redhat

実践 DevOps

- Red Hat DevOps Discovery Workshop –

Yoshikazu YAMADA <yyamada@redhat.com>
Red Hat K.K. DevOps Lead Senior Architect

Agenda

I.導入編

II.実践編

III.テクニカル編

IV.成熟度判定

redhat.

I.導入編

redhat.

DevOps の起源

Back to the origin of DevOps

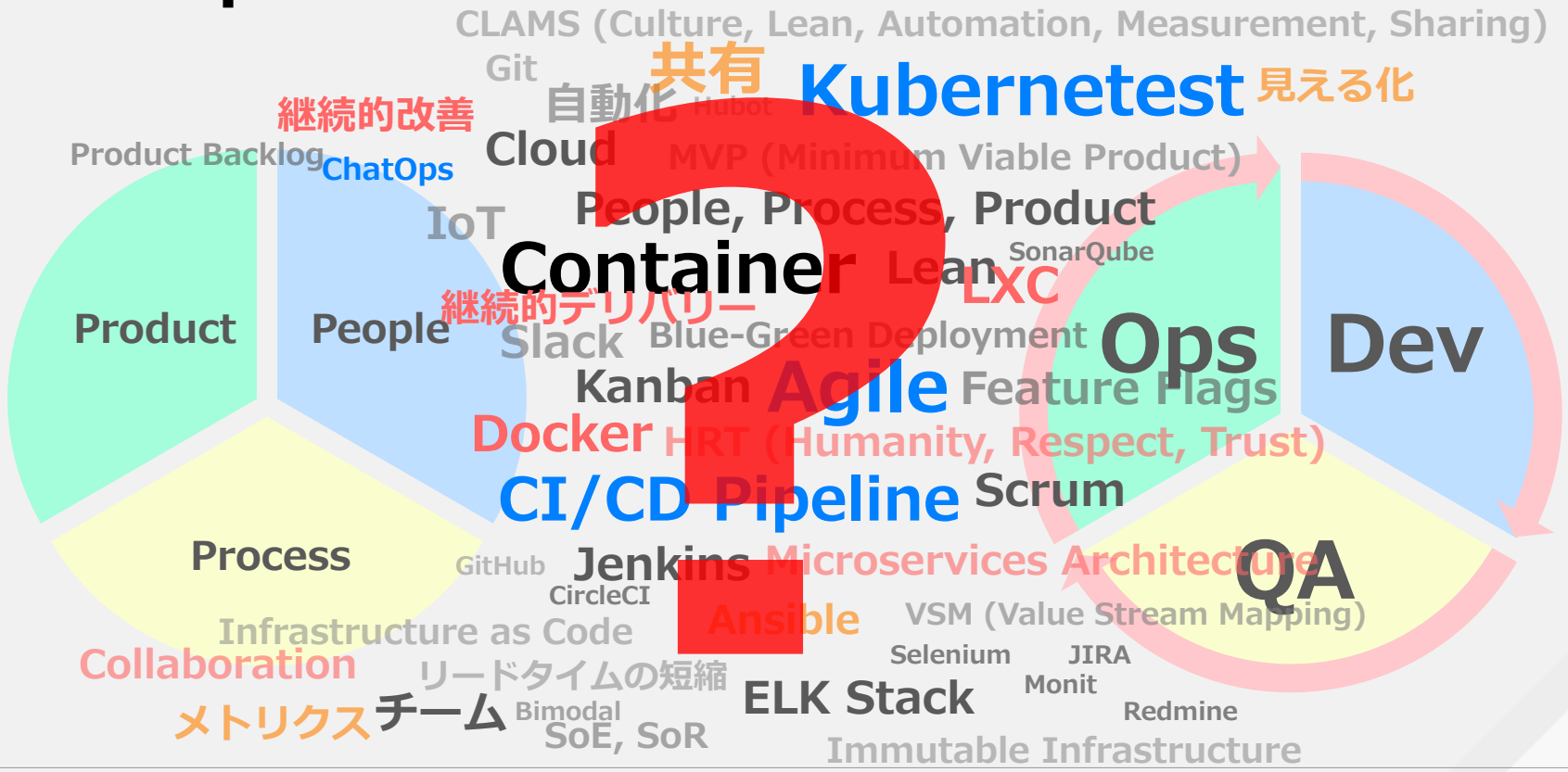
10+ Deploys Per Day: Dev and Ops Cooperation at Flickr [1]

1. Automated Infrastructure
2. Shared version control
3. Feature flags
4. Shared metrics
5. IRC and IM robots

1. Respect
2. Trust
3. Healthy attitude about failure
4. Avoiding Blame

redhat.

DevOps?



redhat.

DevOps?

- 目的は何なのか？
- どんな知識が必要なのか？
- 具体的に何を実践すれば良いのか？
- どのように始めれば良いのか？

redhat.

Agenda – 導入編

- DevOps 周辺の歴史
- DevOps と ビジネス
- DevOps と IT 運営プロセス
- DevOps と 技術・ツール
- DevOps の 目的 と 主要成功要因
- Red Hat DevOps Discovery Workshop による DevOps の実践モデル
- Workshop 実施に向けたディスカッション

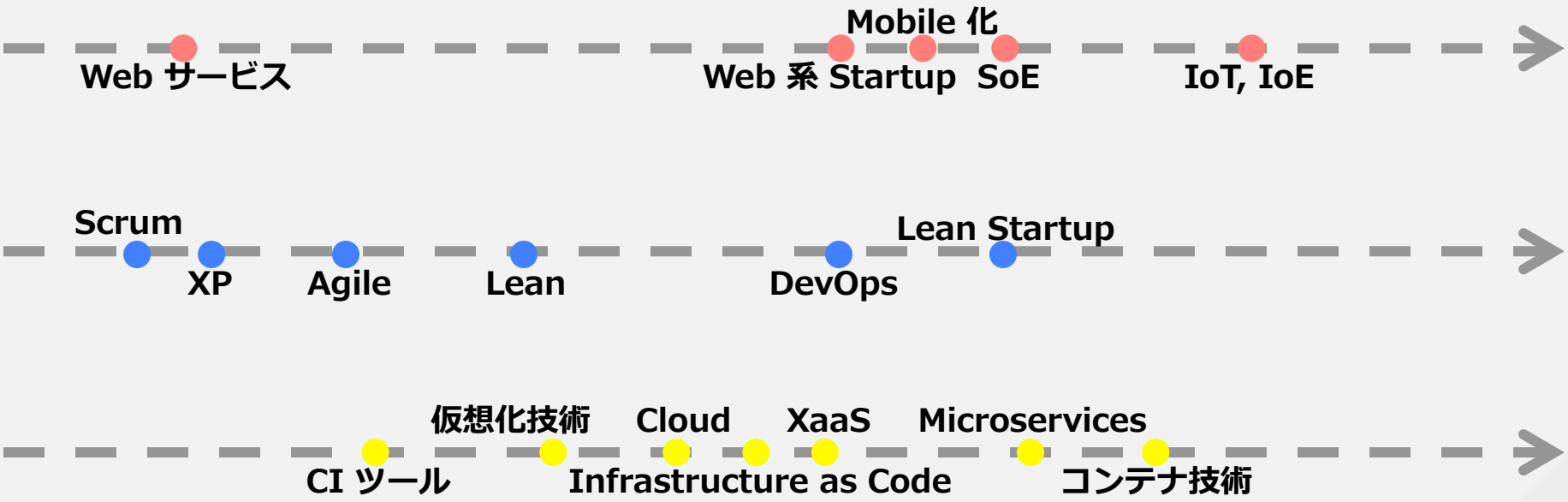
redhat.

