


Oracle CloudでDevOps!? Javaアプリケーションのモダン開発を支援する Oracle Cloudサービスのご紹介

日本オラクル株式会社
クラウド事業戦略統括
クラウドソリューション推進本部
茂 こと

Modern Cloud Day Tokyo  #oracleMCD

次世代クラウドが変える日本のビジネス

以下の事項は、弊社の一般的な製品の方向性に関する概要を説明するものです。また、情報提供を唯一の目的とするものであり、いかなる契約にも組み込むことはできません。以下の事項は、マテリアルやコード、機能を提供することをコミットメント（確約）するものではないため、購買決定を行う際の判断材料になさらないで下さい。オラクル製品に関して記載されている機能の開発、リリースおよび時期については、弊社の裁量により決定されます。

OracleとJavaは、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。文中の社名、商品名等は各社の商標または登録商標である場合があります。

本日のテーマ

モダンなJavaアプリケーション開発・運用を
Oracle Cloudでどう実現するか

Agenda

- システムに求められるニーズと開発トレンド
- Javaアプリケーションに求められる
アーキテクチャの変化と現実解
- Oracle Cloudを活用したDevOps

システムに求められるニーズと 開発トレンド

企業システムに求められるニーズの変化

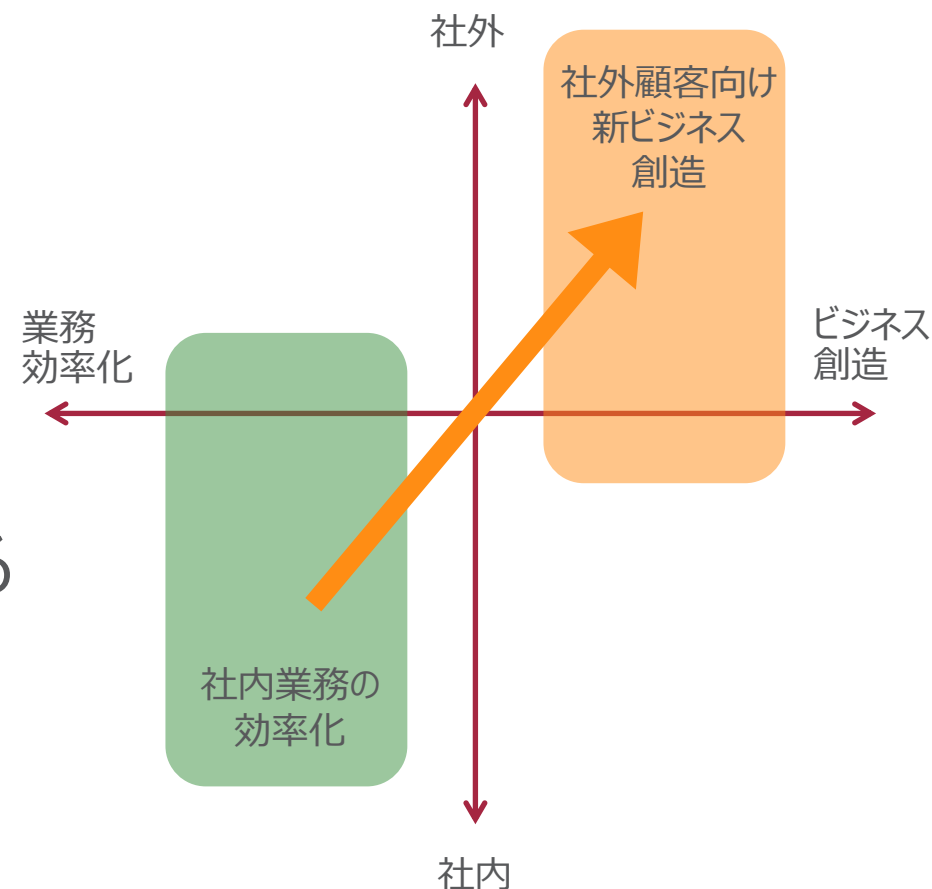
デジタルトランスフォーメーション | 社内業務の効率化から新ビジネスの創造へ

- ビジネスにもとめられる**進化のスピードと品質**

- ✓ サービスの早期リリース
- ✓ 市場の動向/反応を早期フィードバックして対応
- ✓ サービス停止による機会損失をなくす

- システムは変化するビジネス要件に合わせて**短い時間で高頻度**にリリースすることが求められる

- ✓ アプリの変更をすぐに本番環境に適用
- ✓ 変化を許容できるシステムを構築
- ✓ 停止時間の短縮と運用作業の効率化

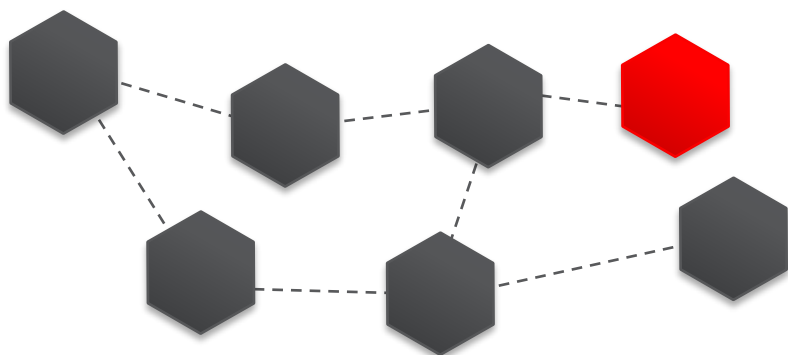


高頻度リリースを実現するための開発トレンド

アーキテクチャと開発手法の両面から高頻度リリースの取り組みが行われている

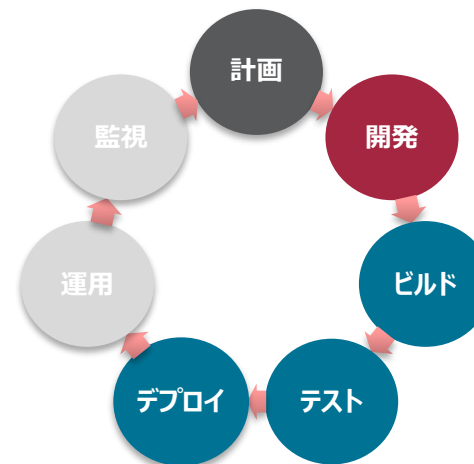
- マイクロサービス・アーキテクチャ

- 大規模なシステムを疎結合な複数のサービスの組み合わせで実現する設計方式
- 変更による影響範囲をサービス単位に極小化し高頻度のアプリケーション更新を実現



- DevOps

- 開発・運用の組織的な隔たりを無くし、サービスの継続的な更新を実現する
- CI/CDツールを用いてアプリのテストからデプロイを自動化し、迅速なリリースと品質確保を両立

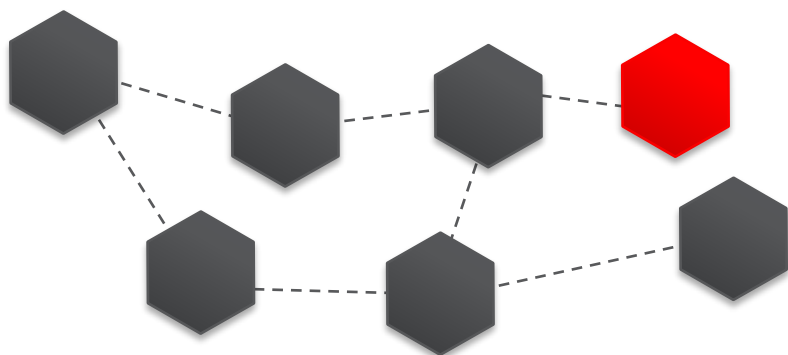


高頻度リリースを実現するための開発トレンド

アーキテクチャと開発手法の両面から高頻度リリースの取り組みが行われている

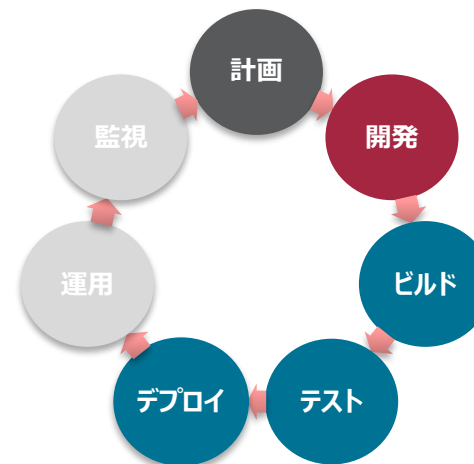
- マイクロサービス・アーキテクチャ

- 大規模なシステムを疎結合な複数のサービスの組み合わせで実現する設計方式
- 変更による影響範囲をサービス単位に極小化し高頻度のアプリケーション更新を実現



- DevOps

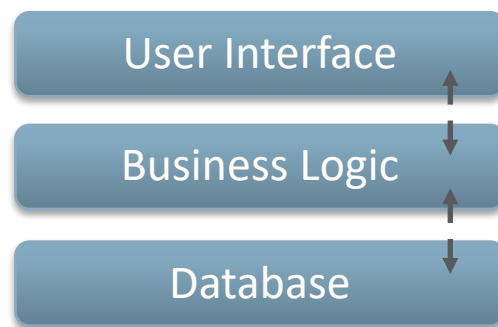
- 開発・運用の組織的な隔たりを無くし、サービスの継続的な更新を実現する
- CI/CDツールを用いてアプリのテストからデプロイを自動化し、迅速なリリースと品質確保を両立



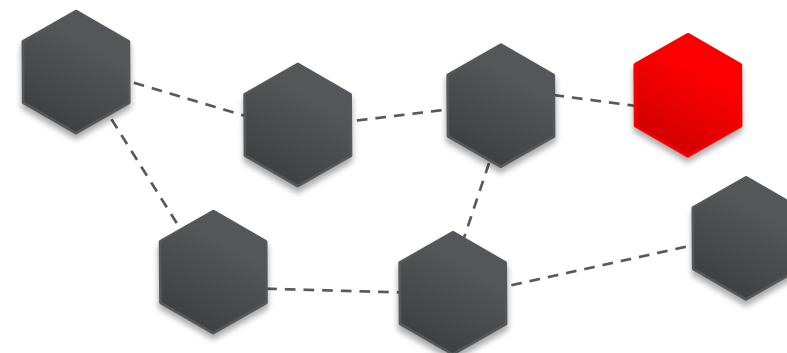
Javaアプリケーションに求められる アーキテクチャの変化と現実解

モノリシックとマイクロサービスの違い

全業務を1アプリケーションとして稼働 vs.ドメインモデルでサービス単位に分割して稼働



- モノリシック・アーキテクチャ
 - 全業務を1アプリケーションにパッケージ化
 - 共有ランタイムにデプロイして稼働
 - データソースをアプリケーション全体で共有



