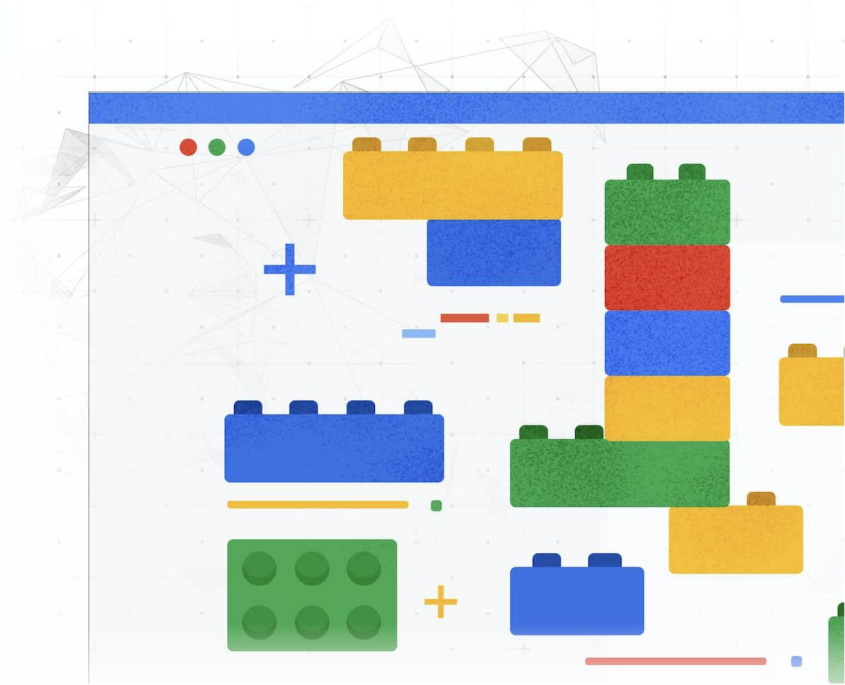
Google Cloud を使用した アプリケーション モダナイゼーションの道のり



グーグル・クラウド・ジャパン合同会社
ソリューションズ アーキテクト
長谷部 光治

長谷部 光治

Google Cloud Japan
Solutions Architect

Technology area:

- Multi / Hybrid cloud
- Application Development
- Infrastructure

アジェンダ

- ✔ Application Modernization
- ✔ DevOps Research and Assessment
- ✔ Google Cloud での実装例
- ✔ その先へ
- ✔ まとめ