DevOps 周辺の歴史

redhat.

DevOps 周辺の歴史

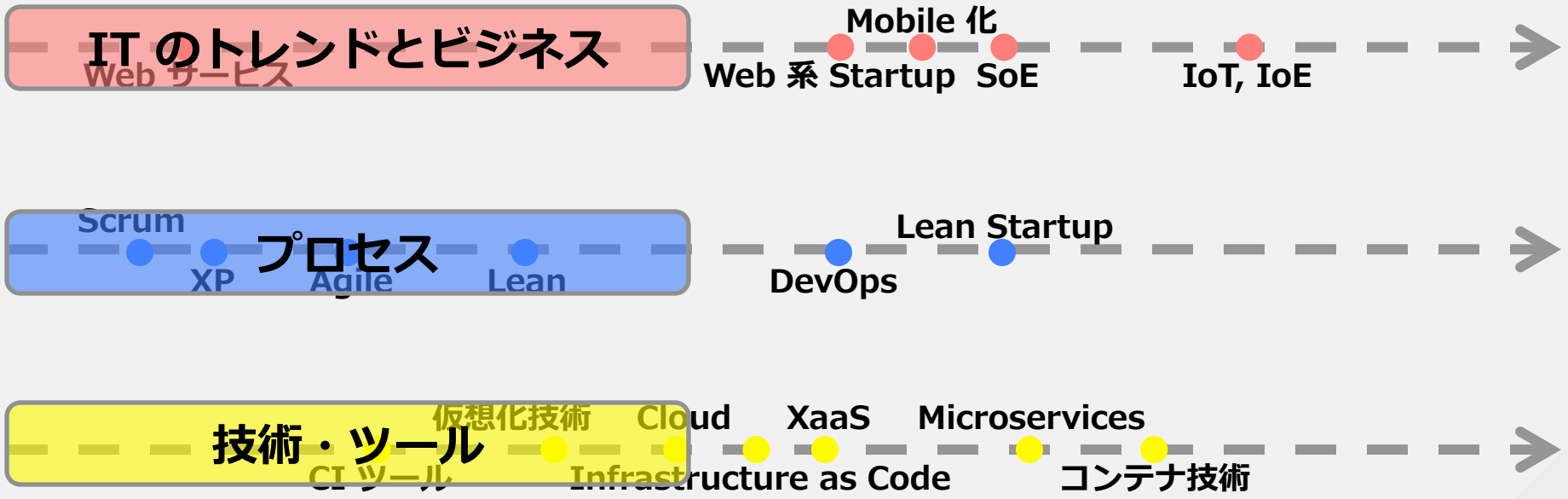
History around DevOps



redhat.

DevOps 周辺の歴史

History around DevOps



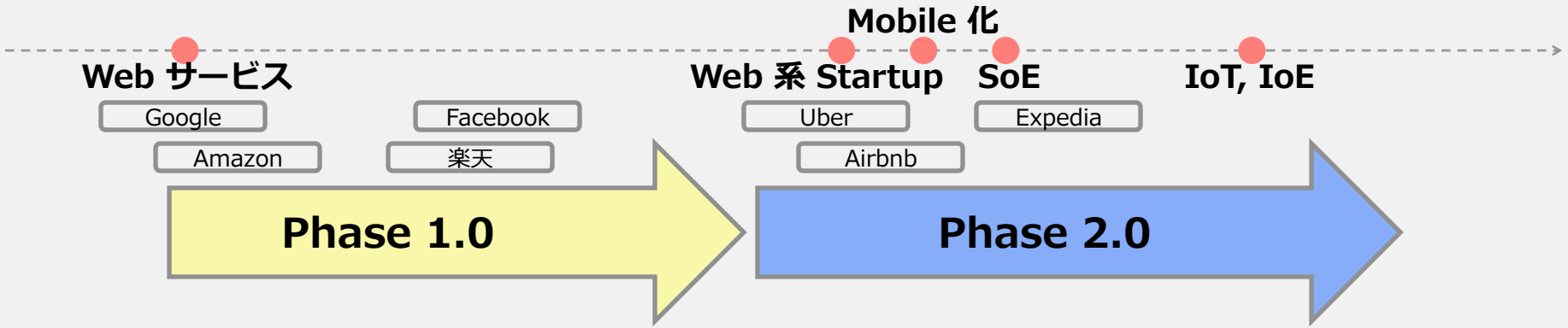
redhat.

DevOps と ビジネス

redhat.

web と ビジネス

IT と ビジネス の 関係 の変化



Phase 1.0

- 既存ビジネスの web への移行

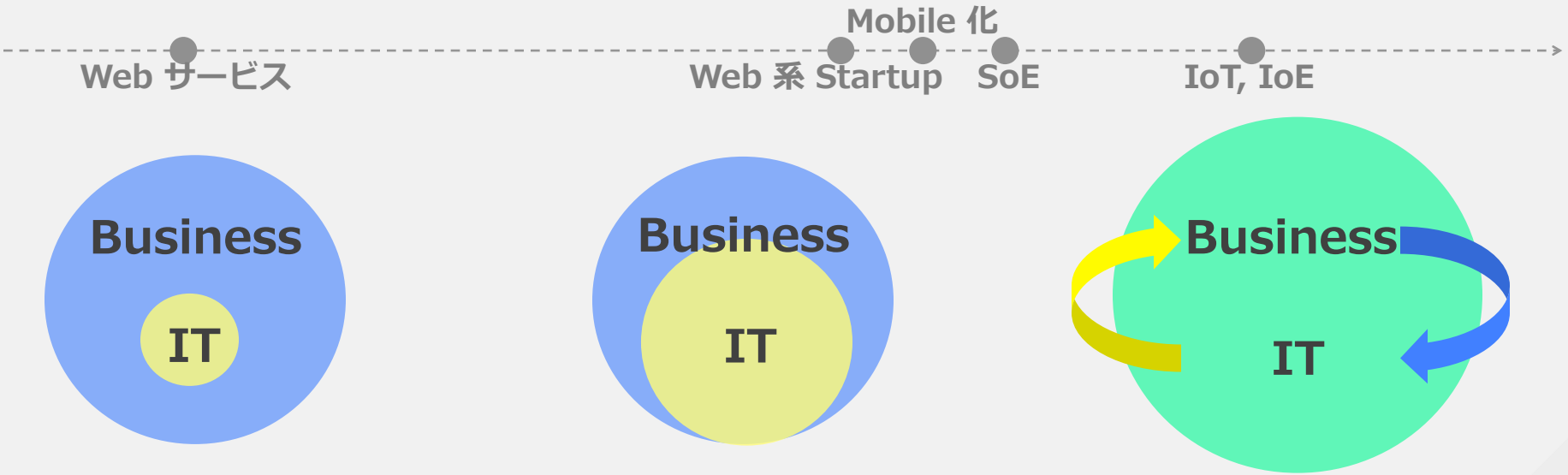
Phase 2.0

- web 上でのビジネスの拡大・多様化・創造・差別化

redhat.

IT と ビジネス

IT と ビジネス の 関係 の変化



redhat.