- マイクロサービス・アーキテクチャ
 - 業務単位等で細かいサービスに分割
 - ランタイムも分割しAPI等で疎結合化
 - データソースをサービス毎に分割して占有

マイクロサービスのトレードオフ

マイクロサービスは万能ではない | システムの要件を見極めて適切な使い分けが必要

モノリシック

変更の影響範囲が把握しにくい
ため、頻繁に更新できない

障害の影響がシステム全体に及
びやすい

通信のオーバーヘッドが無いため、従来
通りのパフォーマンスが出る

基本的にはデータベースがひとつなので、
データの一貫性を保ち易い

管理対象が少ないための、運用・
管理が比較的シンプル

更新頻度

耐障害性

パフォーマンス

データの一貫性

運用・管理性

マイクロサービス

影響範囲がサービスにとどまるた
め、高頻度で更新できる

障害の影響範囲をサービスに留
めるように設計できる

サービス間通信のオーバーヘッド
のため、パフォーマンスが落ちやすい

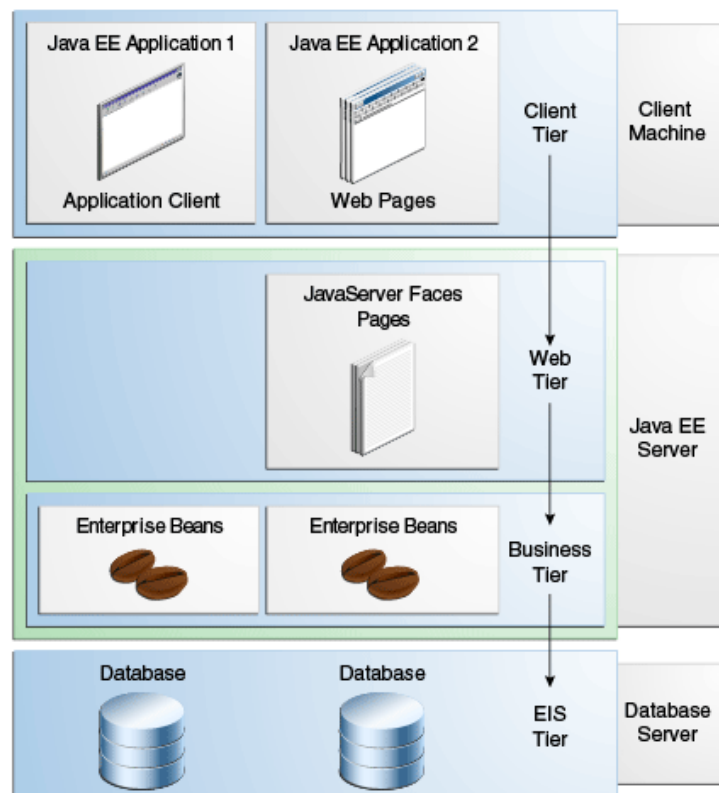
サービスをまたぐトランザクションは
実現困難

サービスが増える分だけ管理対
象が増加



従来のJava EEアプリケーションの特徴

多くは階層だけで分離されたモノリシックな実装



- アプリケーションは階層に基づいて編成
- Java EEによって開発されるためコードはJavaで記述
- アプリケーションは複数のモジュールからなる巨大なEARファイルで構成されている
- データベースはアプリケーションで共有する
- ユーザー・インタフェースはJSFまたはJSPでコーディングされた複数のWARモジュールで構成

Java EEアプリケーションに求められる変化

継続的な変化への対応が求められる

- 安定性は保ったまま、新機能の早期リリース
- アプリケーションの分割の検討
 - 一部のコンポーネントを他のコンポーネントより頻繁に更新する
 - コンポーネント毎にスケール要件が異なる
 - アプリケーションが大きく複雑になりすぎている
- 開発、運用コストの最適化
 - アプリ開発、実行基盤の構築を自動化、高速化
 - 新規開発された他のアプリとの運用の一本化

マイクロサービス適用のジレンマ

モノリシック・アプリケーションの分解の困難さ

複雑なシステムを完全に置き換えるには膨大な作業が発生する

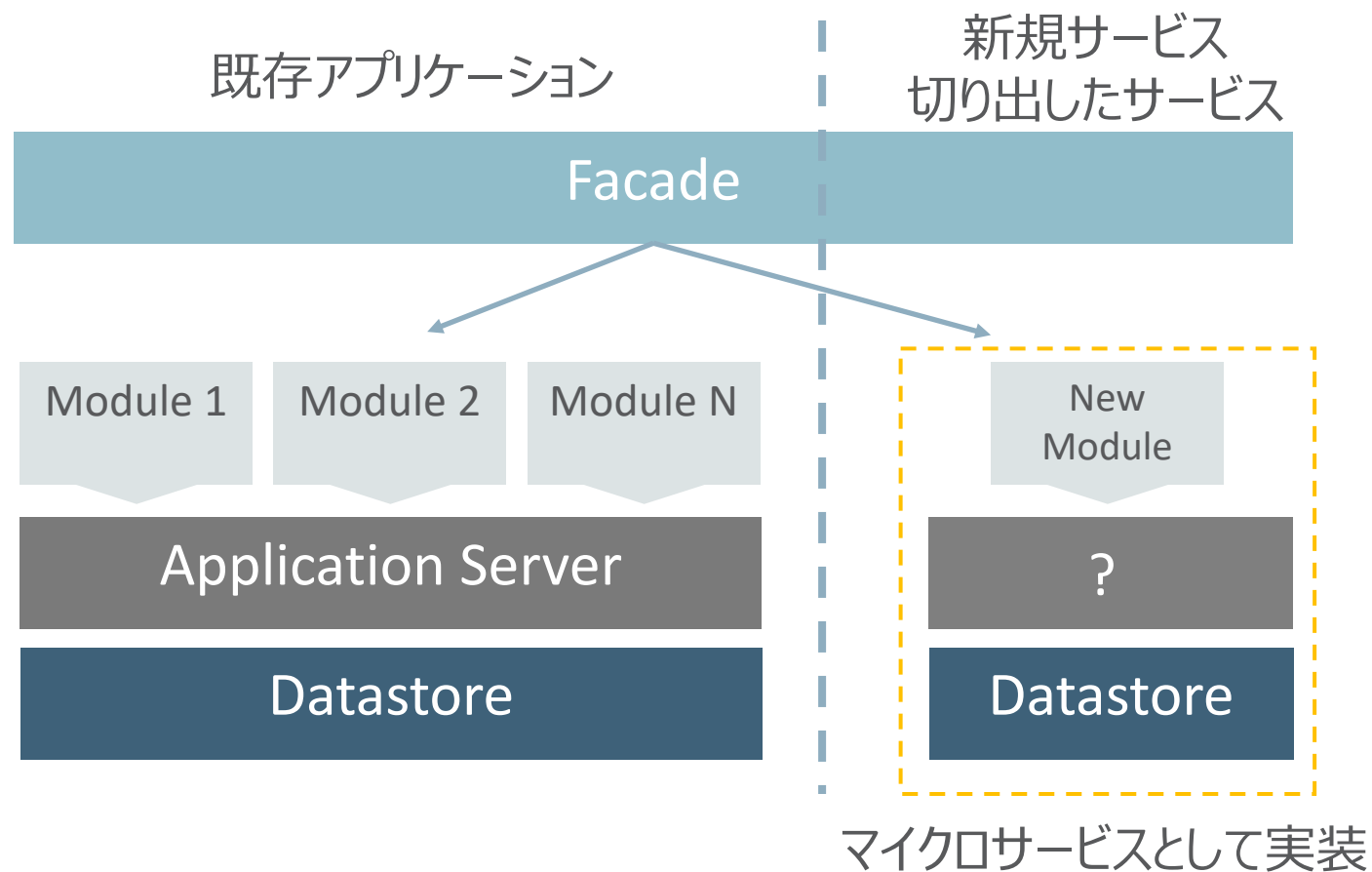
- 大規模なモノリシック・アプリケーションは密結合しやすく依存関係を把握・整理し辛い
- 密結合したモジュール同士は一方の改修により他方の影響が発生し改修範囲が膨らむ
- 稼働中のモノリシック・アプリケーションの更新は影響が全体に及ぶリスクが高い

早期のリリースを優先して早く容易にリリースしたい

- モノリシックなアプリケーションの方が難易度が低く開発自体は容易
- 慣れたアプリケーション開発手法の方が運用の難易度も低く確実

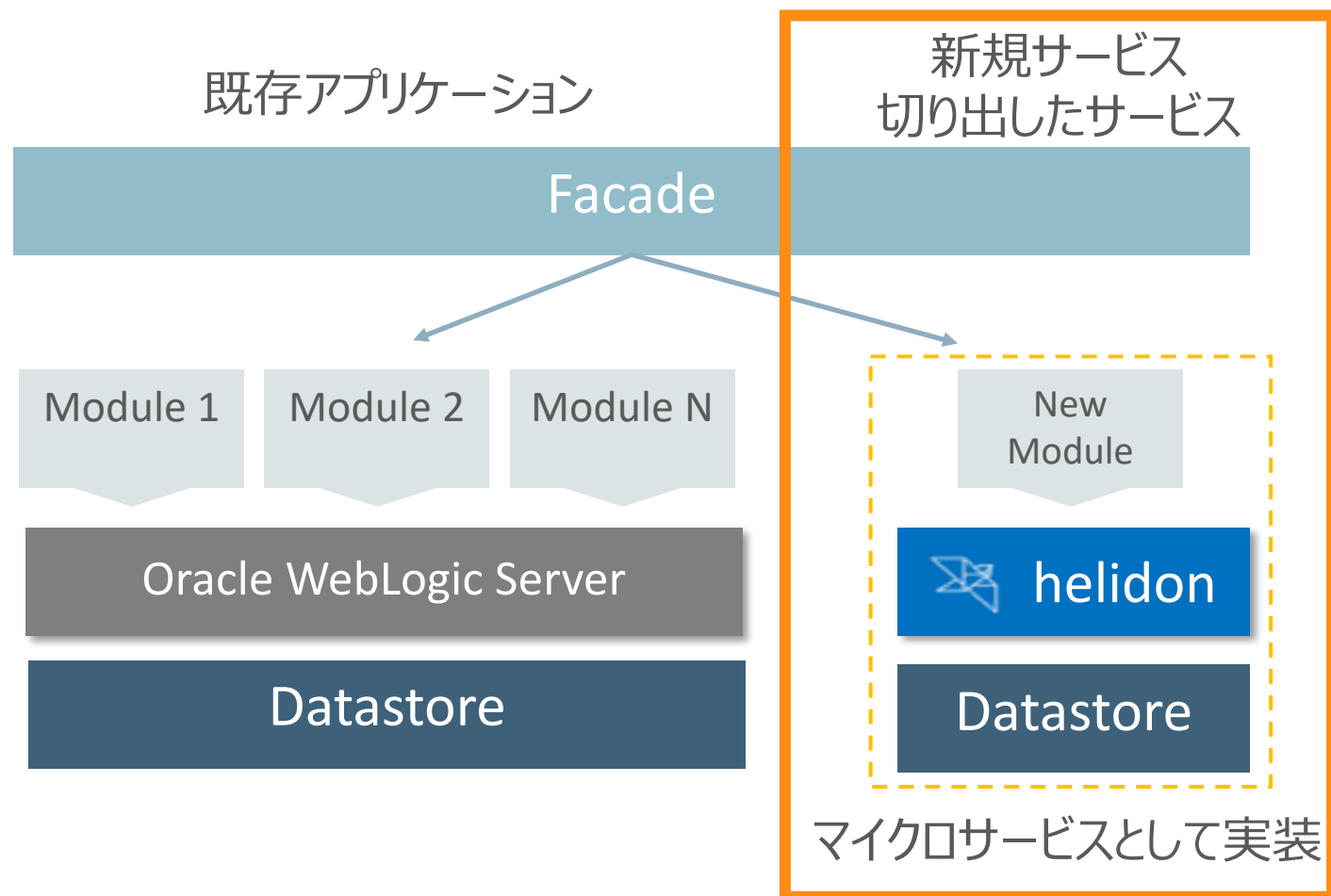
マイクロサービス化に向けた設計パターン

Strangler Pattern(ストラングラー・パターン)



