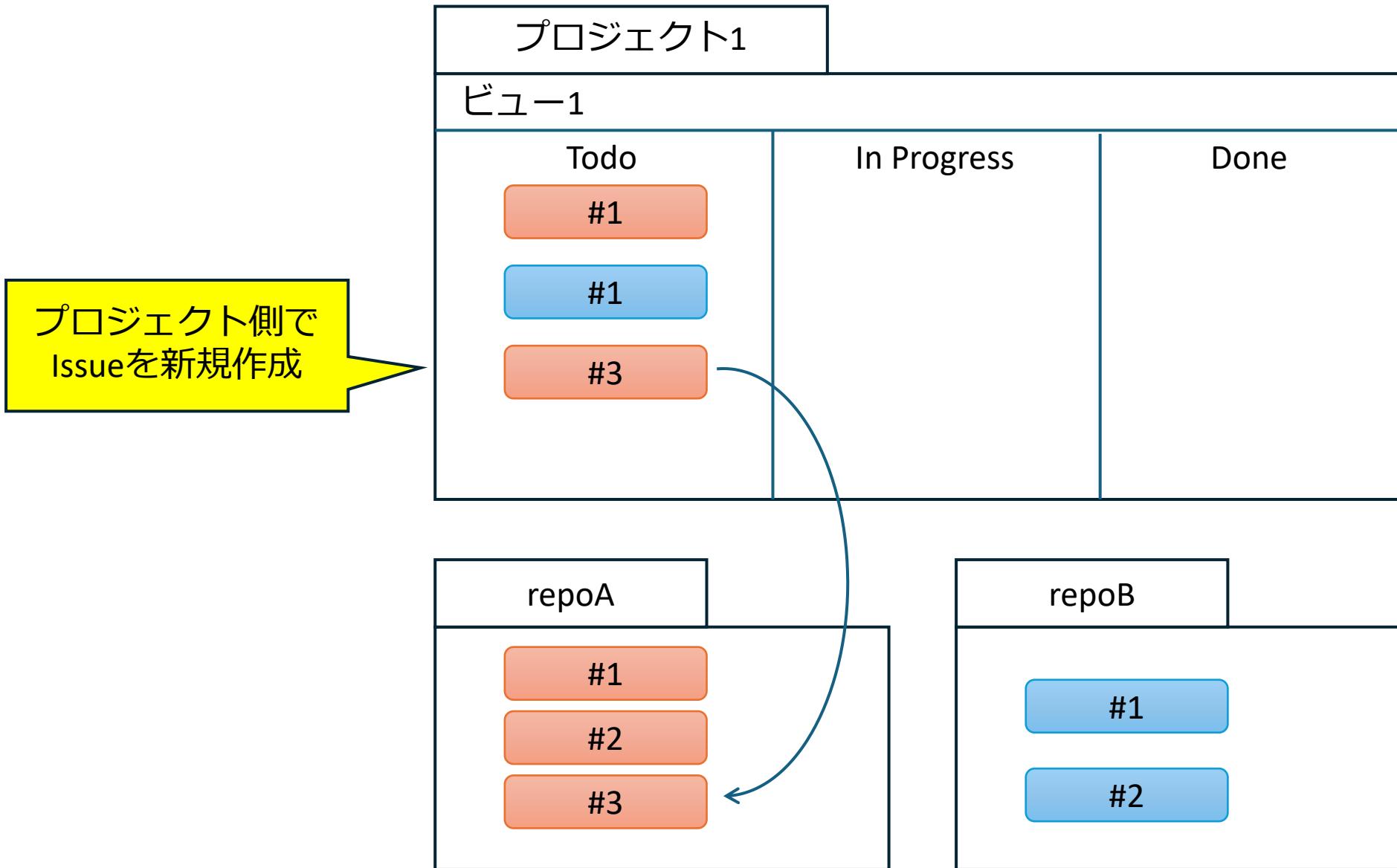
■ただし、リポジトリのすべての Issue がプロジェクトで管理されるわけではないことに注意。



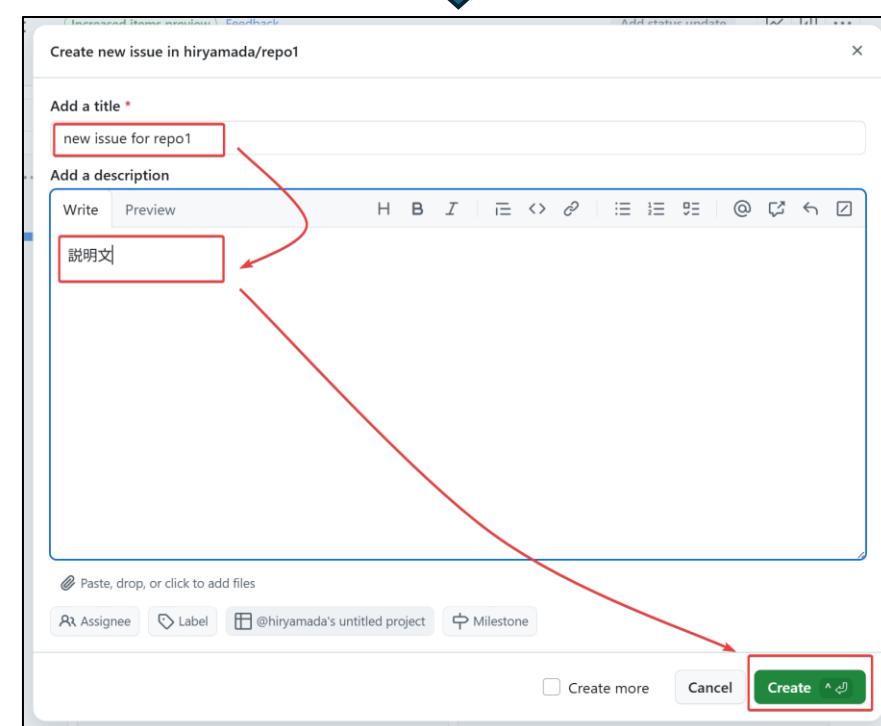
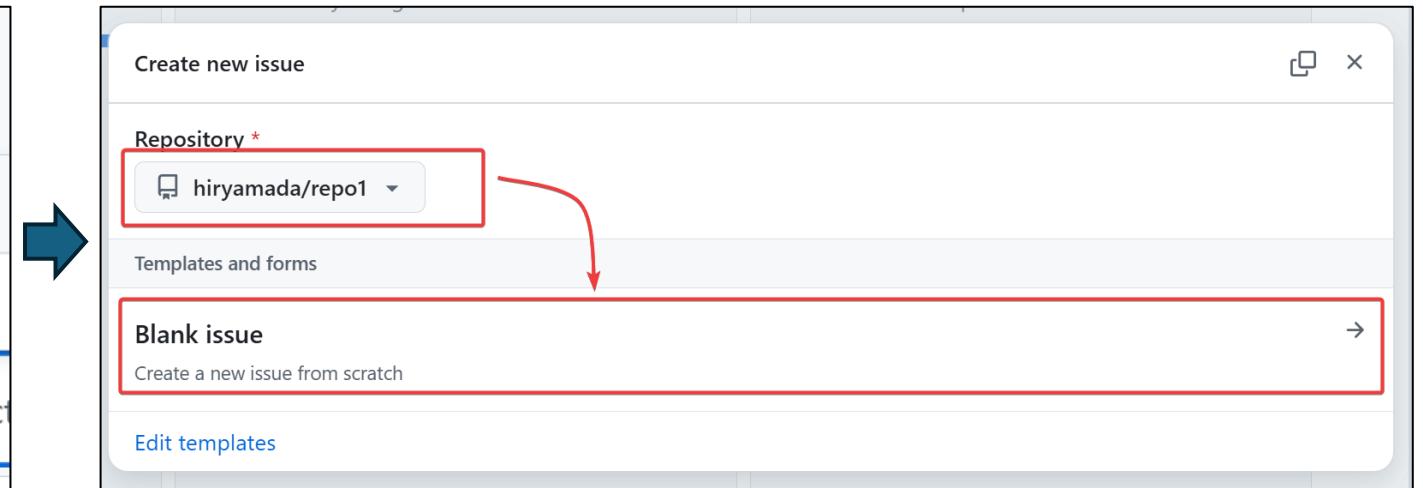
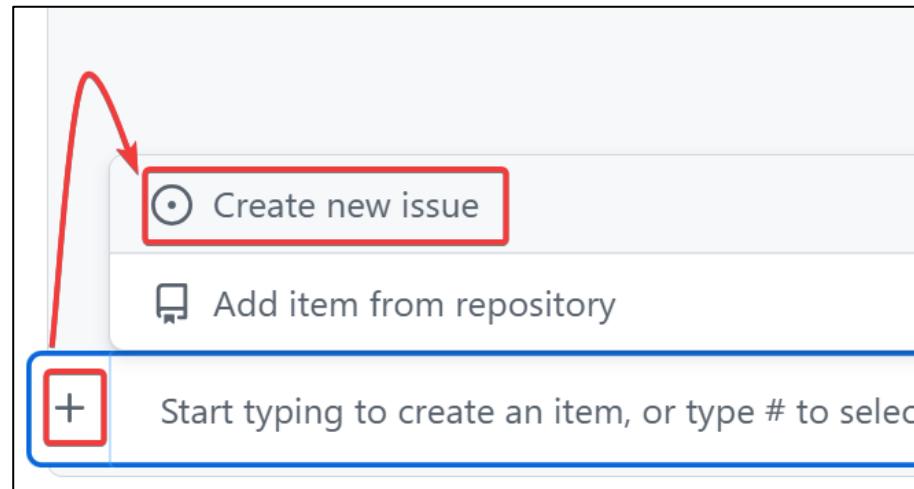
# 参考: Issueを自動的にProjectにも追加する

- Issueを一つ一つ手作業でプロジェクトに追加していくと、手間がかかり、漏れ・抜けが発生する可能性もある。
- リポジトリに追加されたIssueを自動的にプロジェクトにも追加する方法はいくつがある。
  - [アイテムを自動的に追加する - GitHub Docs](#)
  - [GitHub の Issue を自動で Project に追加する方法3選 - NIFTY engineering](#)
  - [開発組織の issue や Pull Request を自動で適切な GitHub Project に割り振っていく](#)

■プロジェクト側で、新しいIssueを作成することもできる。Issueのタイトルを入力し、リポジトリを選択する。



■プロジェクトの画面下部「+」をクリック、「Create new issue」をクリック、リポジトリを選択、「Blank issue」をクリック、Issueのタイトル・説明文を入力して「Create」をクリック。



■ DraftやIssueは、作成されるとビューの一番下に追加される。  
必要に応じて、ビューの中で**上下の並び順**を手動で並び替えることができる。  
(ただしこの順番には、特別な意味はない。単に、ユーザーが好きな順に並べられる、というだけ)

| プロジェクト1                 |             |      |
|-------------------------|-------------|------|
| ビュー1                    |             |      |
| Todo                    | In Progress | Done |
| <a href="#">#3</a>      |             |      |
| <a href="#">(draft)</a> |             |      |
| <a href="#">#1</a>      |             |      |
| <a href="#">#1</a>      |             |      |

WindowsやMacのデスクトップ上でアイコンを好きな位置に配置できるのと似ている

■手動での並び替えは、ビューの「Sort by」オプションが「No sorting (manual)」になっている場合に可能。

The screenshot shows a Jira board titled "project2". The board view is selected. A context menu is open over a "Draft" item, specifically over the "Sort by" option. The "Sort by" menu is displayed, listing various fields like Title, Assignees, Status, etc., with "manual" selected. At the bottom of the "Sort by" menu, there is an option "No sorting" which is also highlighted with a red box.

project2

Board View 2 + New view

Layout

Table Board Roadmap

Configuration

Fields: Title, Assignees, and S... >

Column by: Status >

Group by: none >

Sort by: manual >

Field sum: Count >

Slice by: none >

Generate chart

Rename view

Move view

Duplicate view

Delete view

Export view data

You can use Control + Space to add an item

Sort by

Select up to 2 fields

Title

Assignees

Status

Labels

Linked pull requests

Milestone

Repository

Reviewers

Parent issue

Sub-issues progress

No sorting

■それ以外のオプションに設定した場合は、その指定に沿って自動的に並び替えが行われる。たとえば第1基準「担当者」、第2基準「タイトル」など。

The screenshot shows the Jira interface with a 'Board' view selected. A context menu is open, and a 'Sort by' dialog box is displayed. The 'Sort by' dialog box contains the following content:

**Sort by**  
Select up to 2 fields

Title 2

Assignees 1

Status

Labels

Linked pull requests

Milestone

Repository

Reviewers

Parent issue

Sub-issues progress

No sorting

The 'Title' and 'Assignees' options are highlighted with a red border.

■画面右上「...」をクリック、「Settings」をクリックすると、プロジェクトの設定が変更できる。ここで「カスタムフィールド」を定義できる。たとえば「重要度」というフィールドを作る。フィールドにセットできる値としてたとえば「低」「中」「高」を定義する。値の色も設定できる。

The image consists of three screenshots illustrating the creation of a custom field in a project settings interface.

**Screenshot 1: Project Settings**  
Shows the main project settings menu with various options like Workflows, Archived items, and Settings. The 'Settings' option is highlighted with a red box. A red arrow points from the top right corner of the screen to the 'Settings' button in the sidebar.

**Screenshot 2: Custom Fields**  
Shows the 'Custom fields' section of the settings. A new field is being created with the following details:

- Field name:** 重要度 (Importance)
- Field type:** Single select
- Options:** 低 (Low), 中 (Medium), 高 (High)

A red box highlights the 'Field name' input field. Another red box highlights the 'Single select' dropdown under 'Field type'. A third red box highlights the 'Add option...' button. The 'Save' button at the bottom right is also highlighted with a red box.

**Screenshot 3: Options**  
Shows the list of options for the 'Importance' field. Three colored circles represent the values:

- Low (Blue circle)
- Medium (Yellow circle)
- High (Red circle)

An 'Add option...' button is visible at the bottom.

- （「ボード」ビューでは、カスタムフィールドの値を設定できないため）  
新しいビュー「ビュー2」を作成し、レイアウトは「**テーブル**」とする。  
「**テーブル**」レイアウトのビューでは、Issueなどが表形式で表示される。

| プロジェクト1 |           |        |   |
|---------|-----------|--------|---|
| ビュー2    |           |        |   |
| Title   | Assignees | Status | + |
| #3      |           |        |   |
| (draft) |           |        |   |
| #1      |           |        |   |
| #1      |           |        |   |
|         |           |        |   |

■ テーブルの列名部分の「+」をクリックして、テーブルに列の表示を追加できる。  
ここでは先程追加した「重要度」フィールドの列を表示するようにする。



| プロジェクト1 |           |        |     |  |
|---------|-----------|--------|-----|--|
| ビュー2    |           |        |     |  |
| Title   | Assignees | Status | 重要度 |  |
| #3      |           |        |     |  |
| (draft) |           |        |     |  |
| #1      |           |        |     |  |
| #1      |           |        |     |  |
|         |           |        |     |  |

■テーブルでは、各Issueにすばやく値を設定できる。

| ビュー2    |           |        |     |   |
|---------|-----------|--------|-----|---|
| Title   | Assignees | Status | 重要度 | + |
| #3      |           |        | 高   |   |
| (draft) |           |        | 高   |   |
| #1      |           |        | 中   |   |
| #1      |           |        |     |   |
|         |           |        |     |   |

■各ビューの「Sort by」設定で、作成した「重要度」フィールドの値の順に並び替えて表示するように設定できる。たとえば「重要度の降順」（高、中、低、の順）というようにセットする。

The screenshot shows the Jira interface for managing views. On the left, there's a list of issues: '-issue4 #4', '-issue2 #2', and '-issue3 #3'. The main area is titled 'View 2' and has a 'Layout' tab selected (Table, Board, Roadmap). In the 'Configuration' section, the 'Fields' dropdown shows 'Title, Assignees, Status... >'. The 'Sort by:' dropdown is highlighted with a red box and contains 'Importance >'. Below it, 'Slice by:' is set to 'none >'. The 'Sort by' dropdown also lists other fields: Title, Assignees, Status, Labels, Linked pull requests, Milestone, Repository, Reviewers, Parent issue, Sub-issues progress, and No sorting. The 'Save' button at the bottom right is highlighted with a green box.

■すると、**重要度が高い順**にIssueなどを並べて表示することができる。

project2 Increased items preview Feedback

Board View 2 + New view

Filter by keyword or field

| Title             | Status | 重要度 | ... | + |
|-------------------|--------|-----|-----|---|
| 1 repo2-issue4 #4 | Todo   | 高   | ▼   | ▼ |
| 2 test            | Todo   | 高   | ▼   | ▼ |
| 3 repo2-issue2 #2 | Todo   | 中   | ▼   | ▼ |
| 4 repo2-issue3 #3 | Todo   | 低   | ▼   | ▼ |

+ You can use Control + Space to add an item

# モジュール2 プロジェクト管理

- DevOpsにおけるプロジェクト管理の概要
  - バージョン管理とは？
  - 繙続的インテグレーションと継続的デリバリーとは？
  - タスク管理とは？
  - タスク管理の手法 「かんばん」 vs 「スクラム」
- GitHub を利用したタスク / プロジェクト管理
  - GitHub Issues
  - GitHub Projects
- よくあるご質問
- まとめ

# よくある質問

- ・「Issueを小さいタスクに分割できますか？」
- ・→はい、可能です。2025/4/9、Issueを複数の「Sub-issues」に分割する機能が一般提供開始されています。
- ・<https://github.blog/changelog/2025-04-09-evolving-github-issues-and-projects/>
- ・<https://docs.github.com/ja/issues/tracking-your-work-with-issues/using-issues/adding-sub-issues>

# よくある質問

- ・ 「Issueをグループ化できますか？」
- ・ →いくつかの方法があります。
- ・ (1) 「**ラベル**」を設定する: 「bug」「enhancement」（機能追加リクエスト）などのラベルを設定してIssueを分類します。
- ・ (2) 「**マイルストーン**」を設定する: 任意の名称と期限（年月日）を設定した「マイルストーン」をリポジトリに定義し、Issueでマイルストーンを選択することで、Issueを分類します。
- ・ (3) 「**タスクリスト**」を使う: コメントの中に、他のIssueへのリンクと、それらのIssueの完了状態を表すチェックボックスを設置できます。このリストで複数のIssueの状態を1箇所に集約できます。
- ・ (4) 「**カスタムフィールド**」を使う: 重要度「高」「中」「低」などのフィールドを定義し、その値で分類・グループ化できます。

[ラベルを管理する - GitHub Docs](#)

[マイルストーンについて - GitHub Docs](#)

[タスクリストについて - GitHub Docs](#)

[単一選択フィールドについて - GitHub Docs](#)

# よくある質問

- ・「GitHub Projects でガントチャートは作れますか？」
  - ・「WBSは作れますか？」
  - ・「土日・祝日や従業員の年休等を考慮した予定は組めますか？」
  - ・「メンバーの時給を考慮した予算管理はできますか？」
  - ・「○○というプロジェクト管理ソフトが持つ××機能は GitHub Project にはありますか？」
- 
- ・→**いいえ**。残念ながら GitHub Projects は、他のプロジェクト管理ソフトが備えているようなプロジェクト管理機能、スケジューリング機能、人員の管理機能をすべて備えているわけではありません。
  - ・GitHub Projects はあくまで 「**複数のリポジトリのIssue等をまとめてわかりやすく管理するビュー**」 を提供するもの、と考えていただくとよいと思います。

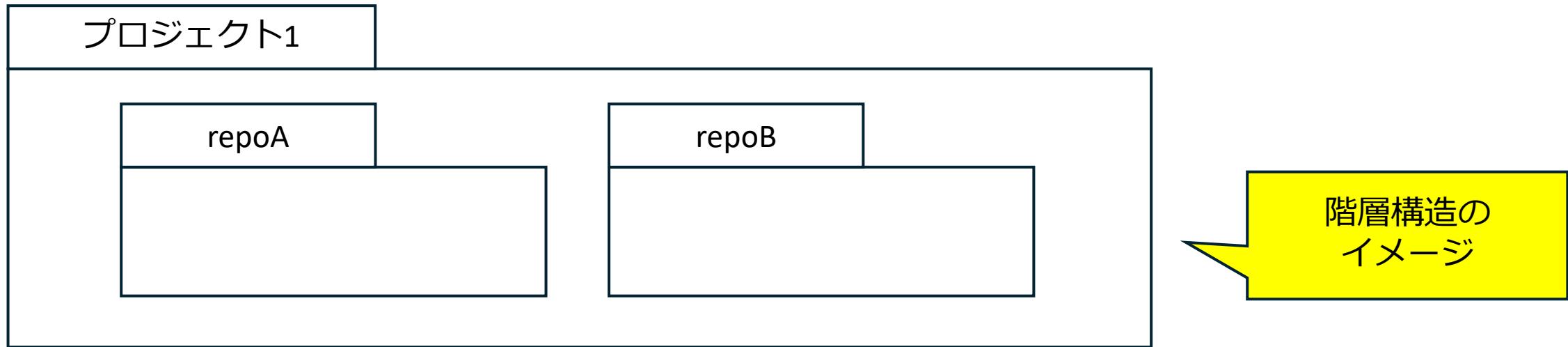
ユーザーレベルで土日祝を頑張って設定することは可能ですが・・・

# よくある質問

- ・「プロジェクトを作った際、誰がそのプロジェクトにアクセスできますか？」
- ・「プロジェクトに対して誰が何を操作できるかを設定できますか？」
- ・→プロジェクトは（リポジトリと同様に） 「**可視性**」 をパブリックまたはプライベートに設定できます。
  - ・パブリックのプロジェクトは、インターネット上のすべてのユーザーに表示されます。
  - ・プライベートのプロジェクトは、プロジェクトに追加された、読み取りアクセスの許可を持つユーザーが表示できます。
- ・→プロジェクトにユーザーを「**コラボレータ**」として追加できます。
  - ・コラボレータには「読み取り」「書き込み」「管理者」のロールのいずれかを設定できます。「書き込み」はプロジェクトの編集が可能、「管理者」は編集とコラボレータ招待が可能です。
- ・→それ以上の細かいアクセス権設定はできません。

# よくある質問

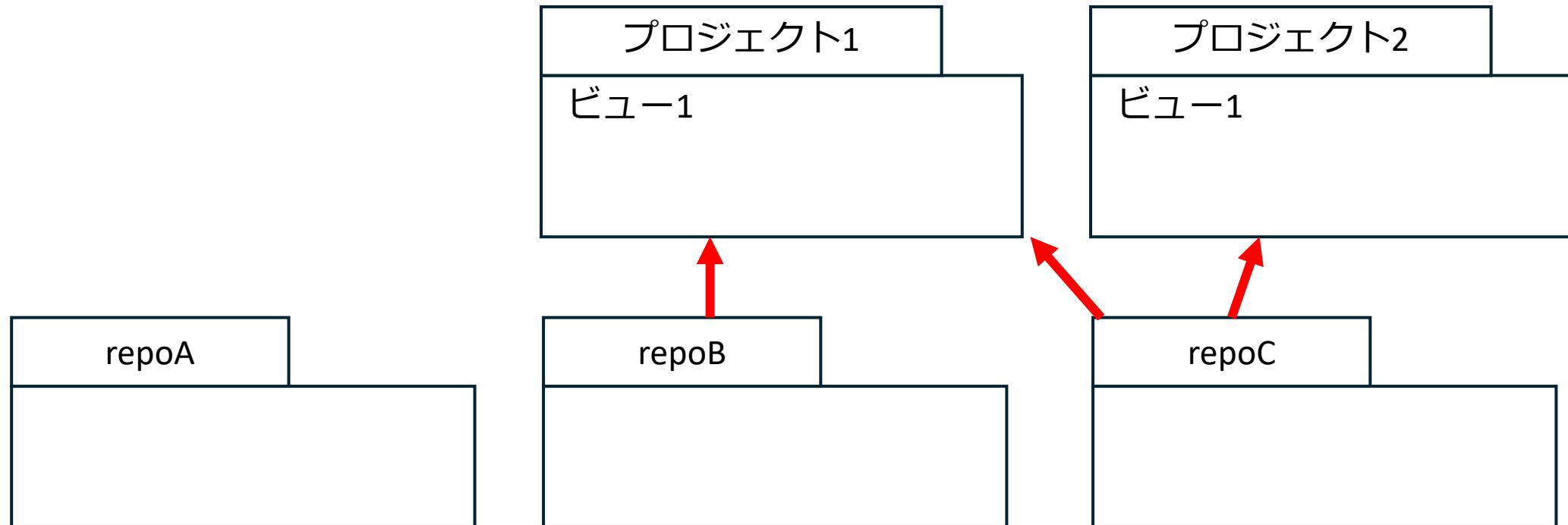
- ・「プロジェクト」の中に「リポジトリ」を入れる（階層構造にする）ことはできますか？
- ・「プロジェクト」の中に「リポジトリ」を作つて、Issueをすべてプロジェクト側で管理するようなことはできますか？



- ・→いいえ。

# 参考: プロジェクトのリンク

- ・**リポジトリ側**で、0個以上のプロジェクトに対するリンクを設定できます。
- ・ただし、リンクしたからなにか便利な使いができるというわけでもなく、単にリポジトリの「Projects」タブに、リンクしたプロジェクトが表示される（クリックするとそのプロジェクトに移動できる）だけです。
- ・特にリンクをしなくても、プロジェクトの利用は可能です。
- ・リンクできるのはリポジトリの所有者のプロジェクト（またはリポジトリの所有者の組織のプロジェクト）のみです。



# よくある質問

- ・「GitHubが用意しているプロジェクトのテンプレートは自分でも作れますか？」
- ・「プロジェクトのカスタムフィールドの設定、ビューの設定などを、プロジェクトを作るたびにやり直すのは面倒です」
- ・→テンプレート作成の機能はないですが、すべてのプロジェクトは別のプロジェクトのテンプレートとして使えます。作成済みの**プロジェクトのコピーを作成**できます。コピーによって作成された新しいプロジェクトは、コピー元と同じ設定を持ちますが、コピー元が持っていたIssueなどの情報は含まれません。
- ・GitHub のコマンド（GitHub CLI）やAPIを使用してプロジェクトの設定を自動化する手もあります。

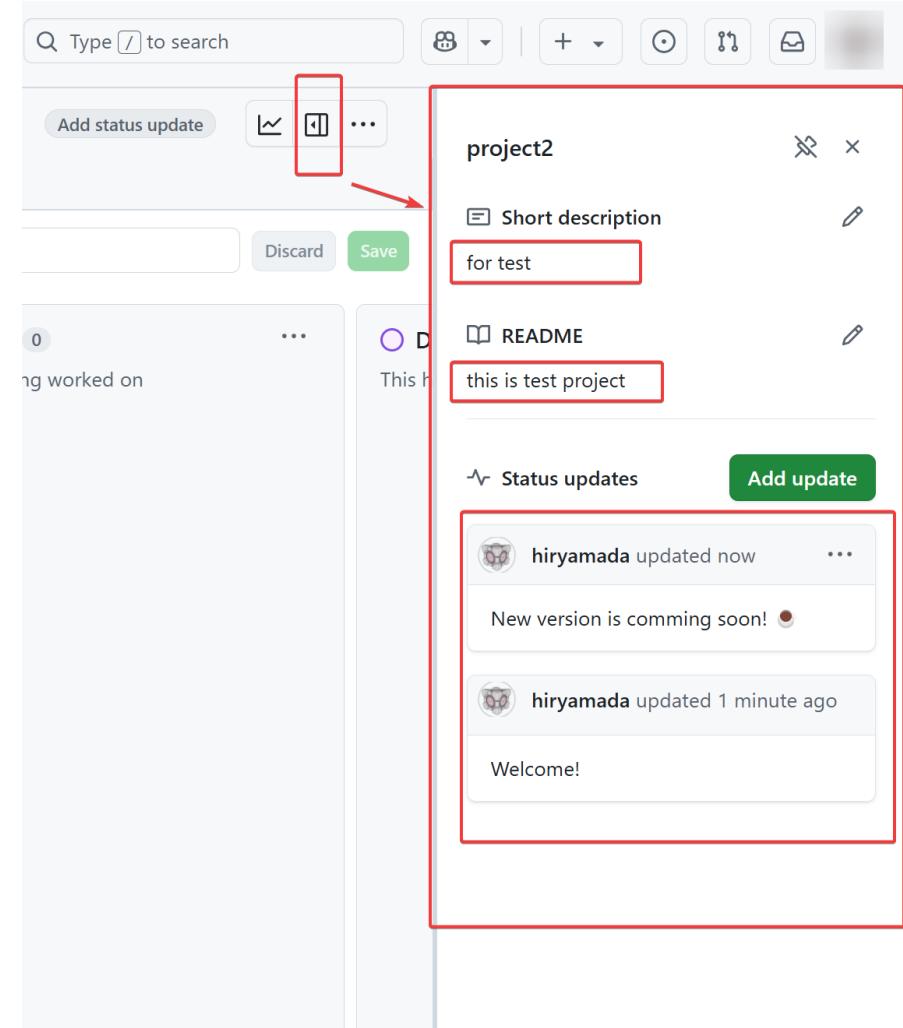
[既存の project のコピー - GitHub Docs](#)

[GitHub CLI | Take GitHub to the command line](#)

[API を使用して Projects を管理する - GitHub Docs](#)

# よくある質問

- ・「プロジェクトに任意のドキュメント（設計書など）を追加できますか？」
- ・→**いいえ。**プロジェクトには、プロジェクトの説明・短い説明・複数の「ステータス」を設定できますが、任意の複数のドキュメントを追加・管理するような機能はありません。
- ・→**リポジトリ側**で、「docs」といったフォルダ以下にドキュメントのファイルを配置するか、またはリポジトリの「Wiki」を使います。



# モジュール2 プロジェクト管理

- DevOpsにおけるプロジェクト管理の概要
  - バージョン管理とは？
  - 繙続的インテグレーションと継続的デリバリーとは？
  - タスク管理とは？
  - タスク管理の手法 「かんばん」 vs 「スクラム」
- GitHub を利用したタスク / プロジェクト管理
  - GitHub Issues
  - GitHub Projects
- よくあるご質問
- まとめ

# モジュール2 まとめ

|                 |   |
|-----------------|---|
| バージョン管理         | ソースコードや文書の変更履歴の管理。Gitが主流となっている。   |
| 継続的インテグレーション    | コードの変更をマージし、ビルド・テストを行う  |
| 継続的デリバリー        | ビルドされた成果物をステージング環境や本番環境に配置する  |
| タスク管理           | バグの報告、修正依頼、新機能追加の提案などのタスク（1件の作業）の管理。プロジェクト管理ツールを使用してタスクを管理する。   |
| GitHub Issues   | GitHub リポジトリの中で、アイデア、フィードバック、タスク、作業、バグレポートなどを記録・追跡できる仕組み。担当者を割り当てたり、コメントを追記したりできる。  |
| GitHub Projects | 複数のGitHubリポジトリにまたがって、それらのリポジトリのIssueをわかりやすく表示・管理するツール。「ボード」ビューではTodo / In Progress / Doneという3つの列でIssueをトラッキングできる。「テーブル」ビューではIssueのタイトル、担当者、状態、重大度など、ユーザーが指定した属性（列）を表示したり、並び替えたりできる。 |

# 講義



AZ-2008  
DevOpsの基礎:  
中心となる原則と実践

DevOps foundations: The core principles and practices

- ・モジュール1 DevOps の概要
- ・モジュール2 DevOps を使用した計画 (Plan with DevOps)
- ・モジュール3 DevOps を使用した開発 (Develop with DevOps)
- ・モジュール4 DevOps を使用した提供 (Deliver with DevOps)
- ・モジュール5 DevOps を使用した操作 (Operate with DevOps)
- ・まとめ

# モジュール3



Git バージョン管理

## DevOps を使用した開発 (Develop with DevOps)

Git と GitHub を使ってバージョン管理を適用することで、ソフトウェア開発プロジェクトの更新を効率化します。継続的インテグレーション、シフトレフトテスト、シフトレフトセキュリティを実装することで、ソフトウェア ライフサイクルを改善します。

# モジュール3 バージョン管理、開発

- バージョン管理 - Git

- GitHub の基本
  - リポジトリの作成、クローン、コミット、プッシュ
- GitHubでのコラボレーション（共同開発）
  - GitHub flow
  - フォーク、プルリクエスト、マージ
- 繙続的インテグレーション
- よくあるご質問
- まとめ

# バージョン管理 - Git

- ・現代の開発におけるバージョン管理では主に Git が使用される
- ・GitHub でも Git が使用される
- ・Gitは、Windows、Mac、Linuxで使用できる
  - ・Windowsの場合は「Git for Windows」をインストールする
- ・Visual Studio、Visual Studio Code、Eclipse、PyCharmなどの主な開発ツールやIDEには、Gitとの連携機能がある
  - ・Gitコマンドを覚えなくてもある程度Gitの操作ができる