IT と ビジネス の関係

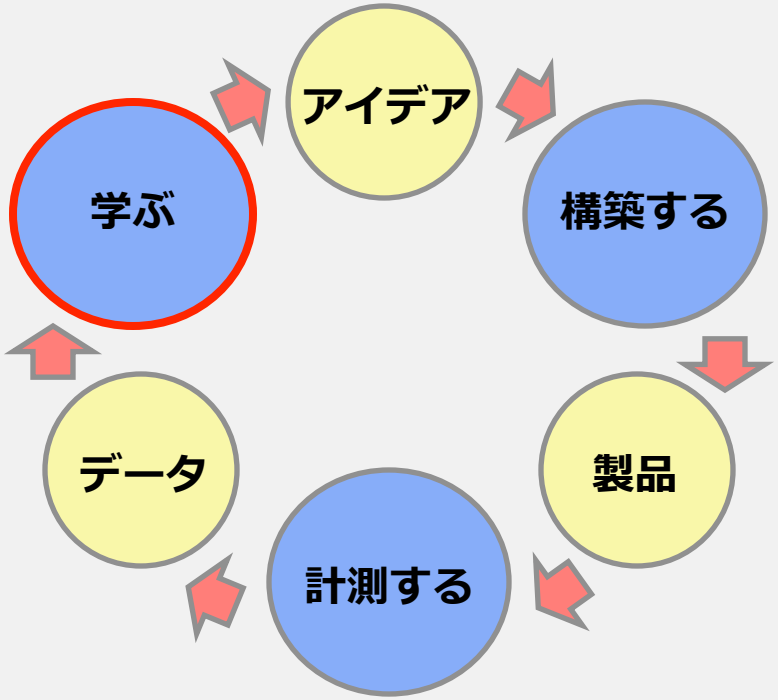
IT と ビジネス の 関係 の変化

- IT によるビジネス領域の拡大
- IT によるビジネスの多様化・創造・差別化

- IT が**ビジネスの価値**に直結する領域の拡大と競争の激化
- IT が**ビジネスの成功**に大きく起因

redhat.

IT ビジネス のフィードバックループ



構築・計測・学習 ループに要する時間を最小化

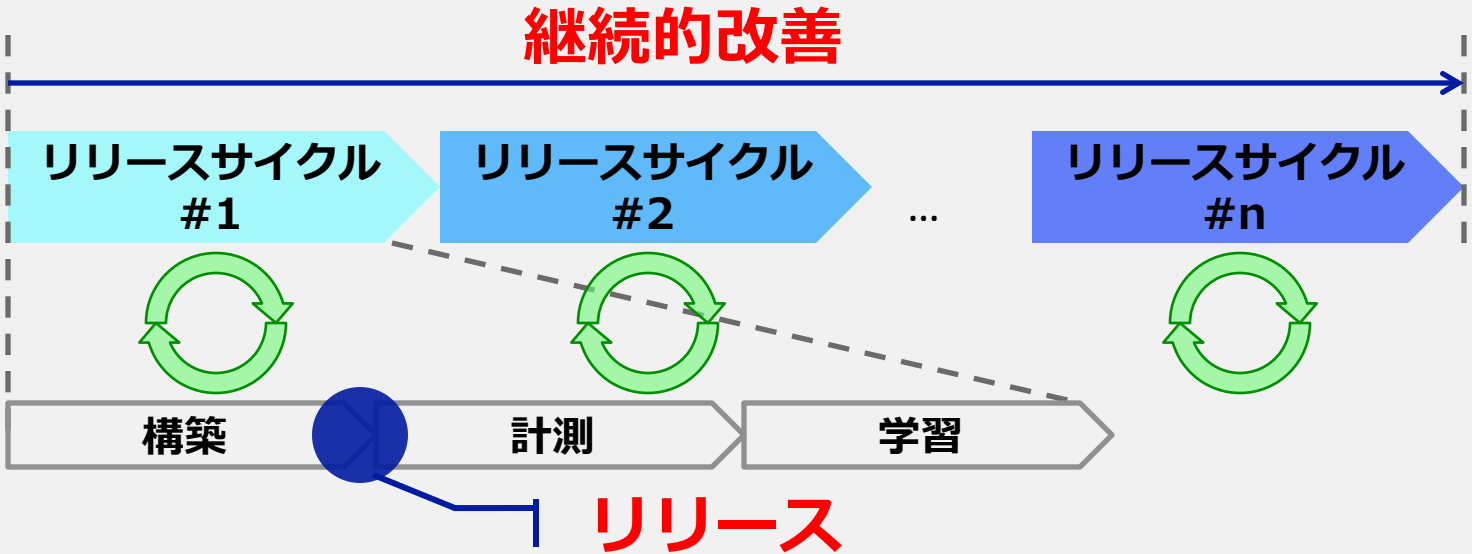
改善・変化への対応の高速化

改善・変化への対応の機会の増加

プロダクト・サービスの価値の向上

redhat.

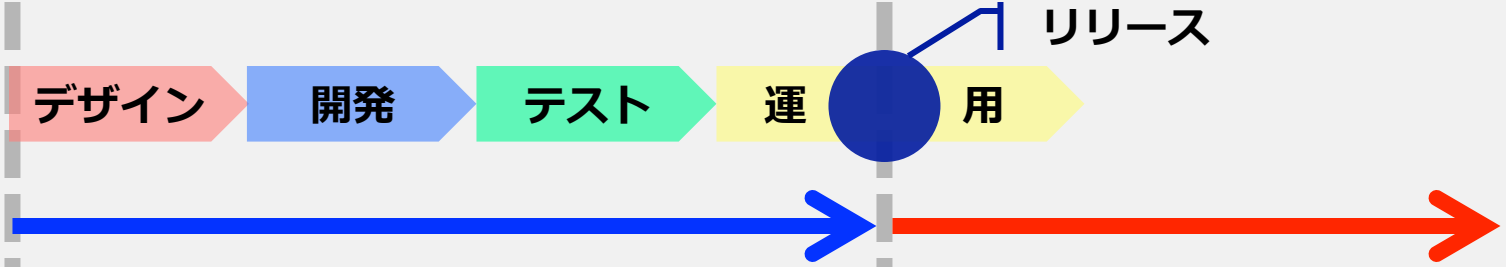
フィードバックループと IT サービス運営



redhat.

DevOps な IT サービス運営 – リリース

IT 運営の考え方



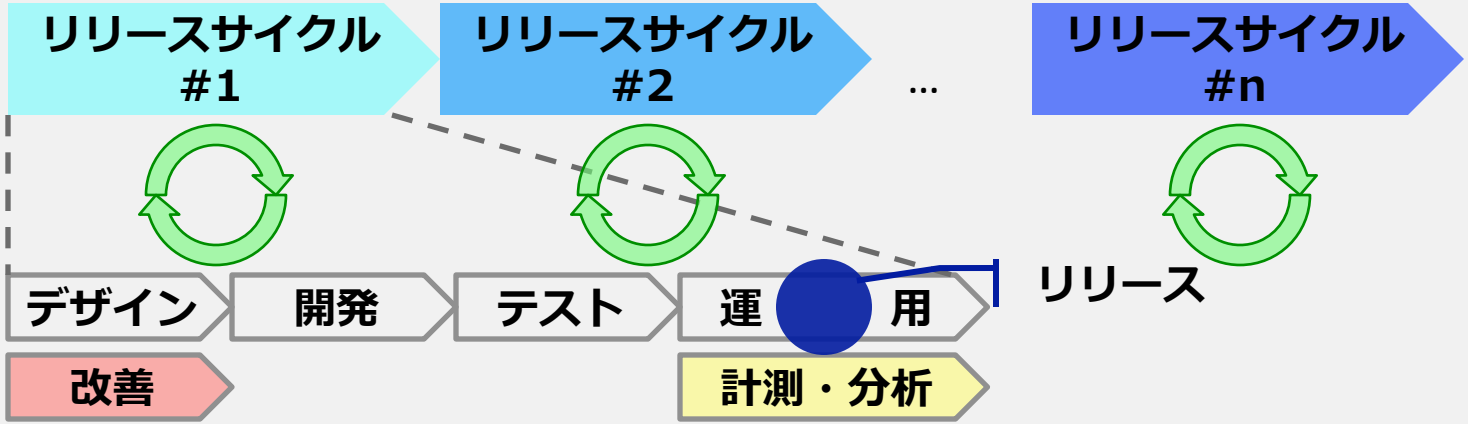
- アイデアを早く実現
- 変化への早い対応

- プロダクト・サービスの価値
- 改善のための学習機会

redhat.