- アプリケーションの特定の機能を徐々に改善していく
 - 低コスト、高速、保守が容易
- 段階的に新しいアプリケーションやサービスに置き換え
 - ファサード(窓口)がユーザーからの要求を既存アプリケーションまたは新しいサービスにルーティング
 - 置き換えが完了したら既存アプリケーションから古いサービスを削除

Java EEアプリケーションのマイクロサービス化



- ✓ マイクロサービスに適したJavaのフレームワークを選択する
- ✓ Javaのスキルを活かした開発が出来る
- ✓ マイクロサービスに適した基盤、監視ツールを取り入れる

Project Helidon

Lightweight. Fast. Crafted for Microservices.



Project Helidon

Javaのマイクロサービス・フレームワーク

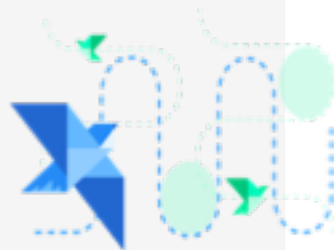
- OracleがホストするOSSプロジェクト
 - GitHubでソースコードを公開 : <https://github.com/oracle/helidon>
- マイクロサービスアプリが必要とする機能(Webサーバ、コンフィギュレーション、セキュリティ)を提供するJavaライブラリの集合体
 - 単体のJVMとして動作 アプリケーションサーバー不要
- 2つのプログラミングモデルを提供
 - Helidon MP: 宣言的な記法、Java EE開発者フレンドリー
 - Helidon SE: 関数型の記法





helidon SE

- マイクロフレームワーク
- 関数型の記法
- リアクティブなWebサーバを構築



```
Routing routing = Routing.builder()
    .get("/hello", (req, res) ->
        res.send("Hello World"))
    .build();

WebServer.create(routing)
    .start();
```



helidon MP

- MicroProfile 2.2
- 宣言的な記法
- CDI, JAX-RS, JSON-P/B
- Java EEスキルを有効活用



```
@Path("hello")
public class HelloWorld {
    @GET
    public String hello() {
        return "Hello World";
    }
}
```

HelidonはDocker, Kubernetesネイティブ

- Dockerfile, Kubernetes YAMLを自動生成
- GraalVM Native Imageに対応
- マイクロサービスの監視に適合



Dockerfile, Kubernetes YAMLを自動生成

MavenでProjectを作成 (quickstart-se)

```
mvn archetype:generate -DinteractiveMode=false ¥  
-DarchetypeGroupId=io.helidon.archetypes ¥  
-DarchetypeArtifactId=helidon-quickstart-se ¥  
-DarchetypeVersion=1.2.0 ¥  
-DgroupId=io.helidon.examples ¥  
-DartifactId=quickstart-se ¥  
-Dpackage=io.helidon.examples.quickstart.se
```

```
mvn package
```

Dockerfile

```
FROM openjdk:8-jre-alpine  
  
RUN mkdir /app  
COPY libs /app/libs  
COPY ${project.artifactId}.jar /app  
  
CMD ["java", "-jar", "/app/${project.artifactId}.jar"]
```

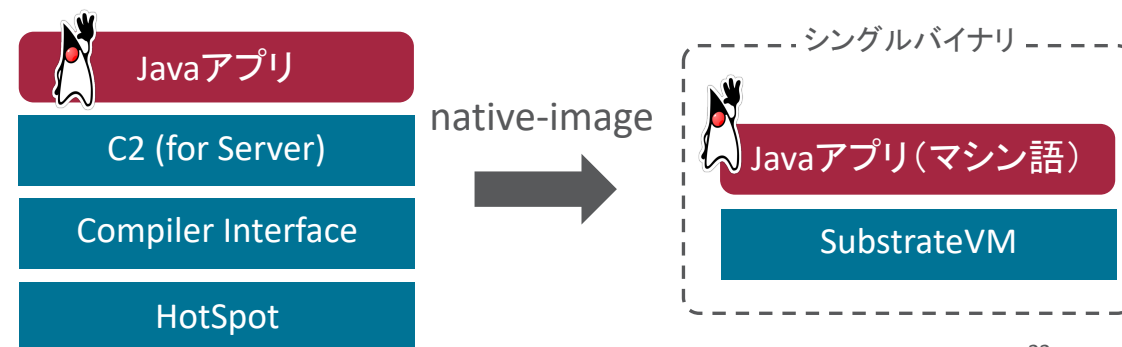
~/quickstart-se\$ tree

```
.  
├── pom.xml  
├── README.md  
└── src  
    ├── main  
    │   ├── docker  
    │   │   └── Dockerfile  
    │   ├── java  
    │   │   ├── io  
    │   │   │   ├── helidon  
    │   │   │   │   ├── examples  
    │   │   │   │   │   ├── quickstart  
    │   │   │   │   │   │   ├── se  
    │   │   │   │   │   │   │   ├── GreetService.java  
    │   │   │   │   │   │   │   ├── Main.java  
    │   │   │   │   │   │   │   └── package-info.java  
    │   │   └── resources  
    │   │       ├── application.yaml  
    │   │       └── logging.properties  
    └── test  
        ├── java  
        │   ├── io  
        │   │   ├── helidon  
        │   │   │   ├── examples  
        │   │   │   │   ├── quickstart  
        │   │   │   │   │   ├── se  
        │   │   │   │   │   │   └── MainTest.java
```

GraalVM Native Imageに対応

GraalVMとの統合により、Helidon SEアプリケーションを高速起動、省メモリ

- GraalVMとはOracleがホストするOSS
 - Java HotSpot VMをベースに開発された、多言語対応のプログラミング言語ランタイム
 - アプリケーションとランタイムをセットにしたシングルバイナリを生成する（Native Image）など、特徴的な機能を持つ
 - GraalVM Native Imageの利用により軽量で高速化起動が可能に
 - ランタイムの起動時間、及びプログラム実行初期のパフォーマンスの向上
 - メモリフットプリントの極小化
- ⇒ マイクロサービスとの相性がよい



Helidonアプリの監視

Kubernetes上でのマイクロサービスの監視に適合

Prometheusフォーマットで
メトリックスを取得可能



Metrics

Instrumentation to expose metrics of your applications.



OpenTracingに対応



Tracing

Profile and monitor your applications across multiple services.



Health Check用
エンドポイントを提供

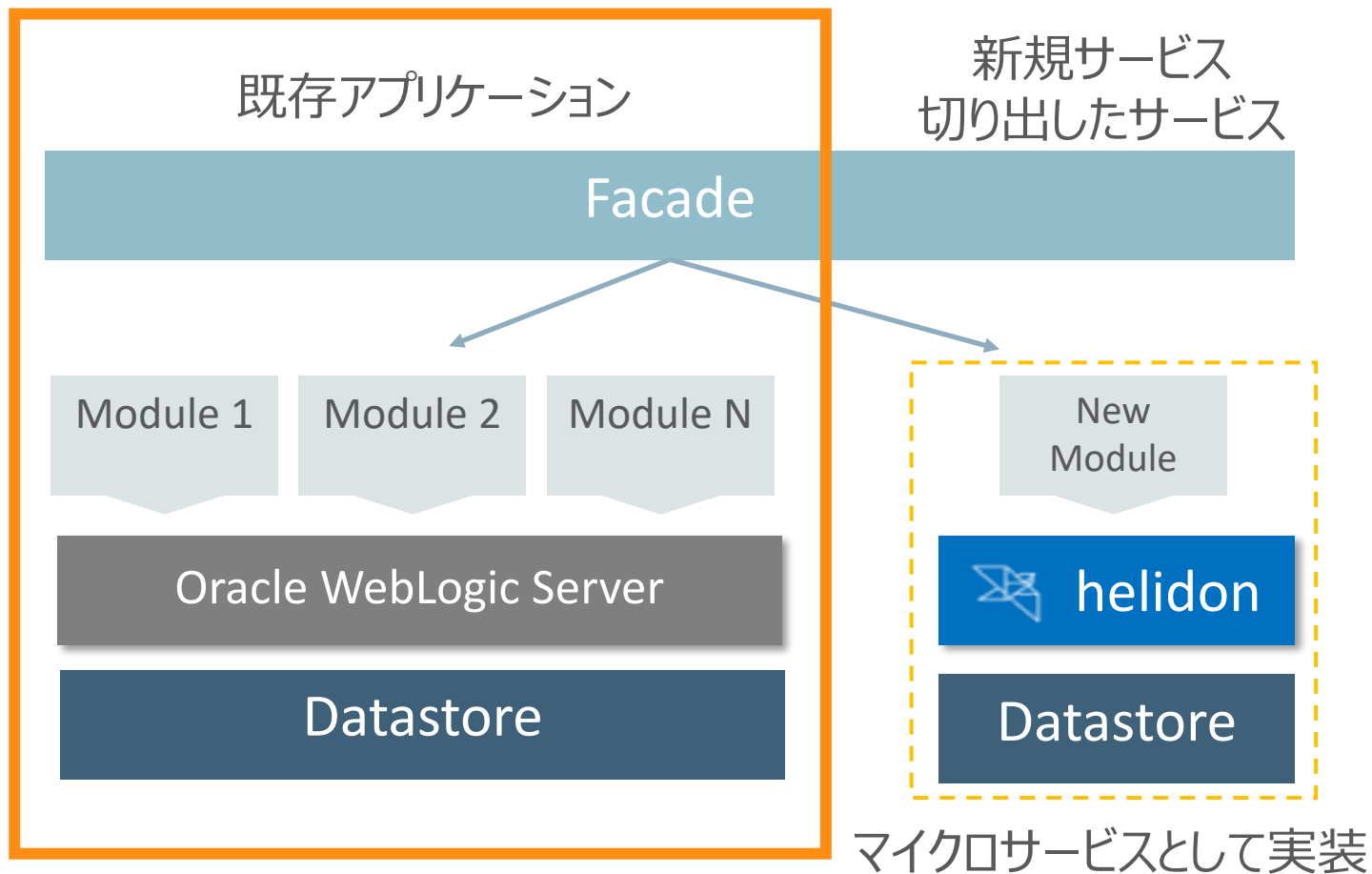


Health Checks

Expose health statuses of your applications.