# モジュール3 バージョン管理、開発

- ・バージョン管理 - Git

- ・GitHub の基本

- ・リポジトリの作成、クローン、コミット、プッシュ

- ・GitHubでのコラボレーション（共同開発）

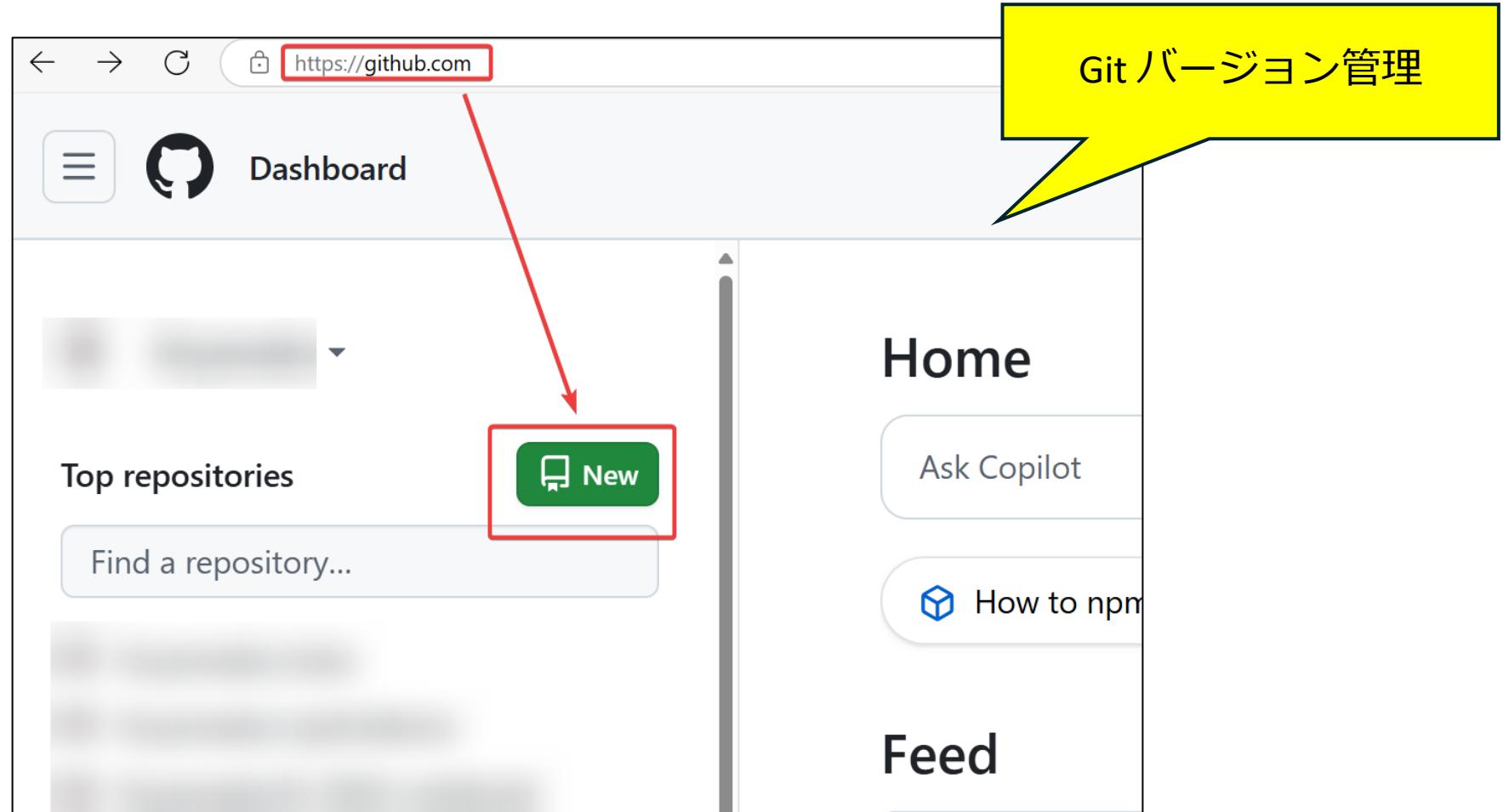
- ・GitHub flow
  - ・フォーク、プルリクエスト、マージ

- ・継続的インテグレーション

- ・よくあるご質問

- ・まとめ

# GitHubリポジトリの作成

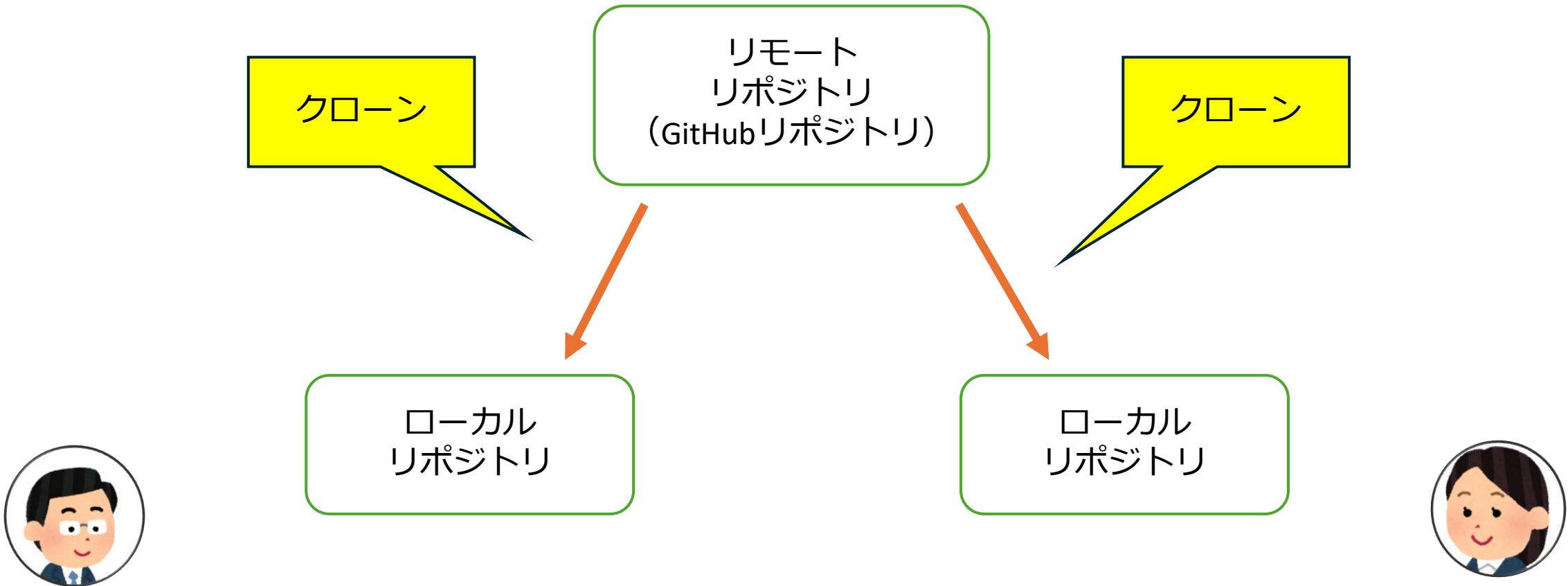


# モジュール3 バージョン管理、開発

- ・バージョン管理 - Git
- ・GitHub の基本
  - ・リポジトリの作成、**クローン**、コミット、プッシュ
- ・GitHubでのコラボレーション（共同開発）
  - ・GitHub flow
  - ・フォーク、プルリクエスト、マージ
- ・継続的インテグレーション
- ・よくあるご質問
- ・まとめ

# クローン (git clone)

- ・リモートリポジトリのコピーを作成する
- ・作業用のローカルリポジトリができる

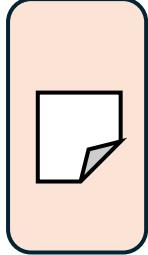


- ・変更作業中

リモート  
リポジトリ  
(GitHubリポジトリ)

ローカル  
リポジトリ

ローカル  
リポジトリ

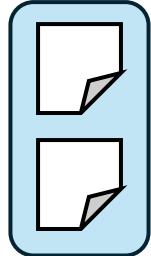
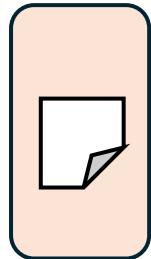


- ・変更作業中

リモート  
リポジトリ  
(GitHubリポジトリ)

ローカル  
リポジトリ

ローカル  
リポジトリ



# モジュール3 バージョン管理、開発

- ・バージョン管理 - Git
- ・GitHub の基本
  - ・リポジトリの作成、クローン、コミット、プッシュ
- ・GitHubでのコラボレーション（共同開発）
  - ・GitHub flow
  - ・フォーク、プルリクエスト、マージ
- ・継続的インテグレーション
- ・よくあるご質問
- ・まとめ

# コミット (git commit)

- ファイルの追加・変更・削除をローカルリポジトリで登録・確定する処理

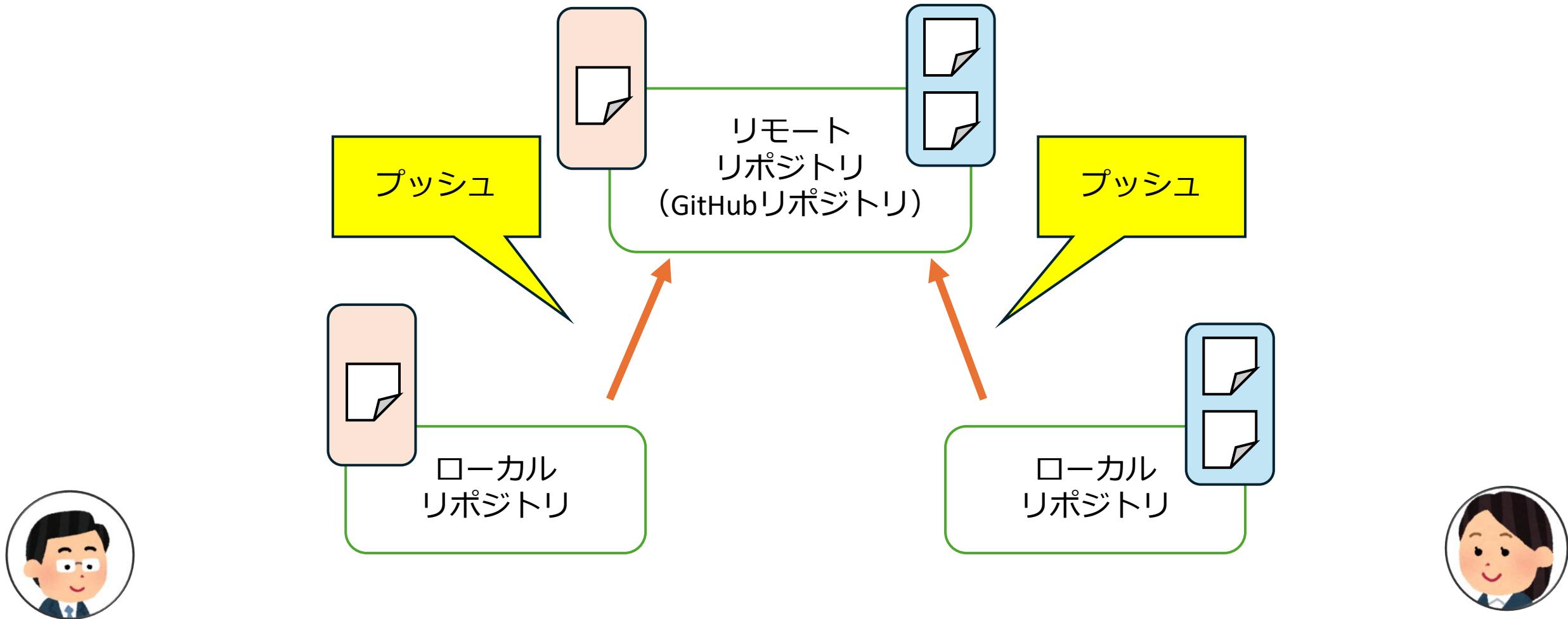


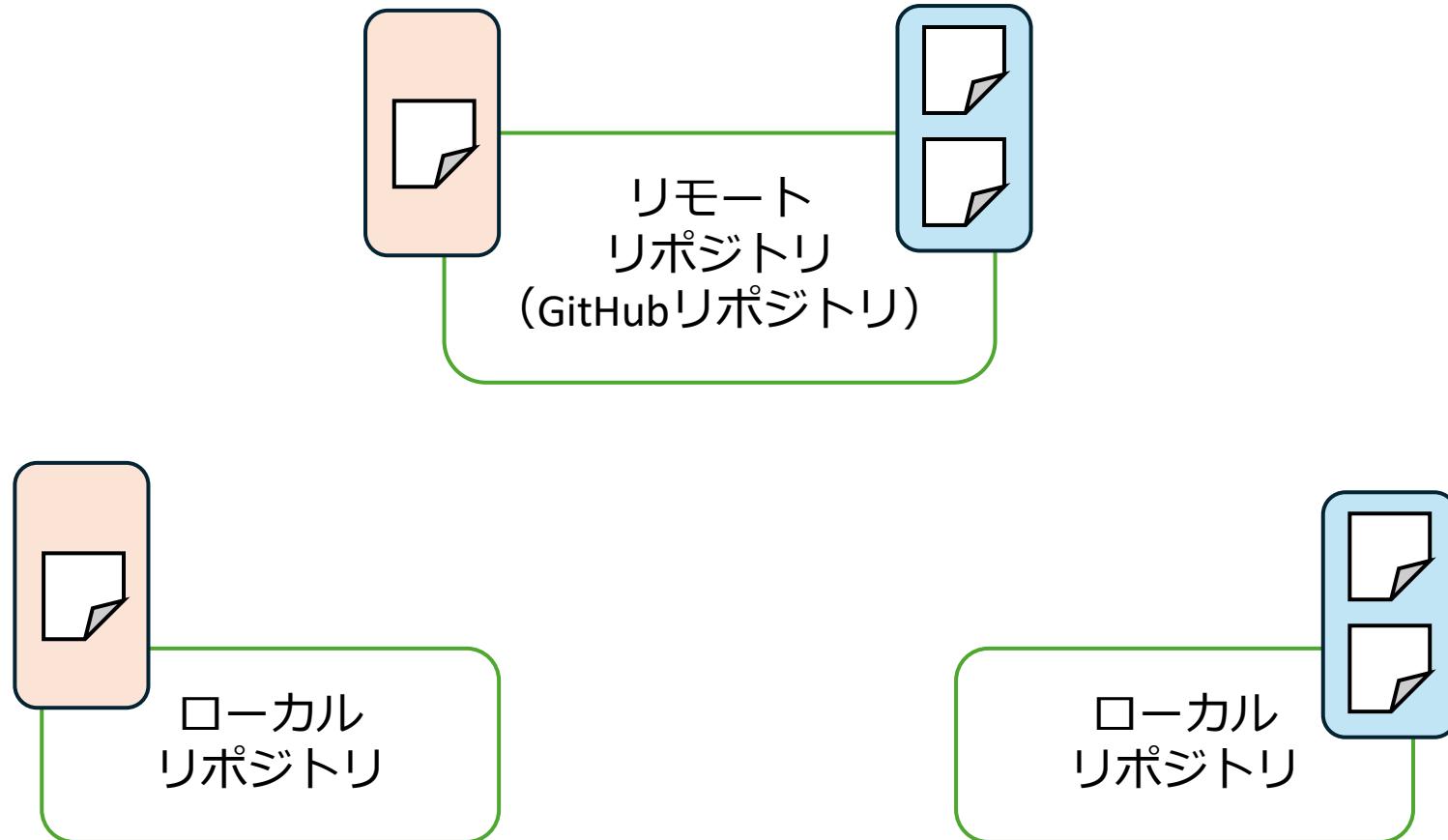
# モジュール3 バージョン管理、開発

- ・バージョン管理 - Git
- ・GitHub の基本
  - ・リポジトリの作成、クローン、コミット、**プッシュ**
- ・GitHubでのコラボレーション（共同開発）
  - ・GitHub flow
  - ・フォーク、プルリクエスト、マージ
- ・継続的インテグレーション
- ・よくあるご質問
- ・まとめ

# プッシュ (git push)

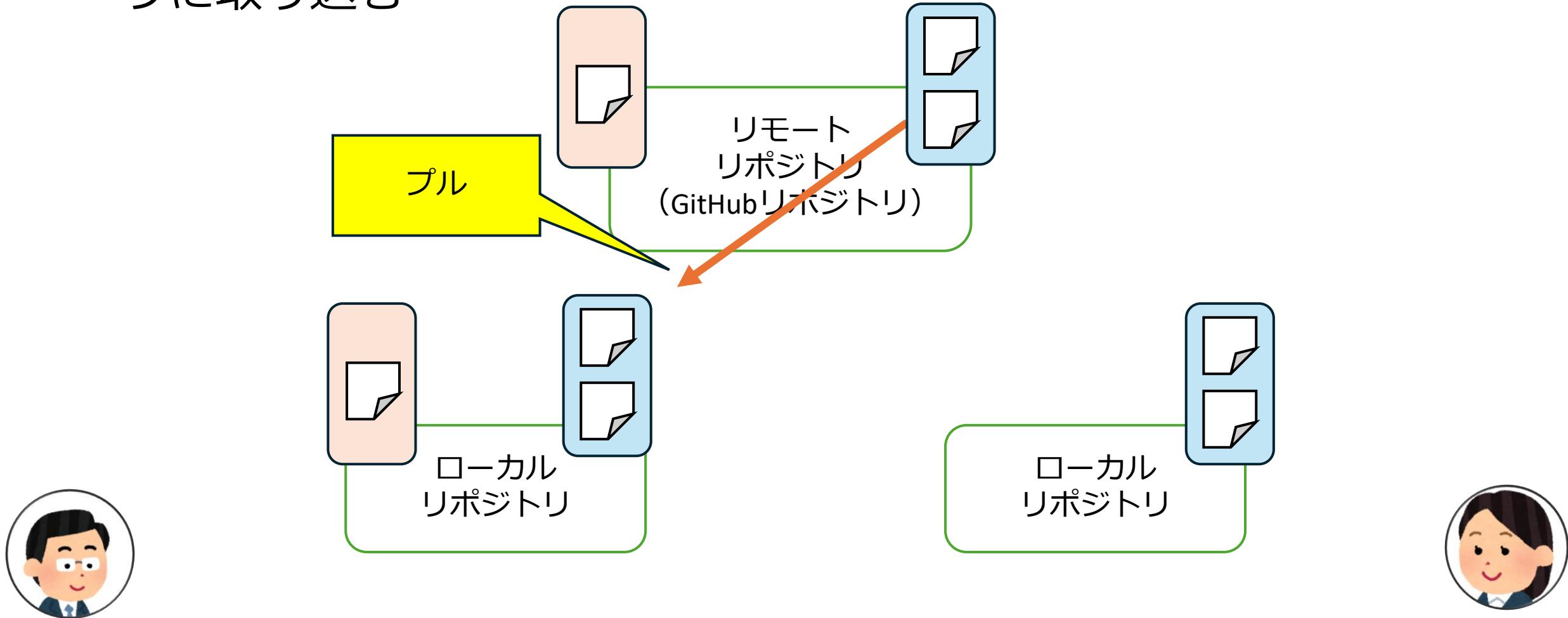
- ・ローカルのコミット（群）を、リモートへ送信する

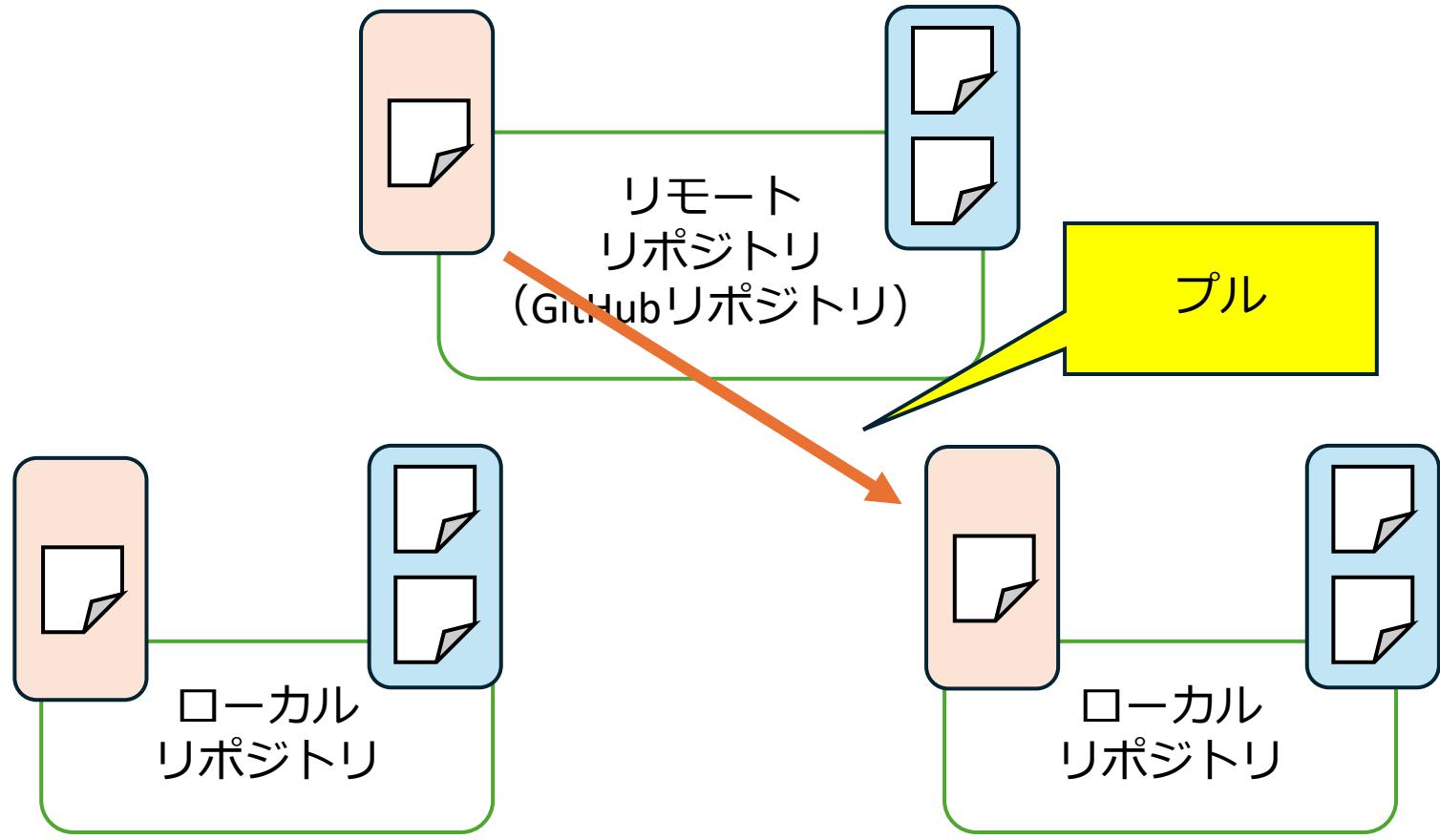


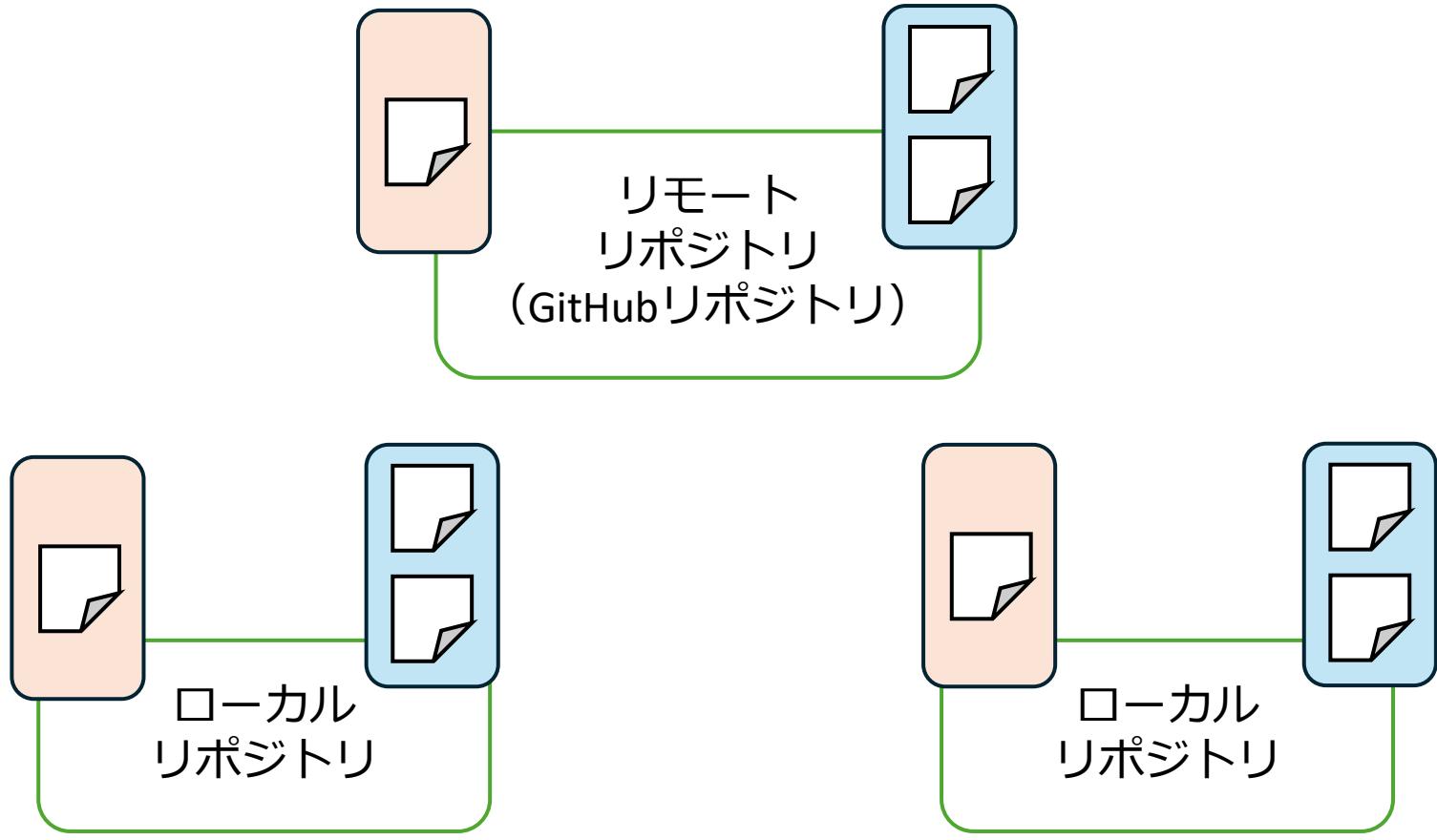


# プル (git pull)

- リモートのコミット（群）をまとめて自分のローカルリポジトリに取り込む







# モジュール3 バージョン管理、開発

- ・バージョン管理 - Git
- ・GitHub の基本
  - ・リポジトリの作成、クローン、コミット、プッシュ
- ・GitHubでのコラボレーション（共同開発）
  - ・GitHub flow
    - ・フォーク、プルリクエスト、マージ
  - ・継続的インテグレーション
  - ・よくあるご質問
  - ・まとめ

# GitHub flow

- GitHub社によって開発されたワークフロー（Gitを使った作業の手順）
- 軽量（lightweight、シンプルであるという意味）
- 比較的理解しやすい

# GitHub flow の基本操作

| 操作           | 意味                                  |
|--------------|-------------------------------------|
| フォーク         | 元のユーザーが所有するリポジトリのコピーを作成する           |
| ブランチ作成       | フォークしたリポジトリ内で新しい作業用のブランチを作成する       |
| 変更・コミット      | ブランチ上で必要な変更を行い、コミットする               |
| プッシュ         | コミット（群）を、フォークした自分のリポジトリへプッシュする      |
| プルリクエスト作成    | 元のユーザー、変更内容のフィードバックを求めるためのリクエストを作成  |
| (レビューコメント対応) | (レビューコメントに対応し、必要に応じて変更を加える)         |
| プルリクエストのマージ  | 元のリポジトリのユーザーが、プルリクエストをマージし、変更を反映させる |
| ブランチ削除       | 作業用のブランチを削除する                       |

# モジュール3 バージョン管理、開発

- ・バージョン管理 - Git
- ・GitHub の基本
  - ・リポジトリの作成、クローン、コミット、プッシュ
- ・GitHubでのコラボレーション（共同開発）
  - ・GitHub flow
  - ・フォーク、プルリクエスト、マージ
- ・継続的インテグレーション
- ・よくあるご質問
- ・まとめ



userA



→ フォーク



userB

元リポジトリの「Fork」ボタンを使用。  
元リポジトリのコピーが作られる。  
元リポジトリとのつながりは維持される

元リポジトリをアップストリーム（上流）、  
フォークしてできたリポジトリをダウンストリーム  
(下流) という。

フォーク後、  
元リポジトリで  
更新が行われた場合

フェッチ&マージを実行して元リポジトリの更新を取り込む



userA

(2) userAが  
編集を行う



userA/repo1  
main ブランチ

README.md

ver1

ver1  
ver2

(3) userAが  
mainブランチに  
直接コミットする

README.md

ver2

(1) userBがリポジトリを  
フォークする



userB

userB/repo1  
main ブランチ

README.md

ver1

※(3)の後にuserB/repo1に表示されるメッセージ  
This branch is 1 commit **behind** userA/repo1

(4) userBがフェッチ&  
マージする



Fetch and merge

README.md

ver2

This branch is **up to date** with userA/repo1

# モジュール3 バージョン管理、開発

- ・バージョン管理 - Git
- ・GitHub の基本
  - ・リポジトリの作成、クローン、コミット、プッシュ
- ・GitHubでのコラボレーション（共同開発）
  - ・GitHub flow
  - ・フォーク、**プルリクエスト、マージ**
- ・継続的インテグレーション
- ・よくあるご質問
- ・まとめ

# フォークしたリポジトリの mainブランチに 更新を行った場合

プルリクエストを作成して元リポジトリへの反映を依頼する。  
反映されたら、フェッチ&マージが必要



userA

Merge pull request  
Confirm merge

(4) プルリクエストを  
マージする  
※userBのコミットと、  
userAによるマージコミットの  
2つのコミットが記録される

userA/repo1  
main ブランチ

README.md

ver1

(1) userBがリポジトリを  
フォークする

userB/repo1  
main ブランチ

README.md

ver1

ver1  
ver2

README.md

ver2

userB

(2) userBが  
編集を行う

(3) userBが  
mainブランチに  
直接コミットする

This branch is 1 commit **ahead** of userA/repo1



Contribute

Open pull request

Create pull request

(3) プルリクエストを作成する

※(4)の後にuserB/repo1に表示されるメッセージ

This branch is 1 commit **behind** userA/repo1

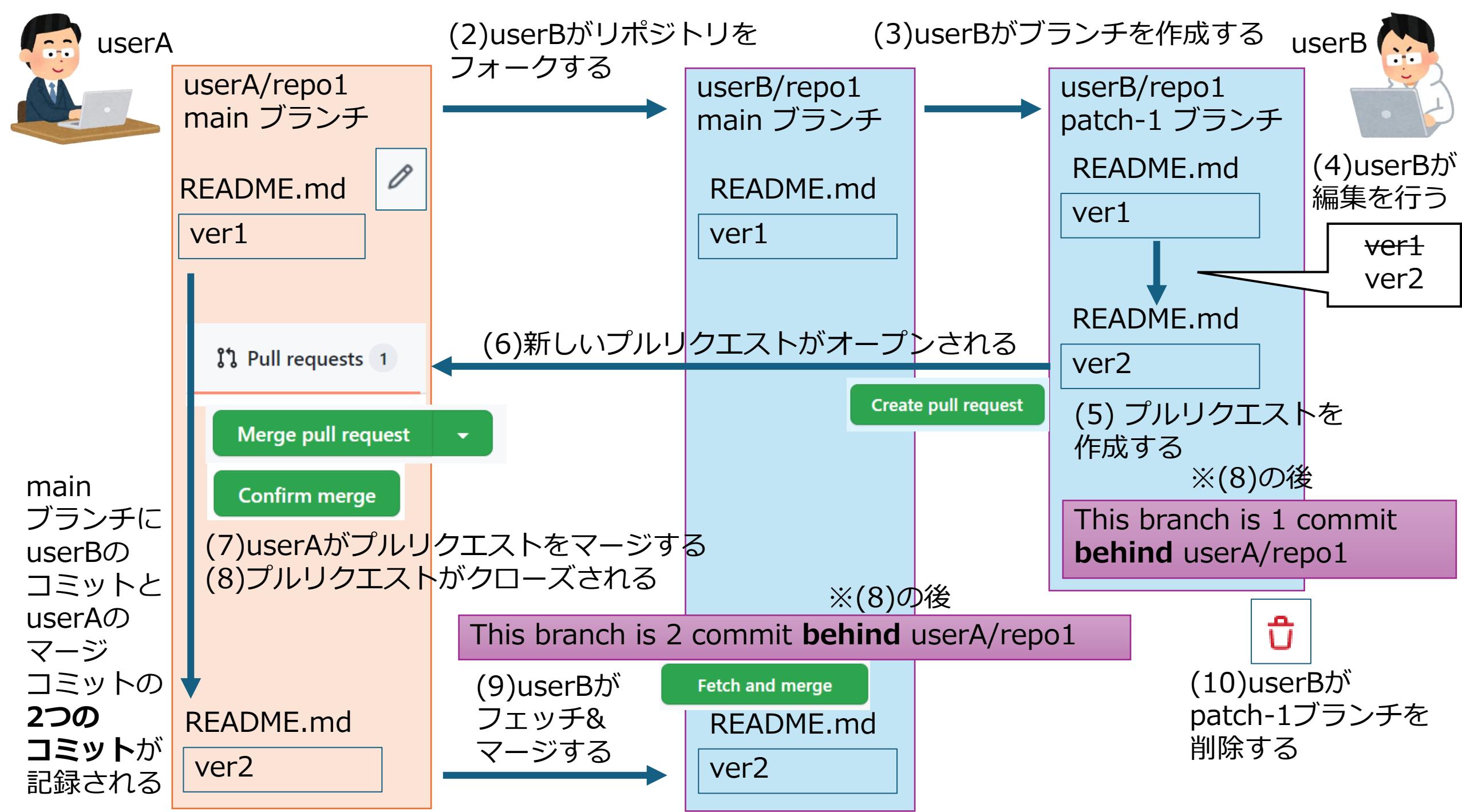
(5) userBがフェッチ&マージする

Fetch and merge

This branch is **up to date** with userA/repo1

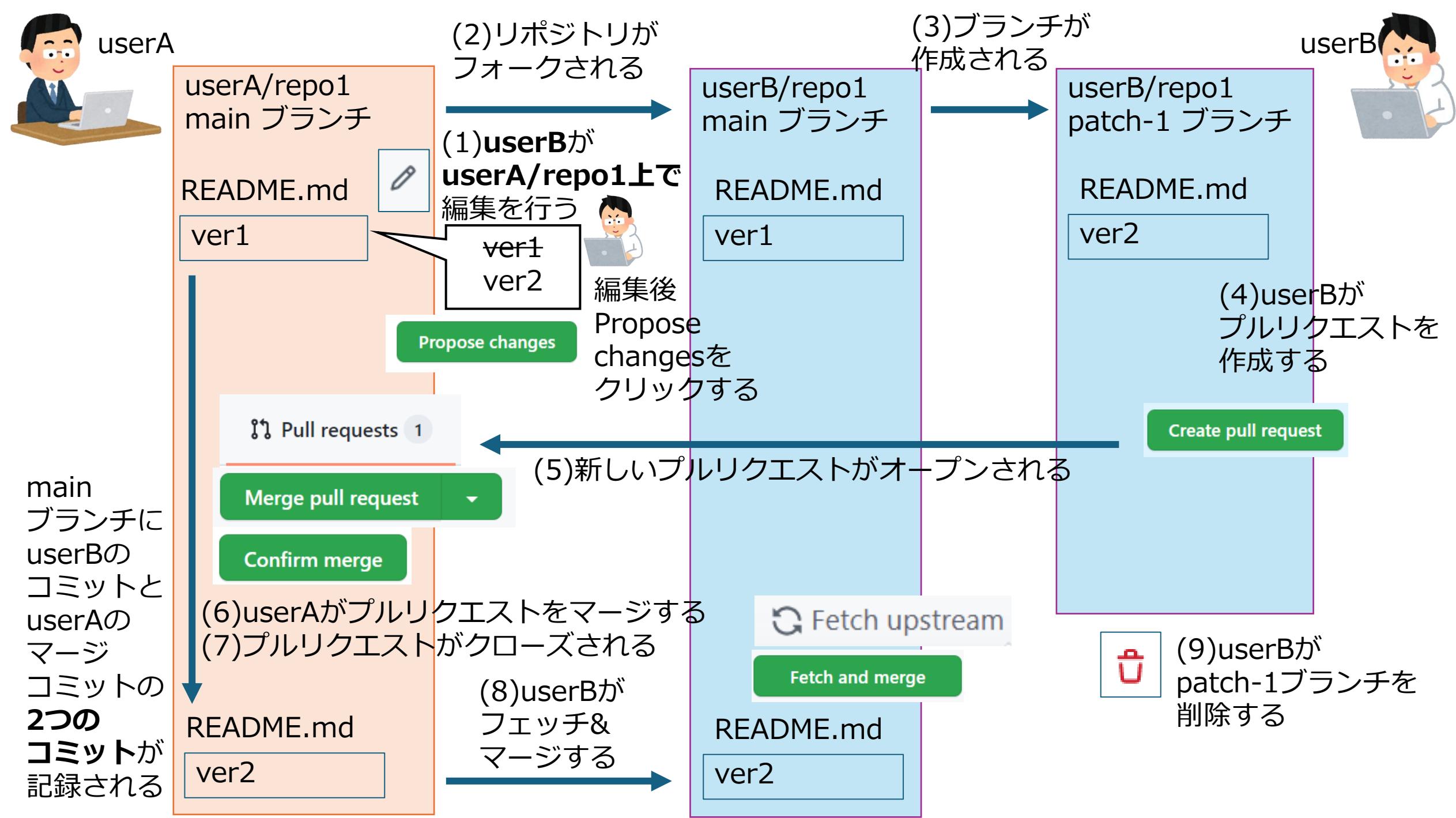
フォークしたリポジトリで  
新しいブランチを作り  
**更新を行った場合**