Application Modernization

01

アプリケーション モダナイゼーションとは？

レガシーなアプリケーション、
プラットフォームを**作り変える**こと

実現手段は様々

- リプラットフォーム
- リホスト
- リアーキテクト
- リファクタリング

つまりアプリケーションに**何らかの変更**が加わる

何のために行う？

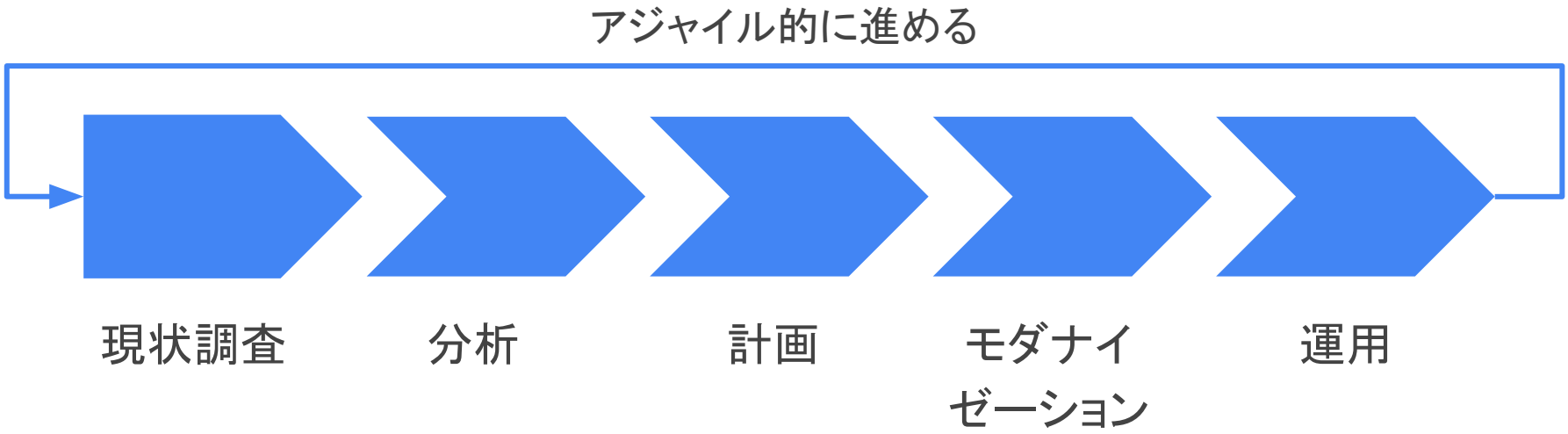
ビジネスの価値を高めたい

具体的には

- 迅速にマーケットに新機能をリリースしたい
- パフォーマンスを高めたい
- 障害を減らしたい
- コストを抑えたい

どのように進める？

典型的なアプリケーション モダナイゼーションのプロセス



詳細から手をつける**その前に**・・・

下準備は整っていますか？

システムに**安全かつ迅速に変更**を行うために

- 問題が起こっているかをひと目で確認できますか？
- 問題が起こった場合すぐに切り戻せますか？
- 少しずつ機能をリリースできますか？
- 本番と同等環境をすぐ用意できますか？

これらの能力を**向上させることが先決**

モダナイゼーションに必要な能力を身につける

- 問題が起こっているかを
  ひと目で確認できますか？
- 問題が起こった場合
  すぐに切り戻せますか？
- 少しずつ機能をリリース
  できますか？
- 本番と同じテスト環境を
  すぐ用意できますか？

DevOps のプラクティス

- DevOps とは？

  ソフトウェア **デリバリの速度**とサービスの**信頼性の向上**、ソフトウェアの関係者間で共有する権限の構築を目的とする**組織的で文化的**な仕組み

DevOps Research and Assessment

02

DORA とは？

DORA
DEVOPS RESEARCH & ASSESSMENT

DevOps に関する**研究**、
リサーチを行っているチーム

- The State of DevOps レポートを作成
- 2018 年に Google の傘下に
- DevOps の成熟度を調べるツール、
  また組織全体で能力を高めるための
  プラクティスを提供

DORA における 4 つのスコープ

DORA では DevOps の能力を **4 つに分類**している

- 技術
- プロセス
- 測定
- 文化

DevOps は**組織的、文化的取り組み**のため
すべての項目が重要となるが、
本セッションでは**技術面**に焦点を当てる

技術に関するプラクティス一覧

- バージョン管理
- トランクベース開発
- 継続的インテグレーション
- デプロイの自動化
- 継続的テスト
- 疎結合なアーキテクチャ
- クラウド
  インフラストラクチャ

- チームのツール選択を
  サポートする
- テストデータ管理
- セキュリティの担保を開発
- プロセスに組み込む
- データベースの変更管理
- コードのメンテナンス性

技術に関するプラクティスの全体像

- バージョン管理
- トランクベース開発
- 継続的インテグレーション
- デプロイの自動化
- 継続的テスト
- 疎結合なアーキテクチャ
- チームのツール選択をサポートする
- テストデータ管理
- セキュリティの担保を開発プロセスに組み込む
- データベースの変更管理
- コードのメンテナンス性
- 包括的なモニタリングとオブザーバビリティ※
- 障害の予兆通知※

継続的デリバリ

組織のパフォーマンス

継続的デリバリが実現できると組織のパフォーマンスが向上

継続的デリバリを実現するための前提

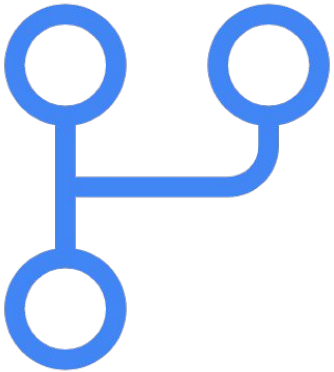
※ この 2 項目は「測定」のスコープに入るプラクティスだがこれらも継続的デリバリの前提になっている