既存WebLogicアプリケーションはどうする？



- 既存アプリケーションはそのままに運用効率の最大化を目指す
 - ✓ 基盤全体のクラウド化により、迅速な環境構築を実現
 - ✓ Kubernetesベースの管理により他のコンテナ・アプリケーションとの統合運用を実現

Oracle WebLogic Server 製品戦略

クラウド化を見据えたミッション・クリティカルプラットフォームとしての投資

戦略製品としての継続してリリース

- Oracle CloudのSaaS/PaaSやFusion Application/Middlewareの**基盤となる最重要コンポーネント**
- ミッション・クリティカルなワークロードを支える**エンタープライズ・クラスの可用性/安定性**を維持

ハイブリッド・クラウドを見据えた機能拡張を継続



- システムの**クラウド化や、クラウド上での運用を実現**するための機能の拡充
- プライベート・クラウドを含む**オンプレミス環境との相互運用性**を視野に入れた戦略的な機能

ミッション・クリティカル領域のJava EE/Jakarta EE基盤としての位置付け

- **標準仕様ベース**の生産性/安全性/費用対効果を担保するアプリケーション基盤
- 大規模なエンタープライズ・システムをエコシステムにより支えてきた**オープン・テクノロジーとしての実績**

WebLogicアプリケーションの運用基盤の選択肢

既存WebLogicの資産を活かしつつ、Cloudやコンテナのメリットを取り入れて最適化

	OCI Native	
	 Oracle WebLogic Cloud (VM/BareMetal)	 Oracle WebLogic Server on Kubernetes*
提供形態	Marketplace	Operator** and Marketplace
メリット	迅速な環境構築が可能	Kubernetesによる運用の統一化
課金モデル	BYOL/Subscription	BYOL/Subscription
Coherence (WLS Suite)	利用可能	利用可能
WebLogic Versions	JRF 12.2.1.3/11.1.1.7 Non-JRF 12.2.1.3	JRF 12.2.1.3 Non-JRF 12.2.1.3
Release	BYOL : GA Subscription : Coming Soon	Coming Soon

* 正式なプロダクト名称は未定

** Operatorは現時点ではGitHubからの提供のみ

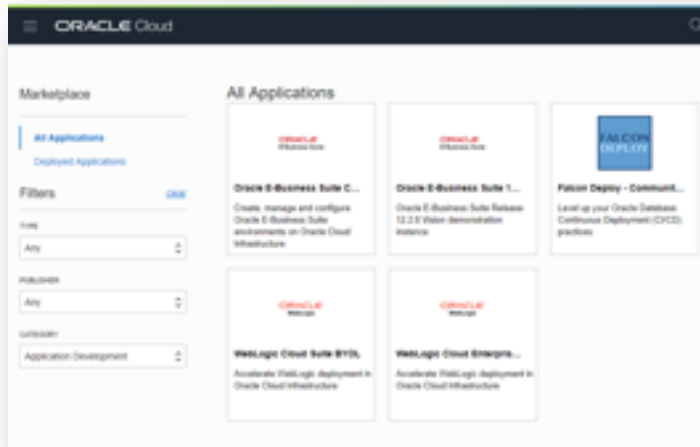
Oracle WebLogic Cloud | OCI Marketplace

僅か3StepでOracle WebLogic Serverの迅速な環境構築を実現

Step 1: OCI Marketplaceから検索

Step 2: 情報の入力

Step 3: インスタンスが起動



ORACLE Cloud

Create Stack

Stack Information
Configure Variables
Review

Configure the variables for the infrastructure resources that this stack will create when you run the apply job for this execution plan.

WebLogic Server Instance

RESOURCE NAME PREFIX
test
The names of all comp

WEBLOGIC SERVER S
VM Standard2.2
The shape for all Web

SSH PUBLIC KEY
ssh-rsa AAAAB3...
Use the corresponding

WEBLOGIC SERVER A
Halw-PHX-AD-1
The name of the availa

WEBLOGIC SERVER N
1
The initial number of W

WEBLOGIC SERVER A
adminadmin

WebLogic Server Instance Advanced

☒ WLS INSTANCE ADVANCED CONFIGURATION

WEBLOGIC SERVER DOMAIN NAME
myDomain
The name of the WebLogic Server domain

WEBLOGIC SERVER NODE MANAGER PORT
5556
The listen port number for the node manager process on all compute instances

WEBLOGIC SERVER ADMIN CONSOLE PORT
7001
The administration server port on which to access the administration console

WEBLOGIC SERVER ADMIN CONSOLE SSL PORT
7002
The administration server SSL port on which to access the administration console

WEBLOGIC CLUSTER PORT
5555
The managed server port on which to send heartbeats and other internal cluster traffic

WEBLOGIC SERVER EXTERNAL ADMIN PORT
9071
The server port on which to send administration traffic

WEBLOGIC SERVER EXTERNAL ADMIN SSL PORT
9072

Back Next Cancel

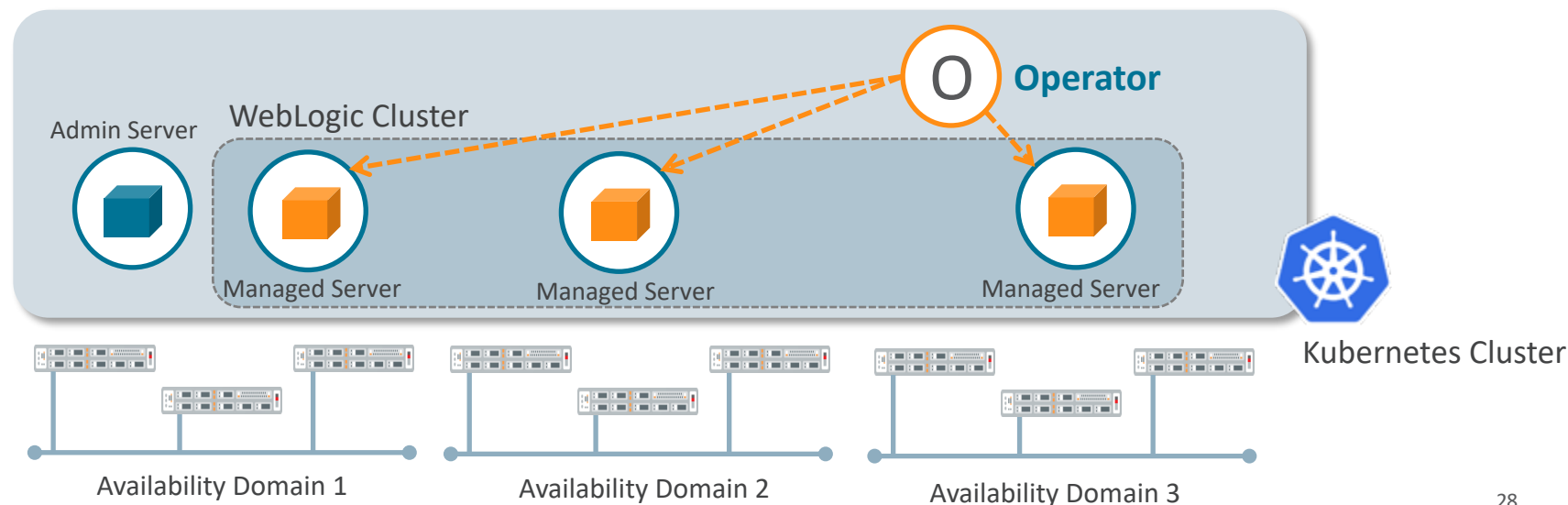


<https://docs.oracle.com/en/cloud/paas/weblogic-cloud/>

WebLogic Server on Kubernetes & WebLogic Operator

WebLogic Serverのランタイム制御をKubernetesに対するオペレーションに集約

- WebLogic OperatorはWebLogicコミュニティにより開発されたオープンソース(*)
 - WebLogic Serverの機能としてOracleがサポート
 - Kubernetes上でのWebLogic Server/WebLogic Clusterの運用自動化をサポートする機能を提供
 - Infrastructure as Code(YAML)をベースとしたWebLogic Server環境の宣言的な構成管理を実現
- Kubernetesベースの管理により他のコンテナ・アプリケーションとの統合運用を実現