プルリクエストを作成して元リポジトリへの反映を依頼する。  
反映されたら、フェッチ&マージが必要



フォークしていない  
リポジトリに  
変更を加えた場合

自動的にフォークが行われ、パッチ用のブランチが作成される

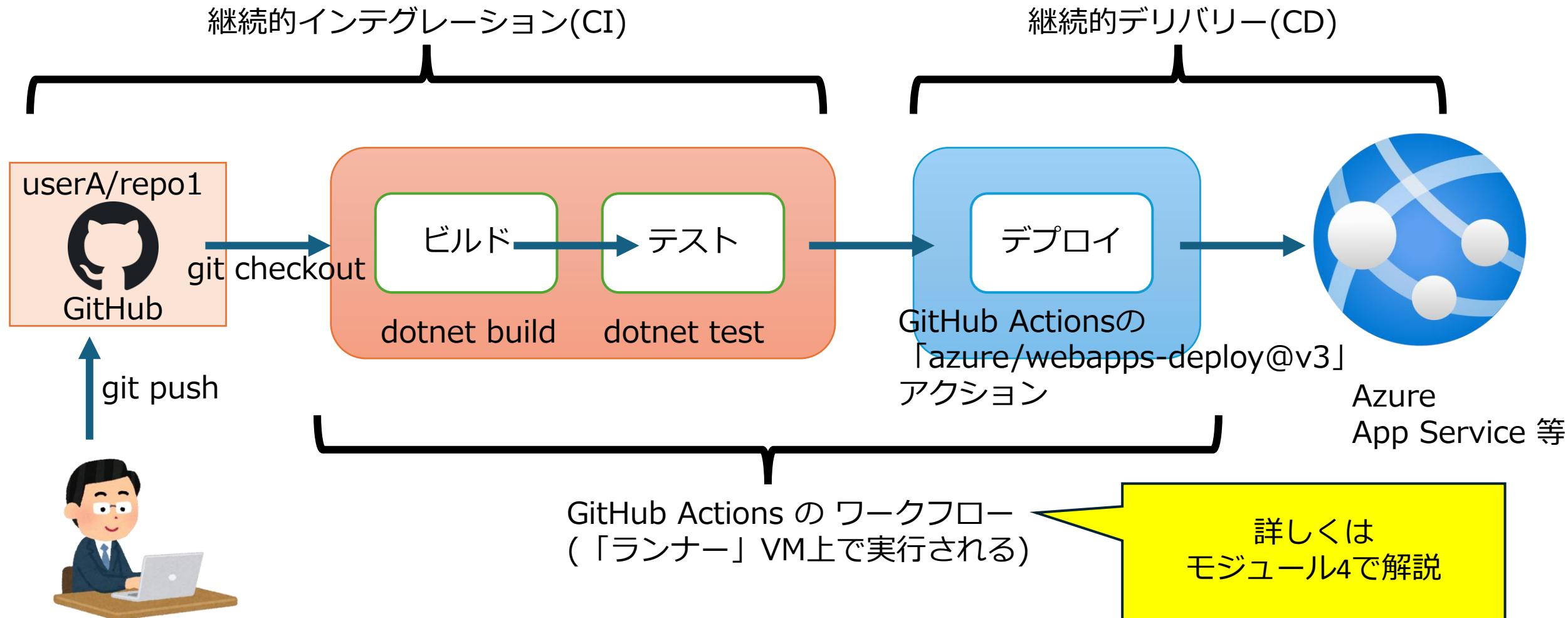


# モジュール3 バージョン管理、開発

- ・バージョン管理 - Git
- ・GitHub の基本
  - ・リポジトリの作成、クローン、コミット、プッシュ
- ・GitHubでのコラボレーション（共同開発）
  - ・GitHub flow
  - ・フォーク、プルリクエスト、マージ
- ・継続的インテグレーション
- ・よくあるご質問
- ・まとめ

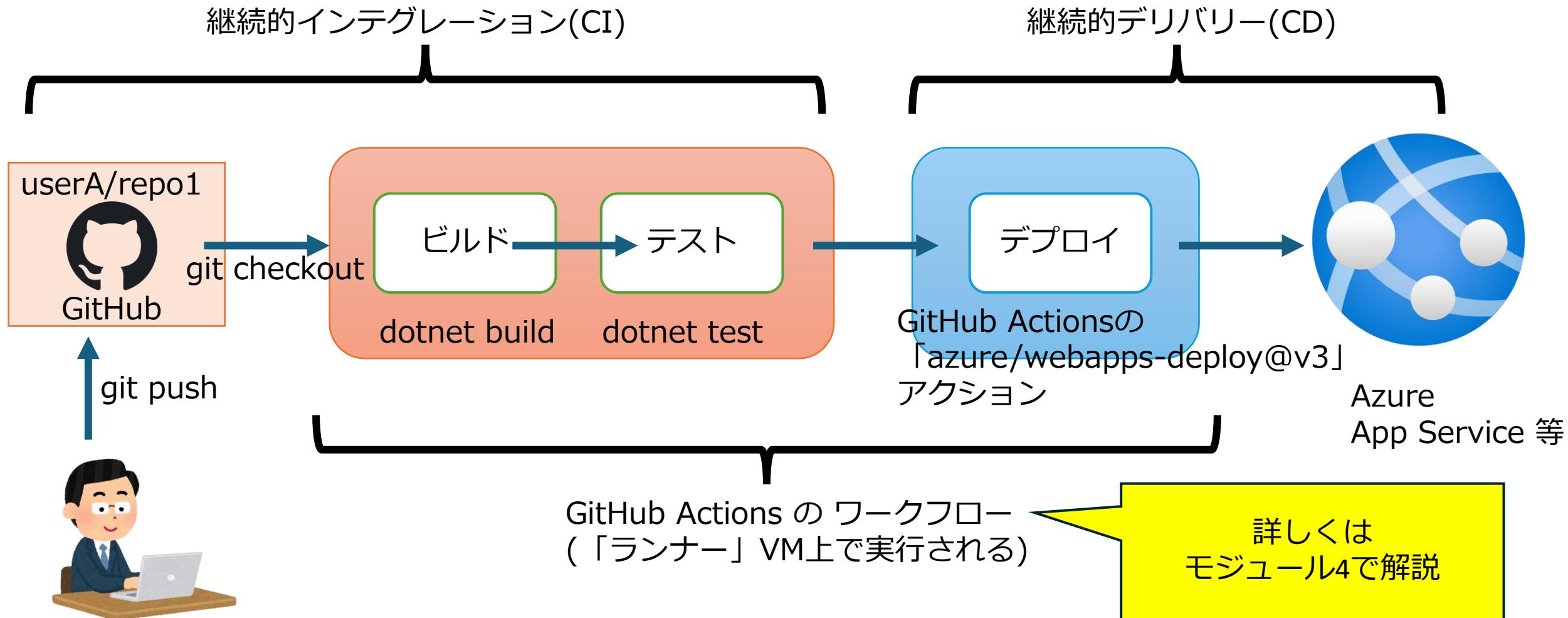
# 継続的インテグレーションとは？

- Gitリポジトリに送信されたコードを自動的にビルド、テストするプロセス
- ビルドやテストが「壊れていない」ことを自動で確認



# 継続的デリバリーとは？

- ・ビルド・テストされた成果物（Webアプリ等）をApp Service等に配置する作業。CIと合わせて一つのワークフローの一つのジョブで実行することも、別のジョブやワークフローで実行することも可能。



# モジュール3 バージョン管理、開発

- ・バージョン管理 - Git
- ・GitHub の基本
  - ・リポジトリの作成、クローン、コミット、プッシュ
- ・GitHubでのコラボレーション（共同開発）
  - ・GitHub flow
  - ・フォーク、プルリクエスト、マージ
- ・継続的インテグレーション
- ・よくあるご質問
- ・まとめ

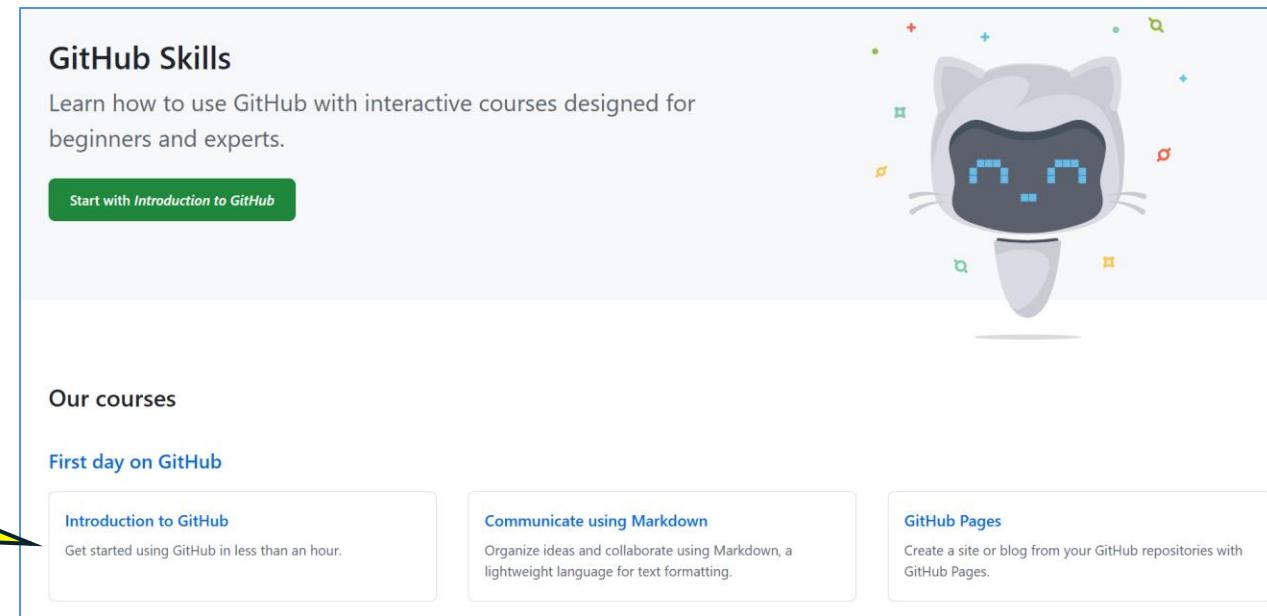
# よくあるご質問

- Git / GitHub の使い方がよくわからない
- → GitHub Desktop という、GitHub の操作をGUIで実行できるツールがあります。Gitのすべての機能が利用できるわけではありませんが、リポジトリの作成やクローン、プッシュなどの操作をわかりやすく実行できます。まずはこちらから使っていき GitHubに慣れるとよいのではないかと思います。
- [GitHub Desktop | Simple collaboration from your desktop](#)

# よくあるご質問

- GitHub の実際の操作方法をもっと詳しく知りたい
- → [GitHub Skills](#) がおすすめです。  
実際に手を動かして  
GitHubの使い方を  
ガイド付きで学べます。

たとえばこの「Introduction to GitHub」では、ブランチの作成、コミット、プルリクエストの作成、マージという一連の基本操作を学べます。



The screenshot shows the GitHub Skills landing page. At the top, there's a heading "GitHub Skills" with a subtext "Learn how to use GitHub with interactive courses designed for beginners and experts." Below this is a green button labeled "Start with *Introduction to GitHub*". To the right is a large, stylized cartoon cat head with glowing blue eyes, surrounded by small floating icons like stars and code snippets. The main content area is titled "Our courses" and lists three courses under the heading "First day on GitHub": "Introduction to GitHub" (described as "Get started using GitHub in less than an hour."), "Communicate using Markdown" (described as "Organize ideas and collaborate using Markdown, a lightweight language for text formatting."), and "GitHub Pages" (described as "Create a site or blog from your GitHub repositories with GitHub Pages").

# モジュール3 バージョン管理、開発

- ・バージョン管理 - Git
- ・GitHub の基本
  - ・リポジトリの作成、クローン、コミット、プッシュ
- ・GitHubでのコラボレーション（共同開発）
  - ・GitHub flow
  - ・フォーク、プルリクエスト、マージ
- ・継続的インテグレーション
- ・よくあるご質問
- ・まとめ

# モジュール3まとめ

|              |   |
|--------------|---|
| Git          | 現在デファクトスタンダードとなっているバージョン管理ツール。Windows/Mac/Linuxなど各種プラットフォームで利用できる。各種開発ツールはGitとの連携機能を持っている場合が多い。 |
| GitHub       | リポジトリの作成、クローン、コミット、プッシュ、プルなどの操作を実行できる。複数人での同時共同開発が可能。   |
| GitHub flow  | フォーク、ブランチ作成、コミット、プルリクエスト作成、マージといった流れで共同作業を行う。   |
| 継続的インテグレーション | Gitリポジトリに送信されたコードを自動的にビルド、テストするプロセス。ビルドやテストが「壊れていない」ことを自動で確認できる。                                |

# 講義



AZ-2008  
DevOpsの基礎:  
中心となる原則と実践

DevOps foundations: The core principles and practices

- ・モジュール1 DevOps の概要
- ・モジュール2 DevOps を使用した計画 (Plan with DevOps)
- ・モジュール3 DevOps を使用した開発 (Develop with DevOps)
- ・モジュール4 DevOps を使用した提供 (Deliver with DevOps)
- ・モジュール5 DevOps を使用した操作 (Operate with DevOps)
- ・まとめ

# モジュール4

デリバリー (CI/CD) ,  
IaC



## DevOps を使用した提供 (Deliver with DevOps)

GitHub Actions を使用して、継続的インテグレーションと継続的デリバリーのワークフローを構築して実行します。そのワークフローでは、リリースサイクルの高速化、回復性の向上、コラボレーションと再利用性の向上、コードとしてのインフラストラクチャが実現されます。

# モジュール4 CI/CD, IaC

- デプロイ
- リリース
- 繼続的デリバリー
- 繼続的デプロイ
- Infrastructure as Code (IaC)
  - AzureのIaC: ARMテンプレートとBicep
- Azure App Service
- GitHub Actionsによる自動化
  - AzureへのWebアプリのデプロイ
- よくあるご質問
- まとめ

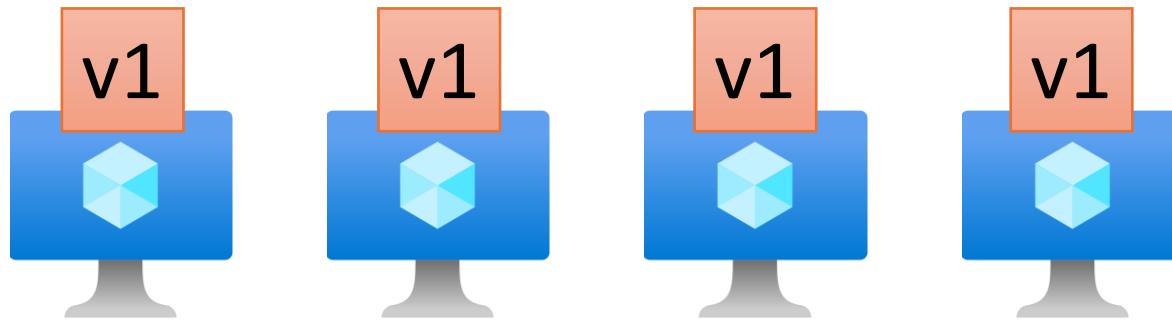
# デプロイとは？

- ・コードやアプリケーションを本番環境のサーバー等に配置すること
- ・いくつかの「デプロイ戦略」がある
  - ・インプレースデプロイ（一括デプロイ）
  - ・ローリングアップデート
  - ・ブルーグリーンデプロイ

# デプロイとは？

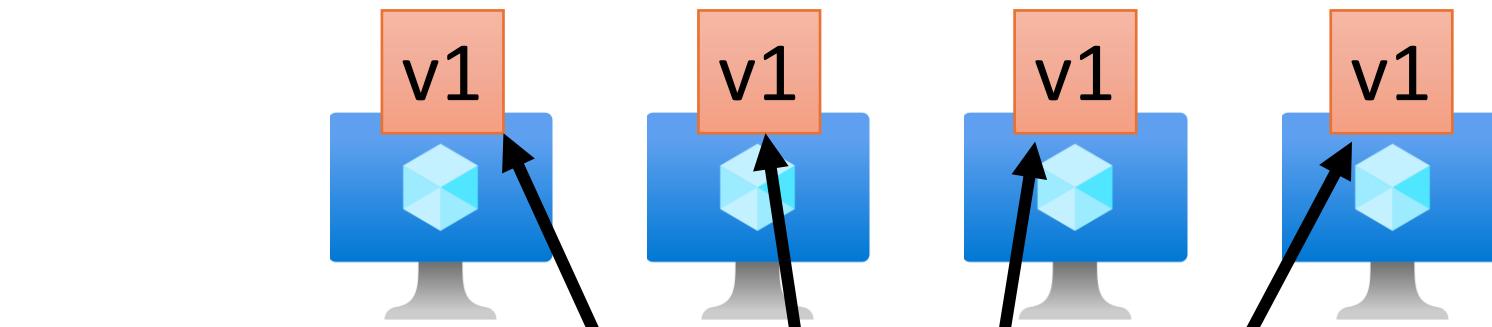
- ・コードやアプリケーションを本番環境のサーバー等に配置すること
- ・いくつかの「デプロイ戦略」がある
  - ・インプレースデプロイ（一括デプロイ）
  - ・ローリングアップデート
  - ・ブルーグリーンデプロイ

Azure



GitHub Actions

v2

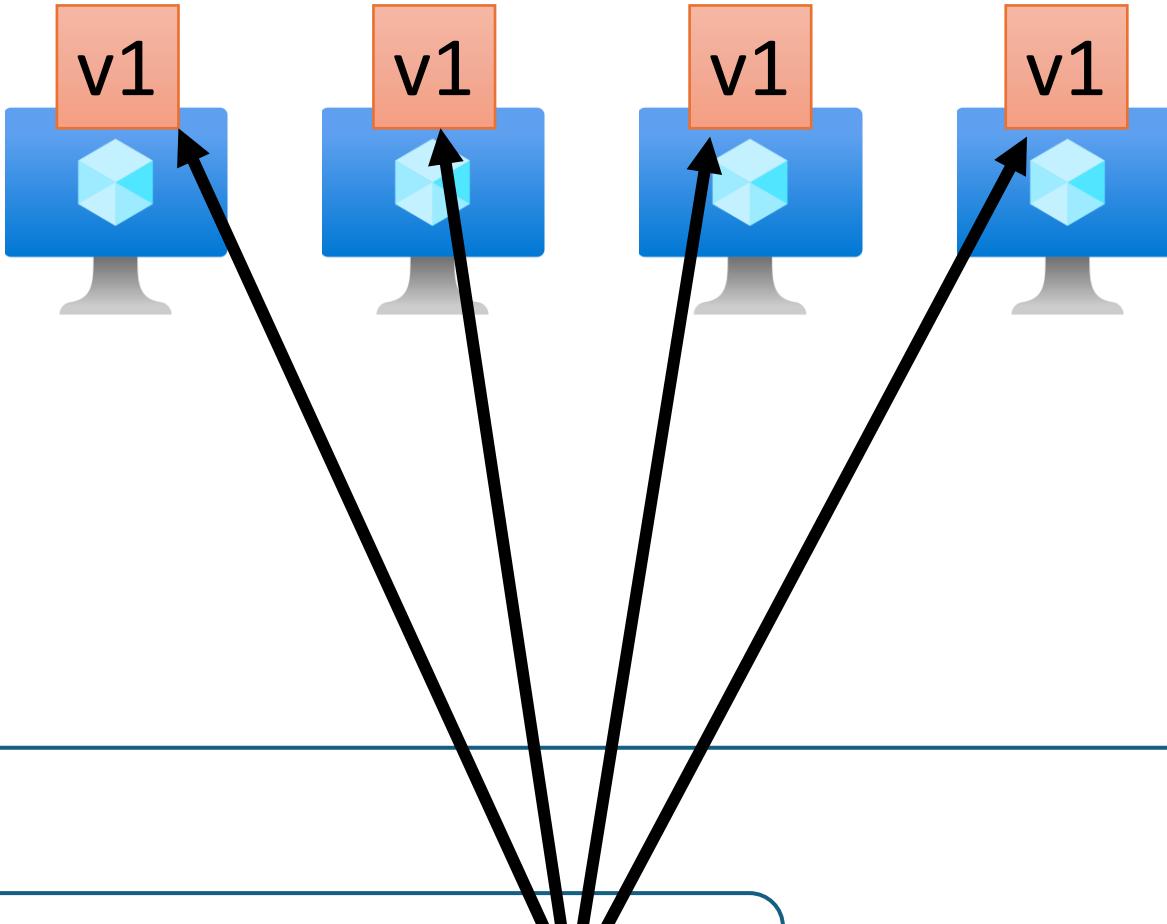


GitHub Actions

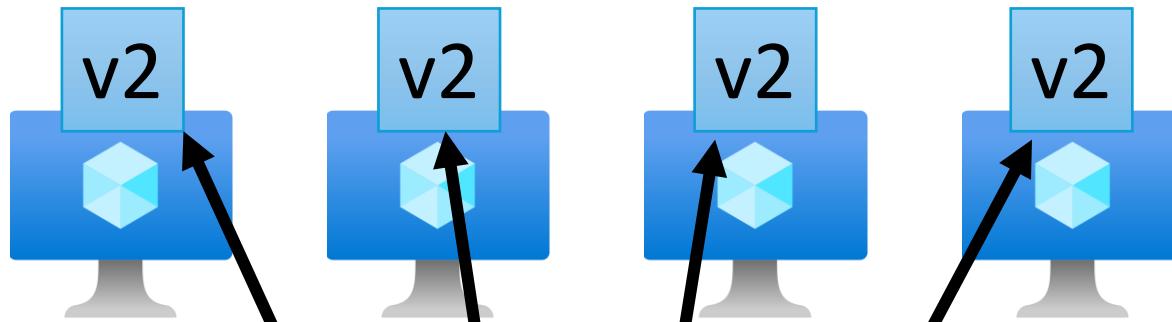
v2

GitHub Actions

v2



Azure



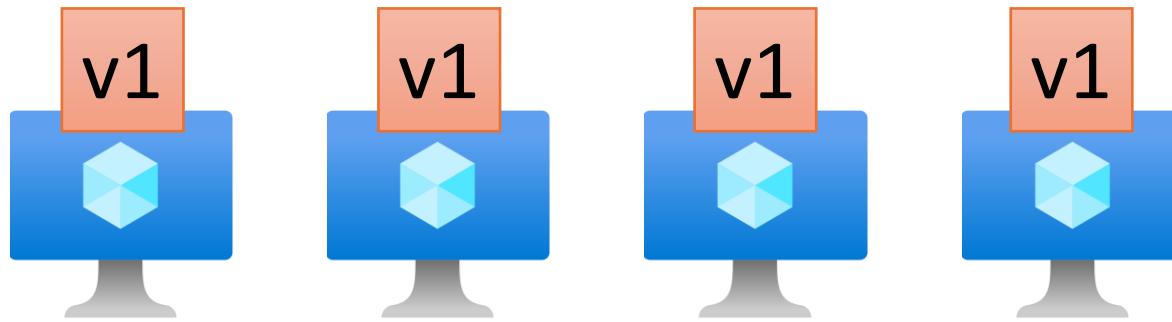
GitHub Actions

v2

# デプロイとは？

- ・コードやアプリケーションを本番環境のサーバー等に配置すること
- ・いくつかの「デプロイ戦略」がある
  - ・インプレースデプロイ（一括デプロイ）
    - ・ローリングアップデート
    - ・ブルーグリーンデプロイ

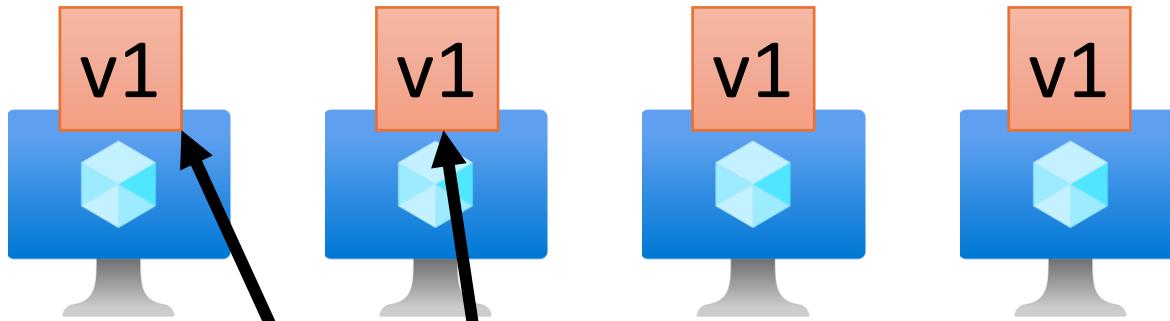
Azure



GitHub Actions

v2

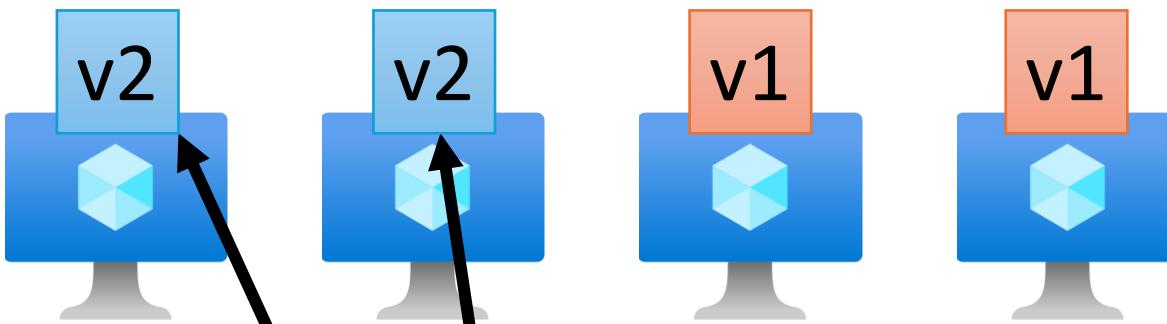
Azure



GitHub Actions

v2

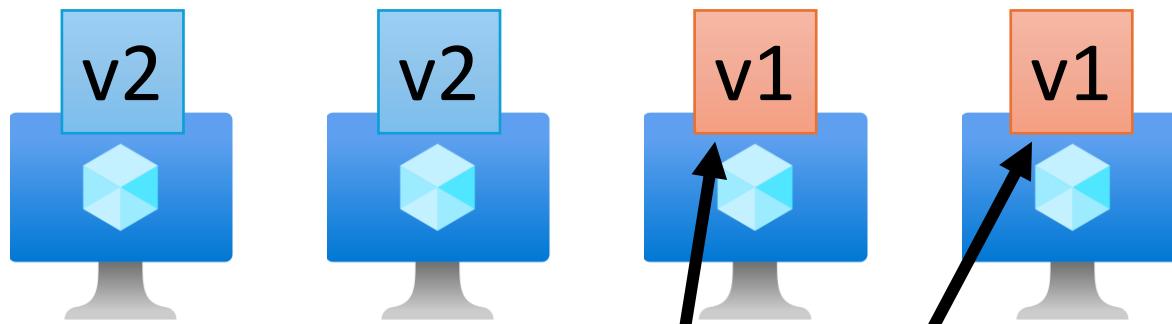
Azure



GitHub Actions

v2

Azure

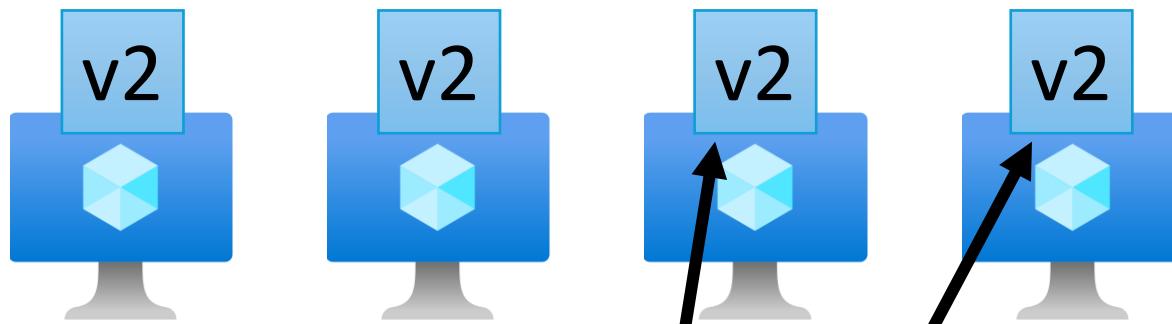


GitHub Actions

v2

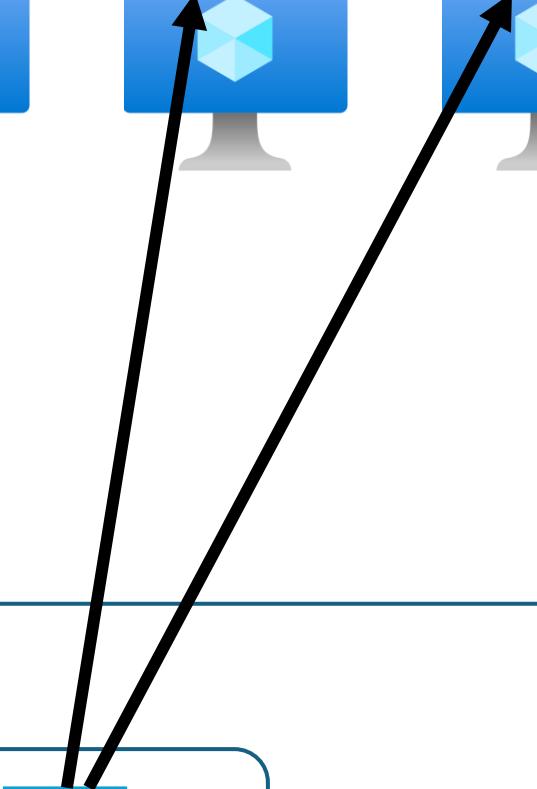


Azure



GitHub Actions

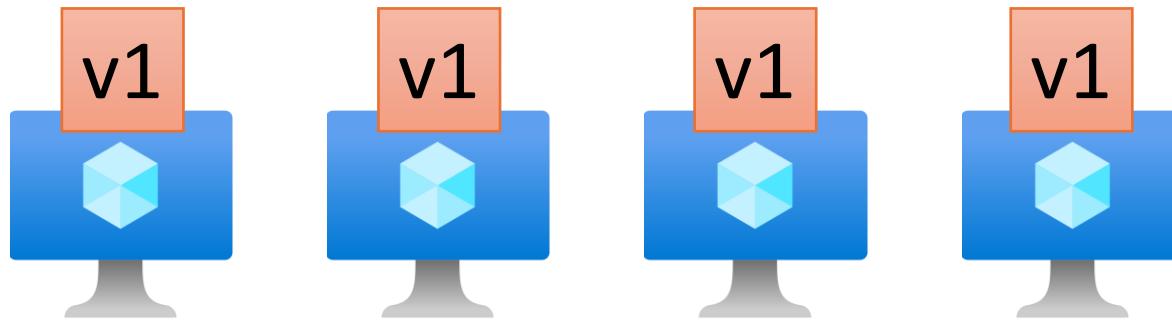
v2



# デプロイとは？

- ・コードやアプリケーションを本番環境のサーバー等に配置すること
- ・いくつかの「デプロイ戦略」がある
  - ・インプレースデプロイ（一括デプロイ）
  - ・ローリングアップデート
  - ・ブルーグリーンデプロイ