各プラクティスについて

バージョン管理

本番環境に関する**全ての資材**を
バージョン管理システムで管理する

- ソースコード
- アプリケーション構成
- システム構成
- 構築自動化スクリプト

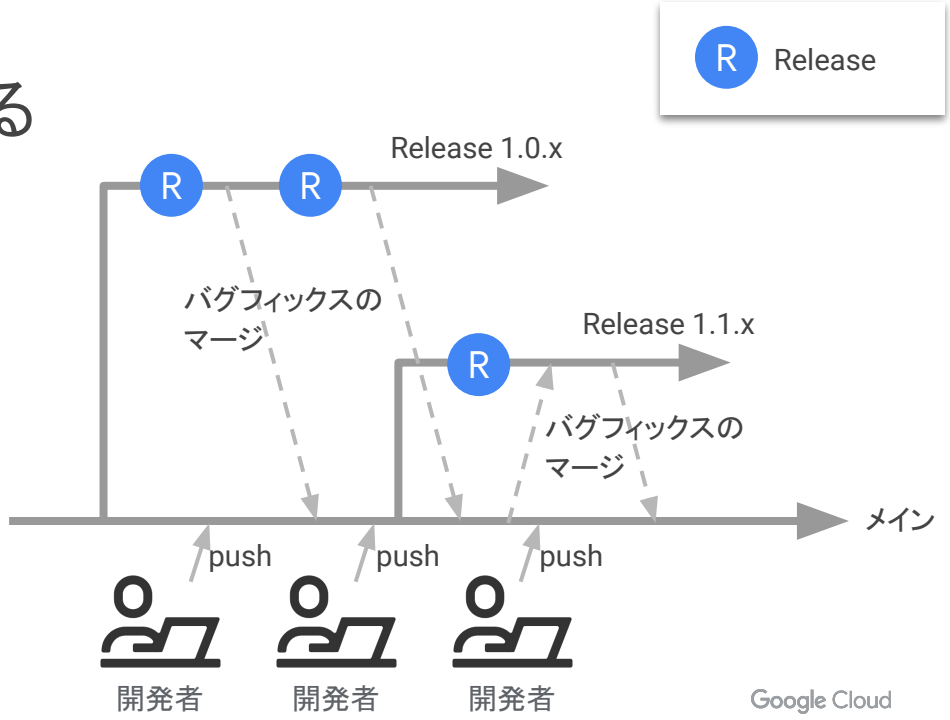


トランクベース開発

開発者はトランク（メイン）に
直接 push（できるように）する

実現のポイント

- **小さい単位**で開発
- **自動テスト**の仕組みを用意
- ビルドを**迅速に実行**する



コードのメンテナンス性

- ソースコード、依存関係を解決するツールの提供
  - 開発者はコードベースから簡単に**検索**ができ、他の人のコードの再利用、変更が行える
  - **依存関係**に影響がある問題を早期に発見できる

- Google の場合
  - **単一リポジトリ構成**で、誰でもコードベースの検索、再利用、変更が行えるようになっている
  - 依存関係に関する問題はツールにより発見、解決される
  - 変更はコードオーナーの**レビューが必要**