DevOps な IT サービス運営 – 継続的改善

IT 運営の考え方



- サイクル数 を増やし 分析・改善・学習 の機会を得る
- サイクル数 を増やし変化に対応する機会を得る

redhat.

DevOps の目的

- ビジネスの価値・競争力の向上・成長
- ビジネスニーズの変化に迅速に対応

- より早くサービスをリリースする
- より多くフィードバックをし継続的にサービスを改善

redhat.

DevOps は

ビジネスの価値・競争力の向上・成長

を目的とした

リードタイム短縮 および 継続的改善

のための IT 運営プラクティス

redhat.

DevOps 理解のポイント

- ~~既存の IT 運営の何を変えなければならないか？~~
- ~~既存の IT 運営をベースに DevOps は必要か？~~

- リードタイムを短縮し継続的な改善を行うとビジネスの価値がどのように向上するか？

redhat.

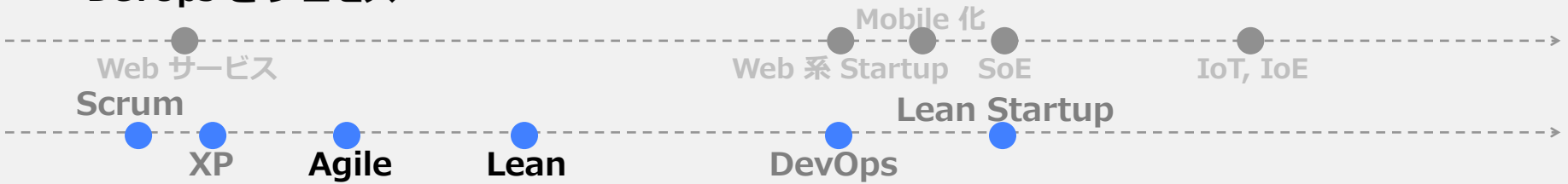
DevOps と IT 運営プロセス

DevOps ≒ Agile, Lean

redhat.

DevOps 周辺の歴史

DevOps とプロセス



Agile

- 顧客満足を最優先
- 価値のあるソフトウェアを早く継続的に提供
- 要求の変更を歓迎
- 短い時間間隔でリリース
- 持続可能な開発を促進
- 定期的な振り返りと最適な調整

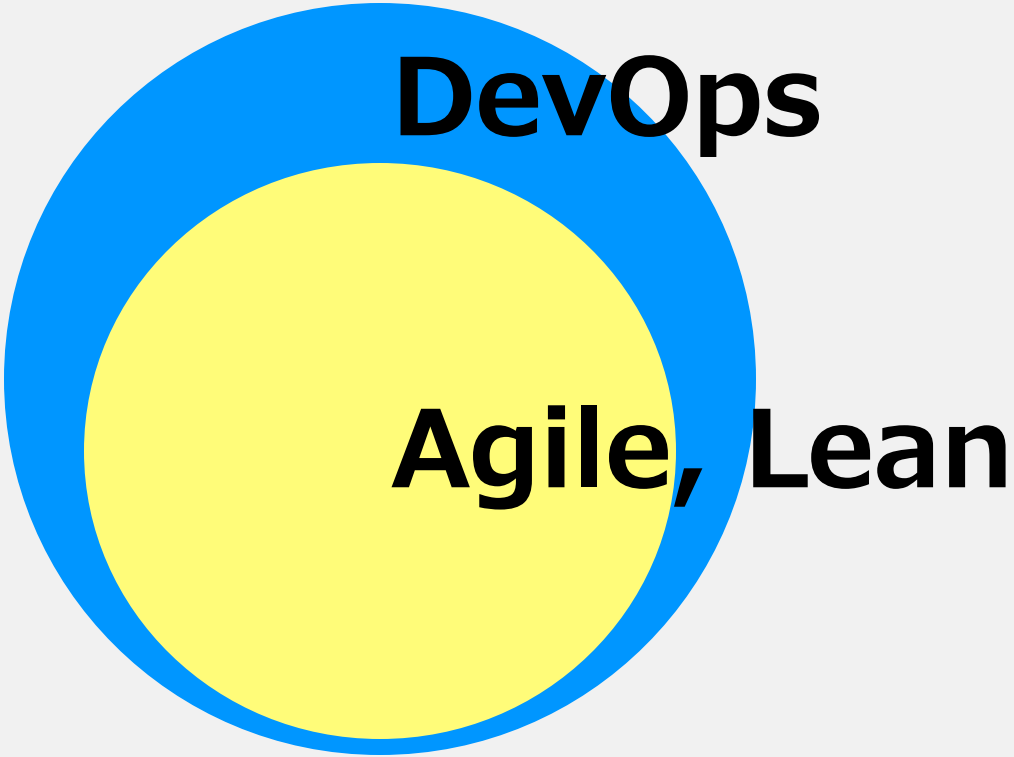
※ アジャイルソフトウェア開発の 12 の原則 [8] の一部抜粋

Lean

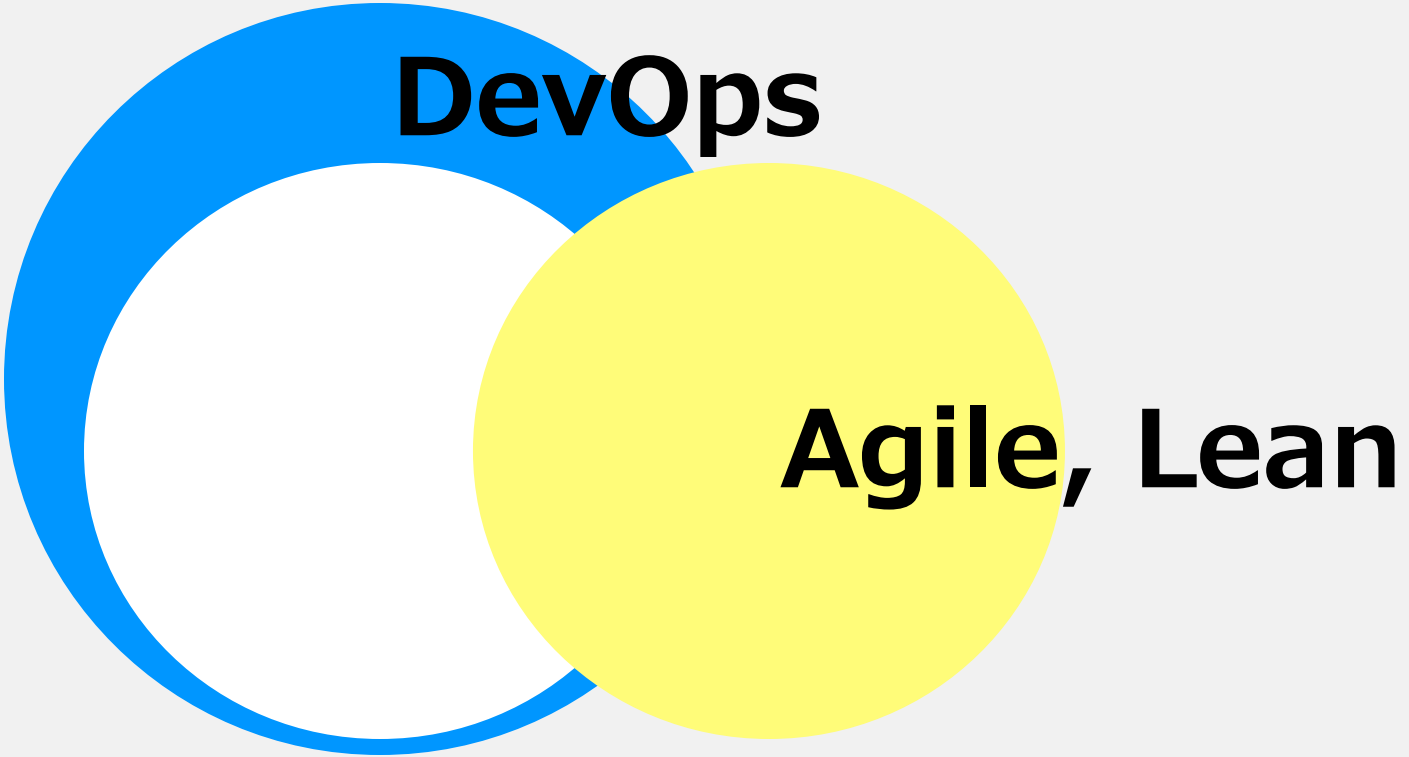
- 顧客重視
- 知識の創造
- 早く提供 (リードタイムの短縮)
- 全体最適化
- フィードバック
- イテレーション