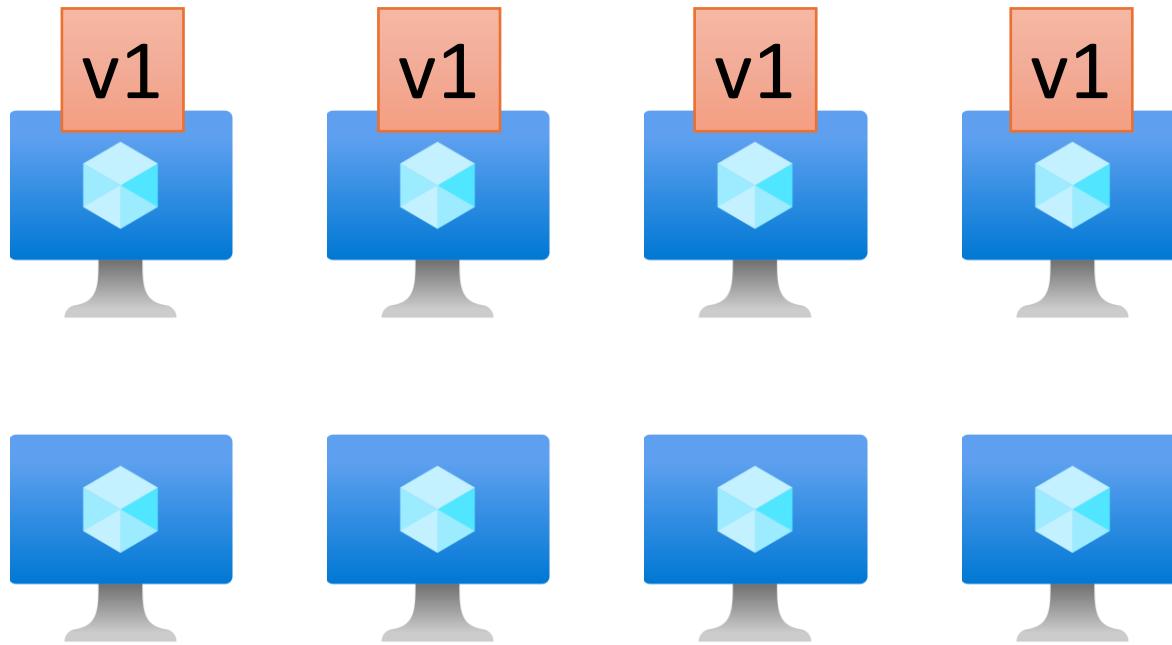
Azure



GitHub Actions

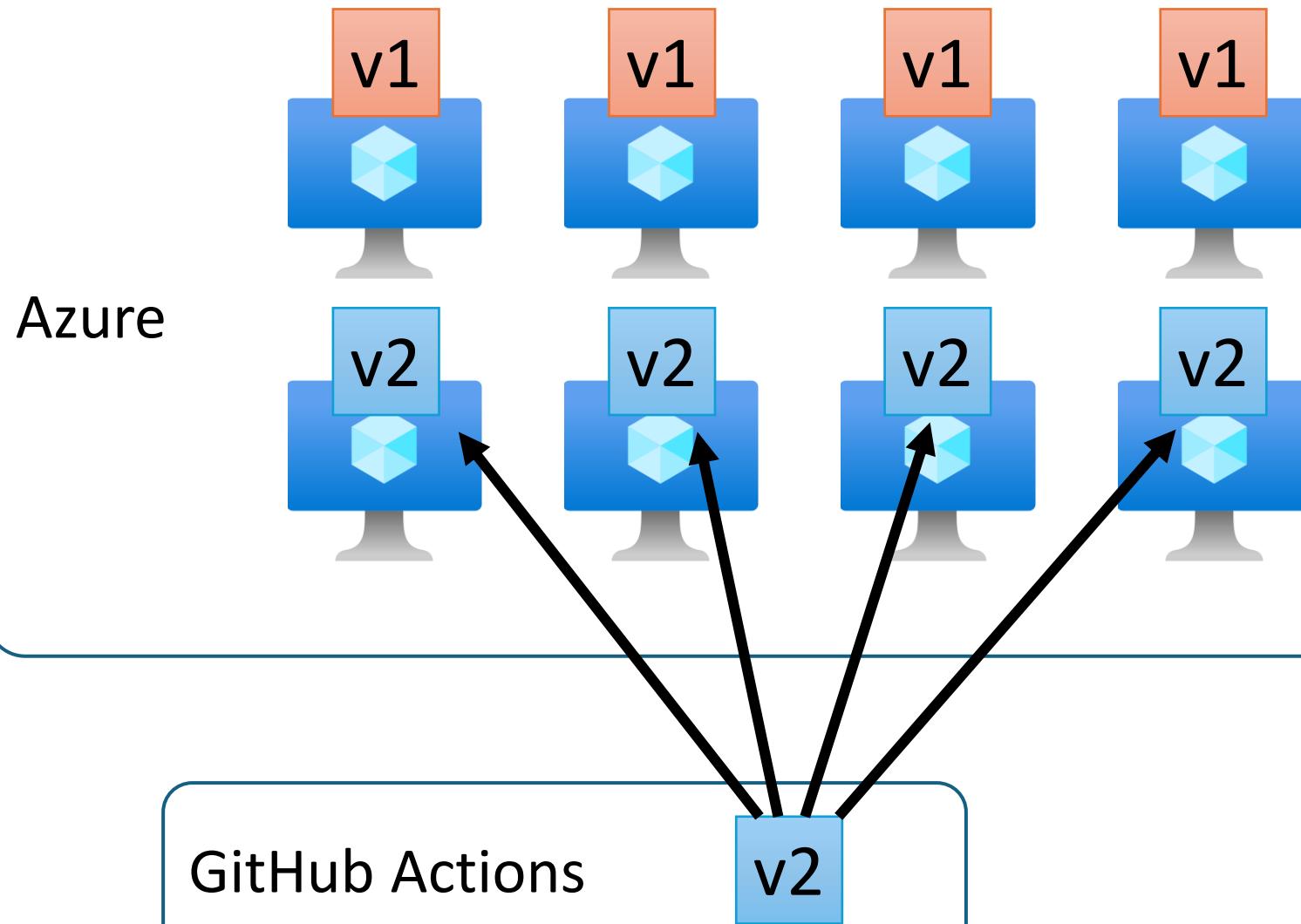
v2

Azure

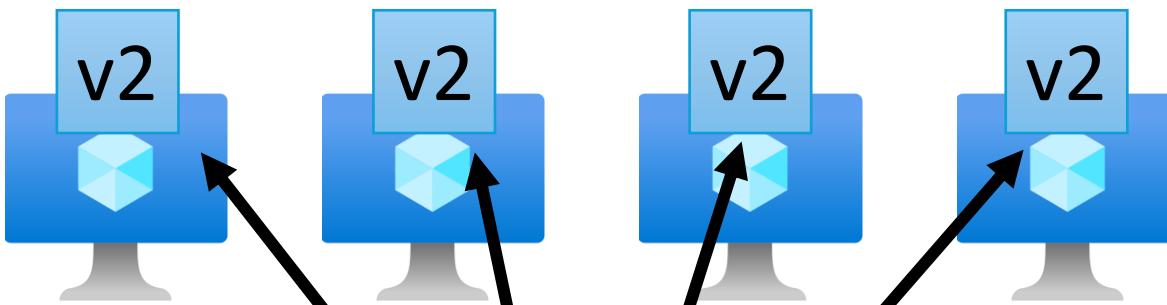


GitHub Actions

v2



Azure



GitHub Actions

v2

# モジュール4 CI/CD, IaC

- デプロイ
- **リリース**
- 繙続的デリバリー
- 繙続的デプロイ
- Infrastructure as Code (IaC)
  - AzureのIaC: ARMテンプレートとBicep
- Azure App Service
- GitHub Actionsによる自動化
  - AzureへのWebアプリのデプロイ
- よくあるご質問
- まとめ

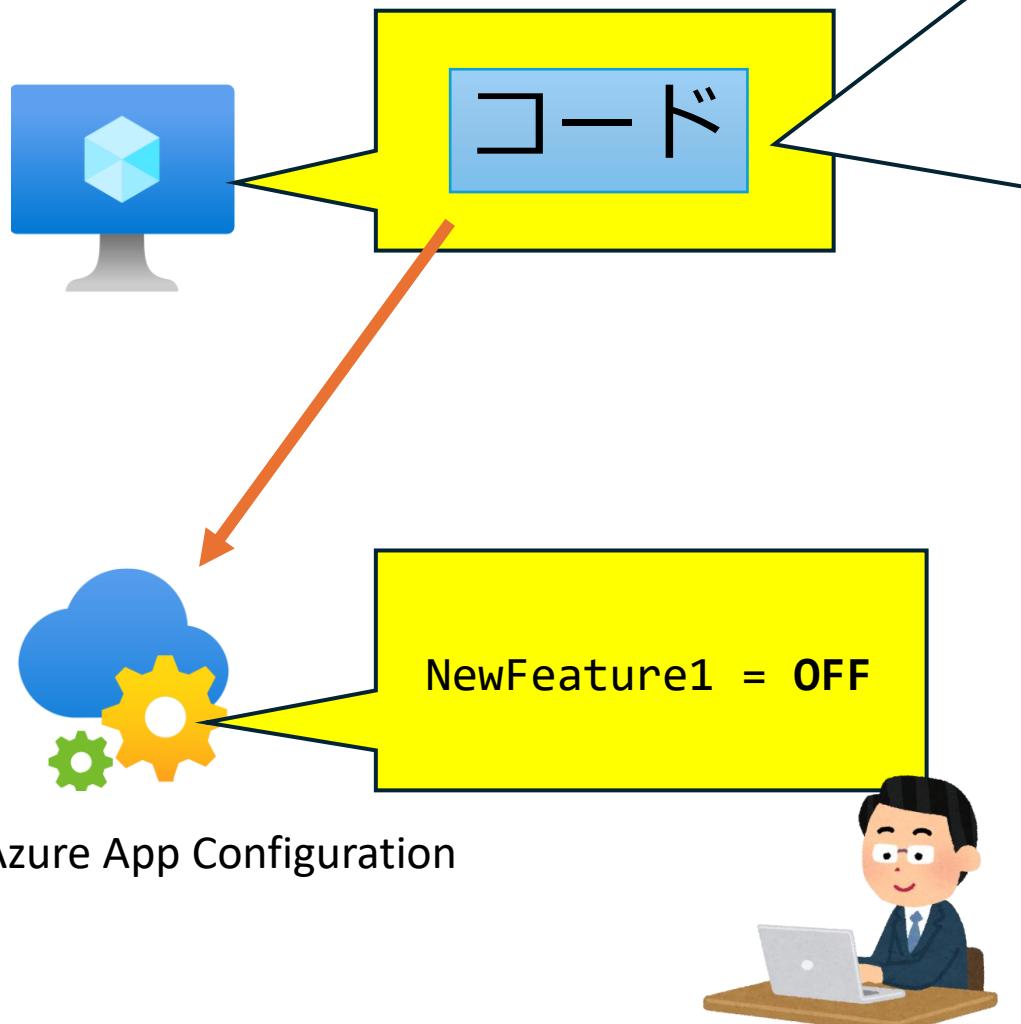
# リリースとは？

- ・デプロイされた新機能・変更を、ユーザーに対して**有効化**すること
- ・いくつかの「リリース戦略」がある
  - ・「機能フラグ」：管理者がフラグをONにすると、ユーザーに新機能・変更が提供され始める。
  - ・「段階的露出」：ユーザーをいくつかの「層」に分け段階的にリリース

# 機能フラグ

- ・コードをデプロイしなおすことなく、特定の機能のON・OFFを簡単に切り替えられるようにする仕組み
- ・管理者は、特定の機能を、デプロイとは別の任意のタイミングでONにする（機能をリリースする）ことができる
  - ・また逆に特定機能をOFFにできる
- ・Azure App Configuration などを使用して実現できる

# 機能フラグ

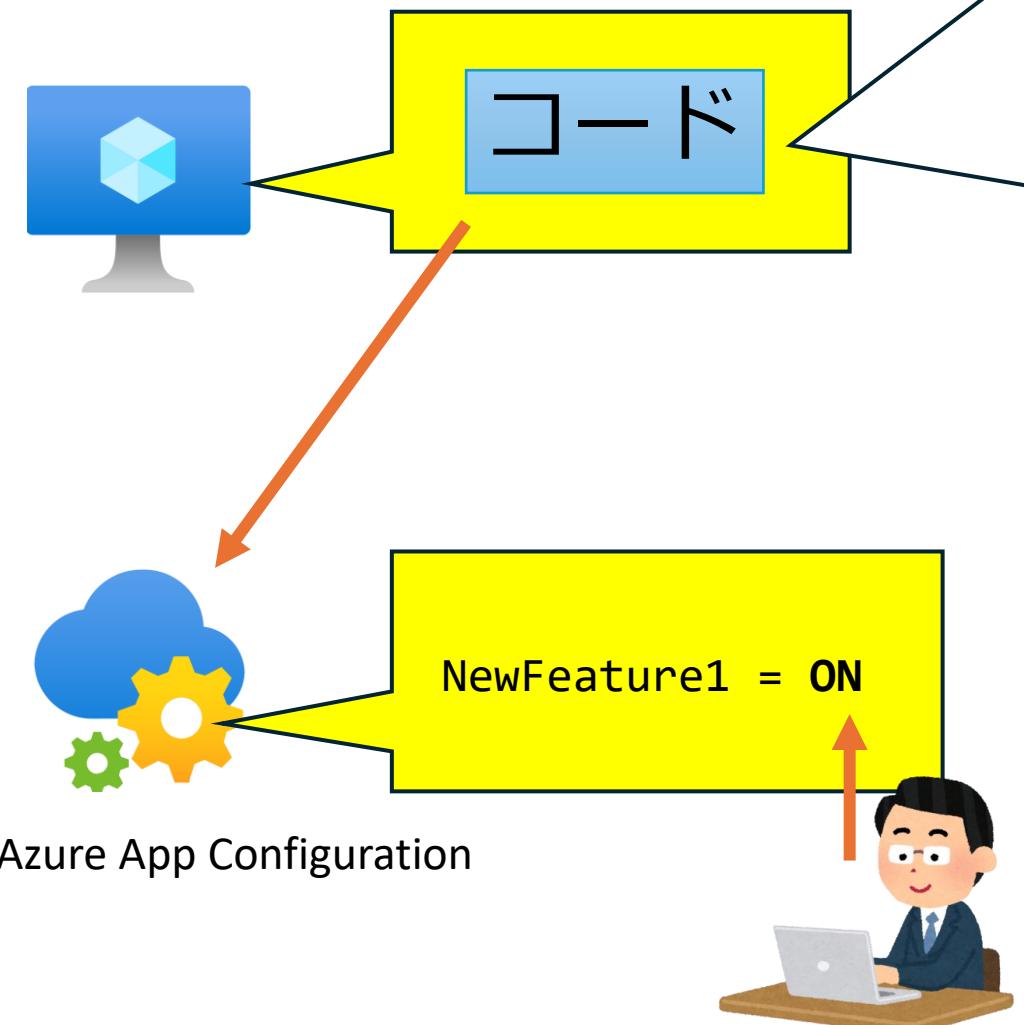


```
private readonly IFeatureManager _featureManager;  
  
if (_featureManager.IsEnabled("NewFeature1"))  
{  
    ... (新機能の処理) ...  
}
```

実行されない



# 機能フラグ

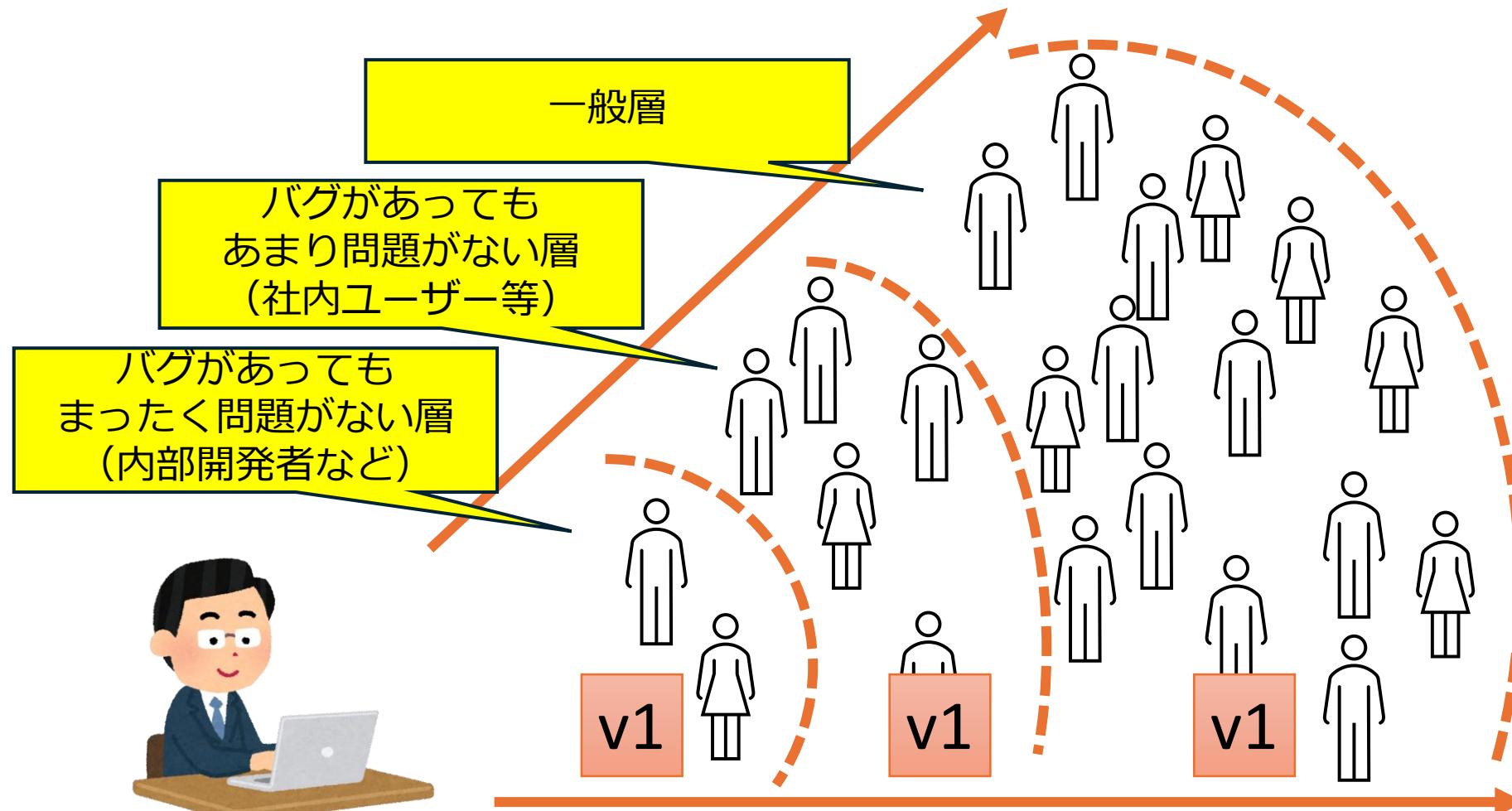


```
private readonly IFeatureManager _featureManager;  
  
if (_featureManager.IsEnabled("NewFeature1"))  
{  
    ... (新機能の処理) ...  
}
```

実行される

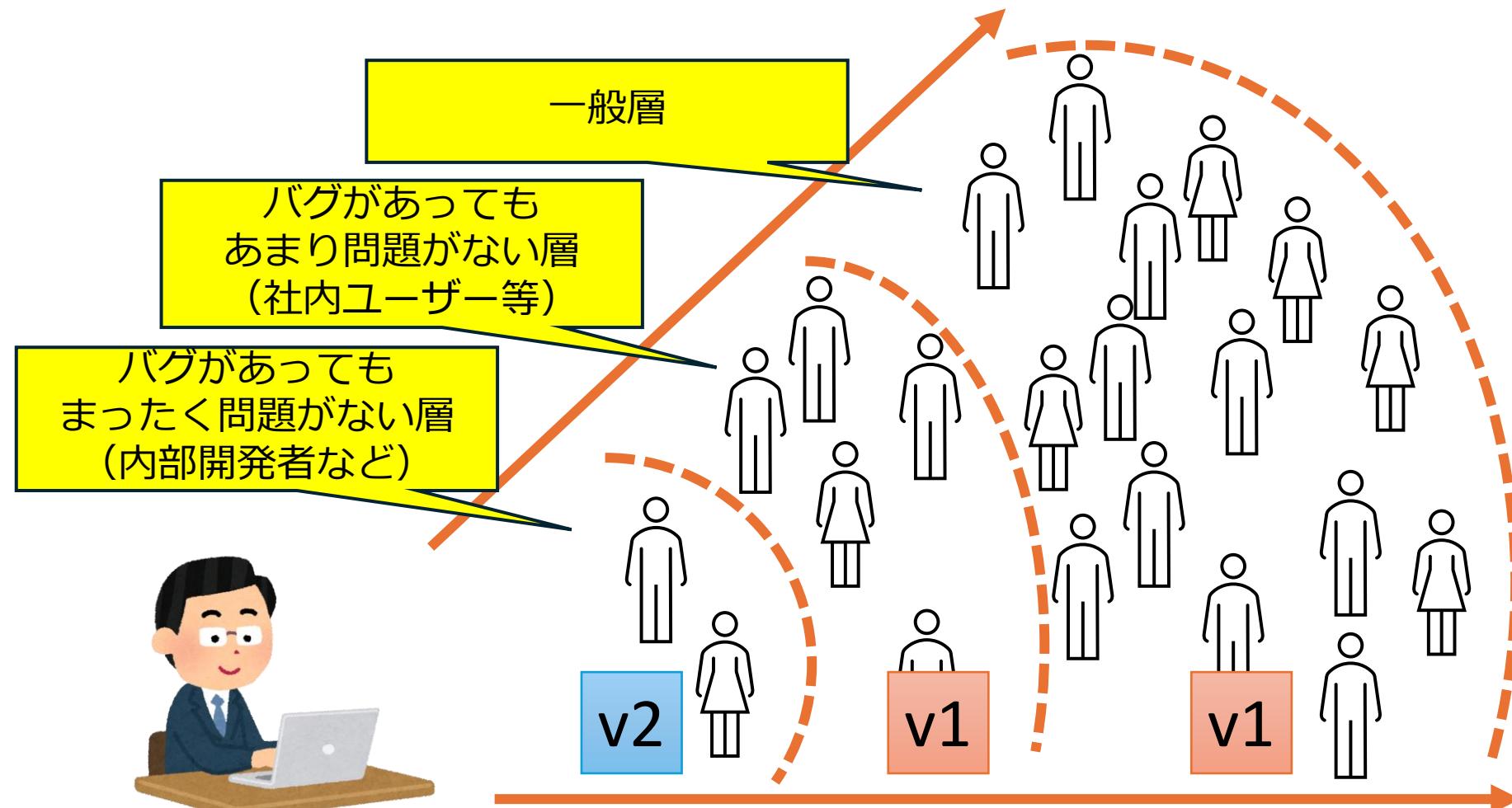
# Progressive exposure (段階的露出)

- ごく少数のユーザー層に対して新機能をリリースし、問題がないことが確認できたら、リリースの範囲を拡大していく手法



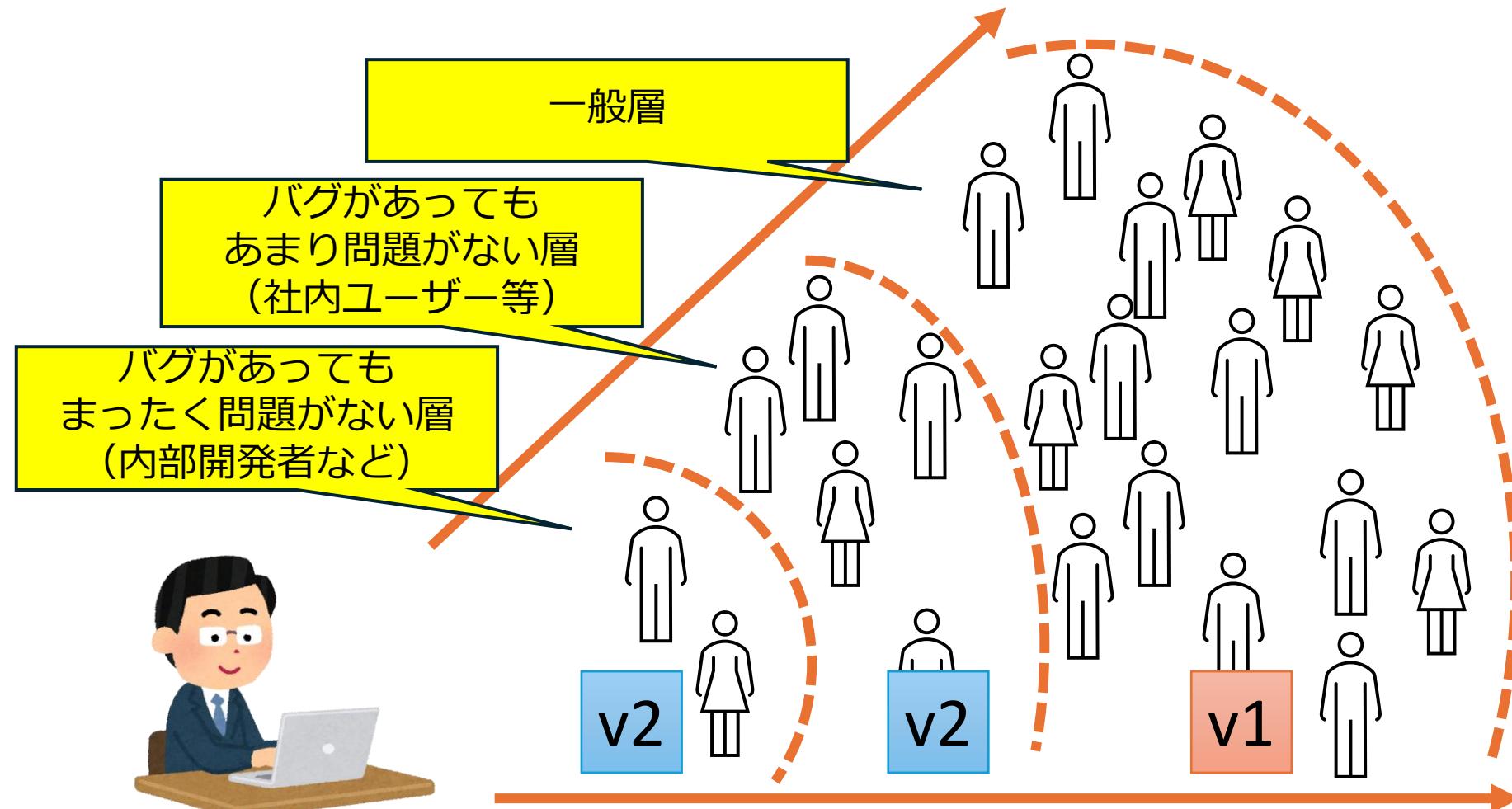
# Progressive exposure (段階的露出)

- ごく少数のユーザー層に対して新機能をリリースし、問題がないことが確認できたら、リリースの範囲を拡大していく手法



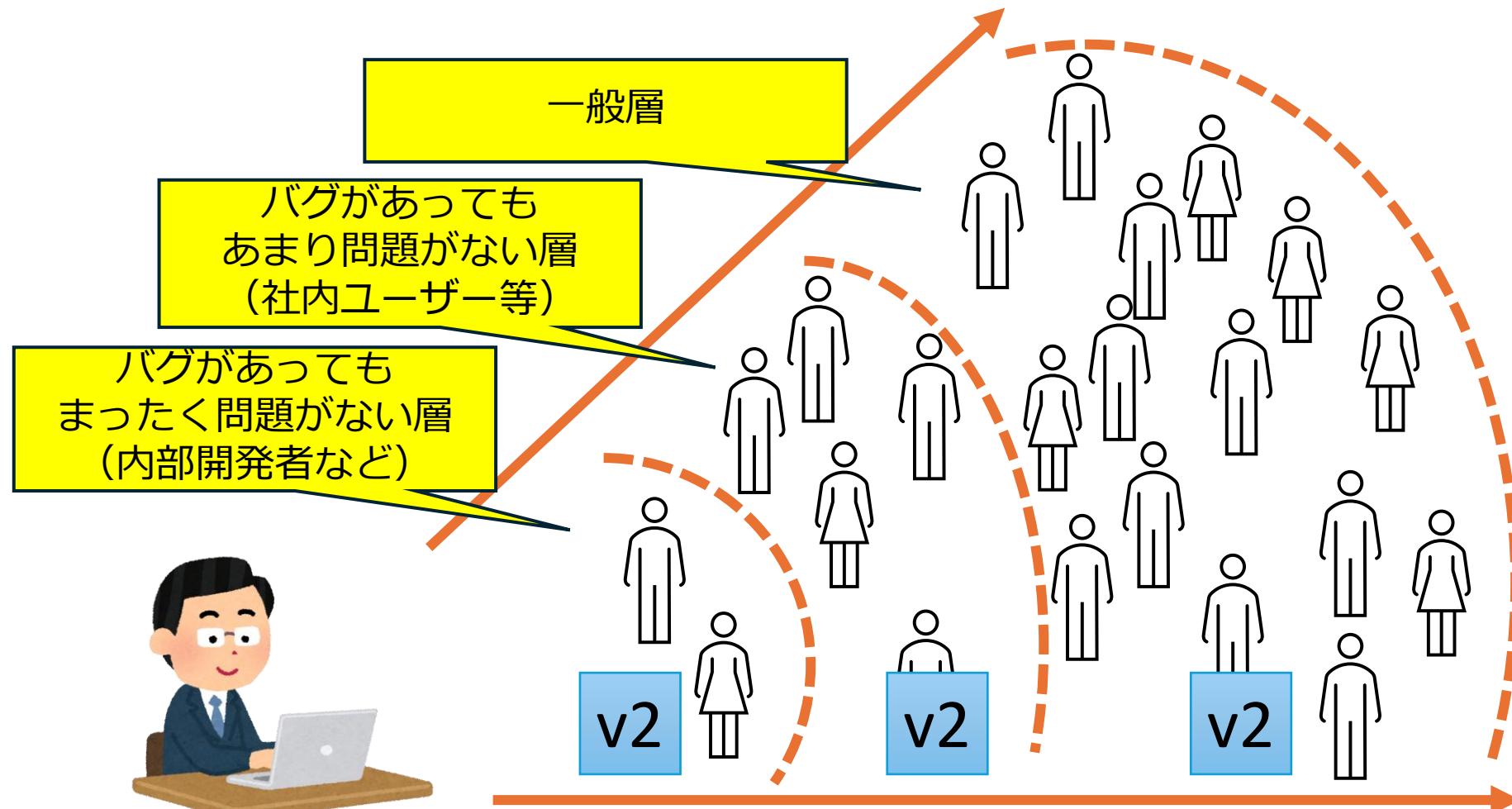
# Progressive exposure (段階的露出)

- ごく少数のユーザー層に対して新機能をリリースし、問題がないことが確認できたら、リリースの範囲を拡大していく手法



# Progressive exposure (段階的露出)

- ごく少数のユーザー層に対して新機能をリリースし、問題がないことが確認できたら、リリースの範囲を拡大していく手法



# 参考: さまざまな「Progressive exposure」（段階的露出）

| 分野         | 意味                                       | 例                                | 利点                             |
|------------|--|----------------------------------|--------------------------------|
| DevOps     | 新機能やコードを一部のユーザーに段階的に展開し、安全性と信頼性を確保する。    | 機能フラグ、リングベース展開、A/Bテスト、カナリアリリース   | 問題の影響を最小化、早期フィードバック、信頼性の高いデプロイ |
| リスク管理      | リスクを段階的に曝露し、影響をコントロールしながら学習・適応する。        | 小規模チームでの新技術の試行、一部サーバーへのパッチ適用     | 実際の影響を観察、失敗時の影響を小さく抑える         |
| UI/UX      | 必要なときに必要な情報や機能だけを段階的に提示する設計手法。           | 初回起動時は基本機能のみ → 慣れたら高度な機能を解放      | ユーザーエクスペリエンスの向上、情報過多の防止        |
| 金融         | 新しい金融商品や投資戦略を市場に段階的に導入する。                | 投資アルゴリズムの社内テスト、地域限定での新商品販売       | 市場の反応を見ながらリスクを管理、顧客の信頼を維持      |
| 心理療法       | 不安障害やPTSD、恐怖症の治療に使われる技法で、段階的に恐怖や不安を軽減する。 | 高所恐怖症の人が、写真 → 動画 → 実際に高所へと段階的に曝露 | 恐怖や不安を段階的に軽減、安全な環境での治療         |
| 教育・学習      | 学習者が新しい情報やスキルに徐々に触れていくことで、理解を深める方法。      | 簡単な構文 → 関数 → クラスなど、段階的に学習内容を高度化  | 学習の効率向上、段階的なスキルアップ             |
| マーケティングと広告 | ブランドや製品を段階的に露出させて関心を高める戦略。               | ロゴ → 製品の一部 → 全体広告と段階的に展開         | 消費者の関心を自然に高める、ブランド認知の向上        |

# モジュール4 CI/CD, IaC

- デプロイ
- リリース
- 繙続的デリバリー
- 繙続的デプロイ
- Infrastructure as Code (IaC)
  - AzureのIaC: ARMテンプレートとBicep
- Azure App Service
- GitHub Actionsによる自動化
  - AzureへのWebアプリのデプロイ
- よくあるご質問
- まとめ

# 継続的デリバリー/継続的デプロイとは？

- 継続的デリバリー

- コードの変更が発生すると、自動的に実稼働環境へのリリースの**準備**が実行されるしくみ
- リリースのタイミングは人が判断し、手動で開始される

- 継続的デプロイ

- デプロイとリリースをすべて自動化する方法
- 特定のテストの成功といった条件により自動でリリースを行う

# モジュール4 CI/CD, IaC

- ・デプロイ
- ・リリース
- ・継続的デリバリー
- ・継続的デプロイ
- ・Infrastructure as Code (IaC)
  - AzureのIaC: ARMテンプレートとBicep
- ・Azure App Service
- ・GitHub Actionsによる自動化
  - AzureへのWebアプリのデプロイ
- ・よくあるご質問
- ・まとめ

# Infrastructure as Code (IaC) とは？

- ・インフラ（特にクラウドの仮想マシンなどのリソース）をスクリプトや定義ファイルで定義すること
- ・Azureでは以下のようなツールを使うことができる
  - Azure CLI
  - Azure PowerShell
  - ARM Template
  - Bicep
  - Terraform
  - Pulumi

# IaC (Infrastructure as Code) のメリット

- 再利用が可能
- リソースのデプロイを自動化できる
  - 人手による作業のミスの防止
- インフラ定義をファイル化し、バージョン管理（Git）などで記録できる
  - インフラの変更管理・履歴管理が可能となる
  - 誰が、いつ、どのリソースを、どのような理由で、どのように変更したかが明確に記録される
  - ロールバック（以前のバージョンのインフラ構成の復元）が可能
- リソースをオンデマンドで作成し、不要になつたら削除する、というダイナミックなリソース運用が可能となる
  - コスト削減
  - セキュリティ向上
  - 新しい構成をすばやく実験できる

# IaC (Infrastructure as Code) のアプローチ

- 命令形アプローチ:
  - Azure CLIやAzure PowerShellを使用してリソースの作成を行う
  - 命令の実行順（リソースの作成順）などに注意する必要がある
- 宣言型アプローチ
  - ARM TemplateやBicepなどの専用の定義フォーマットを使用
  - 最終的なリソース構成を記述する
  - リソースの作成順は（特にBicepの場合はほとんど）考慮しなくてよい

# モジュール4 CI/CD, IaC

- ・デプロイ
- ・リリース
- ・継続的デリバリー
- ・継続的デプロイ
- ・Infrastructure as Code (IaC)
  - ・AzureのIaC: ARMテンプレートとBicep
- ・Azure App Service
- ・GitHub Actionsによる自動化
  - ・AzureへのWebアプリのデプロイ
- ・よくあるご質問
- ・まとめ

# ARM (Azure Resource Manager)

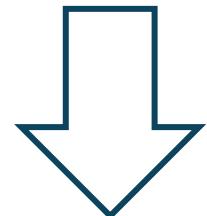
- ARMはAzureのリソース管理のしくみ
- Azureのすべてのリソースを管理する内部的なレイヤー
- Azure portalの操作、 Azure CLIからの操作、 Azure PowerShellからの操作、 ARM Templateのデプロイ、 Bicepのデプロイは、すべて最終的にARMに対するリクエスト（REST API呼び出し）となる
- ARMはリクエストを受けてリソースの作成・変更・削除などを実行する

「ARMテンプレート」を作成し、その中に、作成したいリソースの情報を記述する。  
複数のリソースをまとめて**デプロイ**（作成）できる。

azuredeploy.json



Azure CLIのコマンドの例



```
az deployment group create ¥  
-g testrg1 ¥  
-f azuredeploy.json
```

# ARM テンプレートの例

- JSON形式で、Azure リソースの定義を記述

App Service プランの例

App Service Webアプリの例

```
{  
  "type": "Microsoft.Web/serverfarms",  
  "apiVersion": "2022-03-01",  
  "name": "[parameters('appServicePlanName')]",  
  "location": "[parameters('location')]",  
  "sku": {  
    "name": "F1",  
    "tier": "Free"  
  },  
  "properties": {  
    "reserved": false  
  }  
}
```



```
{  
  "type": "Microsoft.Web/sites",  
  "apiVersion": "2022-03-01",  
  "name": "[parameters('webAppName')]",  
  "location": "[parameters('location')]",  
  "properties": {  
    "serverFarmId": "[resourceId('Microsoft.Web/serverfarms', parameters('appServicePlanName'))]",  
    "httpsOnly": true  
  }  
}
```



# モジュール4 CI/CD, IaC

- ・デプロイ
- ・リリース
- ・継続的デリバリー
- ・継続的デプロイ
- ・Infrastructure as Code (IaC)
  - ・AzureのIaC: ARMテンプレートとBicep
- ・Azure App Service
- ・GitHub Actionsによる自動化
  - ・AzureへのWebアプリのデプロイ
- ・よくあるご質問
- ・まとめ

# Bicep

- 専用の文法により、ARMテンプレートと同等の定義をより短く記述できる ※だいたい 1/3 くらいの字数で書ける
- 記述・メンテ・理解しやすい
- BicepファイルとARMテンプレートは相互に変換が可能
  - az bicep build / az bicep decompile

```
resource appServicePlan 'Microsoft.Web/serverfarms@2022-03-01' = {  
    name: appServicePlanName  
    location: location  
    sku: {  
        name: 'F1', tier: 'Free'  
    }  
    properties: {  
        reserved: false  
    }  
}
```