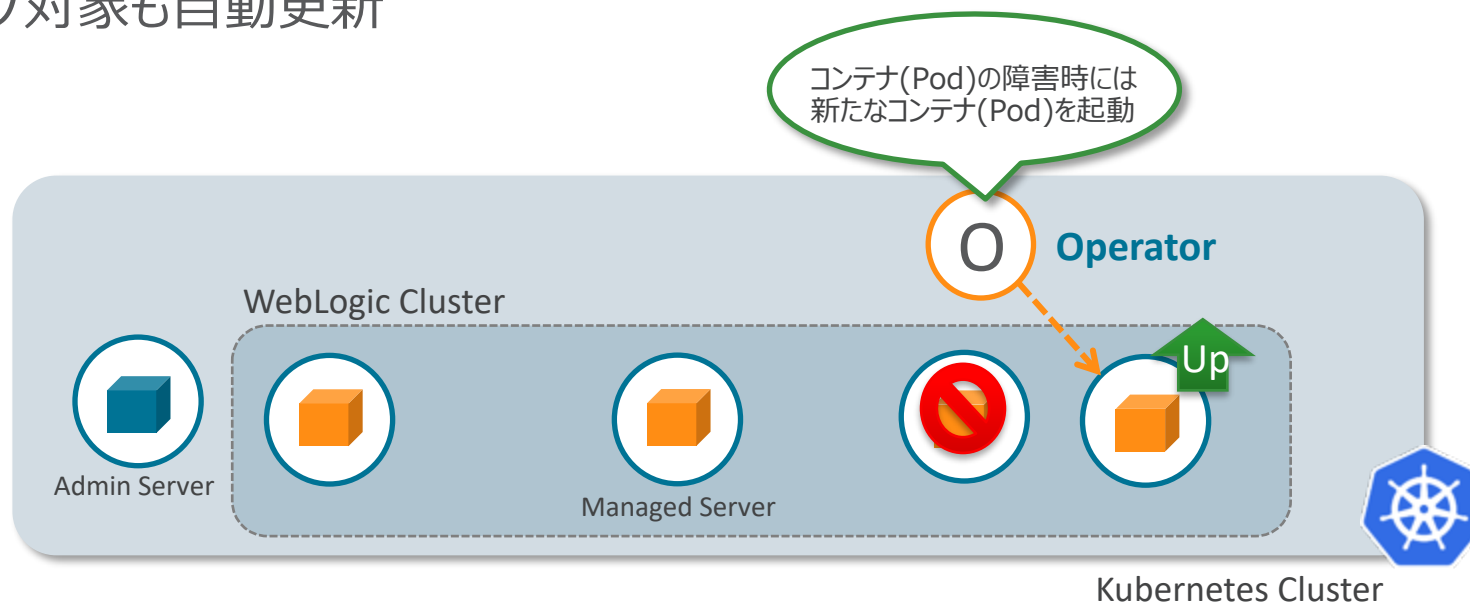


(*) GitHubにて公開:
<https://github.com/oracle/weblogic-kubernetes-operator>

WebLogic on Kubernetesの自動復旧

指定されたコンテナの数を維持し可用性構成を維持

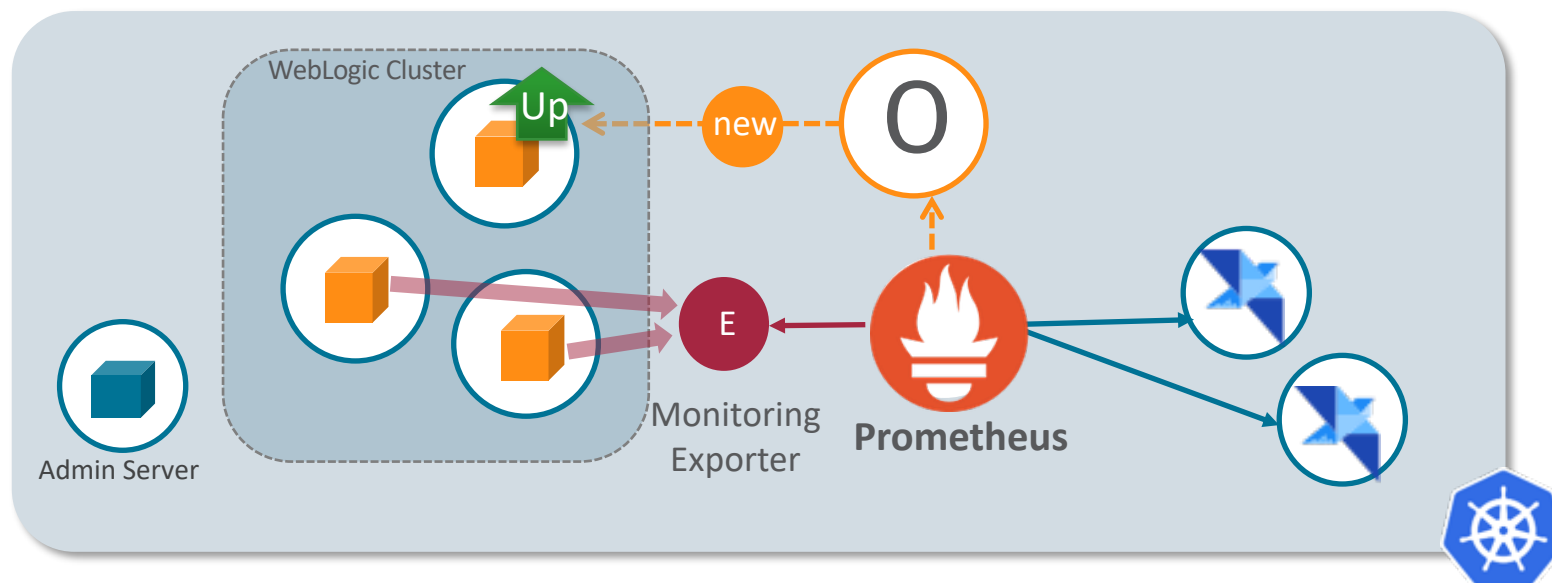
- サーバーやコンテナの障害を検知し、自動的に復旧
 - クラスタ内のコンテナの障害により停止した場合、Kubernetesのセルフヒーリング機能とOperatorによりコンテナを自動で作成・起動
 - マニフェスト (YAML) に定義した数の管理対象サーバーが常に起動するように動作
 - ルーティング対象も自動更新



Prometheusによる監視

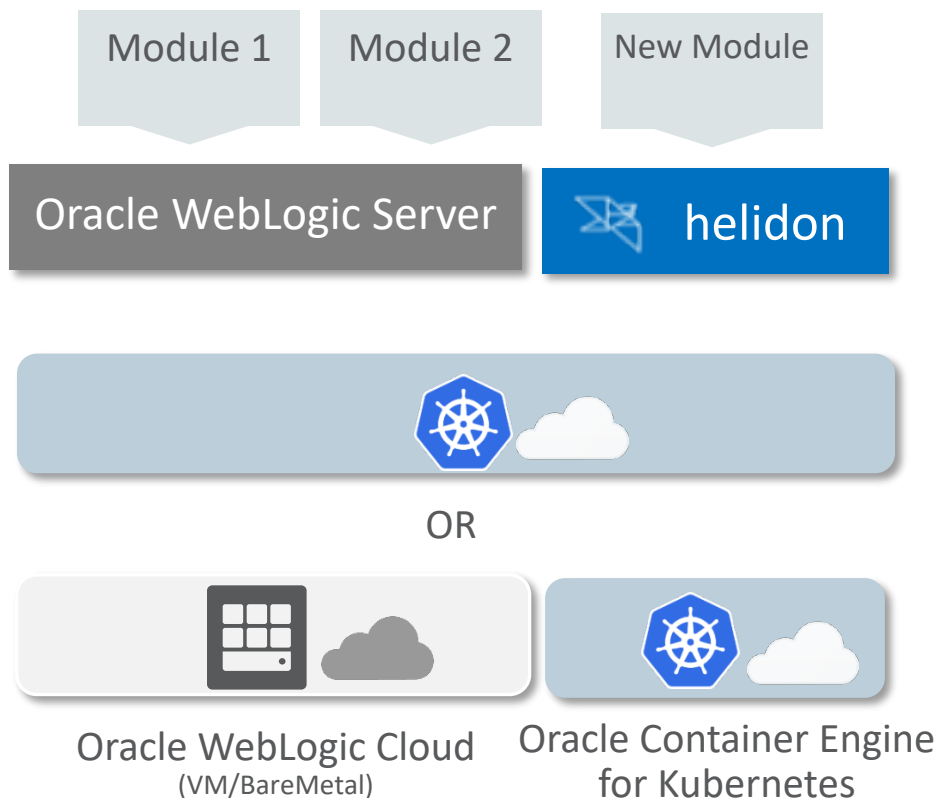
Kubernetes上のアプリケーションの監視をPrometheusにより統一化


- WebLogicのMBeanメトリックをMonitoring ExporterがPrometheusに連携
- Prometheus連携によるWebLogicメトリックに応じた自動スケーリング
 - Prometheus上で設定した閾値を元にWebLogic Operator経由でWebLogicコンテナの起動数を制御



Java EEアプリケーションに求められる変化と現実解

Oracle Cloudは継続的な変化へ対応可能な 柔軟な選択肢 を提供



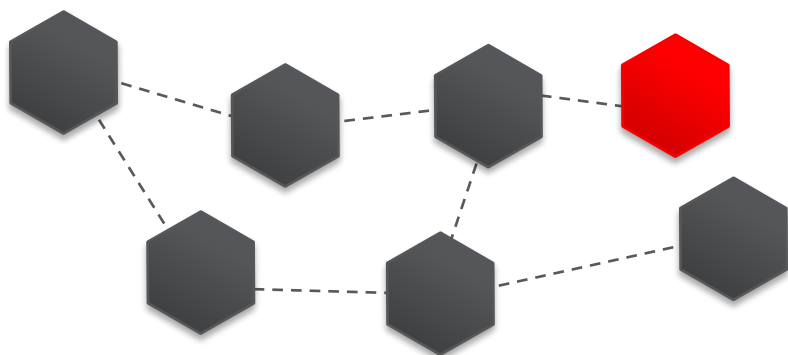
- 安定性を保ったまま、新機能の早期リリース
- アプリケーションの分割の検討
 - ⇒ 従来の**モノリシック**と高頻度リリースに対応する**マイクロサービス**を棲み分けた設計
 - Javaマイクロサービスフレームワーク**Helidon** 
- 基盤の開発、運用コストの最適化
 - ⇒ 既存WebLogicの資産を活かしつつ、**Cloud**や**Kubernetes**のメリットを取り入れて最適化

高頻度リリースを実現するための開発トレンド

アーキテクチャと開発手法の両面から高頻度リリースの取り組みが行われている

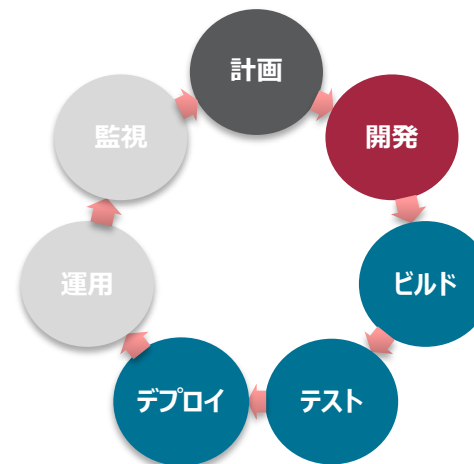
- マイクロサービス・アーキテクチャ

- 大規模なシステムを疎結合な複数のサービスの組み合わせで実現する設計方式
- 変更による影響範囲をサービス単位に極小化し高頻度のアプリケーション更新を実現



- DevOps

- 開発・運用の組織的な隔たりを無くし、サービスの継続的な更新を実現する
- CI/CDツールを用いてアプリのテストからデプロイを自動化し、迅速なリリースと品質確保を両立



Oracle Cloudを活用したDevOps

頻繁な更新を実現するための開発手法

DevOpsが登場した背景

開発チーム(Dev)と運用チーム(Ops)の対立関係が高頻度リリースの障壁になる

- 開発チーム(Dev)と運用チーム(Ops)の対立

“ Devの役割が“システムに新しい機能を追加する”である一方、Opsの役割は“システムの安定稼働”である。
そのため、Devが新しい機能を追加したくても、Opsはシステムの安定稼働のために変更を加えたがらない、という**対立構造**が作られてしまっていた。 ”