※ リーンソフトウェア開発から抜粋 [4], [14]

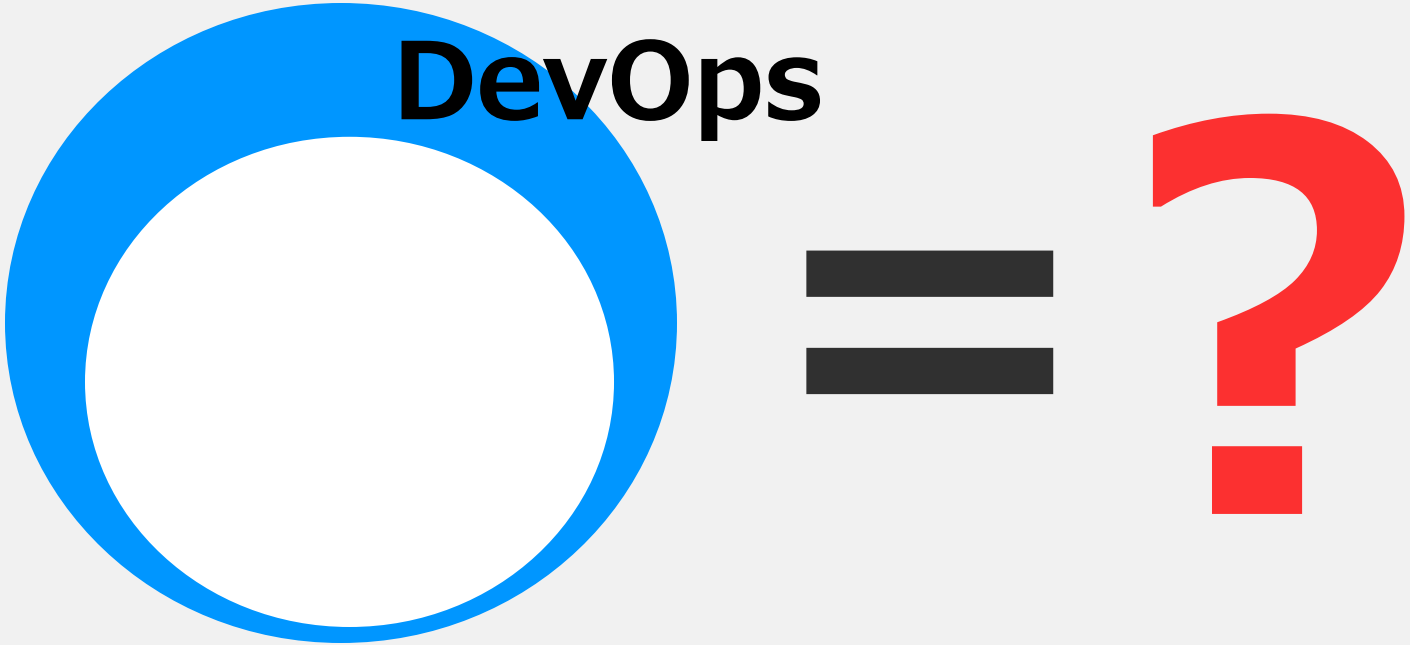
redhat.



redhat.

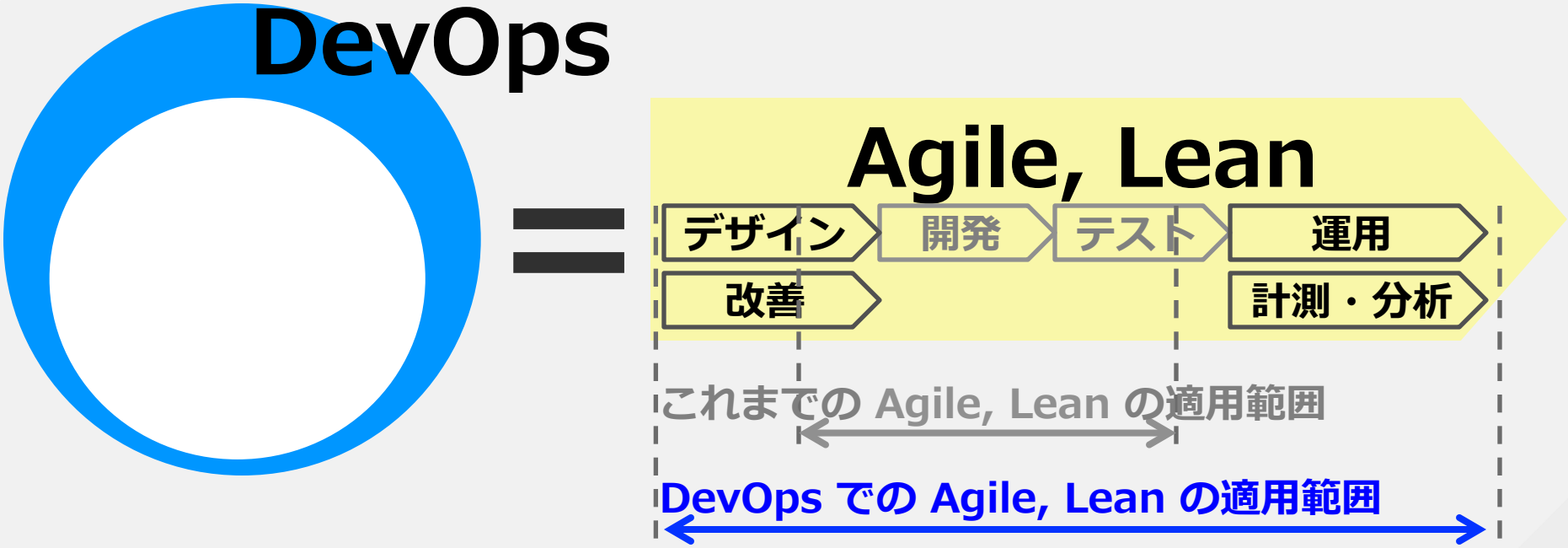


redhat.