App Service プランの例

App Service Webアプリの例

```
resource webApp 'Microsoft.Web/sites@2022-03-01' = {  
    name: webAppName  
    location: location  
    properties: {  
        serverFarmId: appServicePlan.id  
        httpsOnly: true  
    }  
}
```

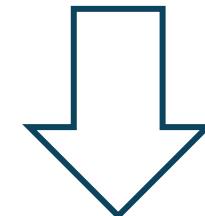


「Bicepファイル」を作成し、その中に、作成したいリソースの情報を記述する。  
複数のリソースをまとめて**デプロイ**（作成）できる。

main.bicep



Azure CLIのコマンドの例



```
az deployment group create ¥  
-g testrg1 ¥  
-f main.bicep
```

# モジュール4 CI/CD, IaC

- デプロイ
- リリース
- 繼続的デリバリー
- 繼続的デプロイ
- Infrastructure as Code (IaC)
  - AzureのIaC: ARMテンプレートとBicep
- Azure App Service
- GitHub Actionsによる自動化
  - AzureへのWebアプリのデプロイ
- よくあるご質問
- まとめ



# Azure App Service

App ServiceはPaaS (Platform as a Service) であり、  
IaaS (Infrastructure as a Service) に比べて、より多くの機能が提供される。



Azure仮想マシン  
IaaS



Azure App Service  
PaaS

OSの管理：必要

OSの管理：不要

言語ランタイム：インストール必要

言語ランタイム：インストール不要  
.NET/Java/Python/JavaScriptなど

スケーリングと負荷分散：運用が必要

スケーリングと負荷分散：組み込み

バックアップ：運用が必要

バックアップ：組み込み

App Serviceでは、「App Serviceプラン」と「App Serviceアプリ」というリソースを使用して運用する。1つのプランでは複数のアプリを運用できる。料金はアプリではなくプランに対して発生する。



App Serviceプラン



App Serviceアプリ



App Serviceプラン



App Serviceアプリ



App Serviceアプリ

App Service プランを作成する際に「価格レベル」を選択。  
これにより、性能や、使用できる機能が変化し、料金も変わる。



価格レベル:  
Basic



価格レベル:  
Standard



価格レベル:  
Premium

|            |        |        |        |
|------------|--------|--------|--------|
| 最大インスタンス数  | 最大 3   | 最大 10  | 最大 30* |
| カスタム ドメイン  | サポート対象 | サポート対象 | サポート対象 |
| 自動スケール     | -      | サポート対象 | サポート対象 |
| ハイブリッド接続   | サポート対象 | サポート対象 | サポート対象 |
| 仮想ネットワーク接続 | サポート対象 | サポート対象 | サポート対象 |

プラン内には最低1つの「インスタンス」が必要。

インスタンスを増やすと、よりたくさんのトラフィックを処理できるが  
よりコストもかかる。

Standard以上のプランでは自動スケールも利用可能。



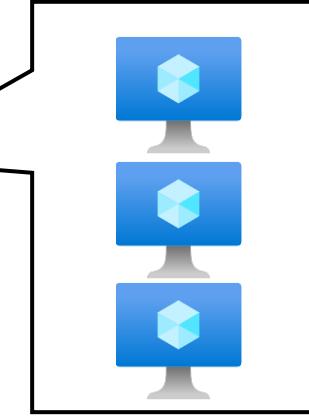
価格レベル:  
Basic



価格レベル:  
Standard

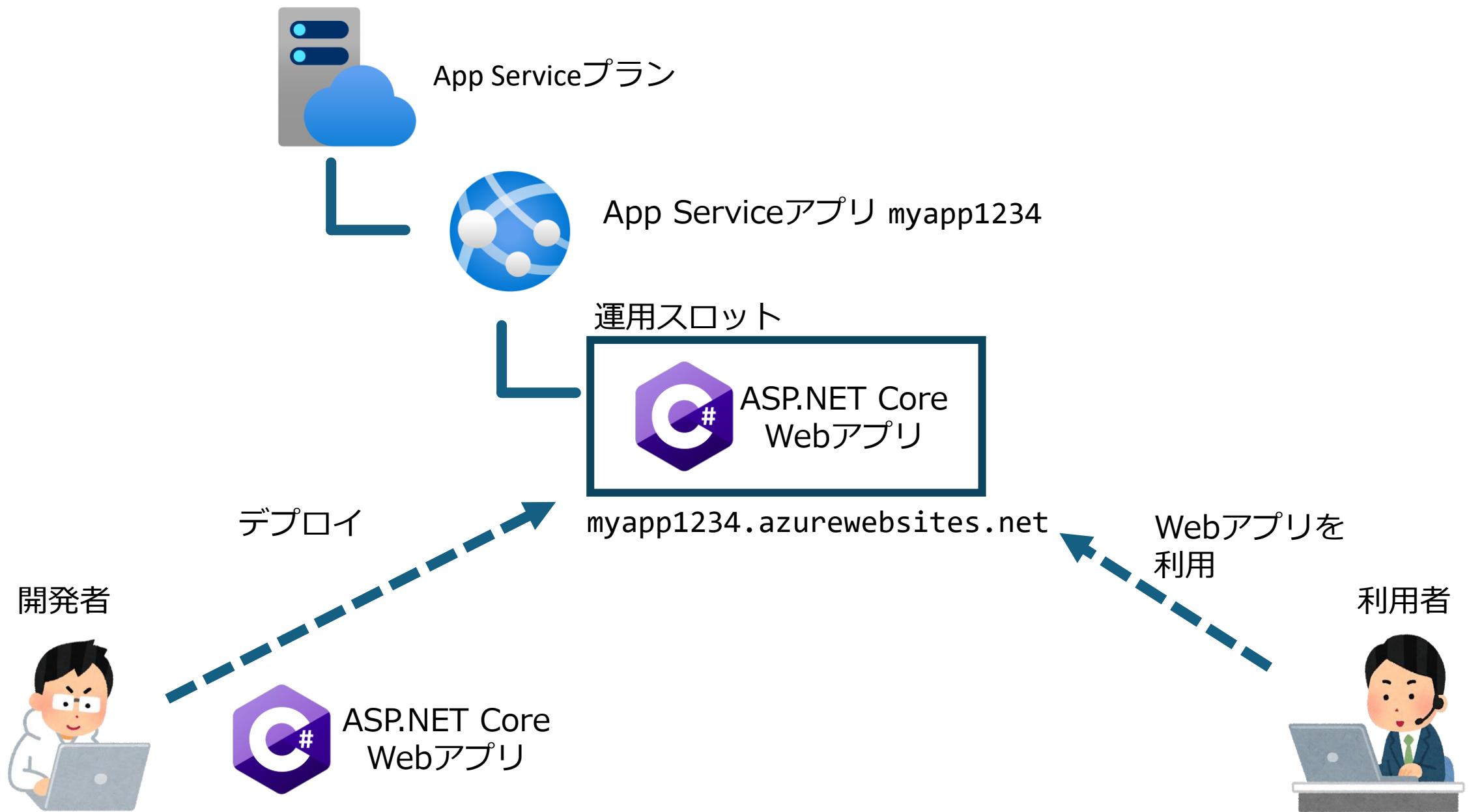


価格レベル:  
Premium



|            |        |        |        |
|------------|--------|--------|--------|
| 最大インスタンス数  | 最大 3   | 最大 10  | 最大 30* |
| カスタム ドメイン  | サポート対象 | サポート対象 | サポート対象 |
| 自動スケール     | -      | サポート対象 | サポート対象 |
| ハイブリッド接続   | サポート対象 | サポート対象 | サポート対象 |
| 仮想ネットワーク接続 | サポート対象 | サポート対象 | サポート対象 |

アプリには「運用スロット」と呼ばれる場所があり、そこにアプリをデプロイする



# モジュール4 CI/CD, IaC

- デプロイ
- リリース
- 繼続的デリバリー
- 繼続的デプロイ
- Infrastructure as Code (IaC)
  - AzureのIaC: ARMテンプレートとBicep
- Azure App Service
- GitHub Actionsによる自動化
  - AzureへのWebアプリのデプロイ
- よくあるご質問
- まとめ

# GitHub Actions

- GitHubリポジトリの中で使えるCI/CDツール
- YAML形式で「ワークフロー」を定義する
  - 「ワークフロー」はGitHubリポジトリの`.github/workflows/`以下に保存する
  - たとえば、リポジトリのWebアプリのコードをチェックアウトし（コードを「ランナー」へとコピーし）、ビルドし、テストを実行し、ビルドしたWebアプリのバイナリをAzure App Serviceへデプロイする、といった一連の動作を定義できる
- リポジトリへのコードのプッシュ（格納）などをトリガーとして「ワークフロー」が実行される
  - 「ワークフロー」は「ランナー」と呼ばれるVM上で実行される

App Service  
プラン・アプリを  
デプロイする  
Bicepファイル

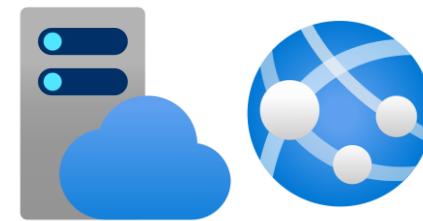
✓ AZ2008SAMPLE

- ✓ .github\workflows
  - ! deploy-app.yml
  - ! deploy-infra.yml
- ✓ app
  - > bin
  - > obj
  - > Pages
  - > Properties
  - > wwwroot
  - rss app.csproj
  - { appsettings.Development.json
  - { appsettings.json
  - C# Program.cs
- ✓ infra
  - fa main.bicep
- diagonal-dot .gitignore
- equiv az2008sample.sln

App Service  
プラン・アプリを  
デプロイする  
Bicepファイル

az2008sample

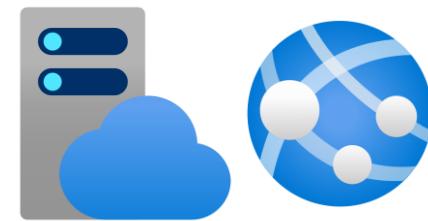
- .github\workflows
  - ! deploy-app.yml
  - ! deploy-infra.yml
- app
  - > bin
  - > obj
  - > Pages
  - > Properties
  - > wwwroot
  - rss app.csproj
  - { appsettings.Development.json
  - { appsettings.json
  - C# Program.cs
- infra
  - main.bicep
- .gitignore
- az2008sample.sln



C#のWebアプリ  
(ASP.NET Core)

App Service  
プラン・アプリを  
デプロイする  
Bicepファイル

```
▽ AZ2008SAMPLE
  ▽ .github\workflows
    ! deploy-app.yml
    ! deploy-infra.yml
  ▽ app
    > bin
    > obj
    > Pages
    > Properties
    > wwwroot
    > app.csproj
    { } appsettings.Development.json
    { } appsettings.json
    C# Program.cs
  ▽ infra
    < main.bicep
  .gitignore
  az2008sample.sln
```



C#のWebアプリ  
(ASP.NET Core)

App Service  
プラン・アプリを  
デプロイする  
Bicepファイル

▽ AZ2008SAMPLE

▽ .github\workflows

! deploy-app.yml

! deploy-infra.yml

▽ app

> bin

> obj

> Pages

> Properties

> wwwroot

rss app.csproj

{ appsettings.Development.json

{ appsettings.json

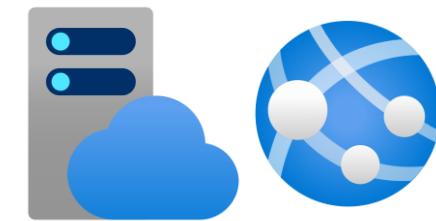
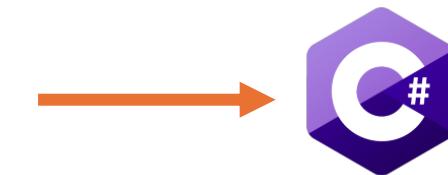
C# Program.cs

▽ infra

fa main.bicep

diagonal-striped .gitignore

equiv az2008sample.sln



C#のWebアプリ  
(ASP.NET Core)

App Service  
プラン・アプリを  
デプロイする  
Bicepファイル

▽ AZ2008SAMPLE

▽ .github\workflows

! deploy-app.yml

! deploy-infra.yml

▽ app

> bin

> obj

> Pages

> Properties

> wwwroot

rss app.csproj

{ } appsettings.Development.json

{ } appsettings.json

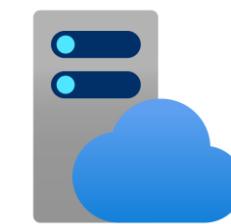
C# Program.cs

▽ infra

fa main.bicep

diagonal-striped-dot .gitignore

equiv az2008sample.sln



C#のWebアプリ  
(ASP.NET Core)

App Service  
プラン・アプリを  
デプロイする  
Bicepファイル

▽ AZ2008SAMPLE

▽ .github\workflows

! deploy-app.yml

! deploy-infra.yml

▽ app

> bin

> obj

> Pages

> Properties

> wwwroot

rss app.csproj

{ } appsettings.Development.json

{ } appsettings.json

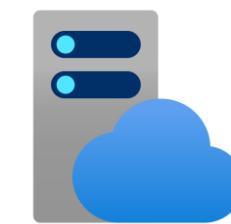
C# Program.cs

▽ infra

fa main.bicep

diagonal-striped-dot .gitignore

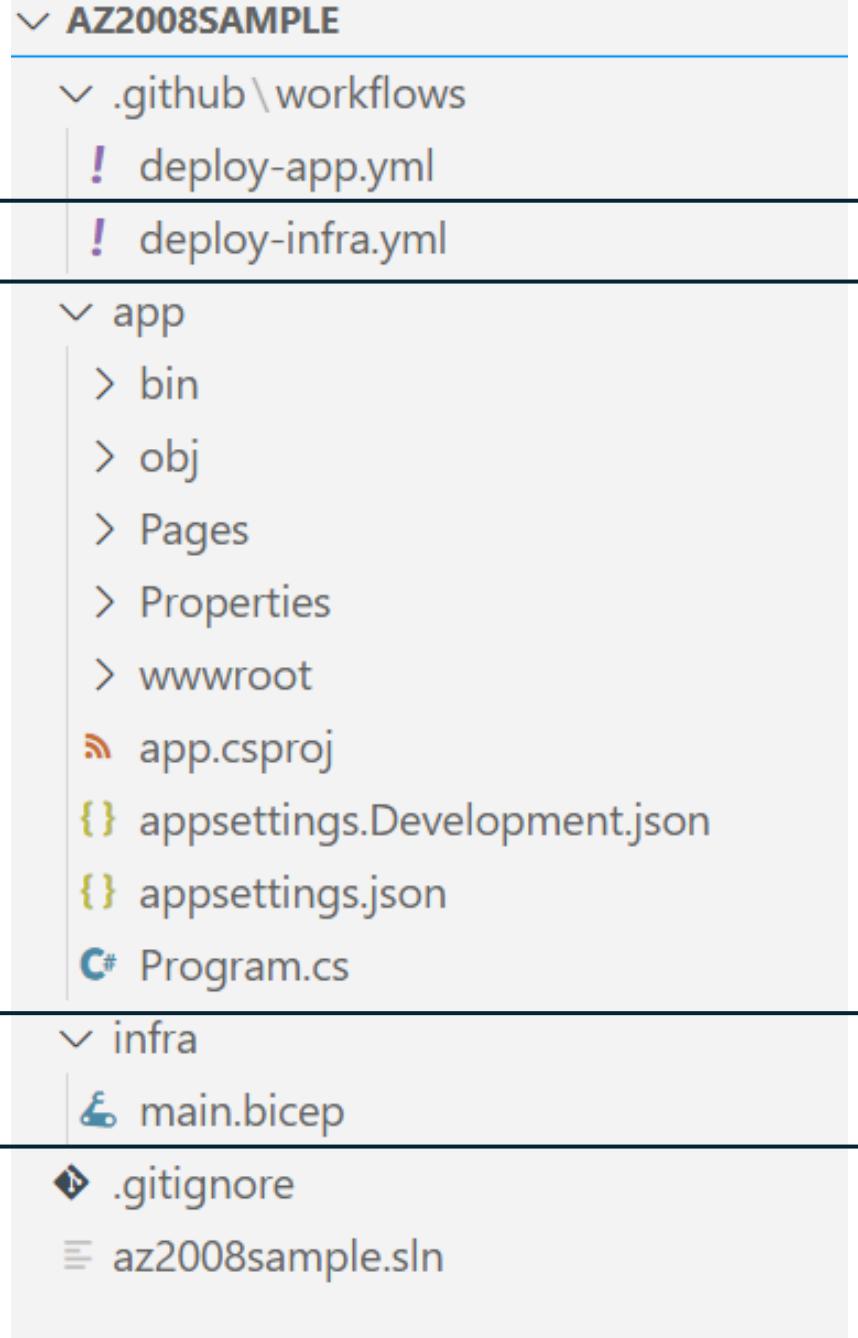
equiv az2008sample.sln



つまりこの場合・・・

- ・「Bicepを使用して Azure App Service のリソースを作成するワークフロー」と
- ・「Webアプリのソースコードをビルド・テストし、Azure App Service にデプロイするワークフロー」
- ・の、2つのワークフローが必要となる。

App Service  
プラン・アプリを  
デプロイする  
Bicepファイル



Bicepファイルが  
変更されたら  
Azureへそれを  
デプロイする  
ワークフロー

# App Serviceの プラン・アプリを デプロイする ワークフロー

git push され、かつ  
infra フォルダ以下に  
変更があった場合に  
このワークフローを  
実行

Azureにサインインし、  
Bicepをデプロイ

```
name: Deploy Infrastructure
on:
  push:
    paths:
      - 'infra/**'
  workflow_dispatch:
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4
      - name: Azure Login
        uses: azure/login@v1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }}
      - name: Deploy Bicep
        run:
          az deployment group create \
            --resource-group az2008samplerg \
            --template-file infra/main.bicep
```

このワークフローの名前

オプション（これを書いておくと、GitHub Actionsの画面でボタンをクリックして、このワークフローを実行できる）

C#のWebアプリ  
(ASP.NET Core)

▽ AZ2008SAMPLE

▽ .github\workflows

! deploy-app.yml

! deploy-infra.yml

▽ app

> bin

> obj

> Pages

> Properties

> wwwroot

rss app.csproj

{ } appsettings.Development.json

{ } appsettings.json

C# Program.cs

▽ infra

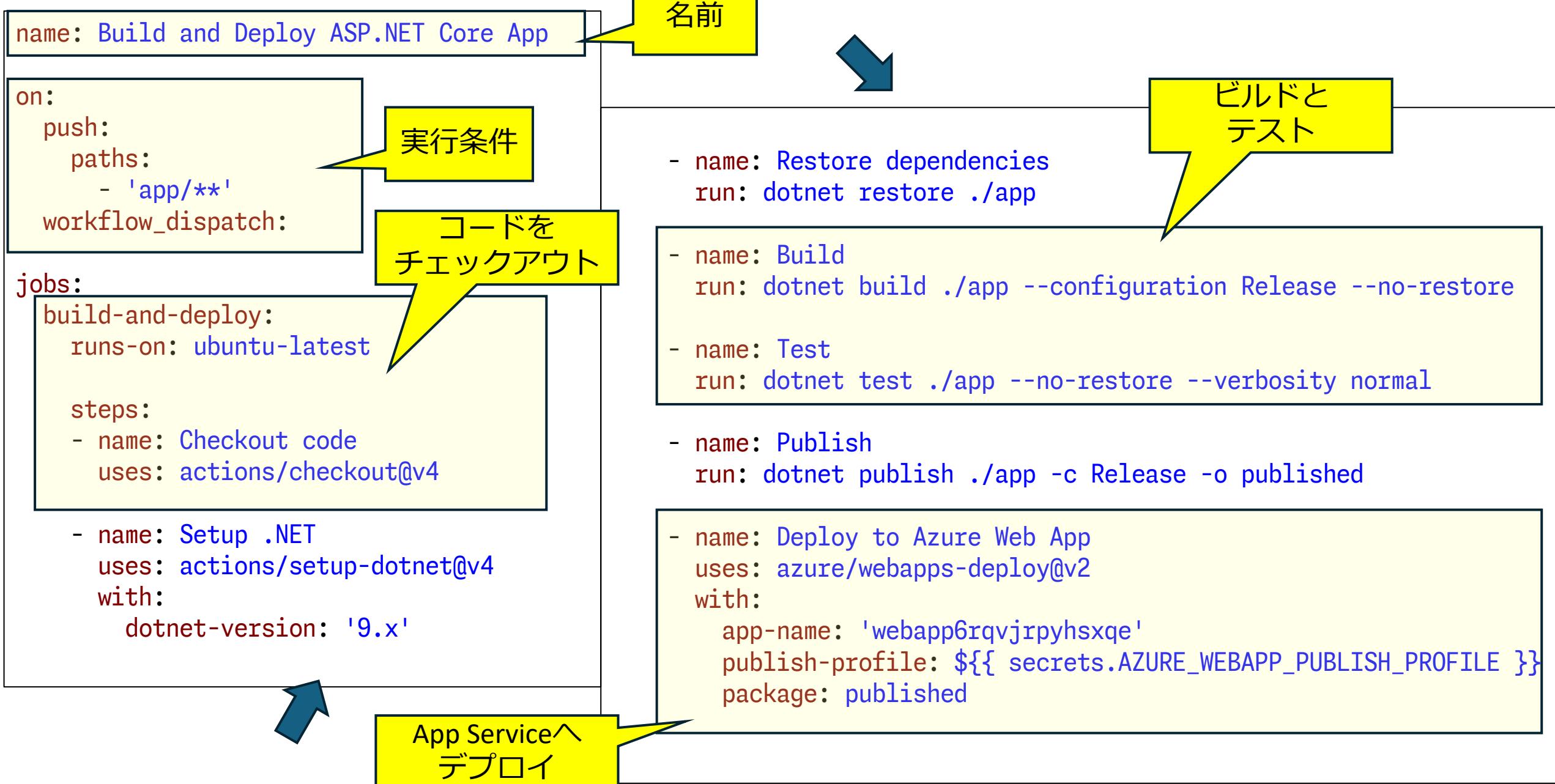
main.bicep

.gitignore

az2008sample.sln

Webアプリの  
コードが変更されたら  
App Serviceに  
Webアプリをデプロイする  
ワークフロー

# Webアプリのビルト・テスト・デプロイを行う GitHub Actions ワークフロー



# モジュール4 CI/CD, IaC

- デプロイ
- リリース
- 繼続的デリバリー
- 繼続的デプロイ
- Infrastructure as Code (IaC)
  - AzureのIaC: ARMテンプレートとBicep
- Azure App Service
- GitHub Actionsによる自動化
  - AzureへのWebアプリのデプロイ
- よくあるご質問
- まとめ

# よくあるご質問

- GitHub Actions のワークフローのYAMLを書いてGitHub上で実際に動かそうとすると、書き方を間違えてしまいエラーが起きる。なんとかうまくデバッグできないでしょうか？
- →実は GitHub Actions をローカルの開発環境で動かすツールがあります。[nektos/act: Run your GitHub Actions locally](#) 
- これを使用すると、開発した GitHub Actions ワークフローをいったんローカルで実行して動作確認をすることができますので、開発が少し楽になります。

# モジュール4 CI/CD, IaC

- デプロイ
- リリース
- 繼続的デリバリー
- 繼続的デプロイ
- Infrastructure as Code (IaC)
  - AzureのIaC: ARMテンプレートとBicep
- Azure App Service
- GitHub Actionsによる自動化
  - AzureへのWebアプリのデプロイ
- よくあるご質問
- まとめ

# モジュール4まとめ

|                              |   |
|------------------------------|---|
| デプロイ                         | コードやアプリケーションを本番環境のサーバー等に配置すること。インプレースデプロイ（一括デプロイ）、ローリングアップデート、ブルーグリーンデプロイなどの戦略（手法）がある。  |
| リリース                         | デプロイされた新機能・変更を、ユーザーに対して有効化すること。「機能フラグ」や「段階的露出」などの戦略（手法）がある。   |
| 継続的デリバリー                     | コードの変更が発生すると、自動的に実稼働環境へのリリースの準備が実行されるしくみ。リリースのタイミングは人が判断し、手動で開始される。   |
| 継続的デプロイ                      | デプロイとリリースをすべて自動化する方法。特定のテストの成功といった条件により自動でリリースを行う。  |
| Infrastructure as Code (IaC) | インフラ（特にクラウドの仮想マシンなどのリソース）をスクリプトや定義ファイルで定義すること。ARM TemplateやBicepなどが使用される。定義の再利用、デプロイの自動化、変更管理、ダイナミックなリソースの管理が可能となる。   |
| Azure App Service            | AzureでWebアプリを運用するためのSaaS。App Serviceプランの中でApp Serviceアプリが稼働。App Serviceアプリに、開発したWebアプリをデプロイできる。   |
| GitHub Actions               | GitHubリポジトリの中で使えるCI/CDツール。YAML形式で「ワークフロー」を定義する。たとえば、リポジトリのWebアプリのコードをチェックアウトし（コードを「ランナー」へとコピーし）、ビルドし、テストを実行し、ビルドしたWebアプリのバイナリをAzure App Serviceへデプロイする、といった一連の動作を定義できる。 |

# 講義



AZ-2008  
DevOpsの基礎:  
中心となる原則と実践

DevOps foundations: The core principles and practices

- ・モジュール1 DevOps の概要
- ・モジュール2 DevOps を使用した計画 (Plan with DevOps)
- ・モジュール3 DevOps を使用した開発 (Develop with DevOps)
- ・モジュール4 DevOps を使用した提供 (Deliver with DevOps)
- ・モジュール5 DevOps を使用した操作 (Operate with DevOps)
- ・まとめ

オペレーション

# モジュール5



運用、監視、改善、SRE、  
プラットフォームエンジニアリング

オペレーショナル・エクセレンスは、オペレーション力、すなわち現場力が卓越し、競争上の優位性にまで高められている状態のこと。

## DevOps を使用した操作 (Operate with DevOps)

組織で DevOps を使用して、オペレーショナルエクセレンスを実現し、開発者エクスペリエンスを強化します。シフトライトテスト、パフォーマンスとセキュリティのモニタリングによる監視、サイト信頼性エンジニアリング、プラットフォームエンジニアリングを実装すると、運用と開発のプラクティスが向上します。

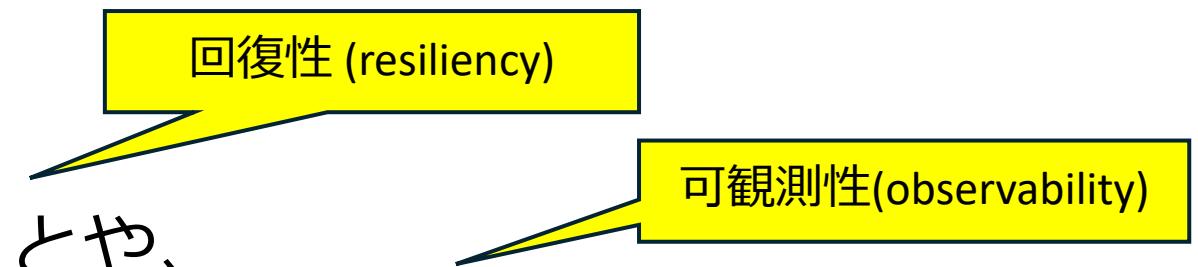
# モジュール5 オペレーション（運用）

- オペレーションエクセレンスとは？
- DevOpsにおけるオペレーションエクセレンスの例
- シフトライトテスト
- カオスエンジニアリング
- Azure Chaos Studio
- Azure Logic Apps
- Azure Traffic Manager
- Azure 上の Web サイトの回復性の強化
- よくあるご質問
- まとめ

# オペレーショナルエクセレンスとは？

- ・組織全体で業務プロセスの最適化を図り、継続的な改善により競争優位を実現すること
- ・「運用卓越性」、「業務遂行の優秀さ」、「業務効率改善」といったように訳すことができる

- ・DevOpsにおいては、システムの耐障害性を高めることや、トラブルを即座に発見できる監視のしくみを構築することが、**オペレーショナルエクセレンスの実現**につながる



# モジュール5 オペレーション（運用）

- ・オペレーションエクセレンスとは？
- ・DevOpsにおけるオペレーションエクセレンスの例
- ・シフトライトテスト
- ・カオスエンジニアリング
- ・Azure Chaos Studio
- ・Azure Logic Apps
- ・Azure Traffic Manager
- ・Azure 上の Web サイトの回復性の強化
- ・よくあるご質問
- ・まとめ

# DevOpsにおける オペレーショナルエクセレンスの例

- ・継続的運用（回復性）
  - ・シフトライトテストの考え方で本番環境での障害注入テストを行う
  - ・Azure Chaos Studioを使用
- ・継続的監視（可観測性）
  - ・パフォーマンスの監視体制を構築する（Azure Monitor等）
  - ・セキュリティの監視体制を構築する（GitHub Advanced Security等）
- ・エンジニアリング（運用技術）
  - ・サイト信頼性エンジニアリング（SRE）
  - ・プラットフォームエンジニアリング

信頼性の高いシステムを実行するための職務、マインドセット、手法

開発者エクスペリエンス向上技術

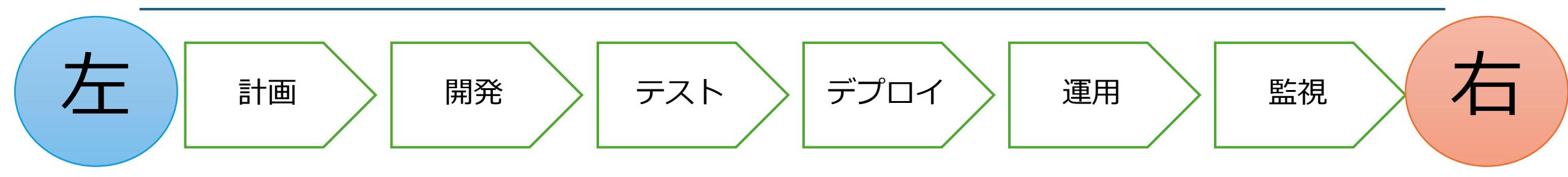
※GitHub Advanced Security は本コースの解説範囲外ですが、  
ご興味のある方は関連コース「GH-500 GitHub Advanced Security」のご受講をぜひご検討ください

# モジュール5 オペレーション（運用）

- ・オペレーションエクセレンスとは？
- ・DevOpsにおけるオペレーションエクセレンスの例
- ・シフトライトテスト
- ・カオスエンジニアリング
- ・Azure Chaos Studio
- ・Azure Logic Apps
- ・Azure Traffic Manager
- ・Azure 上の Web サイトの回復性の強化
- ・よくあるご質問
- ・まとめ

# シフトライトテストとは？

- ・「シフトライトテスト」は、運用の「右側」、つまり本番環境（運用環境）でテストを行う、という考え方
- ・本番環境で障害などが発生しても問題がないようにシステムを設計する
- ・→**カオスエンジニアリング**



# モジュール5 オペレーション（運用）

- ・オペレーションエクセレンスとは？
- ・DevOpsにおけるオペレーションエクセレンスの例
- ・シフトライトテスト
- ・カオスエンジニアリング
- ・Azure Chaos Studio
- ・Azure Logic Apps
- ・Azure Traffic Manager
- ・Azure 上の Web サイトの回復性の強化
- ・よくあるご質問
- ・まとめ

# カオスエンジニアリング

- Netflixが2010年頃に提唱した事例が有名
- Netflixは、世界的な動画ストリーミングサービス
- スケーラブルで複雑なシステムを高可用に保つため、Netflixは**カオスエンジニアリング**を開発
- 各種ツールを使用して、**本番環境で意図的に障害を発生させる**ようにした（障害の注入）
- それでもNetflixサービスが止まらないように冗長化設計を行った

# カオスエンジニアリング

- ・本番環境で意図的に障害を発生（注入）させるためにNetflixが開発したツール
  - ・**Chaos Monkey**: 本番環境のEC2インスタンスをランダムに停止させることで、サービスが他のインスタンスに自動で切り替わるかを検証
  - ・**Latency Monkey**: 通信レイテンシを人工的に挿入。
  - ・**Chaos Gorilla**: 一つのAZ（Availability Zone）全体の停止をシミュレーション。
  - ・**Doctor Monkey**: システムの健全性を監視して不健全なノードを隔離。

NoSQLデータベース  
AzureのCosmos DBに相当

# カオスエンジニアリングの実績

- 2015年9月20日、US-EAST-1リージョンのAmazonのDynamoDBサービスが、問題が発生して停止。
- これは20以上のAWSサービスに影響を及ぼした。
- その影響により、AWSをインフラとする複数のインターネットサービスが6~8時間にわたってダウンしてしまった
- Netflixのサービスはこの大規模障害の影響を受けたものの、**このようなリージョン単位の大規模障害に備えていたことから、重大な事態には至らずに済んだ**

AWS大規模障害を乗り越えたNetflixが語る「障害発生ツールは変化に対応できる勇気を与えてくれる」 | さくらのナレッジ

# カオスエンジニアリングの考え方

- ・システムの各部で障害が発生しても、システムが全体としては問題なくサービスを提供し続けられるように、システムの冗長化設計を行う
  - Fault Tolerance, FT (耐障害性)
- ・そのうえで、専用のツールを使用して、**実際に、本番環境**で、**計画的に**、システムの各部で障害を発生させ続ける
  - Failure Injection Test (障害注入テスト)
- ・冗長化設計に問題がなければ、システムはサービスを提供し続けることができる。耐障害性が極めて高いシステムとなる。

# モジュール5 オペレーション（運用）

- ・オペレーションエクセレンスとは？
- ・DevOpsにおけるオペレーションエクセレンスの例
- ・シフトライトテスト
- ・カオスエンジニアリング
- ・Azure Chaos Studio
- ・Azure Logic Apps
- ・Azure Traffic Manager
- ・Azure 上の Web サイトの回復性の強化
- ・よくあるご質問
- ・まとめ

# Azure Chaos Studio

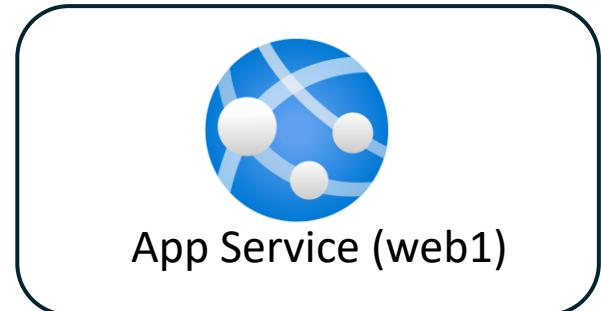
- 2023/11/16 一般提供開始
- 現実世界の障害をシミュレートする障害を**意図的**に発生させることで、アプリケーションの回復力を向上させる実験プラットフォーム。
- **カオスエンジニアリングをAzureで実現できる**
- Azure App Serviceの停止、仮想マシン（VM）の停止、ネットワークの遅延、ストレージの障害、シークレットの期限切れ、データセンターの停止といった現実世界の障害に対するアプリケーションの対応を評価できる

<https://techcommunity.microsoft.com/blog/appsonazureblog/build-resilient-applications-by-simulating-outages-with-azure-chaos-studio/3982260>