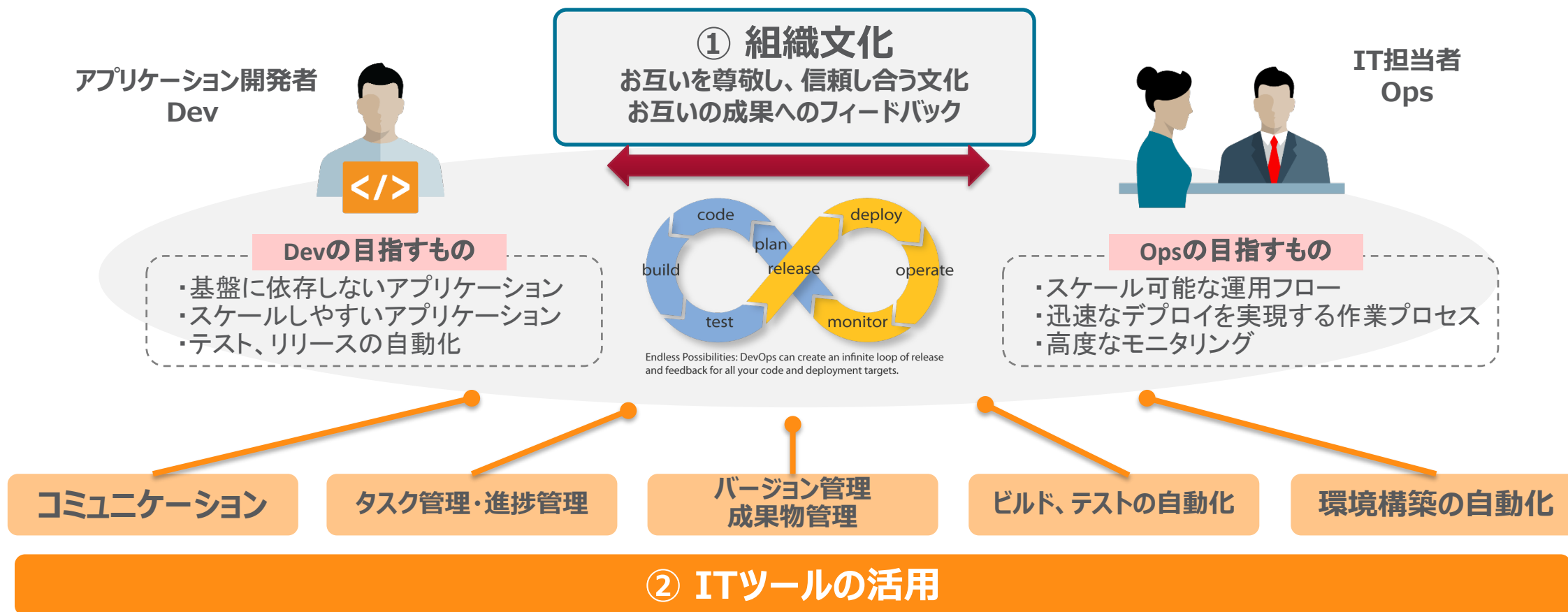
アプリケーション
の問題です！



環境の問題だ！

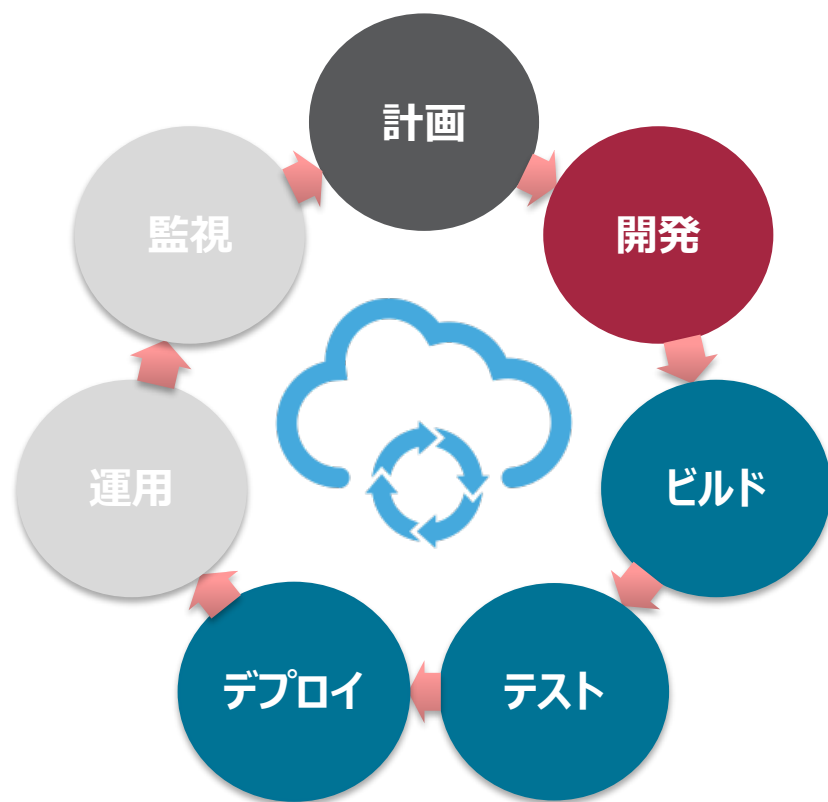
DevOpsで実現する円滑な組織運営

組織文化と様々なITツール活用の両面によって実現



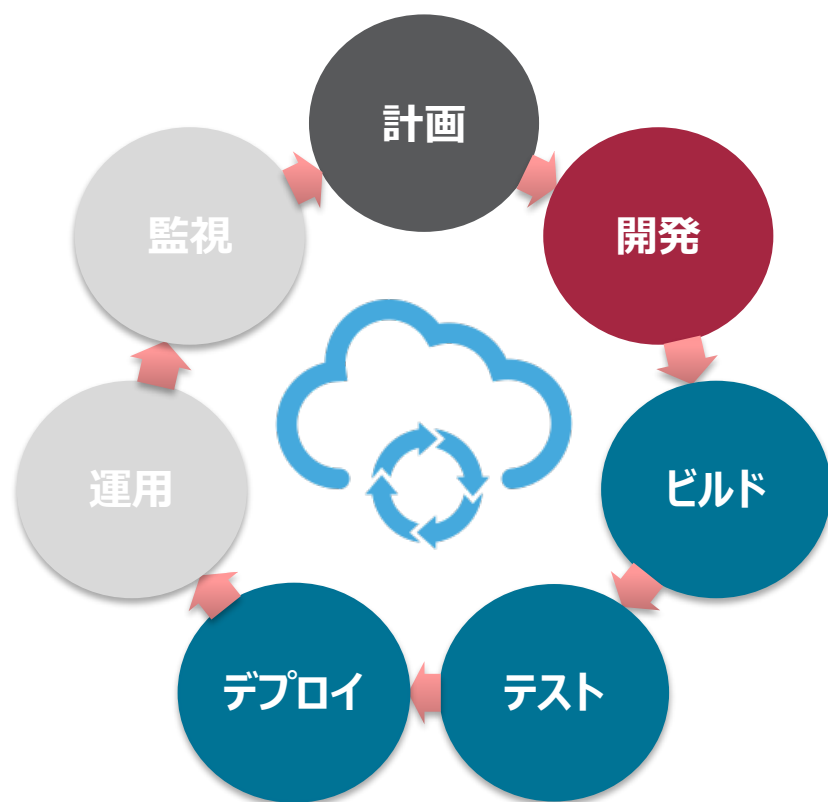
Oracle Developer Cloud Service (DevCS)

DevOpsの実現を支援する事前統合済みのクラウド型チーム開発プラットフォーム



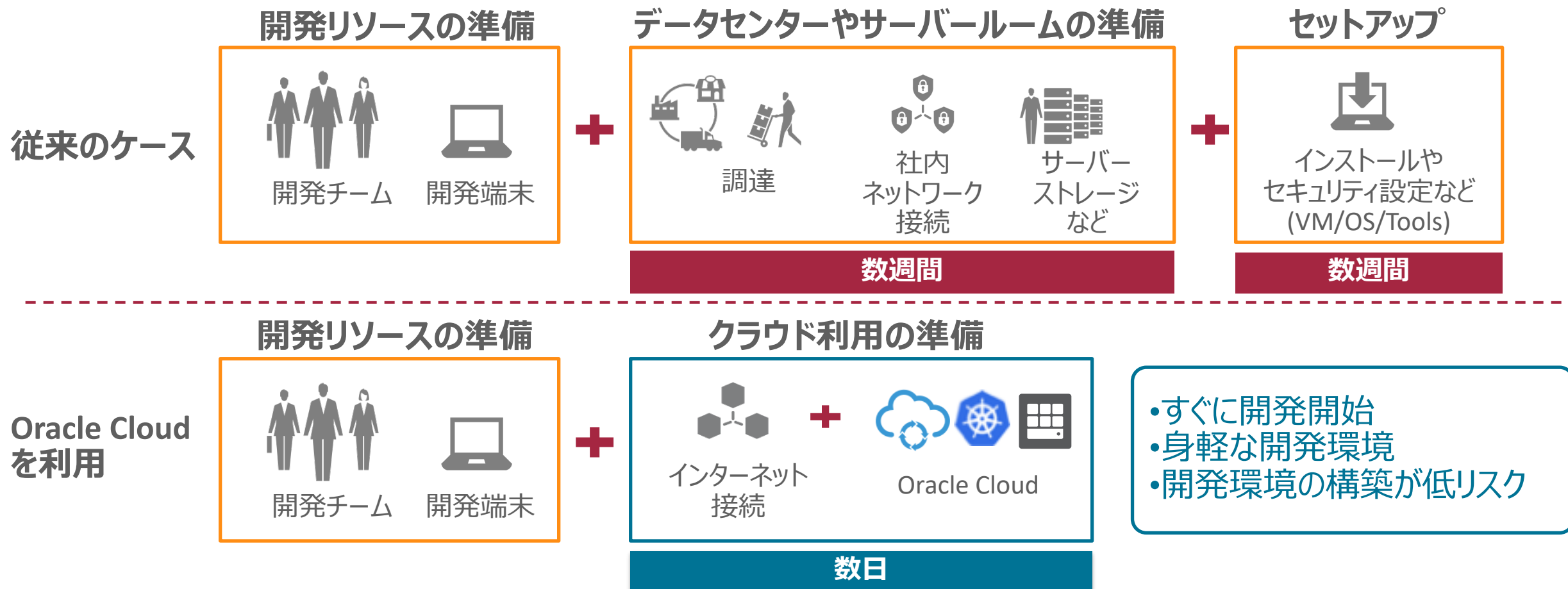
- 迅速なアプリケーション開発のために統合された開発プラットフォーム
 - コード管理 / ビルド管理 / タスク管理 などのツールを**数ステップ**で一括導入
 - 開発からデプロイまでの工程を自動化する**CI/CD**（継続的インテグレーション/デリバリ）を実現
 - Oracle Cloudにご登録の方は無料でプロビジョニング可能