Bazel

継続的テスト

テストを**開発ライフサイクル全体**で実行し続ける

- 単体テスト
- 受け入れテスト
- UI テスト

これらを自動でテストできるようにする

開発者とテスト担当者は**一緒に作業にあたる**

- 問題をすぐに修正できる
- テストを意識したコードを書くようになる

テストデータ管理

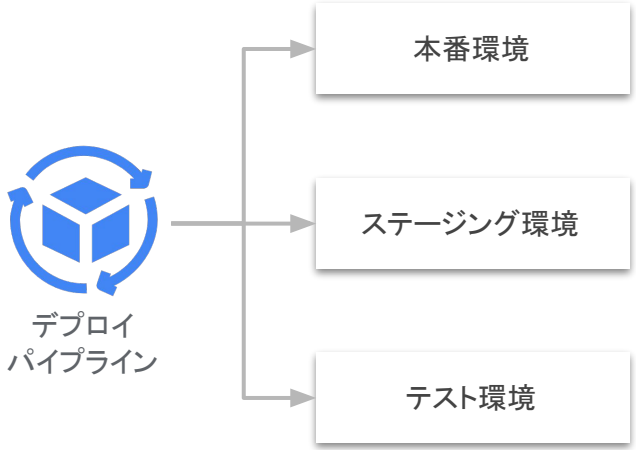
自動テストのパフォーマンス、信頼性を
担保するためにテストデータ管理は重要

- 単体テストを優先する（外部データに依存しない）
- テストデータを**分離**、または**依存を最小限**にする
- テストデータをエクスポートしておき、必要な
  ときに**すぐに利用できるように**しておく

デプロイの自動化

以下を満たす自動デプロイの
パイプラインを用意する

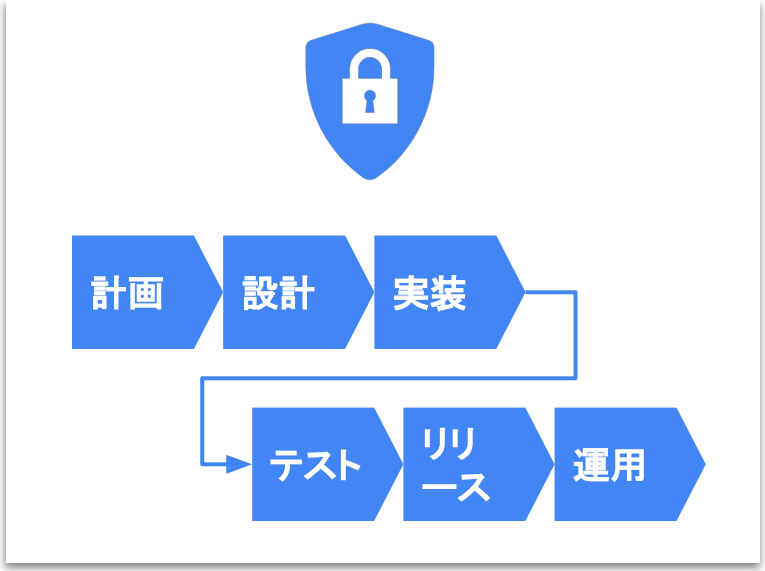
- すべての環境で使える
- 必要な認証情報があれば
  誰でも自動でデプロイできる
- すべての環境で
  同じパッケージを利用する
- バージョン管理システムを通じ
  環境の再現ができる



セキュリティの担保を開発プロセスに組み込む

セキュリティ対応は開発の後ろの方で実施されがち

- **設計フェーズから**
  セキュリティレビューを実施
- セキュリティチーム
  検証済みのライブラリ、
  ツールを利用する
- **自動化テスト**に
  セキュリティテストを組み込む