# Azure Chaos Studio

- Azureのサービスの障害を**意図的に**発生させることが可能
- 意図的な障害: たとえば App Service を一時的に停止させる、など

リージョンA



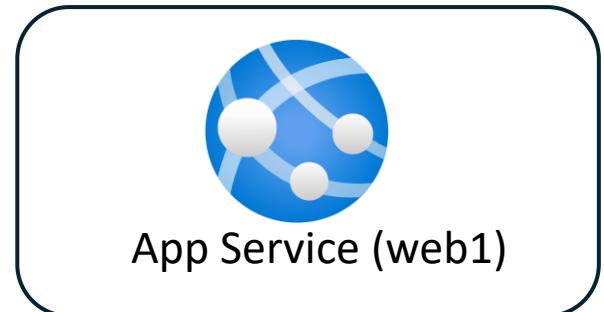
リージョンB



# Azure Chaos Studio

- ・「ターゲット」で、障害を起こすリソースを指定
- ・「実験」で、ターゲットに対しどのように障害を起こすかを指定

リージョンA



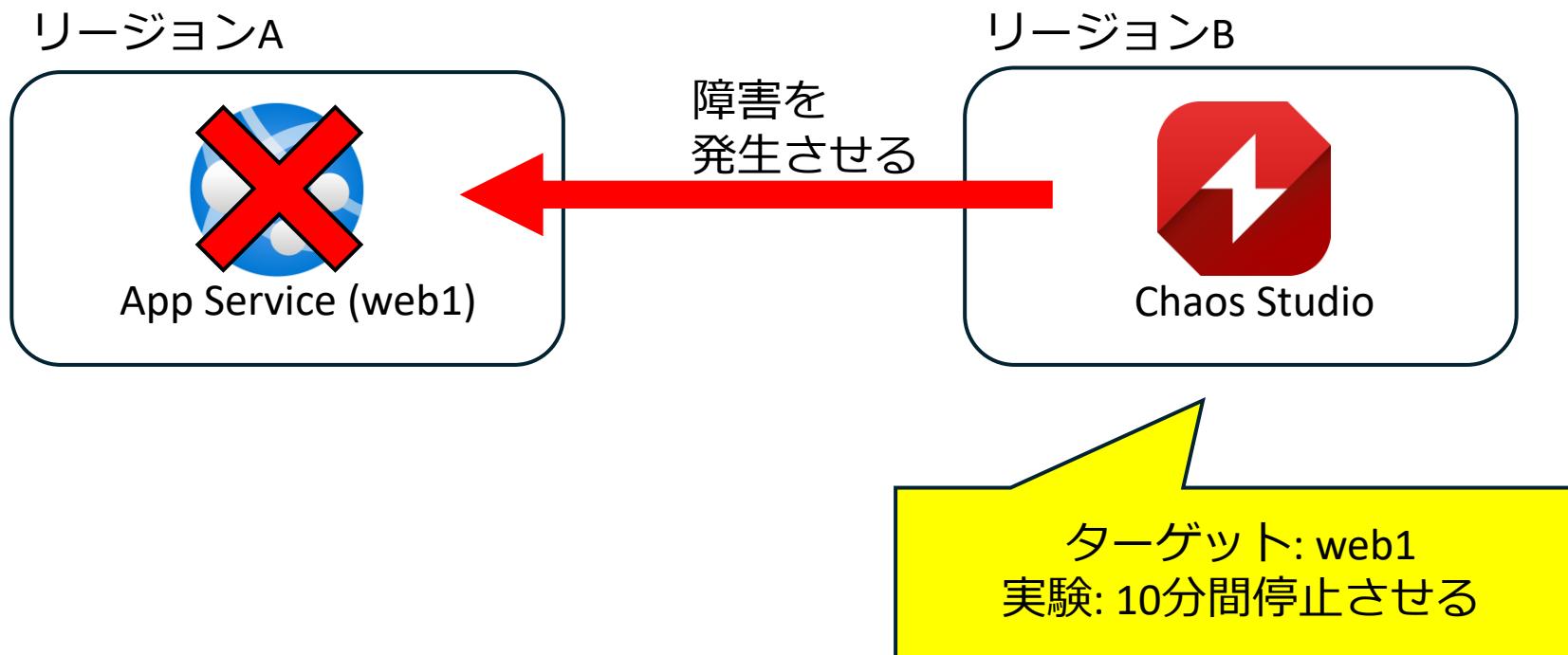
リージョンB



ターゲット: web1  
実験: 10分間停止させる

# Azure Chaos Studio

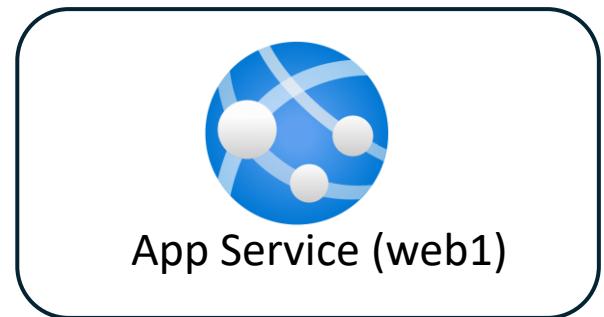
- 「実験」を開始すると、ターゲットで、指定された障害が**実際**に発生する



# Azure Chaos Studio

- 「実験」が終わると、ターゲットは障害から回復する

リージョンA



リージョンB



ターゲット: web1  
実験: 10分間停止させる

Azure Chaos Studioの「ターゲット」の設定画面。  
ここでは、Azure App Serviceをターゲットとして設定している。

The screenshot shows the Microsoft Azure portal interface with the title 'Chaos Studio | 対象'. The 'Targets' tab is selected, indicated by a red box. The main area displays a table of targets:

| 名前         | サブスクリプション  | リソース グループ  | サービス直接 |
|------------|------------|------------|--------|
| [Redacted] | [Redacted] | [Redacted] | 有効     |
| [Redacted] | [Redacted] | [Redacted] | 無効     |
| [Redacted] | [Redacted] | [Redacted] | 無効     |

Additional UI elements include a search bar, a message about subscriptions, and a feedback button.

Azure Chaos Studioの「実験」の設定画面。

ここでは、ターゲットのAzure App Serviceを10分停止させる、という設定を行っている

Microsoft Azure リソース、サービス、ドキュメントの検索 (G+/)

ホーム > Chaos Studio | 実験 > exp1 >

## 実験の編集

以下で実験を構成します。 詳細情報 □

Step 1  
1 個の分岐, 1 個のアクション, 1 個のターゲット

...  
...

ステップ \* Step 1  
分岐 \* Branch 1

| アクション            | パラメーター               | ターゲット リ... | スコープ     |
|------------------|----------------------|------------|----------|
| Stop App Service | duration: 10 minutes | 1 リソース     | -<br>... |

+ アクションの追加 ▼

分岐の追加  
ステップの追加

The screenshot shows the Azure Chaos Studio experiment editor interface. At the top, there's a navigation bar with 'Microsoft Azure' and a search bar. Below it, the path 'Home > Chaos Studio | Experiment > exp1 >' is shown. The main title is 'Experiment Configuration'. A sub-section title 'Step 1' is displayed, indicating '1 個の分岐, 1 個のアクション, 1 個のターゲット' (1 branch, 1 action, 1 target). The configuration area shows a 'Step 1' section with a 'Branch 1' branch. Below this, a table lists the action 'Stop App Service' with a duration of '10 minutes', targeting '1 resource', and no scope defined. There are buttons for adding branches ('分岐の追加') and steps ('ステップの追加').

# モジュール5 オペレーション（運用）

- ・オペレーションエクセレンスとは？
- ・DevOpsにおけるオペレーションエクセレンスの例
- ・シフトライトテスト
- ・カオスエンジニアリング
- ・Azure Chaos Studio
- ・Azure Logic Apps
- ・Azure Traffic Manager
- ・Azure 上の Web サイトの回復性の強化
- ・よくあるご質問
- ・まとめ

# Azure Logic Apps

- ・さまざまなサービスの接続・統合、運用の自動化などを行うためのサービス
- ・画面上でフローチャートのようなもの（ワークフロー）を作成することで処理内容を定義



Microsoft Azure Search resources, services, and docs (G+)

Home > MonitorRecordUpdates | Logic apps designer

Logic app

Save Discard Run Designer Code view Parameters Templates Connectors Help Info

When a record is updated

Transform XML

Register in Datamart

If record requires action

True

Create a new record

Slack Post message

False

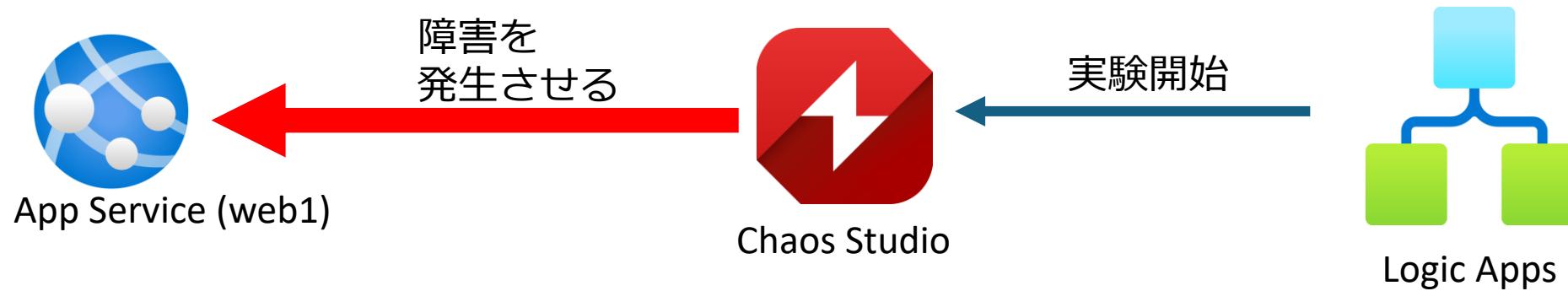
Execute stored procedure

Send email

+ New step

# Azure Logic Apps

- ・スケジュールを設定して、定期的にワークフローを実行することができる
- ・たとえば定期的にChaos Studioの「実験」を開始する、といった自動化が可能

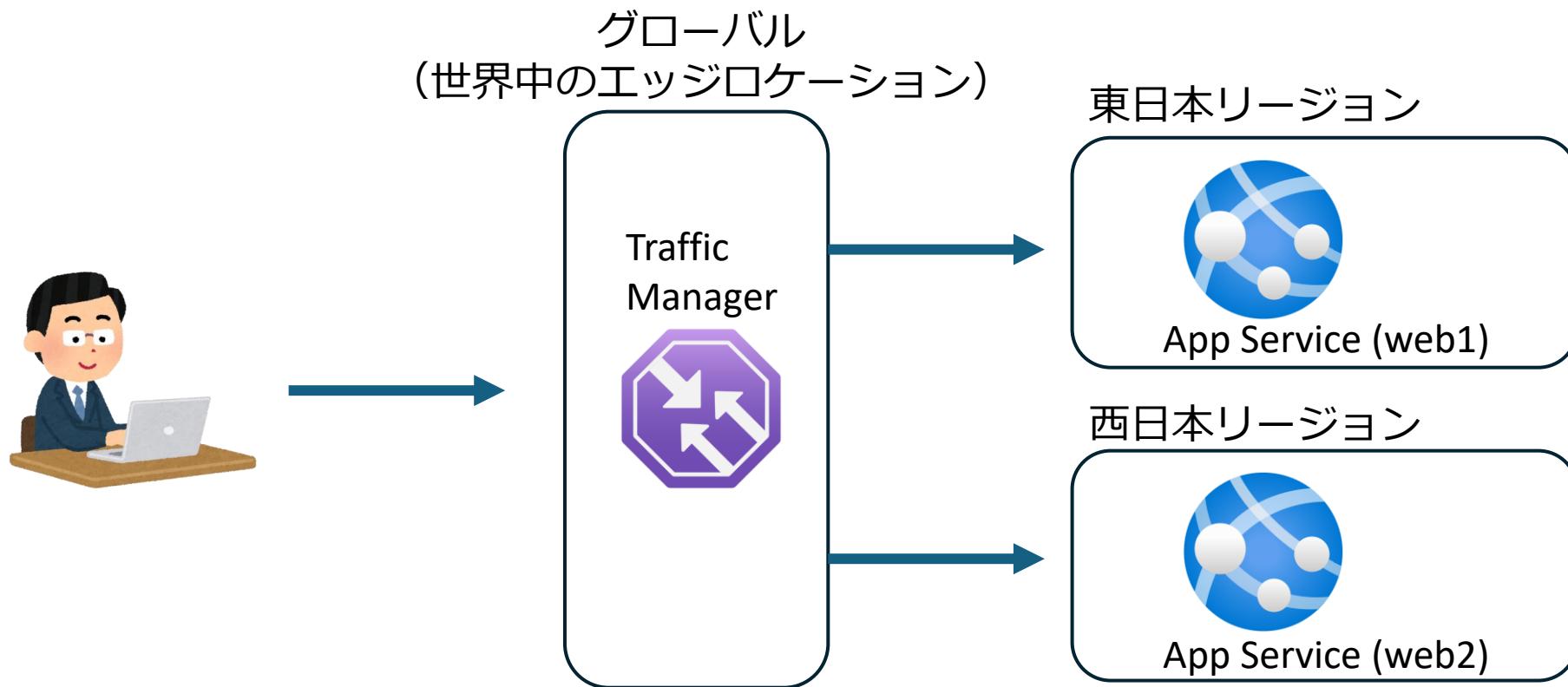


# モジュール5 オペレーション（運用）

- ・オペレーションエクセレンスとは？
- ・DevOpsにおけるオペレーションエクセレンスの例
- ・シフトライトテスト
- ・カオスエンジニアリング
- ・Azure Chaos Studio
- ・Azure Logic Apps
- ・Azure Traffic Manager
- ・Azure 上の Web サイトの回復性の強化
- ・よくあるご質問
- ・まとめ

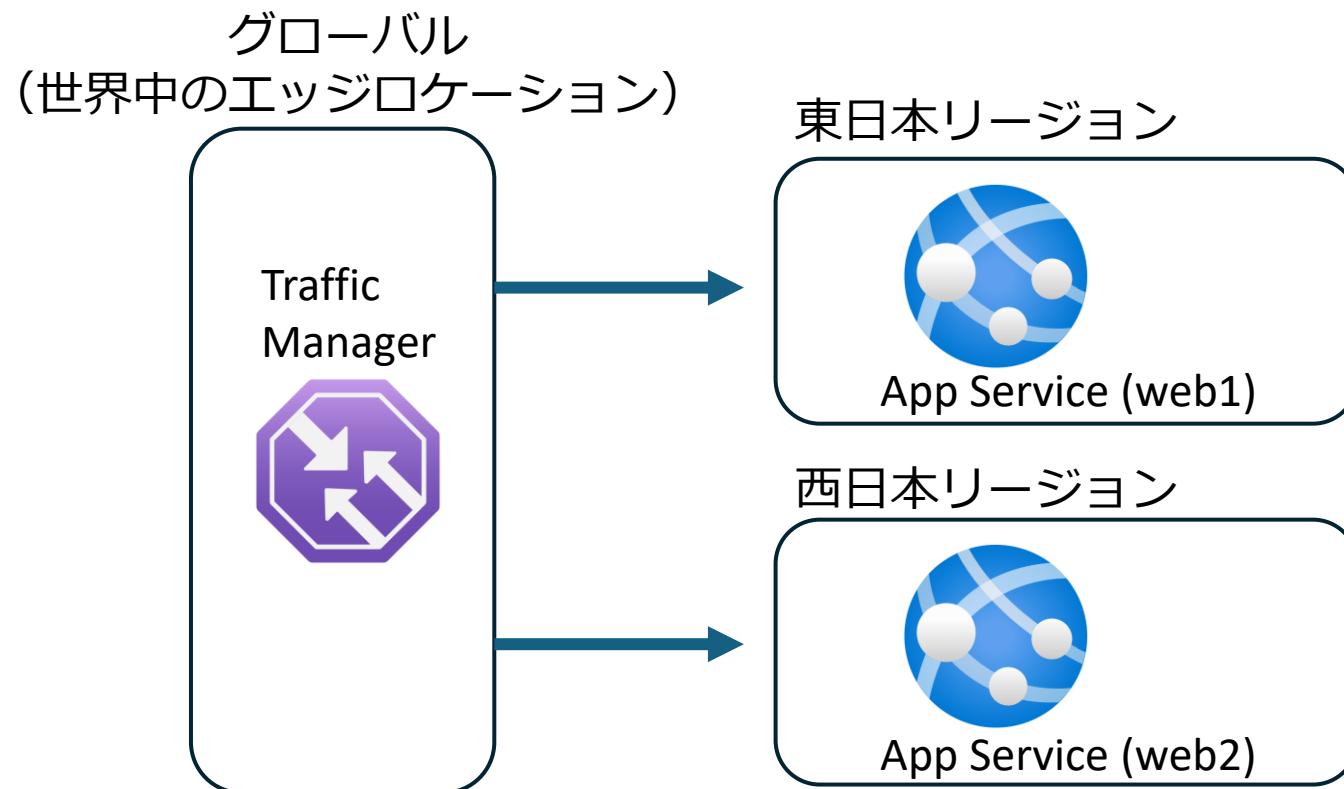
# Azure Traffic Manager

- ・ロードバランサー（負荷分散）サービスの一種
- ・複数のリージョンに対する負荷分散を行える



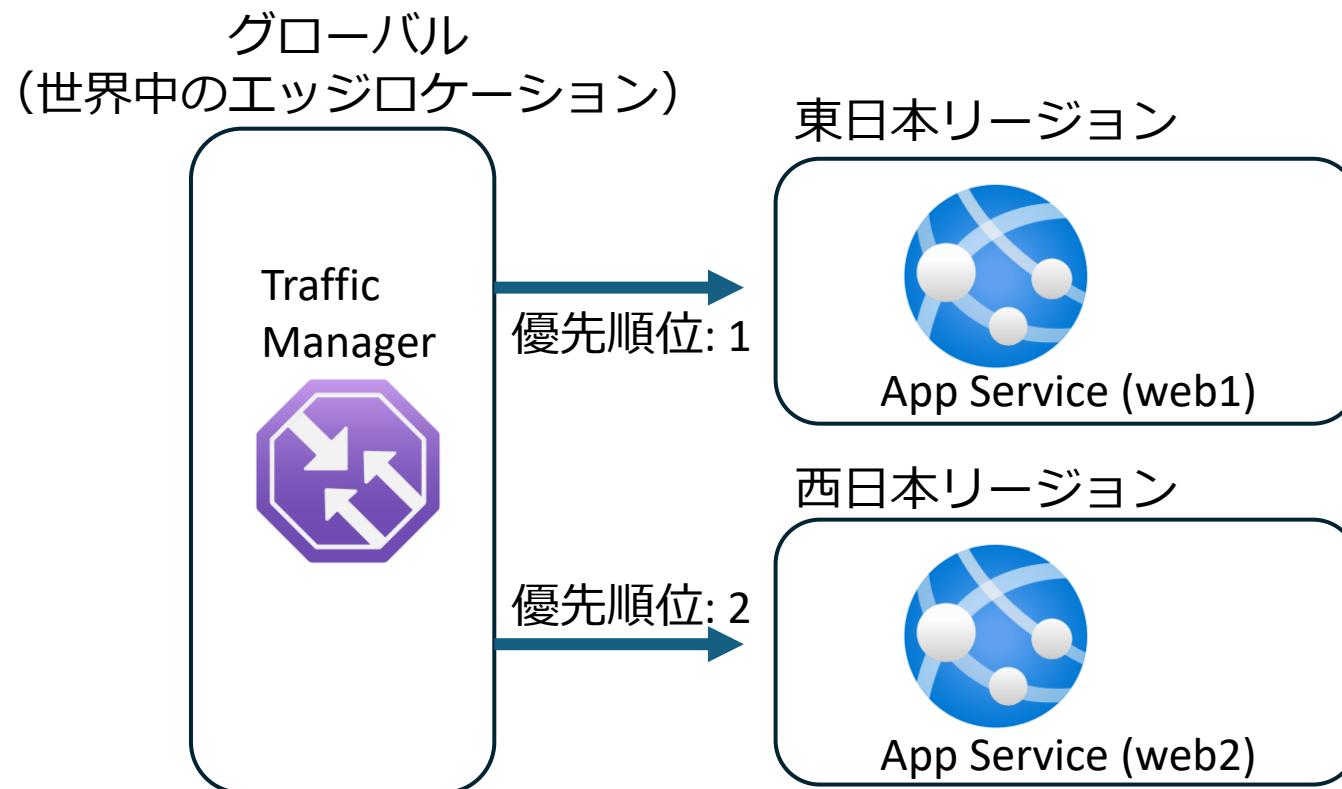
# Azure Traffic Manager

- 世界中の「エッジロケーション」を組み合わせて稼働するサービスであるため、特定リージョンの障害の影響を受けることがない



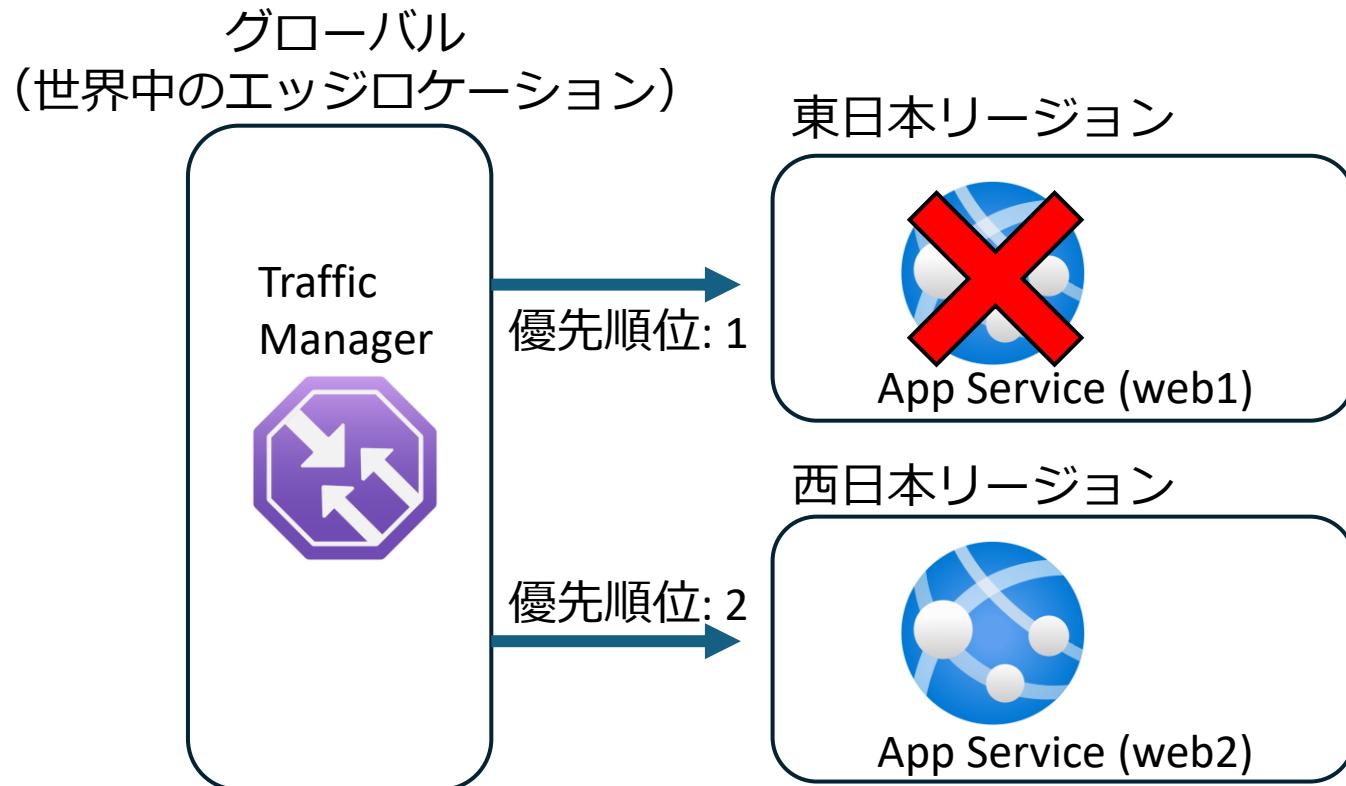
# Azure Traffic Manager

- ・さまざまなルーティングアルゴリズムを使用できる
- ・たとえば「優先順位」アルゴリズムを使用する場合、トラフィックは常に優先順位1のターゲットへと誘導される



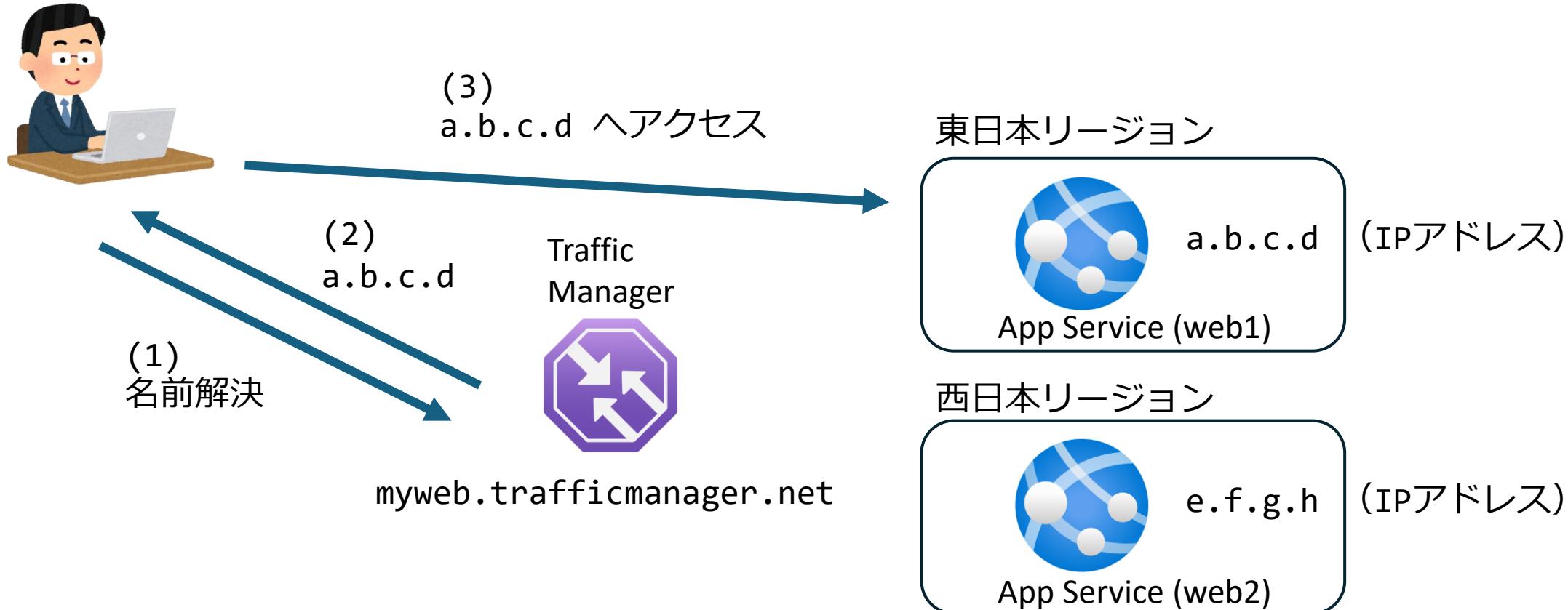
# Azure Traffic Manager

- ただし、優先順位1のターゲットに障害が発生した場合は、トラフィックは優先順位2のターゲットへと誘導される
- このようなしくみで、全体として高い可用性のサービスを運用できる



# Azure Traffic Manager

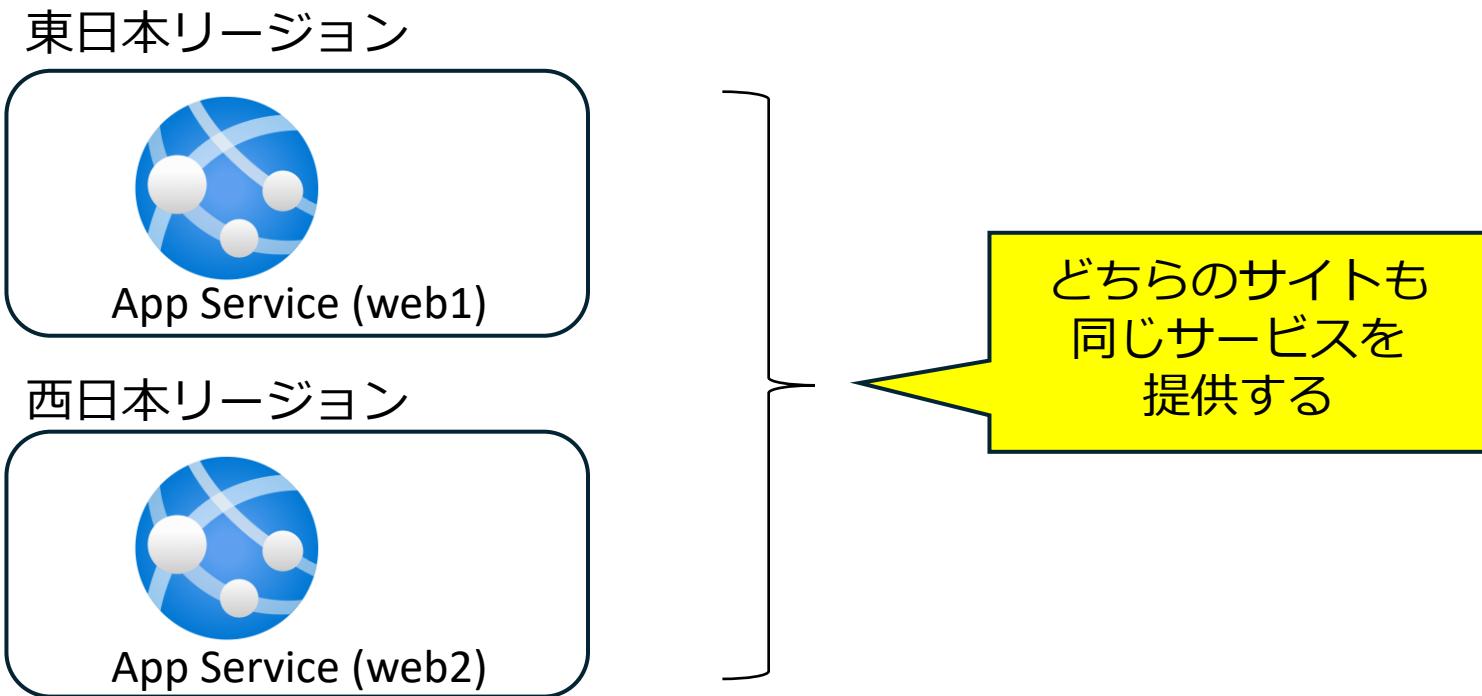
- DNSベースでトラフィックを誘導
- (※以下の図は実際の名前解決の様子を表す)



# モジュール5 オペレーション（運用）

- ・オペレーションエクセレンスとは？
- ・DevOpsにおけるオペレーションエクセレンスの例
- ・シフトライトテスト
- ・カオスエンジニアリング
- ・Azure Chaos Studio
- ・Azure Logic Apps
- ・Azure Traffic Manager
- ・Azure 上の Web サイトの回復性の強化
- ・よくあるご質問
- ・まとめ

# Azure上のWebサイトの回復性の強化



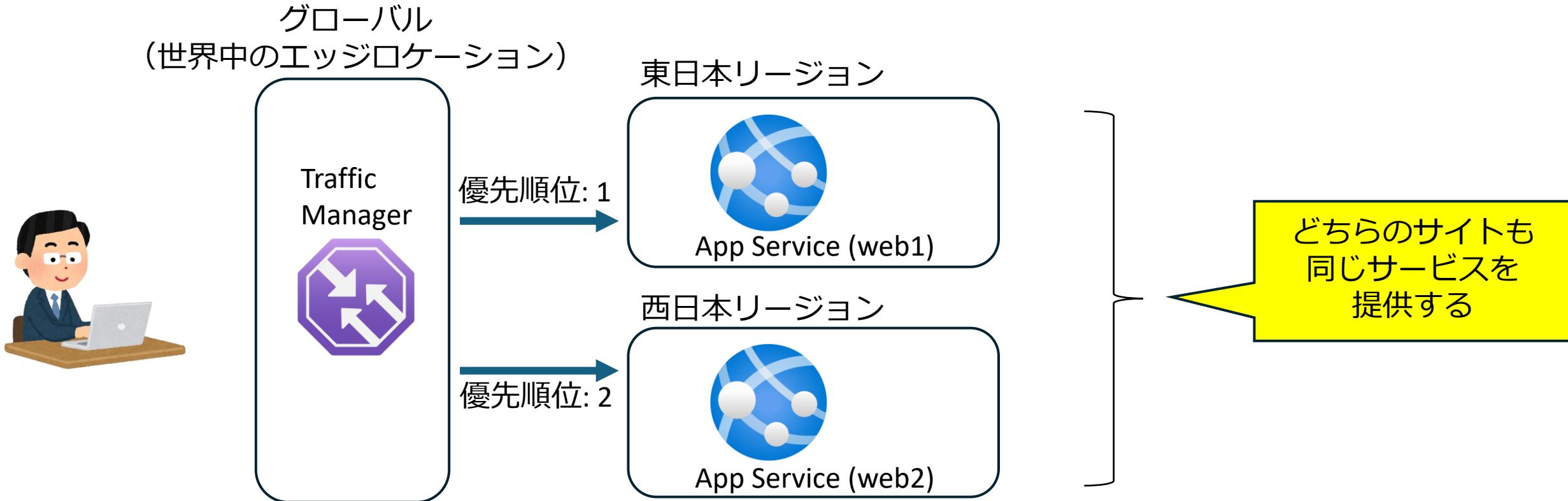
2つのAzure App Serviceをデプロイ。  
この2つはどちらも同じサービスを提供する。