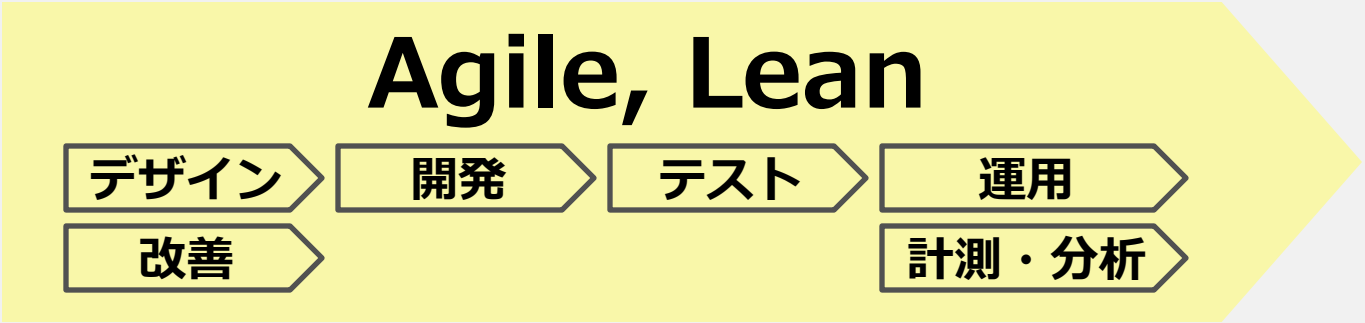
redhat.

DevOps



redhat.

DevOps の本質
プロセス

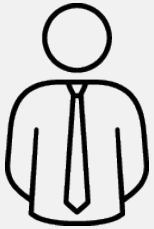


Agile, Lean の IT 運営プロセス 全体への適用

Agile, Lean

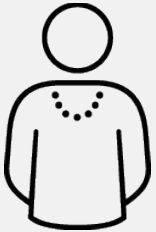
デザイン → 開発 → テスト → 運用

改善 ← 計測・分析



Biz

Dev

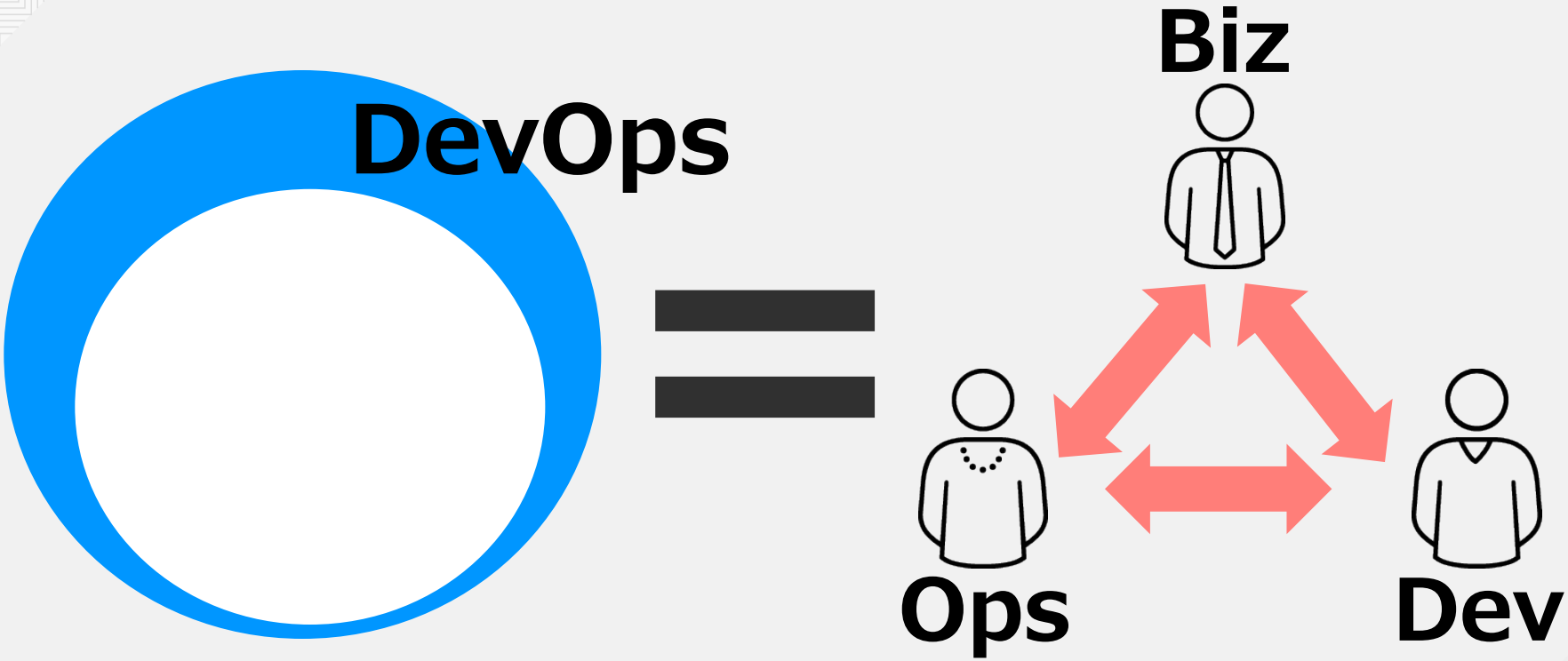


Ops

redhat.

DevOps の本質
プロセス

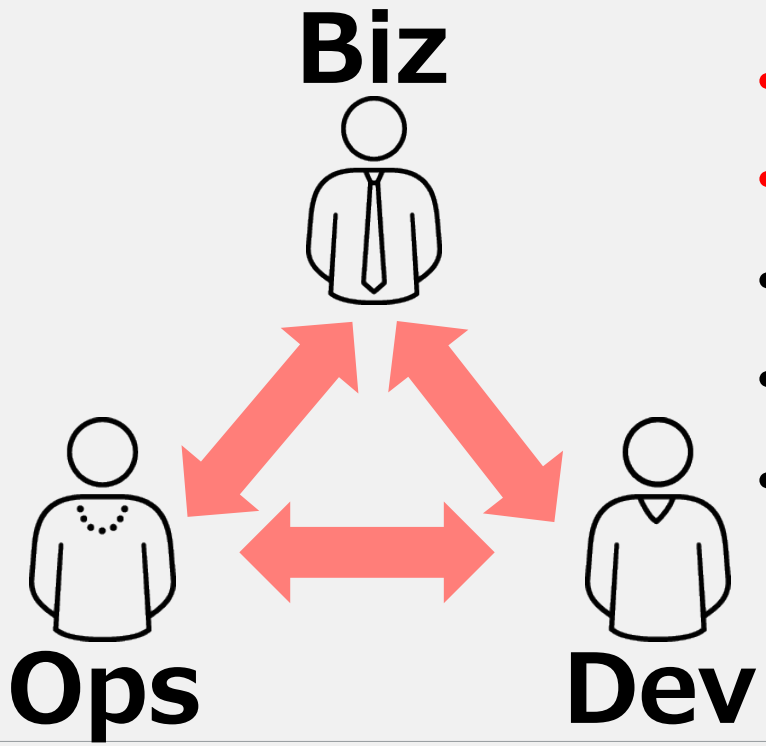
- **Biz, Dev, Ope 横断的な継続的改善活動**
- **Agile, Lean の Biz, Dev, Ops 横断的な実践**



redhat.

DevOps の本質

人・文化



- ビジネスファースト
- 変化・改善に前向きなマインド
- Biz, Dev, Ops の協調
- 協調のためのマインド
- Biz, Dev, Ops チームドリブン

redhat.