疎結合なアーキテクチャ

アプリケーションを他システムに影響を与えずに
必要に応じて**独立してテストおよびデプロイ**できる

アーキテクチャ例

- マイクロサービス アーキテクチャ
- サービス指向アーキテクチャ

マイクロサービス アーキテクチャ例

チームによるツールの選択をサポート

開発に使用するツール（言語も含む）を
選択できる体制を作る

- ベースラインを確立する
- ツールの定期レビューを行う
- 例外プロセスを定義する

参考：Google の場合

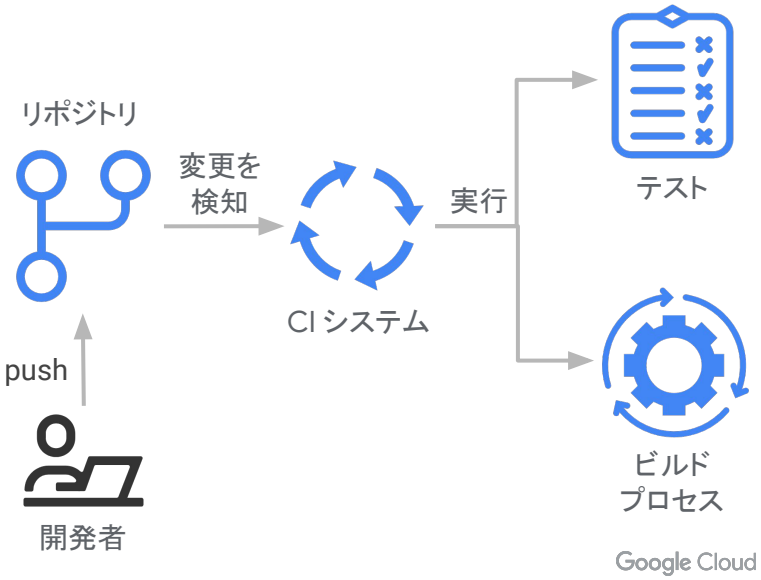
- 推奨ツール群が定義されている
- 推奨ツールをサポートするチームが存在する

継続的インテグレーション（CI）

コミットをトリガーにソフトウェアのテスト、
ビルドが実行される仕組みを作る

必要な要素

- 自動ビルドプロセス
- 自動テスト
- 自動ビルドと自動テストを
  実行する CI システム



包括的なモニタリングとオブザーバビリティ

モニタリング

事前に定義した指標やログを
用いてシステムを状態監視、
現状を把握する仕組み

オブザーバビリティ

事前に定義されていないプロパティ、パターンを調査しシステムの
デバッグなどに利用する仕組み

すべての開発者がツールに習熟し、
データドリブンな意思決定をできるようにする

障害の予兆通知

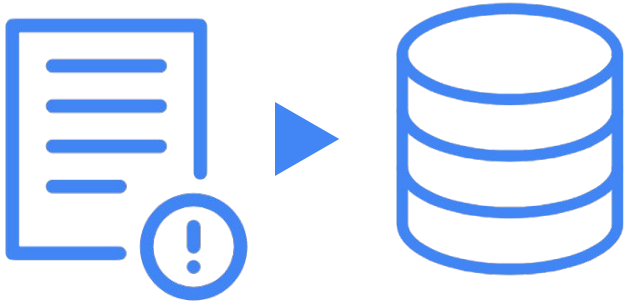
システムの健全性を**積極的にモニタリング**し、
チームが問題を**事前に検出**して軽減できるようにする