# Azure上のWebサイトの回復性の強化

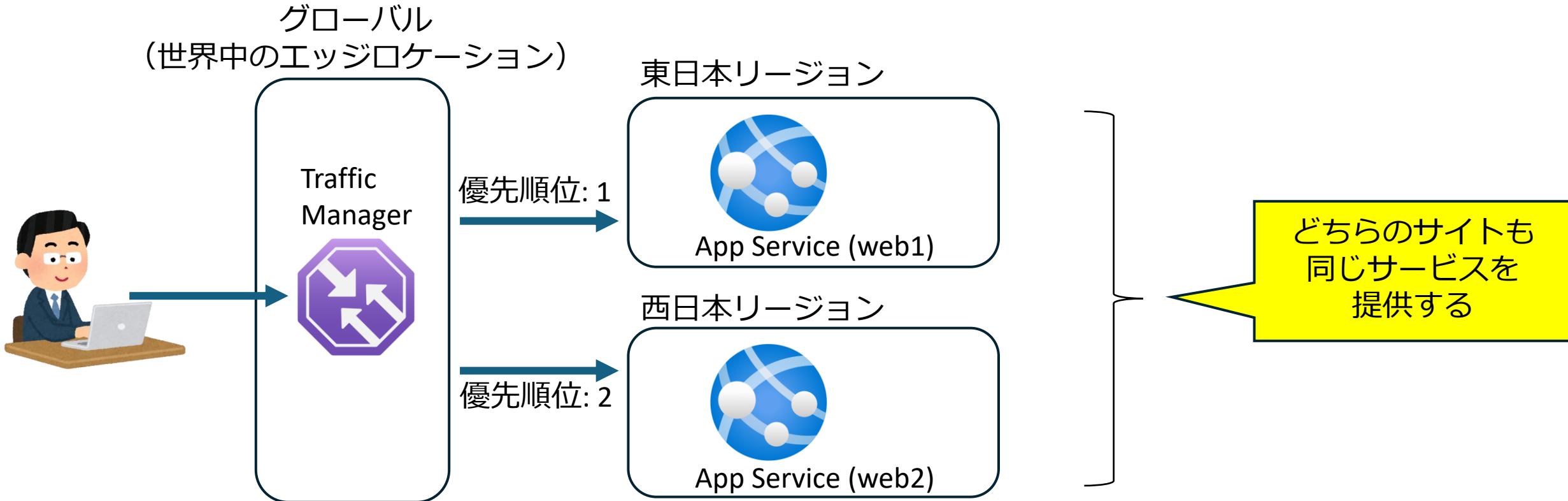


Azure Traffic Managerをデプロイ

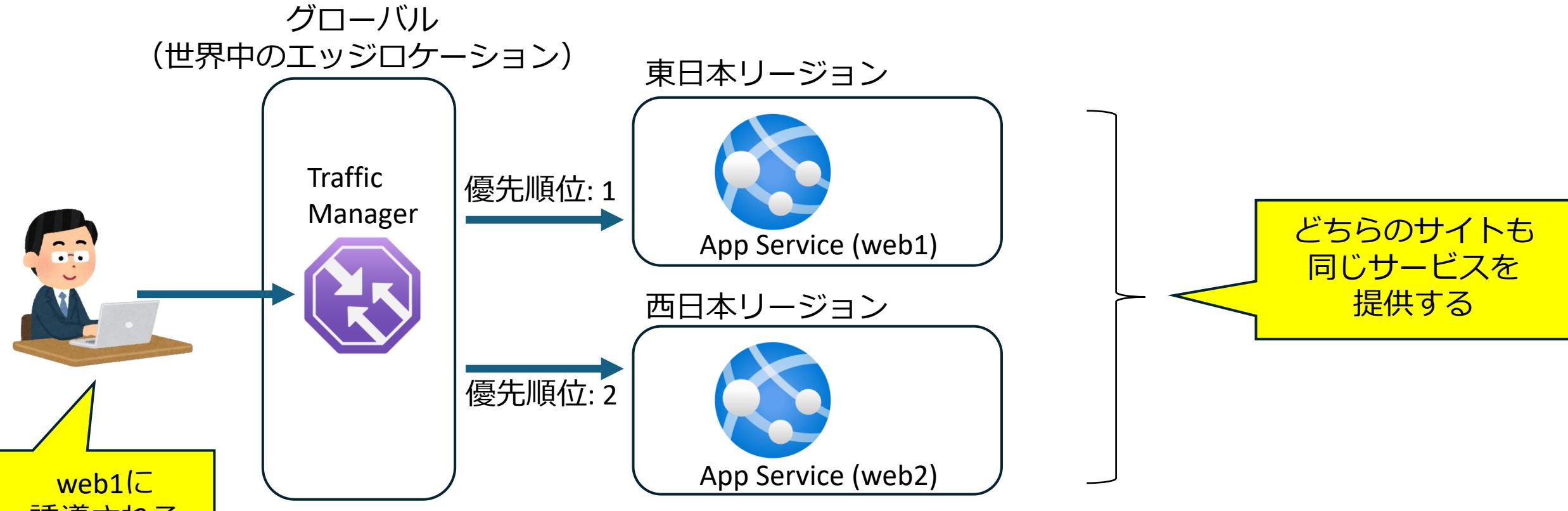
# Azure上のWebサイトの回復性の強化



# Azure上のWebサイトの回復性の強化

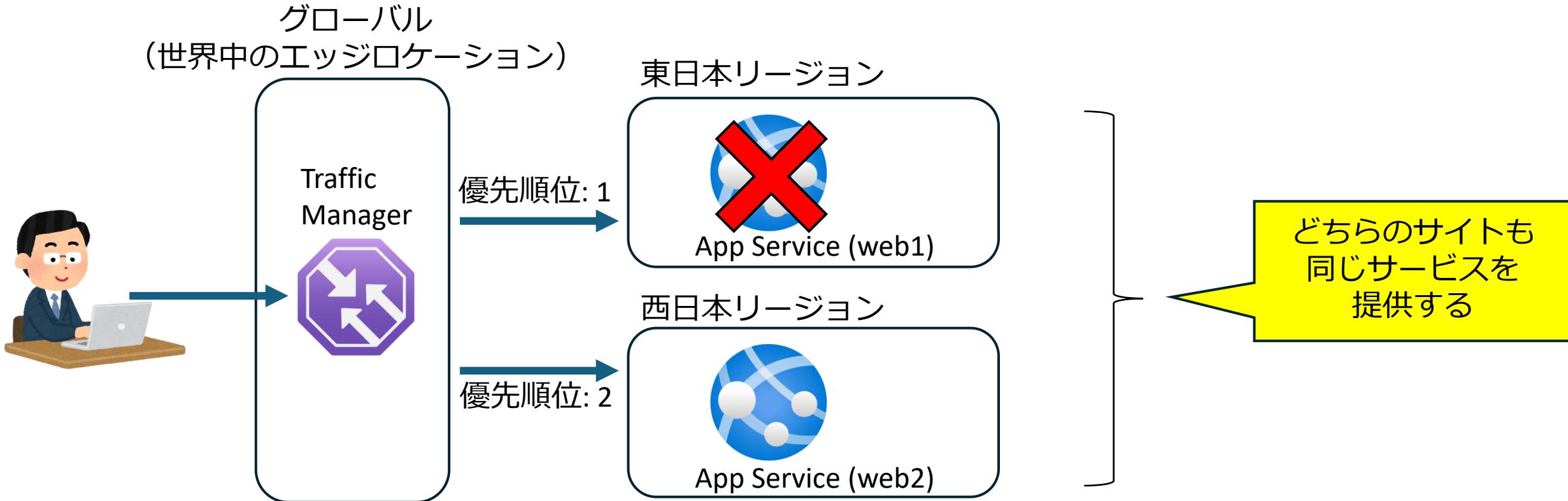


# Azure上のWebサイトの回復性の強化



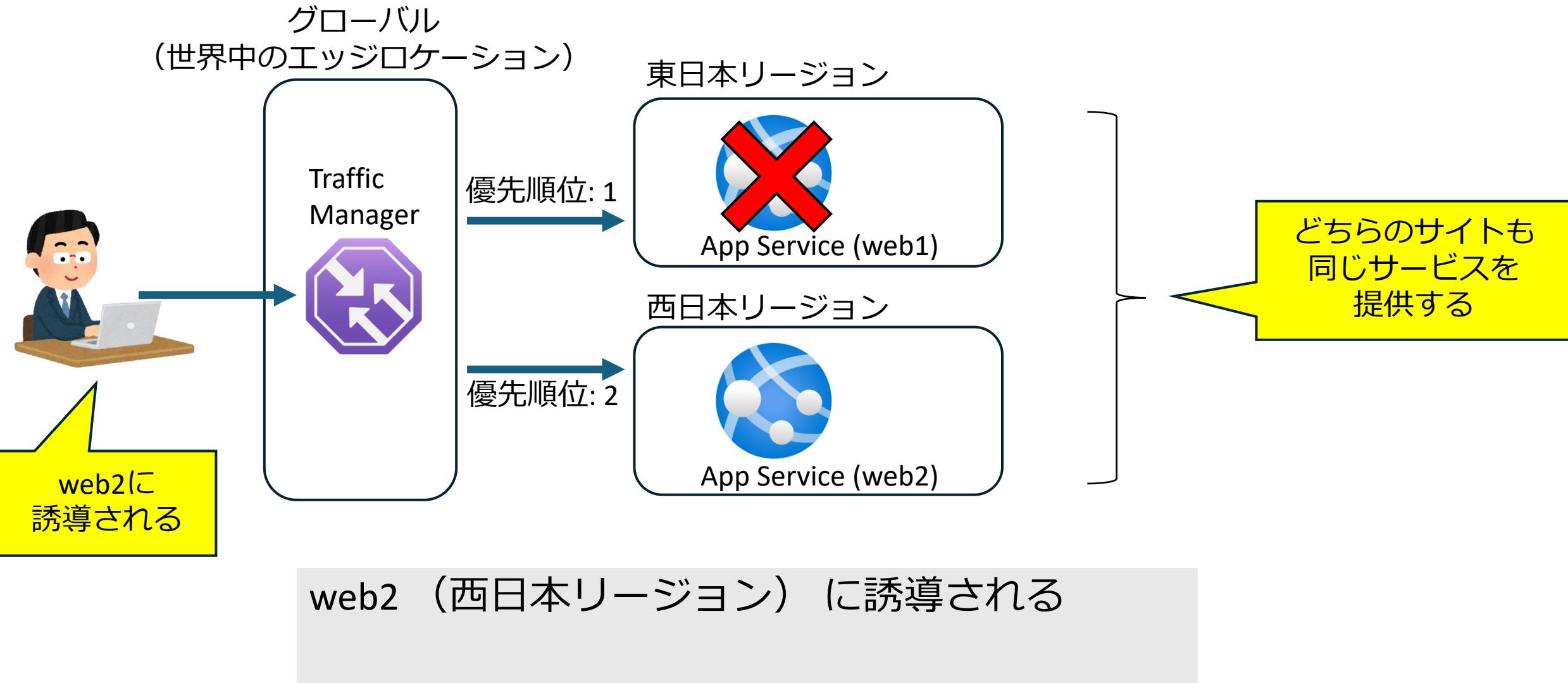
web1 (東日本リージョン) に誘導される

# Azure上のWebサイトの回復性の強化

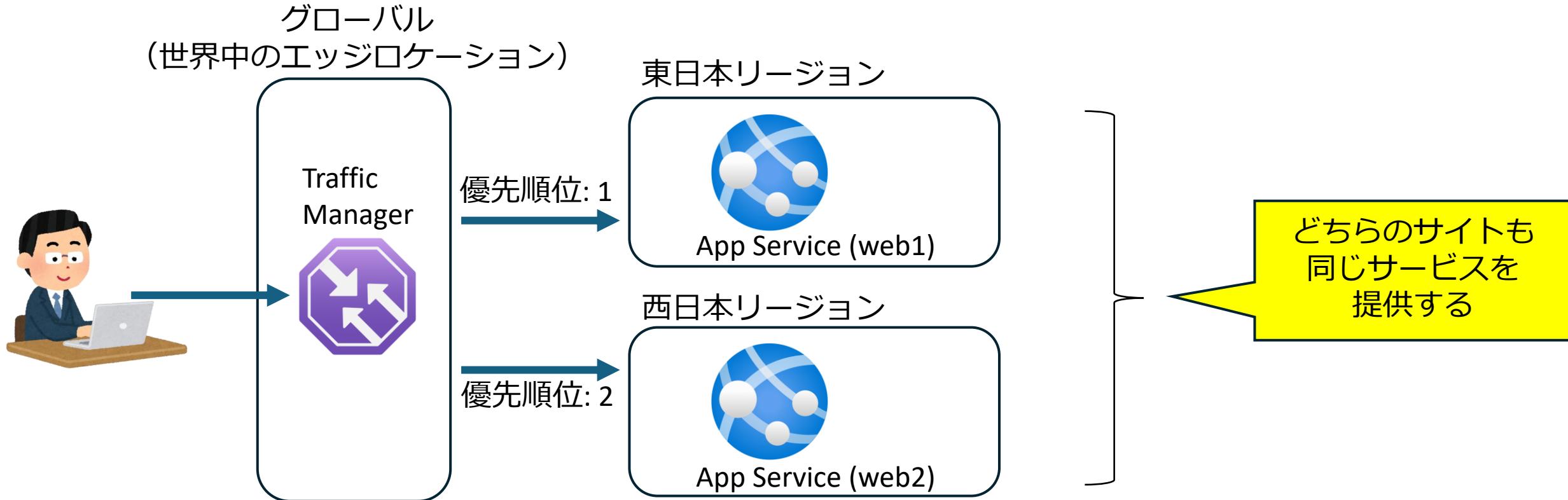


もし、 web1 に障害が発生した場合は・・・

# Azure上のWebサイトの回復性の強化

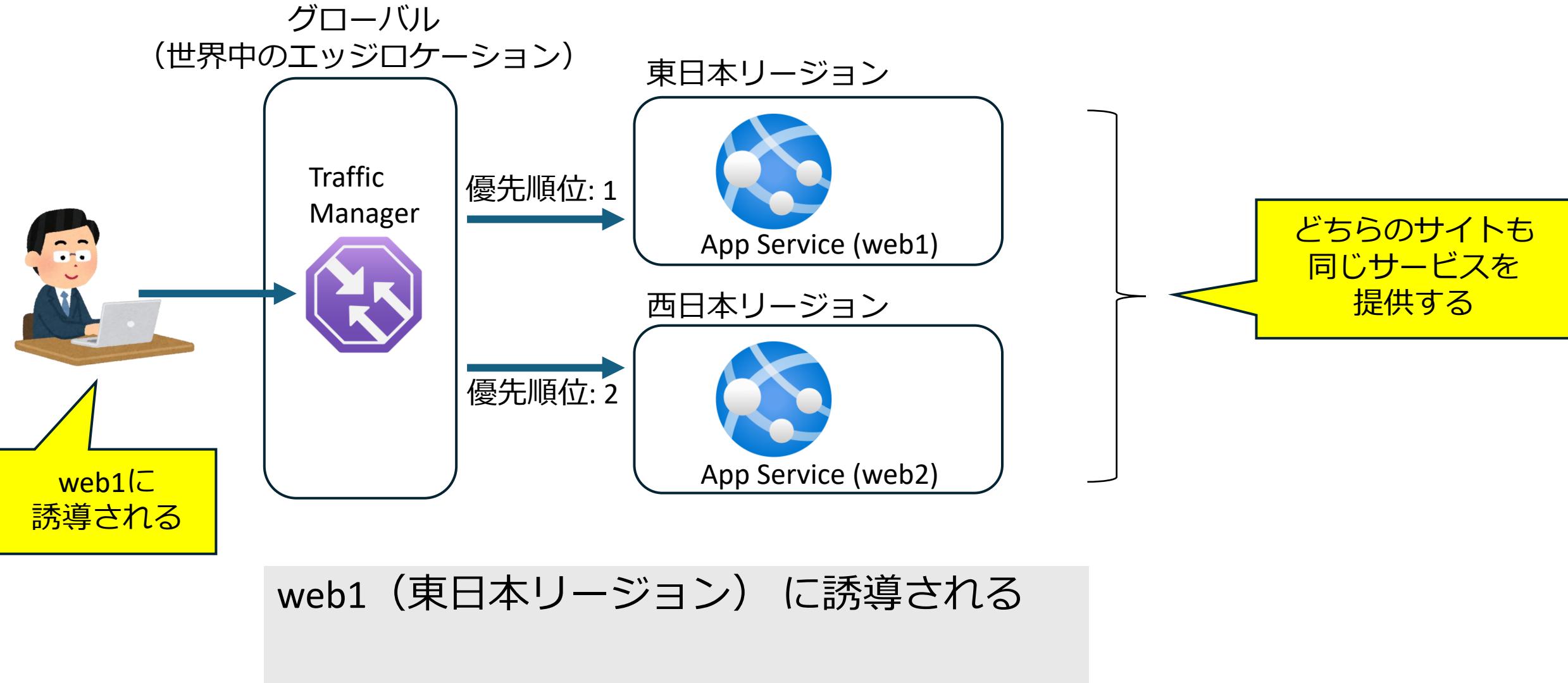


# Azure上のWebサイトの回復性の強化

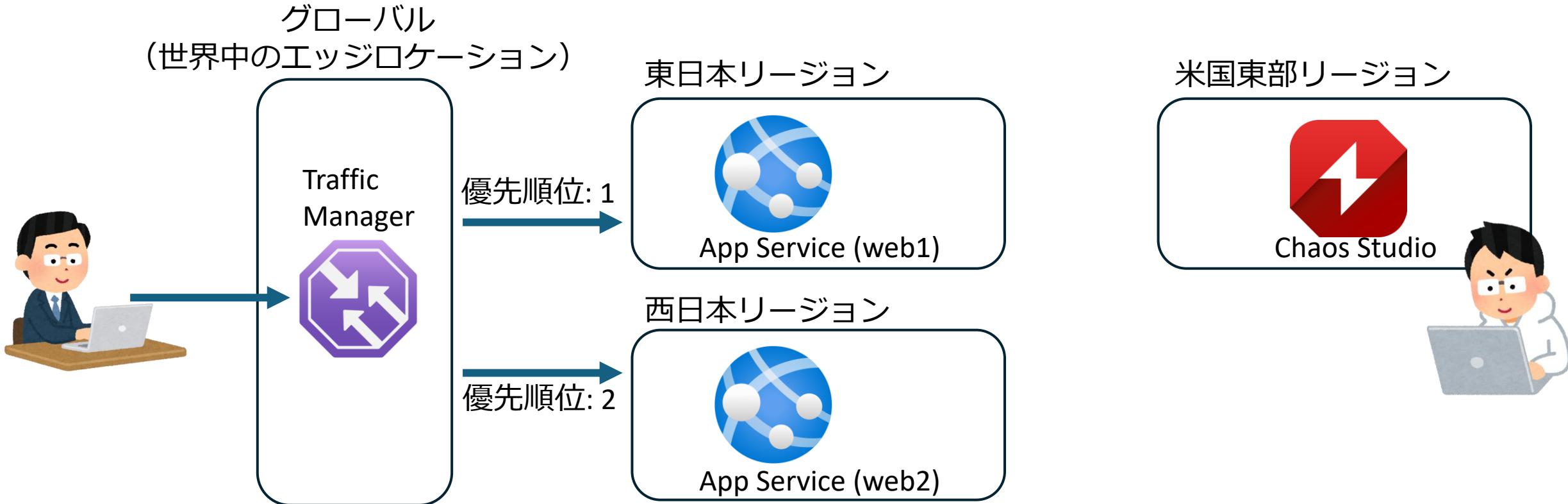


web1 の障害が解消されると・・・

# Azure上のWebサイトの回復性の強化

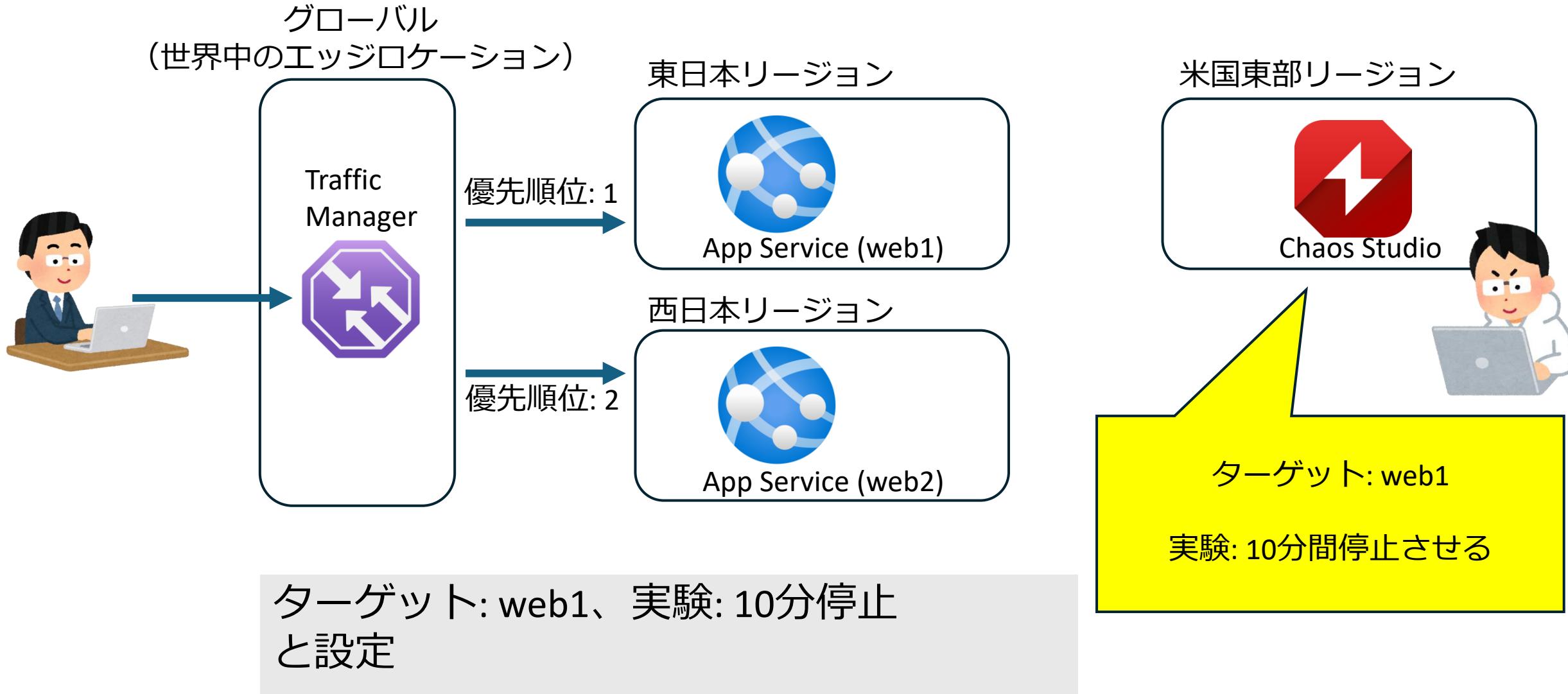


# Azure上のWebサイトの回復性の強化

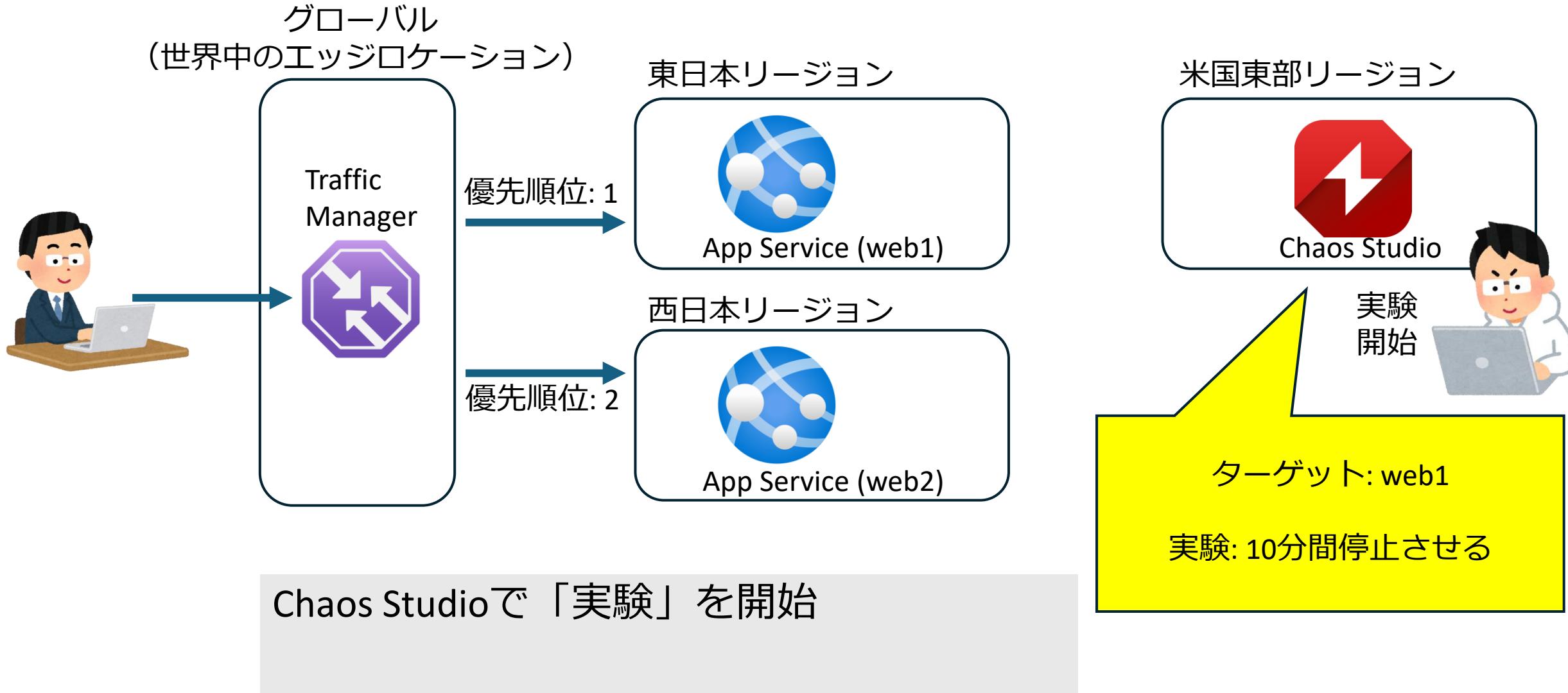


このような障害を意図的に発生させるため  
Azure Chaos Studio をデプロイ

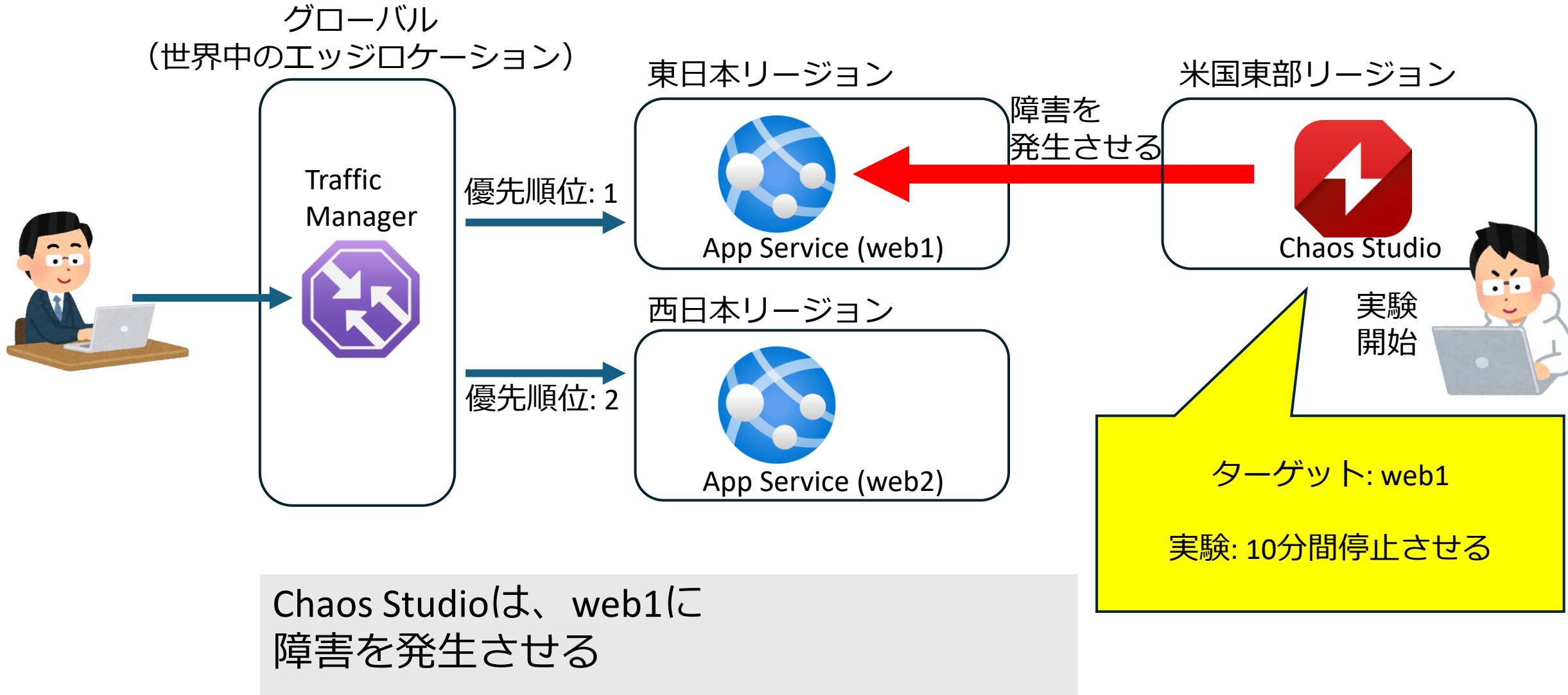
# Azure上のWebサイトの回復性の強化



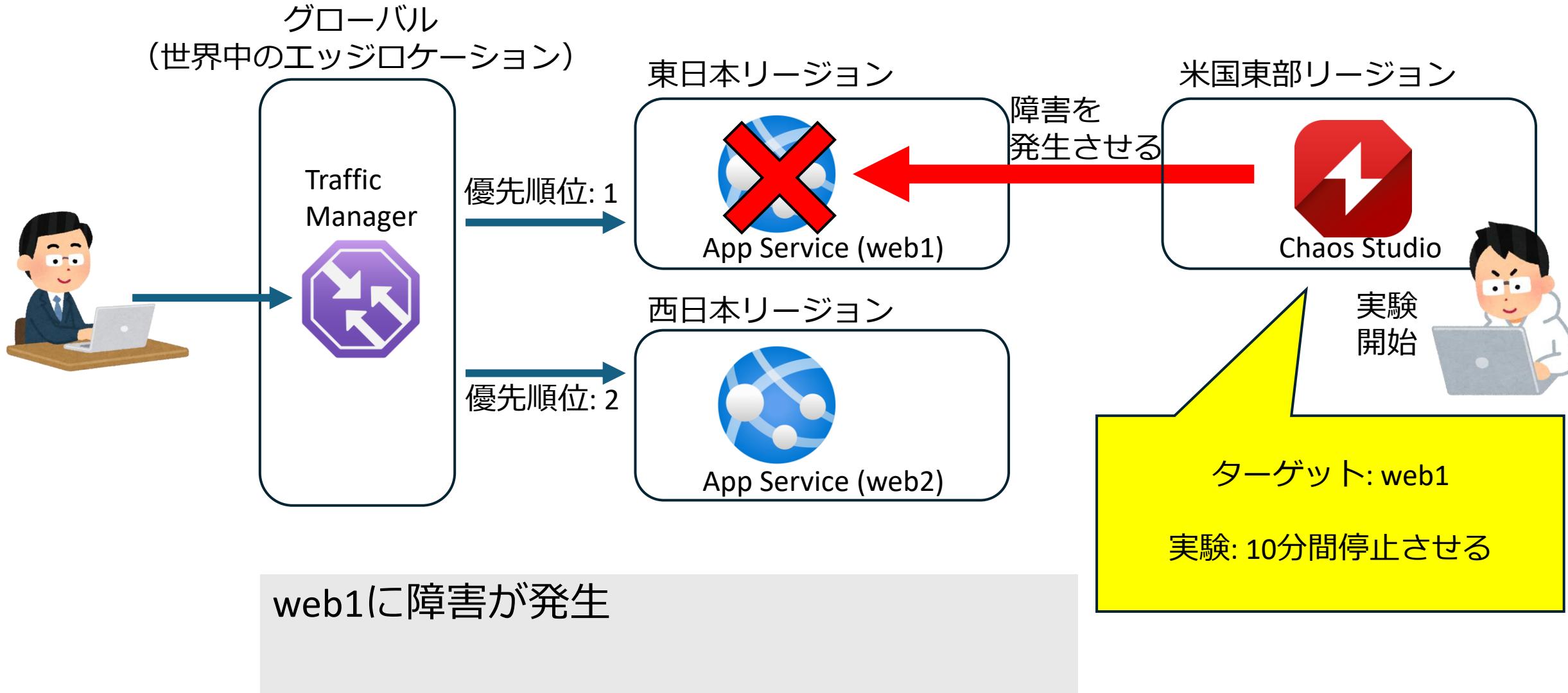
# Azure上のWebサイトの回復性の強化



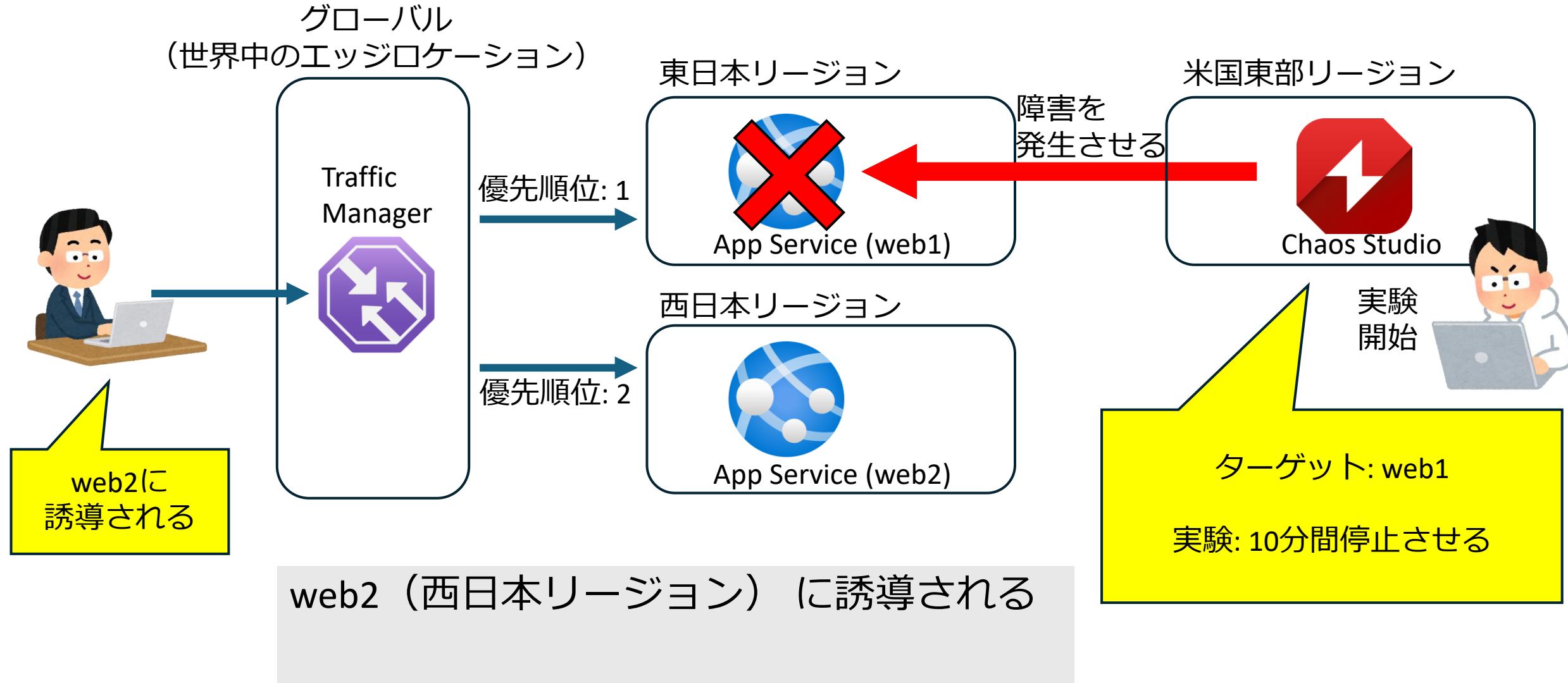
# Azure上のWebサイトの回復性の強化



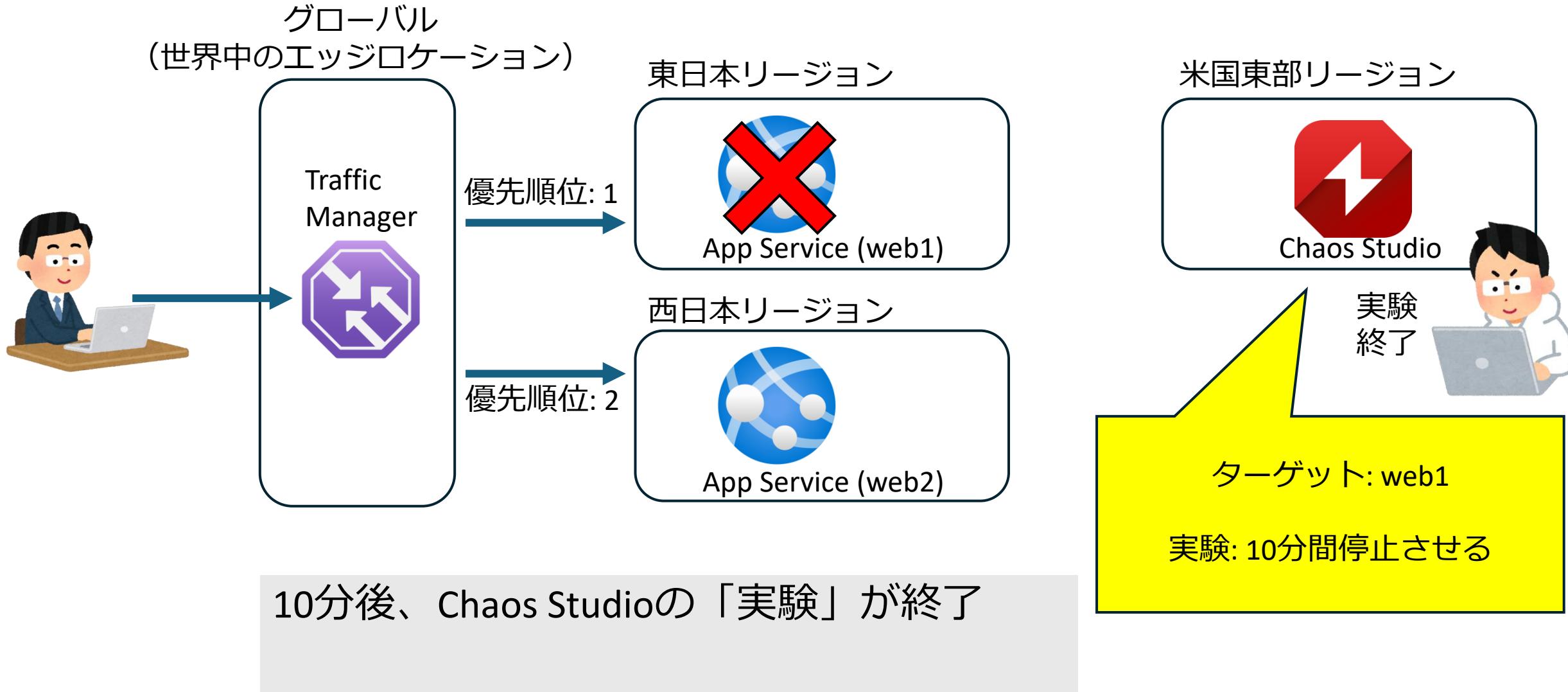
# Azure上のWebサイトの回復性の強化



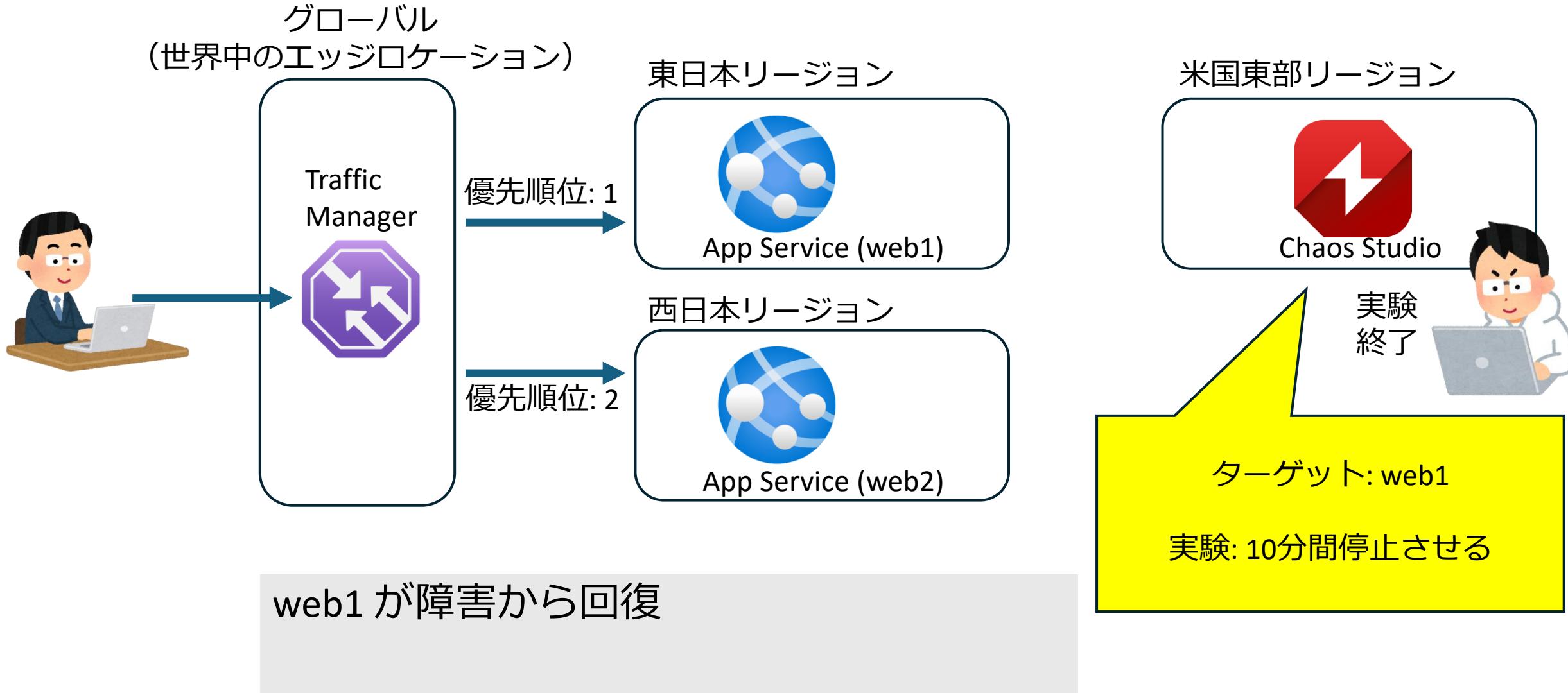
# Azure上のWebサイトの回復性の強化



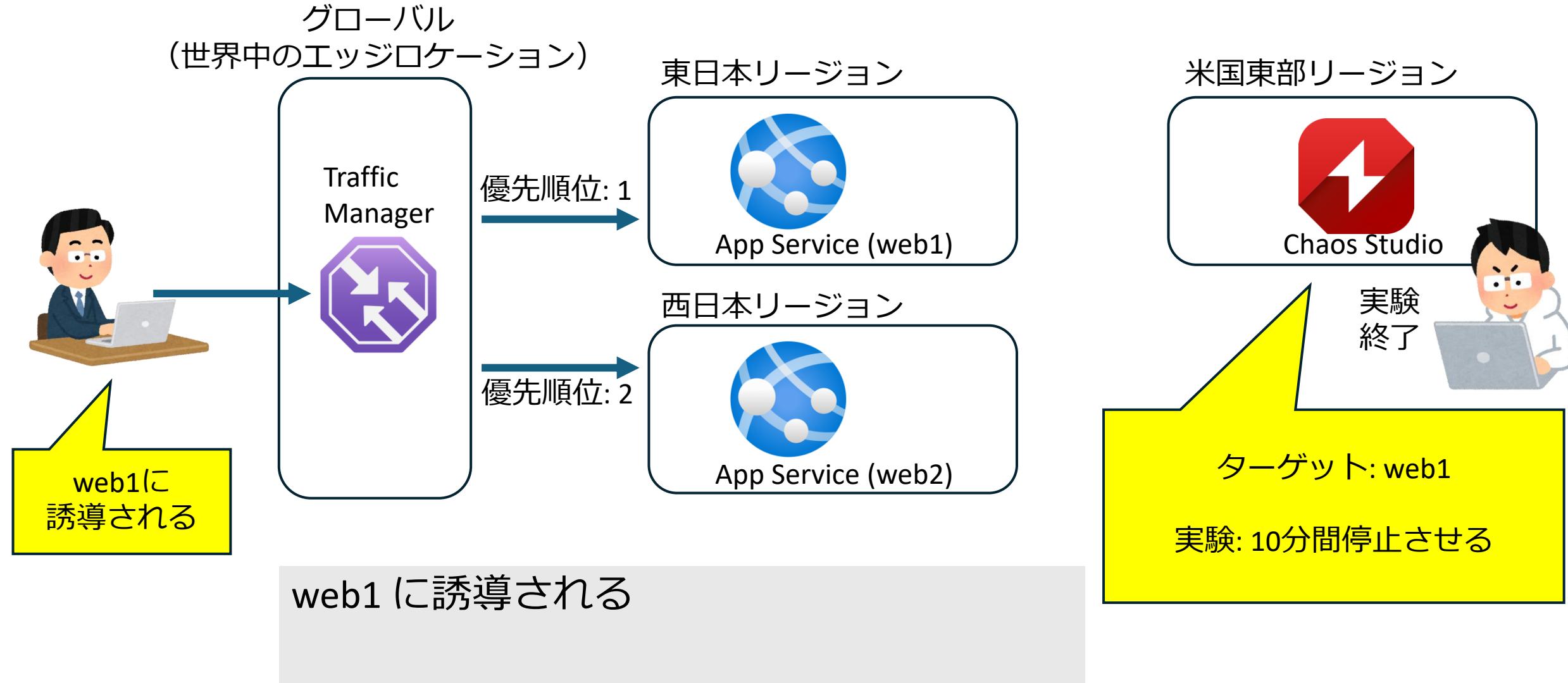
# Azure上のWebサイトの回復性の強化



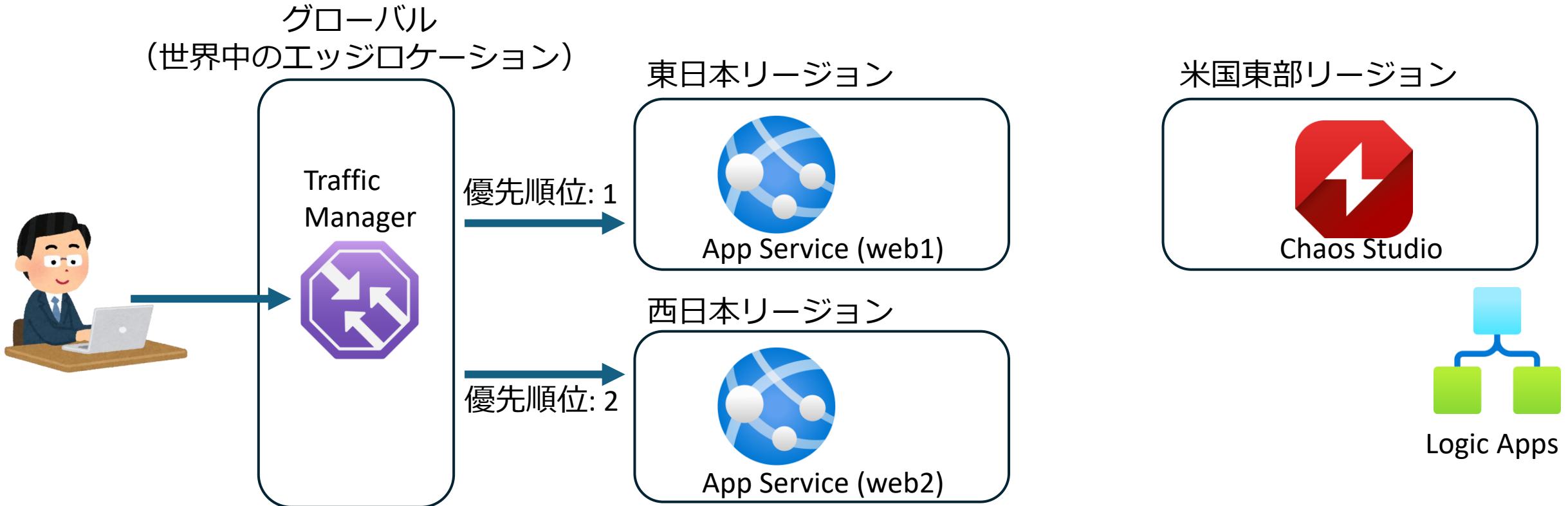
# Azure上のWebサイトの回復性の強化



# Azure上のWebサイトの回復性の強化

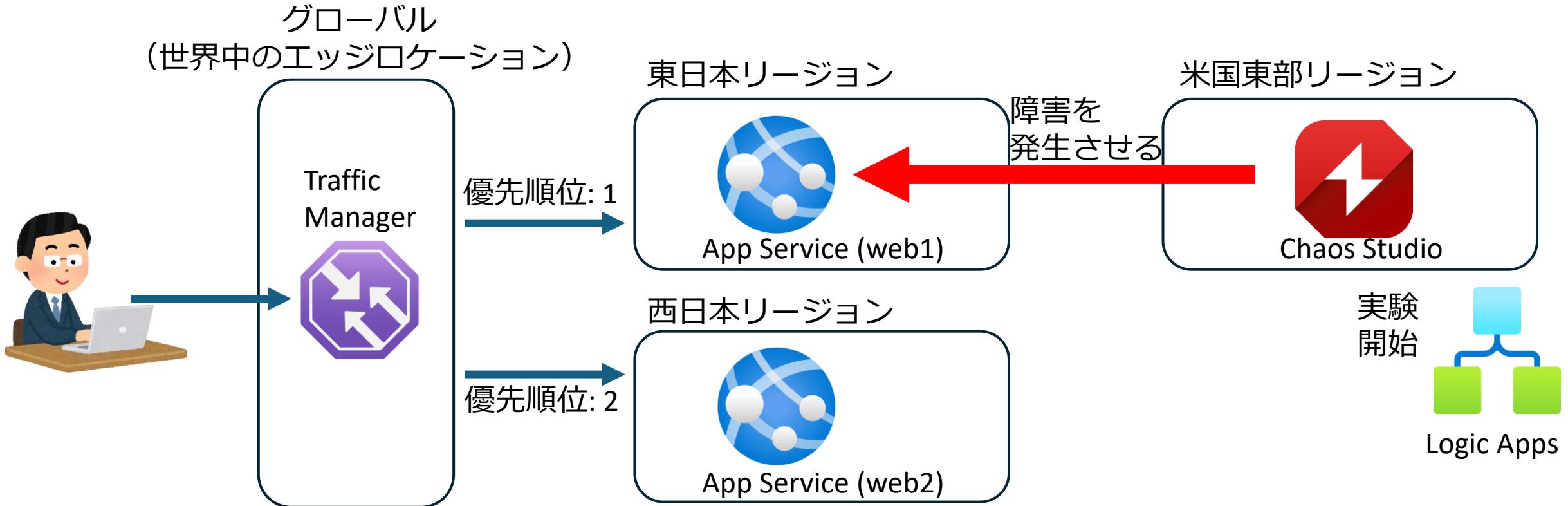


# Azure上のWebサイトの回復性の強化



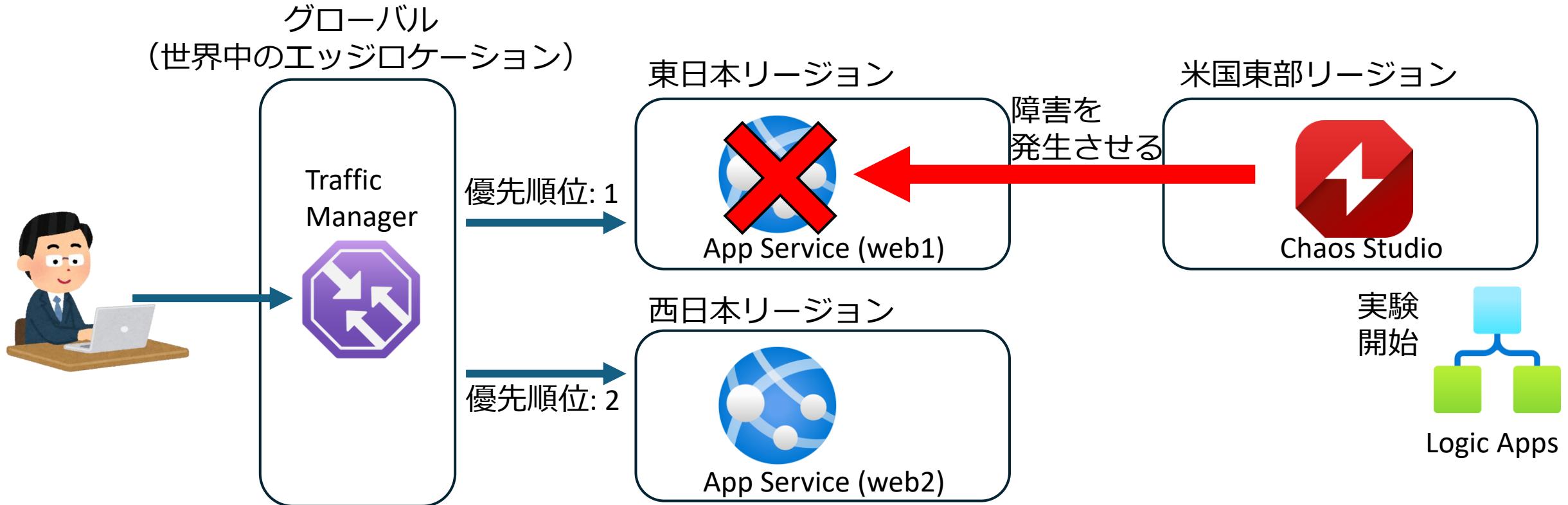
Logic Appsをデプロイし、  
実験の開始・終了をスケジューリング

# Azure上のWebサイトの回復性の強化



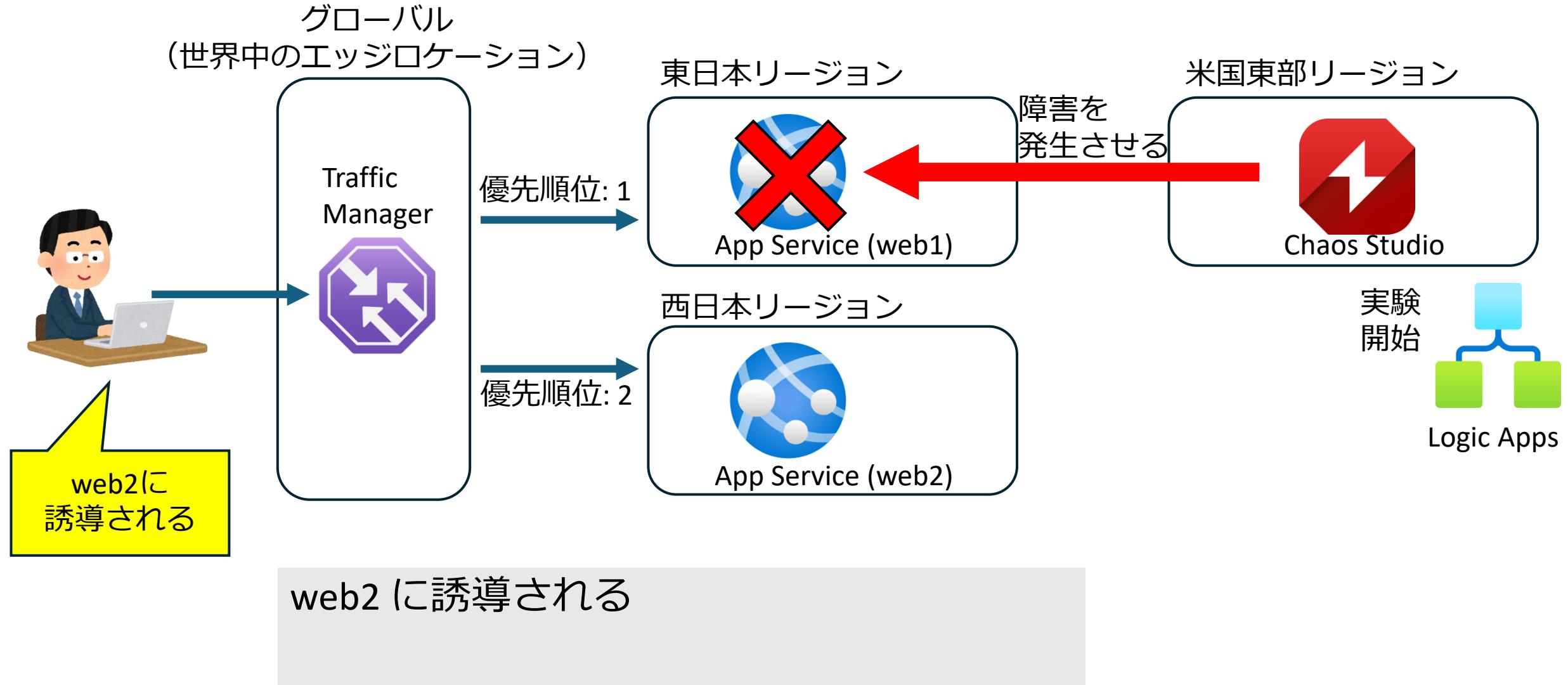
Logic Appsが、定期的に、  
実験を自動的に開始

# Azure上のWebサイトの回復性の強化

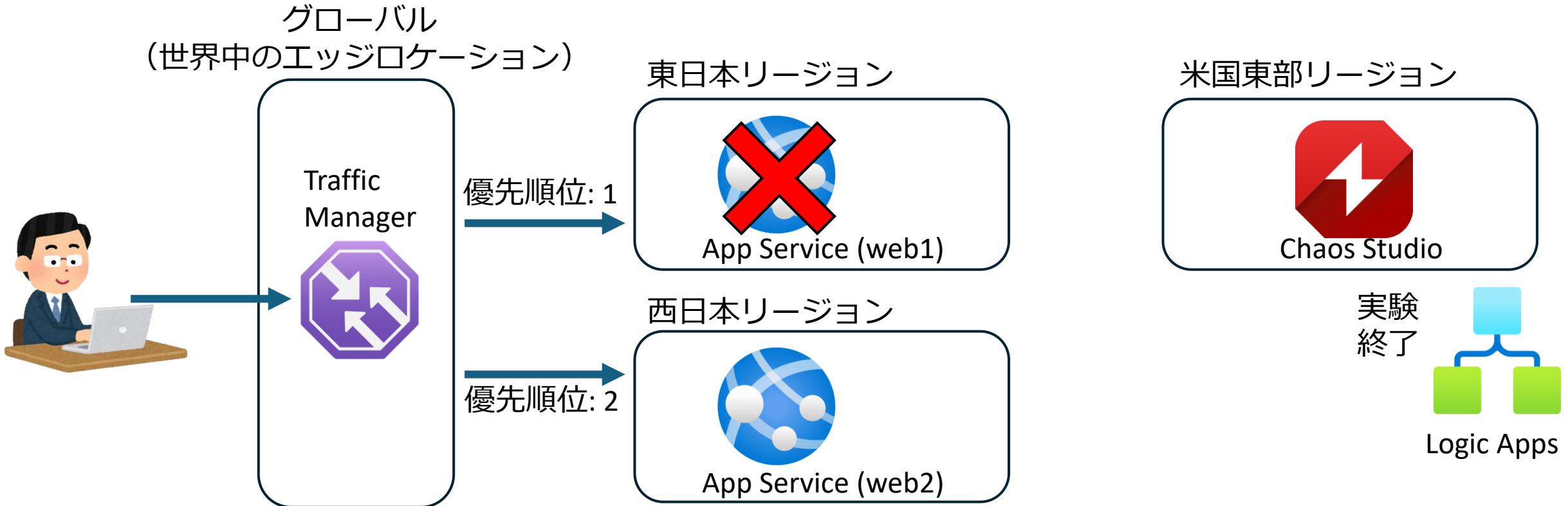


web1 に障害が発生

# Azure上のWebサイトの回復性の強化

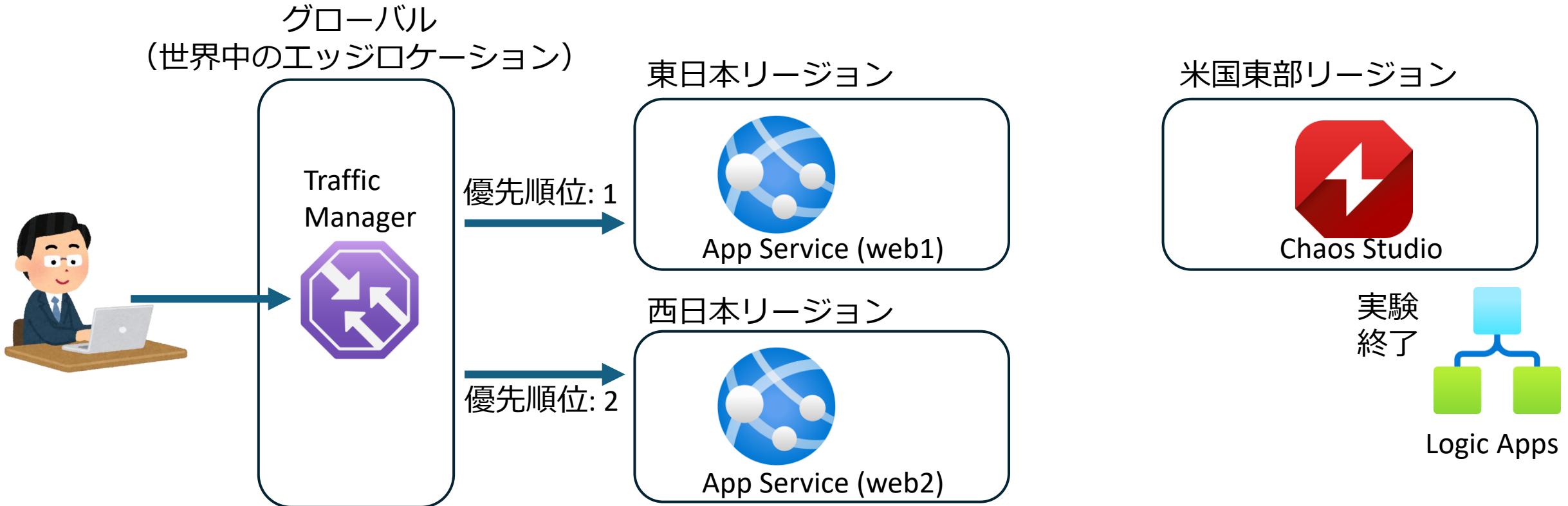


# Azure上のWebサイトの回復性の強化



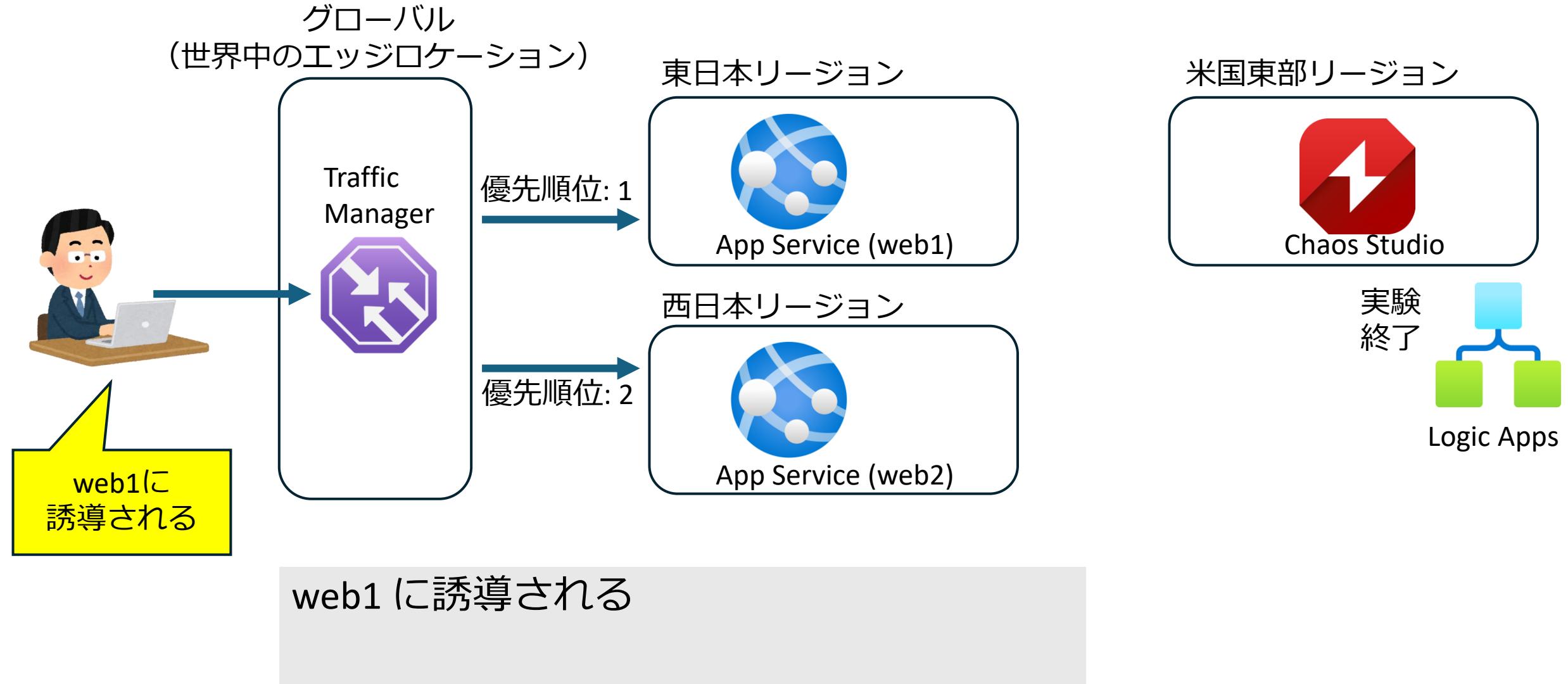
Logic Appsによって開始された実験が  
10分後に終了

# Azure上のWebサイトの回復性の強化



web1 が障害から回復

# Azure上のWebサイトの回復性の強化



# モジュール5 オペレーション（運用）

- ・オペレーションエクセレンスとは？
- ・DevOpsにおけるオペレーションエクセレンスの例
- ・シフトライトテスト
- ・カオスエンジニアリング
- ・Azure Chaos Studio
- ・Azure Logic Apps
- ・Azure Traffic Manager
- ・Azure 上の Web サイトの回復性の強化
- ・よくあるご質問
- ・まとめ

# よくあるご質問

- Netflixが開発したカオスエンジニアリングのツールはオープンソースなどで公開されていますか？ 自分たちのシステム開発でも利用できますか？
- → 「Chaos Monkey」に関してはossとして公開されており利用可能です
  - <https://github.com/Netflix/chaosmonkey>
- ただしNetflixはすべてのツールをOSSで公開しているわけではありません
- OSSまたは有償のカオスエンジニアリングツールとしては LitmusChaos、 Chaos Mesh、 Chaos Toolkit、 Gremlin などを利用できます。たとえば Chaos Meshは、 KubernetesのPodに障害を注入するOSSツールです。
  - <https://github.com/chaos-mesh/chaos-mesh>