DevOps と 技術・ツール

redhat.

DevOps が 技術・ツール に求めるもの

DevOps と 技術・ツール

適切な 品質、コスト を伴った

- 継続的変更 (継続的改善) の実現
- 変更スピード (リードタイム短縮) の向上
- チーム・プロセス への適合

redhat.

継続的変更・変更スピードに関連する品質特性

保守性

生産性

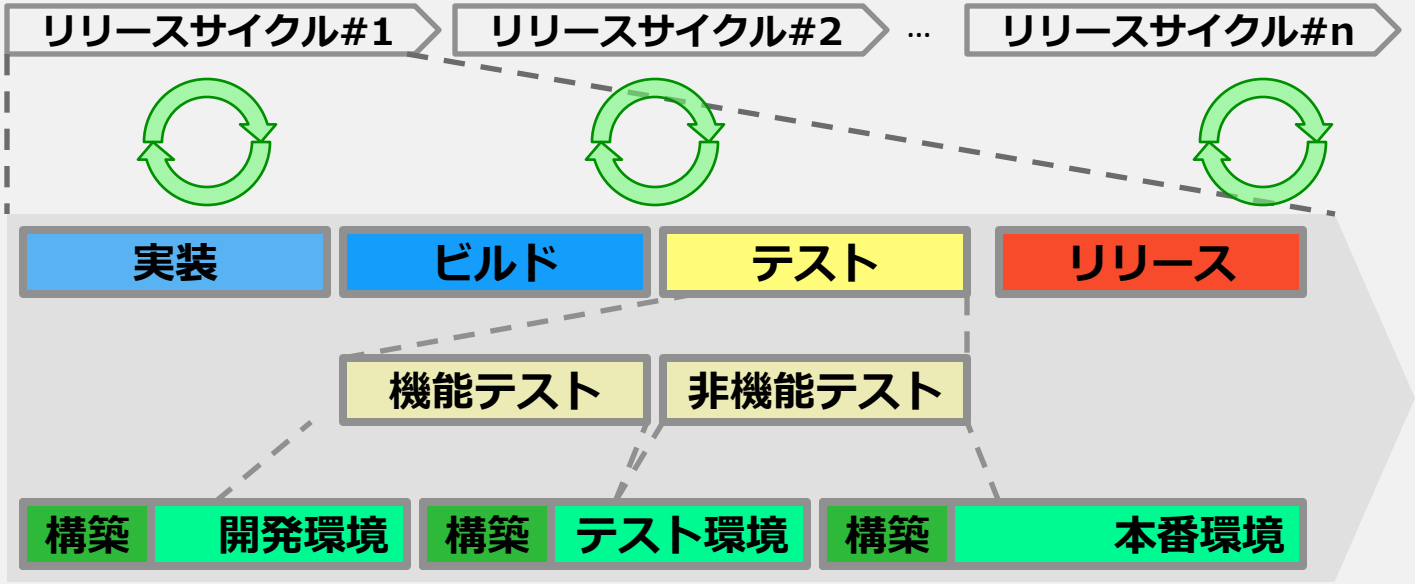
モジュール性	変更の影響範囲が局所化されている度合いやモジュール間の依存性の度合
一貫性	記法・用語・概念が一貫していること
再利用性	ソフトウェアコンポーネントが他のシステムを構築する際に利用できる度合
解析性	障害・変更箇所の識別のし易さや変更の影響範囲の特定のし易さの度合
変更容易性	欠陥や品質の低下なく変更が効果的・効率的に行える度合
テスト容易性	テストポリシー・テスト評価・テスト実装の効果性・効率性の度合
可読性	ソースコードを読む際の、その目的や処理の流れの理解のし易さの度合
簡潔性	実行されない・冗長性・複雑性の少なさの度合

redhat.

品質特性を担保するためのアプローチ

システムインテグレーション

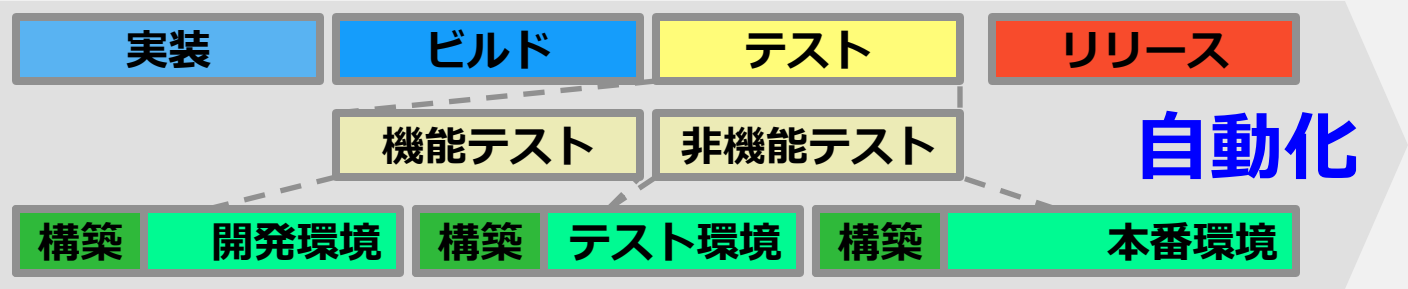
DevOps と 技術・ツール



redhat.

CI/CD パイプライン

DevOps と 技術・ツール



繰返し行われる手順の**自動化**

- ビルド・構築 の 保守性・生産性 の向上
- テスト・リリース の 保守性・生産性 の向上

継続的変更の効率と変更スピードの向上

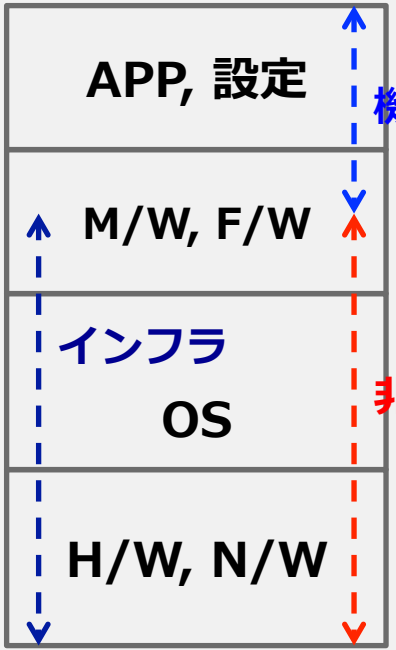
redhat.

自動化とインフラレイヤの特性

DevOps と 技術・ツール

インフラレイヤの特性

- 構築のコード化：難
- 非機能要件のコード化：難
- 環境依存性：高
- 移植性：低
- テスト容易性・効率性：低
- 構成管理の容易性：低



- 開発生産性
- トランザクション
- ロギング
- リソース管理
- 認証・認可
- 保守性
- セキュリティ
- パフォーマンス
- 可用性
- 拡張性

redhat.