- 適切なしきい値でアラートを設定する
- **事後分析**を行う
- アラートの戦略を立てる

データベースの変更管理

データベースに対する変更は
リスクが高い

- データベース変更スクリプトを
  アプリケーションコードと
  同じようにバージョン管理する
- データベースの**変更履歴を**
  記録し可視化する



継続的デリバリ（CD）

すべての変更を**必要に応じて**、**迅速に**、**安全に**、**かつ継続的**にリリースするプラクティス

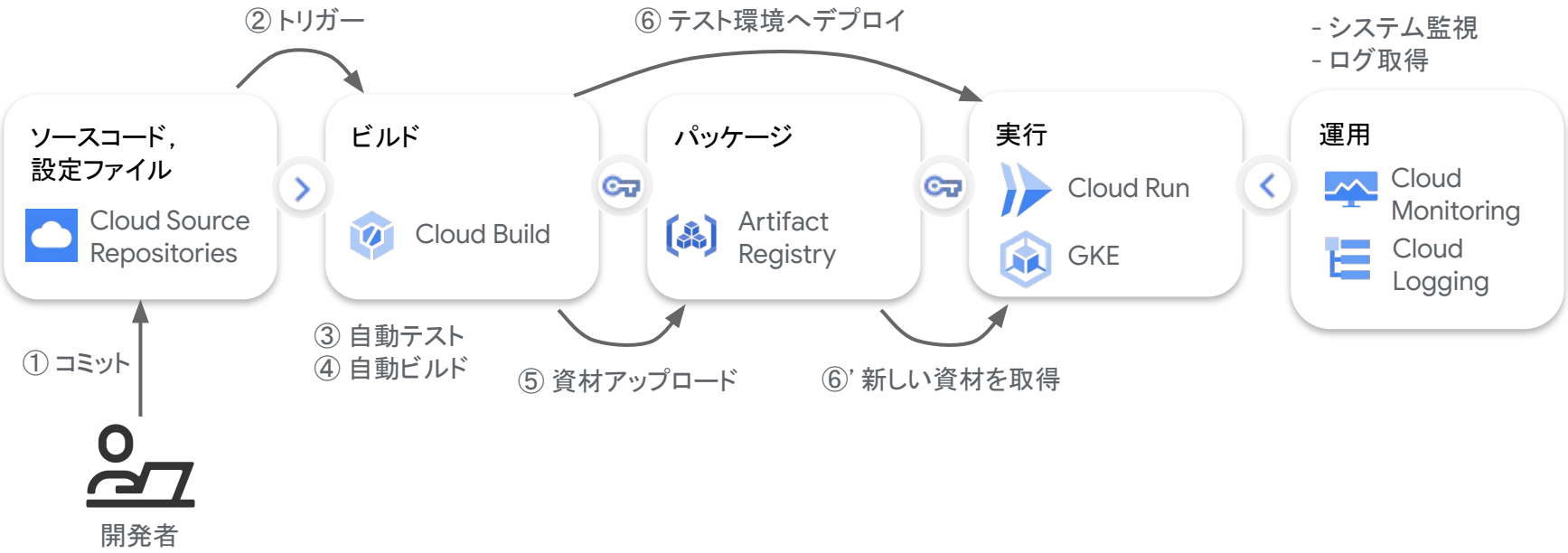
継続的デリバリは前提となるプラクティスを実装することで実現できる

- バージョン管理
- トランクベース開発
- 継続的インテグレーション
- デプロイの自動化
- 継続的テスト
- 疎結合なアーキテクチャ
- チームのツール選択をサポートする
- テストデータ管理
- セキュリティの担保を開発プロセスに組み込む
- データベースの変更管理
- コードのメンテナンス性
- 包括的なモニタリングとオブザーバビリティ
- 障害の予兆通知

Google Cloud での
実装例

03

Google Cloud における CI / CD パイプライン例 - コンテナベース -



Cloud Source Repository

Google Cloud にホストされている
プライベートな Git リポジトリ

- GitHub , Bitbucket との**ミラーリング**
- Cloud Build との連携
- 強力な正規表現による**コードの検索**

Cloud Build

サーバーレスな CI / CD プラットフォーム

- パイプラインを**宣言的**に作成
- **スケーラブル**なビルド（マシンサイズを選択可能）
- ビルドを走らせた**時間のみ**の課金

Artifact Registry

あらゆる資材の管理

- コンテナイメージに加え、
  Maven、**npm パッケージ**を管理
- ビルド、デプロイを自動化
- コンテナイメージの脆弱性をスキャン

Cloud Run

コンテナをサーバーレスで利用する

- 高速に **0 to N スケール**
- 言語やライブラリの制約なし
- OSS の Knative 互換、**ロックインの排除**
- 外部公開やロギングの設定まで一括で自動設定

Google Kubernetes Engine（GKE）

Google の 10 年以上に渡るコンテナ管理経験に基づいた**エンタープライズグレードの Kubernetes 環境**

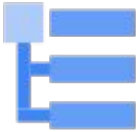
- マネージドなコントロールプレーン
- Google Cloud Platform が提供するハイパフォーマンス、信頼性が基盤
- エンタープライズでの利用を前提
  HIPAA and PCI DSS 3.2 に準拠

Operations

運用担当者（主に監視、ログ分析）、
開発者（トレーシング、デバッグ、プロファイリング）をカバーするフルマネージドなモニタリング基盤

Monitoring

トレンド監視
アラート



Logging

リアルタイム
ログ管理、分析

Trace

分散
トレーシング

Debugger

デバッグ

Profiler

プロファイリング

その先へ

04

DevOps の能力がつくと

アプリケーションは

- **必要に応じて**デプロイ可能
- 変更のリードタイムが **1 日以内**
- 障害が発生してから修正まで **1 時間以内**
- デプロイに問題がある割合が **15 % 以下**