# モジュール5 オペレーション（運用）

- ・オペレーションエクセレンスとは？
- ・DevOpsにおけるオペレーションエクセレンスの例
- ・シフトライトテスト
- ・カオスエンジニアリング
- ・Azure Chaos Studio
- ・Azure Logic Apps
- ・Azure Traffic Manager
- ・Azure 上の Web サイトの回復性の強化
- ・よくあるご質問
- ・まとめ

# モジュール5まとめ

|                          |   |
|--------------------------|---|
| オペレーションエクセルエンス           | 「運用卓越性」とも。組織全体で業務プロセスの最適化を図り、継続的な改善により競争優位を実現すること。  |
| DevOpsにおけるオペレーションエクセルエンス | DevOpsにおいては、システムの耐障害性を高めることやトラブルを即座に発見できる監視のしくみを構築することなどがオペレーションエクセルエンス該当する。  |
| カオスエンジニアリング              | システムの各部で障害が発生しても、システムが全体としては問題なくサービスを提供し続けられるように、システムの冗長化を行う手法。専用のツールを使用して、実際に、本番環境で、計画的に、システムの各部で障害を発生させ続けてテストする（障害の注入）。 |
| Azure Chaos Studio       | 現実世界の障害をシミュレートする障害を意図的に発生させることで、アプリケーションの回復力を向上させる実験プラットフォーム。カオスエンジニアリングをAzureで実現できる。                                     |
| Azure Logic Apps         | さまざまなサービスの接続・統合、運用の自動化などを行うためのサービス。画面上でフローチャートのようなもの（ワークフロー）を作成することで処理内容を定義する。たとえばAzure Chaos Studioの「実験」を定期的に開始したりできる。   |
| Azure Traffic Manager    | ロードバランサー（負荷分散）サービスの一種。複数のリージョンに対する負荷分散を行える。   |

# 講義



AZ-2008  
DevOpsの基礎:  
中心となる原則と実践

DevOps foundations: The core principles and practices

- ・モジュール1 DevOps の概要
- ・モジュール2 DevOps を使用した計画 (Plan with DevOps)
- ・モジュール3 DevOps を使用した開発 (Develop with DevOps)
- ・モジュール4 DevOps を使用した提供 (Deliver with DevOps)
- ・モジュール5 DevOps を使用した操作 (Operate with DevOps)
- ・まとめ

# 全体のまとめ

※各モジュールの重要概念

| モジュールNo | モジュールトピック         | DevOps   | GitHub  | ラボ   |
|---------|-------------------|--|---|--|
| 1       | DevOpsの概要         | 概要<br>メリット<br>歴史<br>従来型開発との比較                  |   |  |
| 2       | プロジェクト管理<br>タスク管理 | アジャイル<br>かんばん<br>スクラム                          | GitHub Issues<br>GitHub Projects              | GitHub Issues<br>GitHub Projects             |
| 3       | 開発                | バージョン管理<br>Git<br>継続的インテグレーション<br>シフトレフト(テスト)  | GitHub flow<br>フォーク<br>ブランチ<br>プルリクエスト<br>マージ | GitHub flow                                  |
| 4       | デリバリー             | デプロイ戦略<br>リリース戦略<br>継続的デリバリー<br>継続的デプロイ<br>IaC | GitHub Actions                                | GitHub Actions<br>Bicep<br>Azure App Service |
| 5       | オペレーション<br>(運用)   | オペレーションエクセレンス<br>シフトライト(テスト)                   |   | Azure Chaos Studio<br>Traffic Manager        |

# 関連認定試験のご紹介

- GitHub社の認定試験
  - [GitHub Certifications](#)
- Microsoftの認定試験
  - [Microsoft Certified: DevOps Engineer Expert](#)
- 皆様の実力のアピールに役立ちます
- 本コースのご受講と合わせてぜひチャレンジしていただければと思います