Oracle Developer Cloud Serviceが提供する機能の例



- **計画 :**
タスク管理, バーンダウンチャート, かんばんボードによる可視化
- **開発 :**
Gitによるバージョン管理, コードレビュー, Mavenリポジトリ
- **ビルド :**
CI/CD Pipelineの設定, YAMLによる定義
Maven/Gradleなど一般的なビルドツールを提供
- **テスト :**
JUnit & Selenium, FindBugs, SonarQube
NVDに基づいたセキュリティ検証
- **デプロイ :**
Oracle Cloud デプロイ環境のプロビジョニング
PaaS、Kubernetes、IaaSへのデプロイ

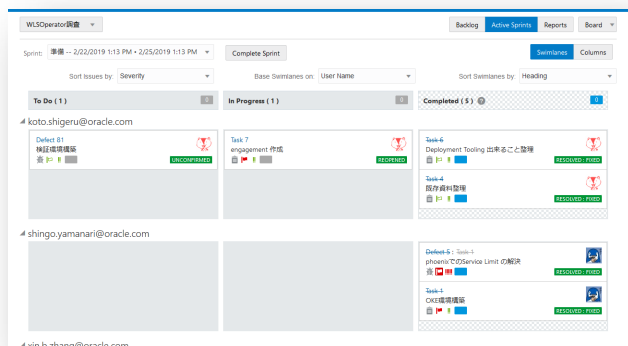
DevCSによる開発・テスト用サーバーを持たない開発 クラウド型チーム開発プラットフォーム、準備が速い・低リスク・身軽な開発環境



DevCSによる高頻度リリース開発

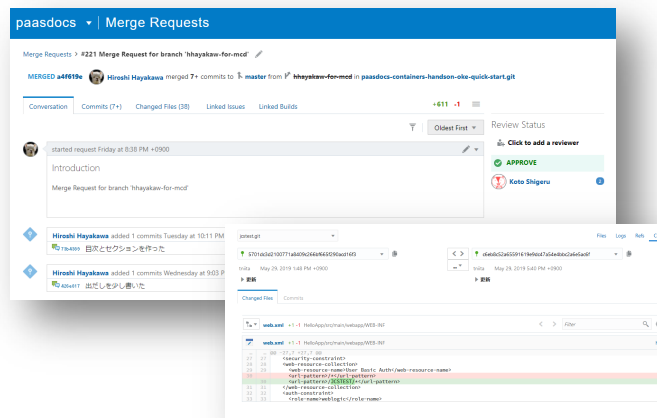
・タスク・進捗管理

- Issue機能によるタスク管理
- バーンダウンチャート、かんばんボードによる可視化
- スクラム開発、スプリントの作成



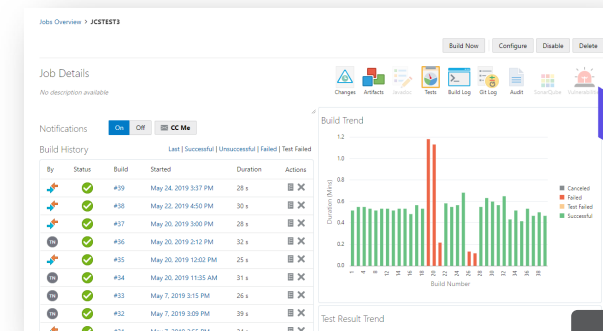
・開発・実装

- Gitコードリポジトリによるコード管理
- オンラインレビューツール
- 主要IDEとの連携



・CI/CD

- 継続的インテグレーションツールでの自動ビルド・テスト
- 環境の自動構築
- 継続的デリバリー自動リリース



Developer Cloud Serviceでシームレスな開発を実現

タスク・進捗管理

JETProject ▾ | Issues Search Issues

Advanced Searches | Open issues Clear Selection Select All Update Selected + New Issue

Standard Searches

- All Issues
- Assigned to me
- Open issues**
- Recently changed
- Related to me

My Searches

- High Priority Status

Shared Searches

Global Searches

Open issues

Page size: 20 Refresh

!	!	Type	ID	Summary
!	!	査	5	Wrong title for header page
!	!	!	1	Load initial code to repository
!	!	!	2	Need microservices for vacati
!	!	!	3	Add NBox employee rating co
!	!	査	4	Missing REST service to query
!	!	査	21	Fix dashboard title
!	!	査	75	Failed Test

Page 1 of 1 (1-7 of 7 items) < 1 >

かんばんボードによる可視化

WLSOperator調査 ▾ Backlog Active Sprints Reports Board ▾

Sprint: 準備 -- 2/22/2019 1:13 PM • 2/25/2019 1:13 PM Complete Sprint

Sort Issues by: Severity Base Swimlanes on: User Name Sort Swimlanes by: Heading

To Do (1) 0 In Progress (1) 0 Completed (5) ? 0

◀ koto.shigeru@oracle.com

Defect 81
検証環境構築 UNCONFIRMED

Task 7
engagement 作成 REOPENED

Task 6
Deployment Tooling 出来ること整理 RESOLVED : FIXED

Task 4
既存資料整理 RESOLVED : FIXED

◀ shingo.yamanari@oracle.com

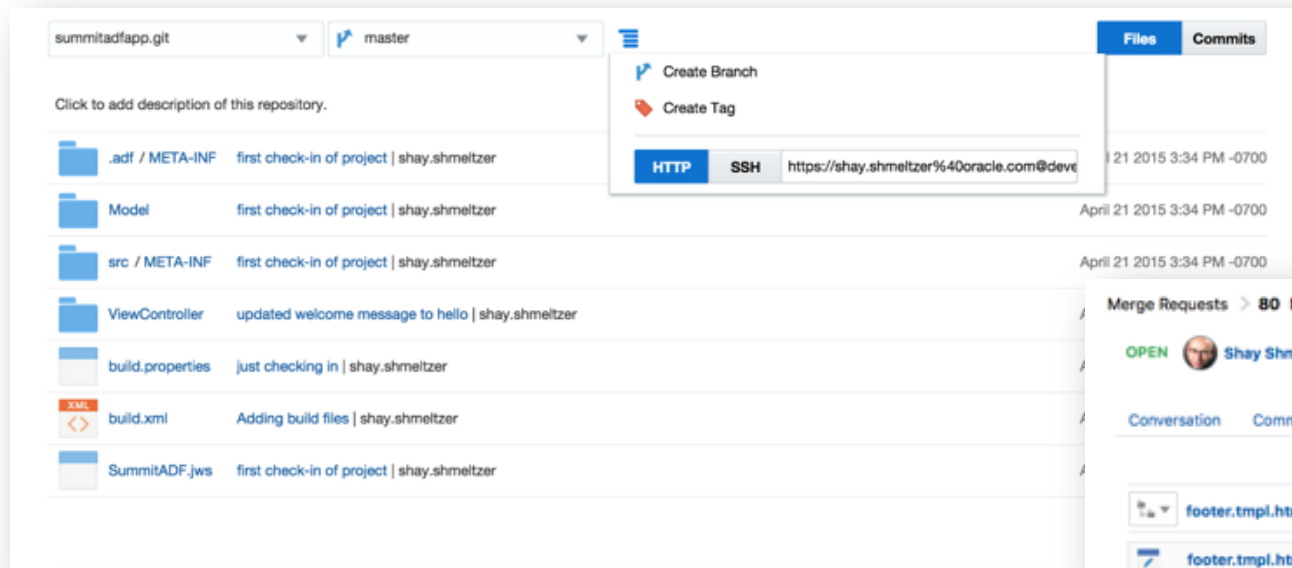
Defect 5 : Task 4
phoenixでのService Limit の解決 RESOLVED : FIXED