が実現できる

DevOps の能力を向上させたその後は

アプリケーションごとの**ユースケースに合わせた**
モダナイゼーションを進める

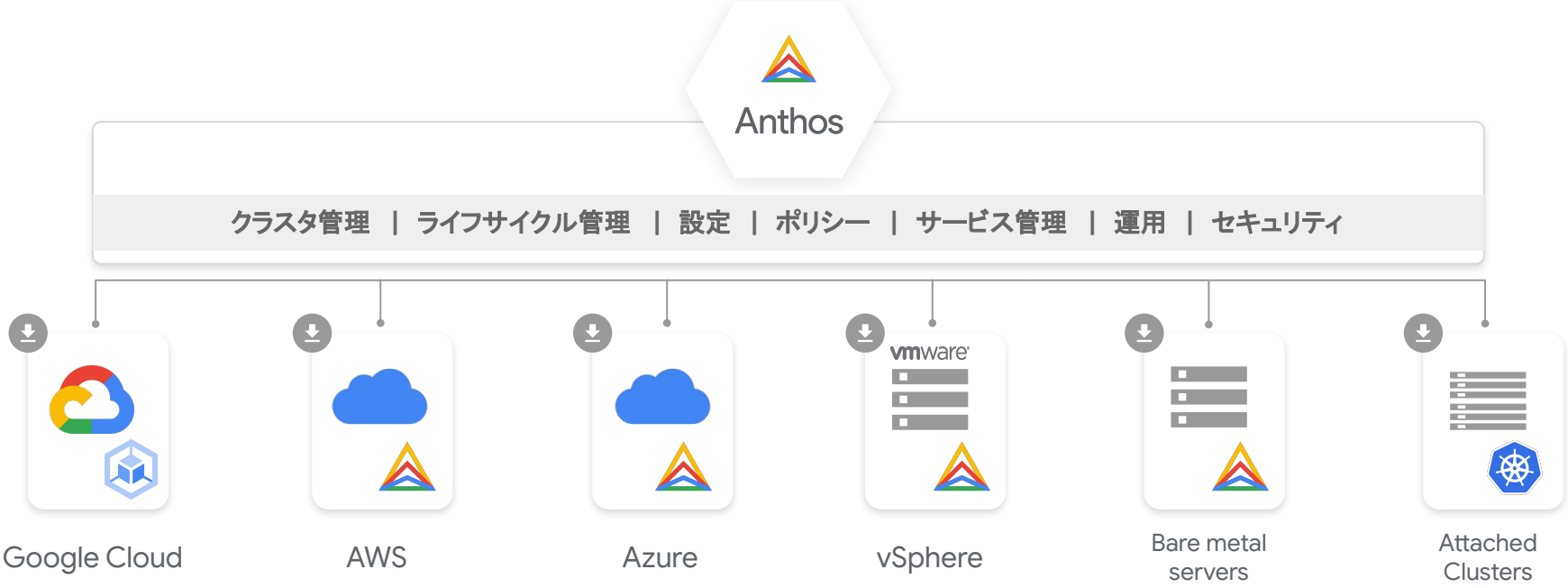
- Java モダナイゼーション
- メインフレーム モダナイゼーション
- .NET モダナイゼーション
- エッジ アプリケーション

DevOps の能力を向上させたその後は

プラットフォームをモダナイズする

- **あらゆる場所**で動く（マルチ、ハイブリッドクラウド）
- **一貫性**のある
  - 管理の仕組み
  - アプリケーション実行環境
- **継続的デリバリ**と
  相性がよいプラットフォーム

Anthos があらゆる場所で
アプリケーション モダナイゼーションを実現



まとめ

05

まとめ

- ✔ アプリケーション モダナイゼーションには
  まず **DevOps の能力**を向上させよう
- ✔ **DORA** の各プラクティスを参考にしよう
- ✔ 能力が身についたあとは、**Anthos** を使い
  アプリケーション モダナイゼーションを
  あらゆる場所で実現しよう

DORA のプラクティスを適用したい方へ

近日、DORA のワークショップを Google Cloud から提供予定

今回紹介した技術面のプラクティスにとどまらず、その他についてもご紹介、ディスカッションを行い、個々のお客様に合わせたインサイトをご提供いたします

DORA
DEVOPS RESEARCH & ASSESSMENT

※ご興味のある方は、担当営業までご連絡ください

Thank you