Task 4
OKE環境構築 RESOLVED : FIXED

◀ xin.b.zhang@oracle.com

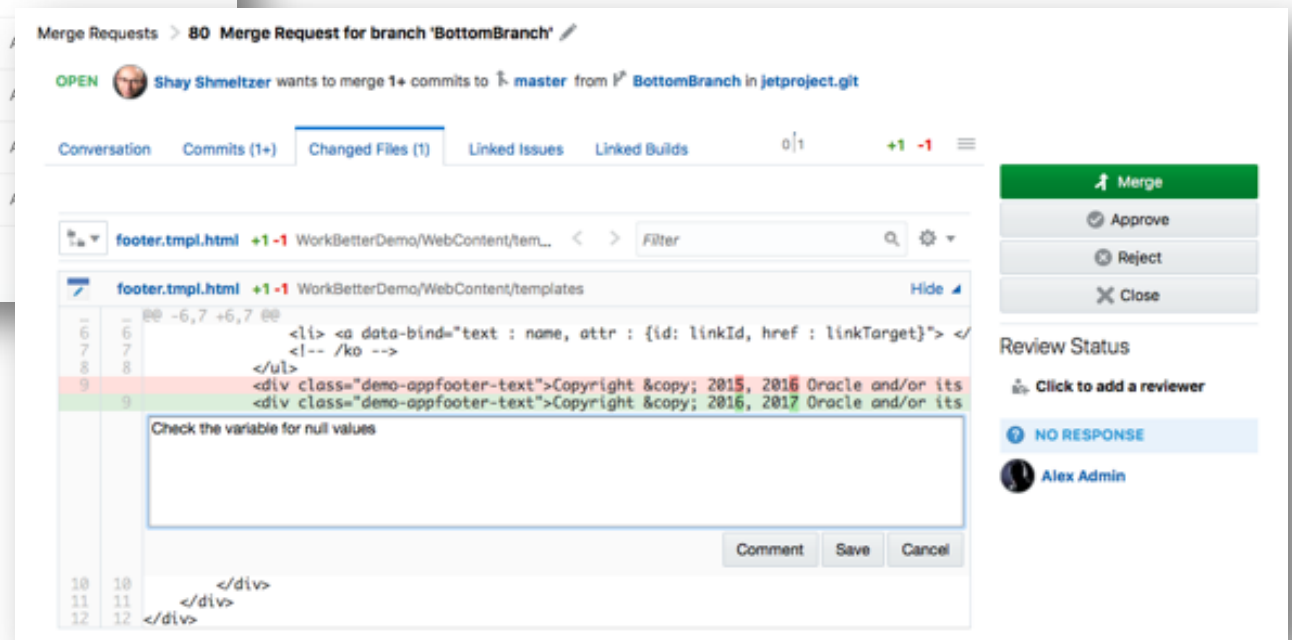
Issue、タスク管理

実装



Gitコードリポジトリによるコード管理

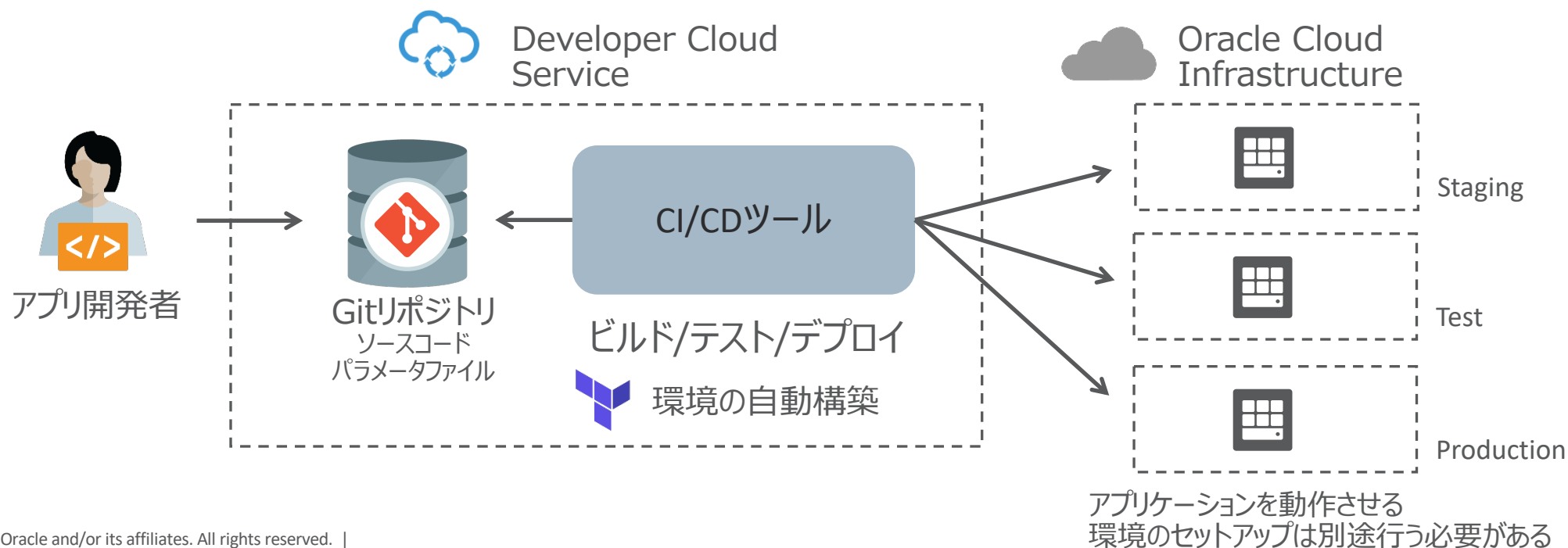
マージリクエスト



CI/CD(継続的インテグレーション/継続的デリバリー)

開発、運用プロセスにおける高頻度リリースのための取り組み

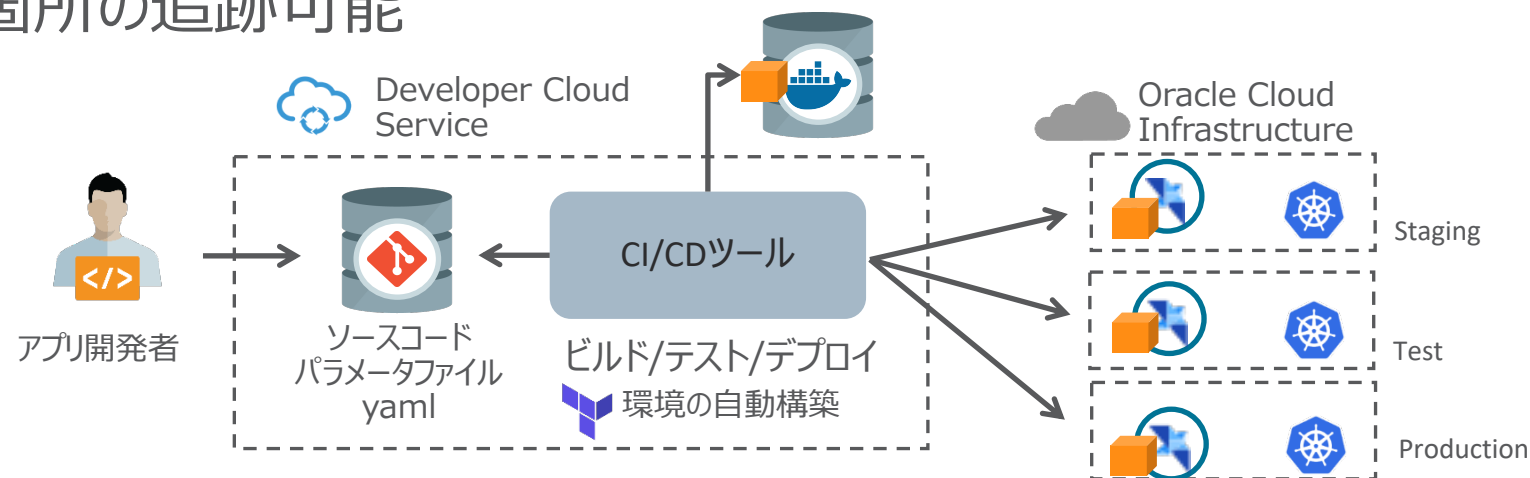
- ソースコードの変更から本番環境へのリリースまでの流れを極力自動化
- 人手をかえさないテスト・リリースにより、システムの品質と安全な高頻度リリースを実現する



コンテナとKubernetes登場後のCI/CD

テスト・運用環境の構築が省力化されCI/CDを実現しやすくなった

- コンテナの可搬性
 - アプリケーションに与える設定、依存ライブラリなどの状態を小容量のコンテナイメージとしてパッケージング、多数の環境に容易にアプリケーションを展開可能
- Kubernetesの宣言的オペレーションと環境のコード化
 - 実行環境の構成をコード（YAML）に記述することで、複数クラスターに同等の環境を容易に構築、また変更箇所の追跡可能



Oracle Container Engine for Kubernetes (OKE)

- Oracle Cloud Infrastructureの機能として提供される
マネージドKubernetesサービスとコンテナレジストリサービスを提供



- Oracle Container Engine for Kubernetes (OKE)
 - マネージドKubernetesサービス

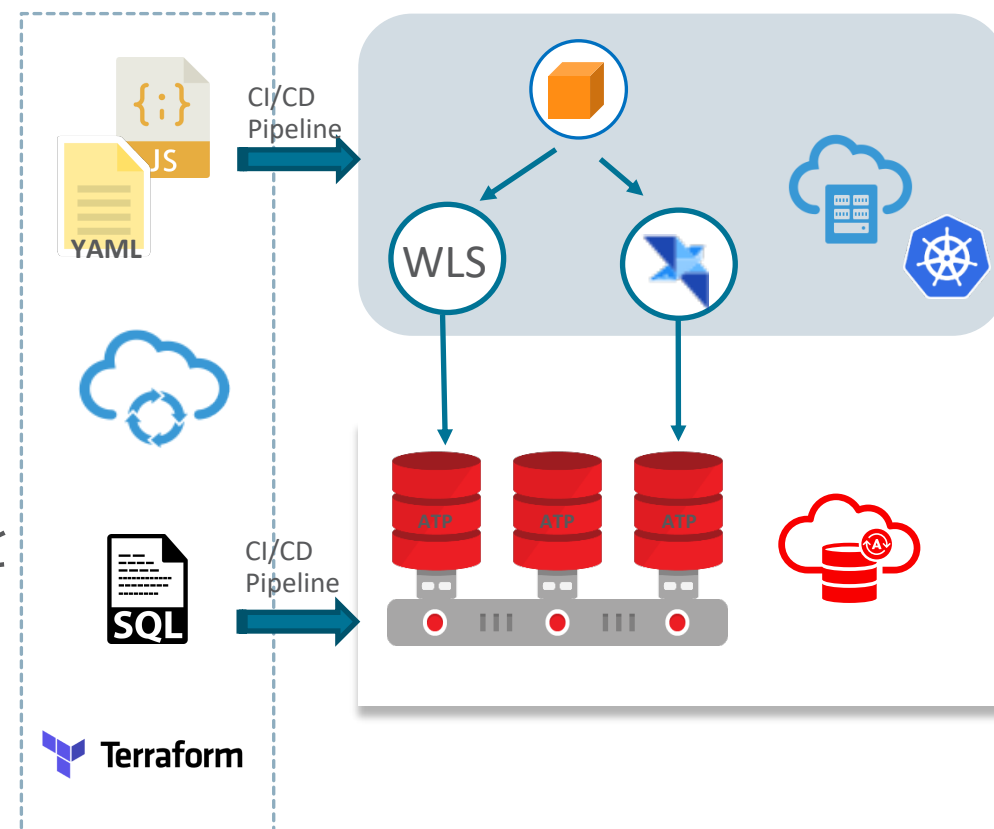


- Oracle Cloud Infrastructure Registry (OCIR)
 - マネージドなコンテナレジストリ

Autonomous Databaseとの組み合わせ

ATPをKubernetesと組み合わせることで開発効率を最大化

- Developer Cloud ServiceのCI/CDツールの利用により、データベース環境構築からアプリケーションのデプロイまで完全に自動化
 - 削除も簡単に実施可能
- 環境構築・運用作業を削減し、開発効率を向上
 - OKEにより複数コンテナで動作するアプリケーションを自動管理可能
 - ATPによりエンタープライズレベルのパフォーマンスをDBの管理、保守作業なしに実現可能



振り返り

【開発トレンド】高頻度リリースを実現する2つのアプローチ

- 頻繁なアプリケーションの更新に対応するためのマイクロサービスアーキテクチャ
- DevOps により短期間での開発と高頻度リリースを実現

【Java EEアプリ】最適化ステージに合わせた最適なアーキテクチャの選択

- 従来の**モノリシック**と高頻度リリースに対応する**マイクロサービス**を棲み分けた設計
JavaマイクロサービスフレームワークHelidon
- 既存WebLogicの資産を活かしつつ、**Cloud**や**Kubernetes**のメリットを取り入れ環境管理の高度化
Oracle WebLogic Cloud/WebLogic on Kubernetes+WebLogic Operator

【DevOps】Oracle Cloudを活用したアプリケーション開発手法のモダナイズ

- **Developer Cloud Service**を活用した**DevOps**の実現
- **DevCS/OKE/Autonomous**を活用した**継続的インテグレーション/デリバリー**

こんな時、かけこむ会社が増えています。



ビジネスプロセスを
改善したい!



今のシステムは
使いにくい!



システムコストを
下げたい!



パフォーマンスを
良くしたい!



経営分析を
したいのだが...



どんなソリューションが
あるの?



見積りはどれくらい
なんだろう?



楽に管理を
したい!

Oracle Digitalは、オラクル製品の導入をご検討いただく際の総合窓口。
電話とインターネットによるダイレクトなコミュニケーションで、どんなお問い合わせにもすばやく対応します。
もちろん、無償。どんなことでも、ご相談ください。



お問い合わせは電話またはWebフォーム

☎ 0120-155-096

受付時間 月～金 9:00-12:00 / 13:00-17:00
(祝日および年末年始休業日を除きます)

<http://www.oracle.com/jp/contact-us>

ORACLE®