自動化における技術的課題

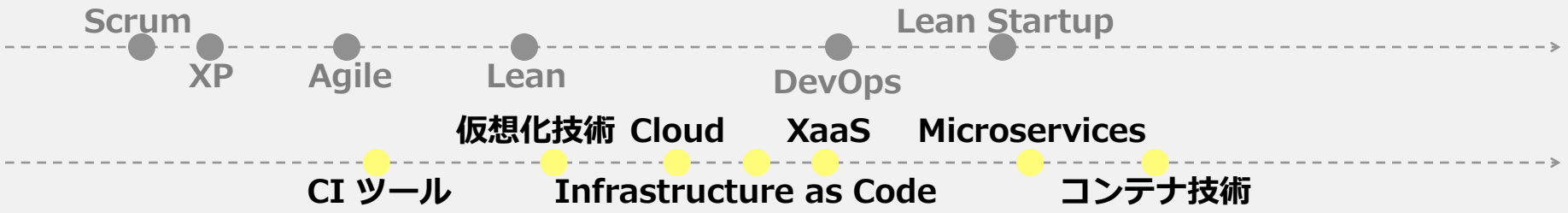
DevOps と 技術・ツール

- インフラ構築の自動化
- インフラ要件のコード化
- インフラに依存するテストの自動化
- インフラの構成管理の効率化

redhat.

DevOps 周辺の歴史

DevOps と 技術・ツール



コンテナ

- Docker
- Kubernetes
- OpenShift

インフラの自動化

- Ansible
- OpenStack

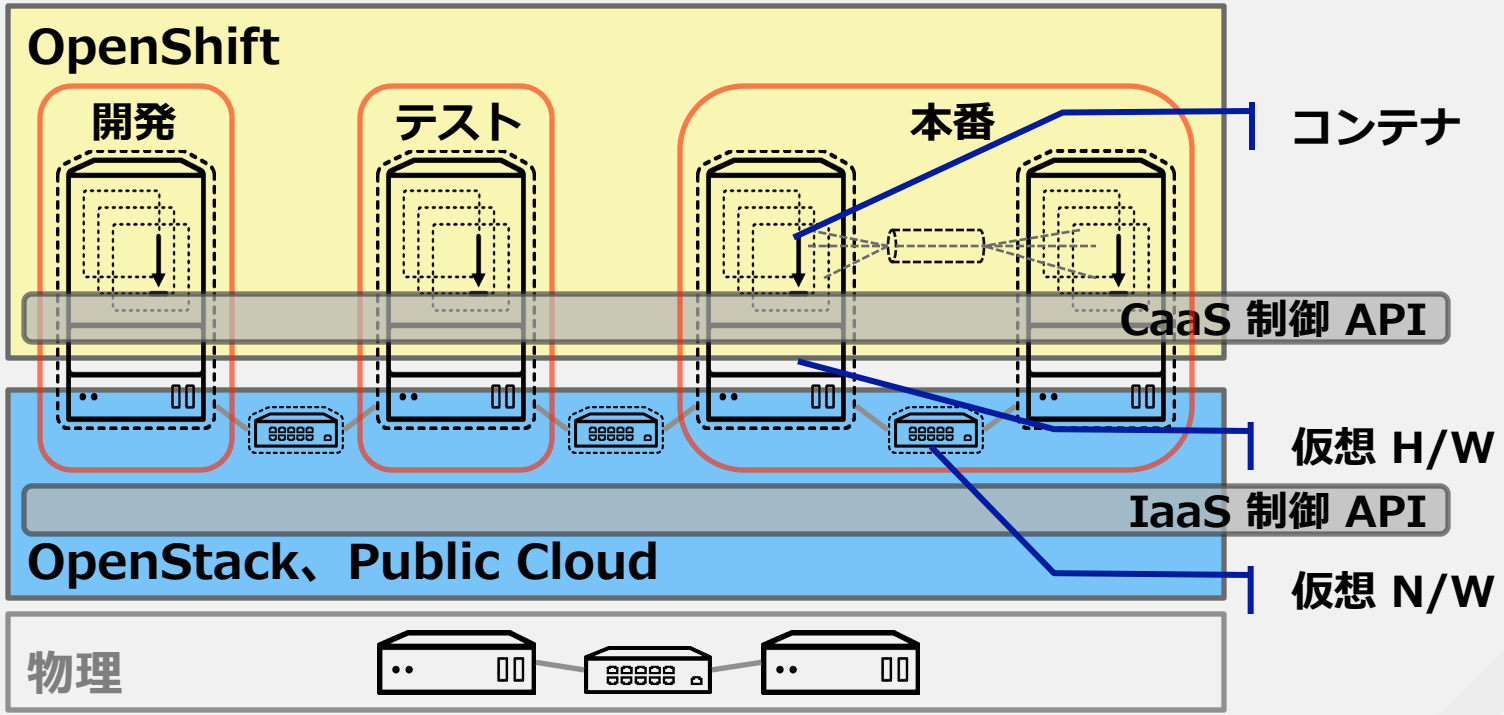
その他

- Git, Jenkins, Serverspec

redhat.

DevOps なインフラ

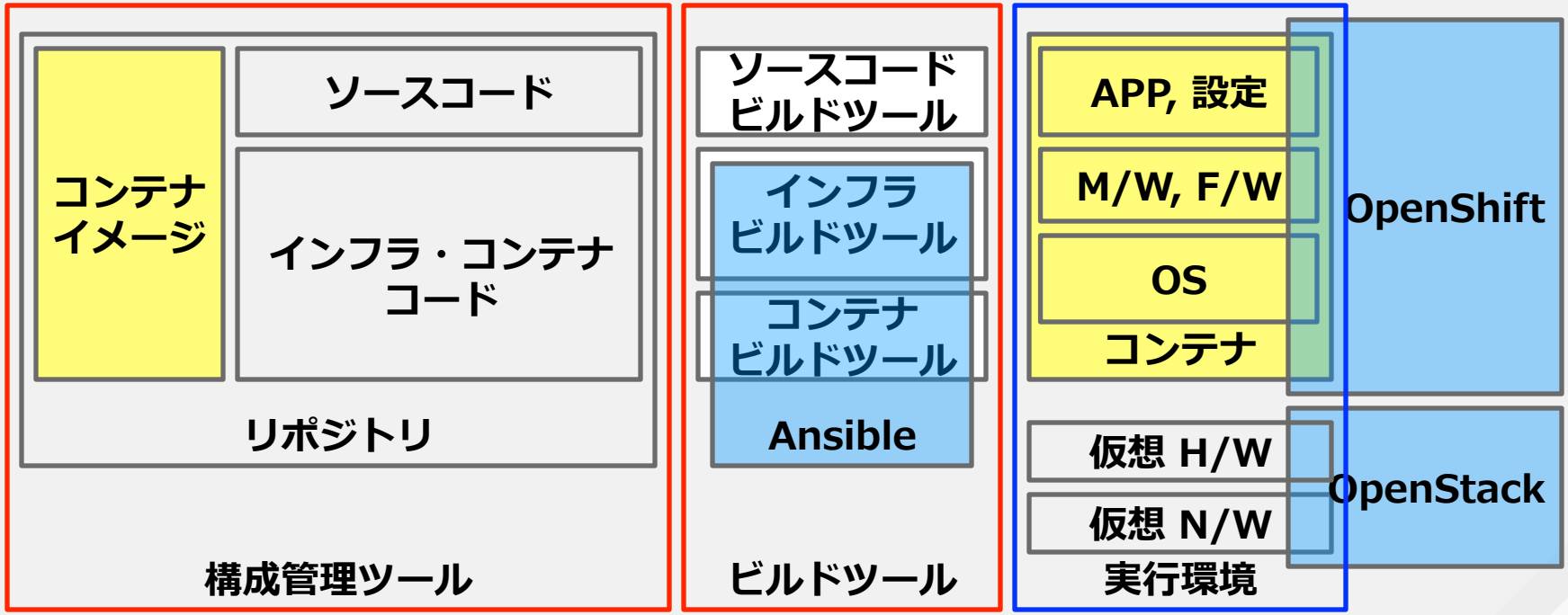
DevOps と 技術・ツール



redhat.

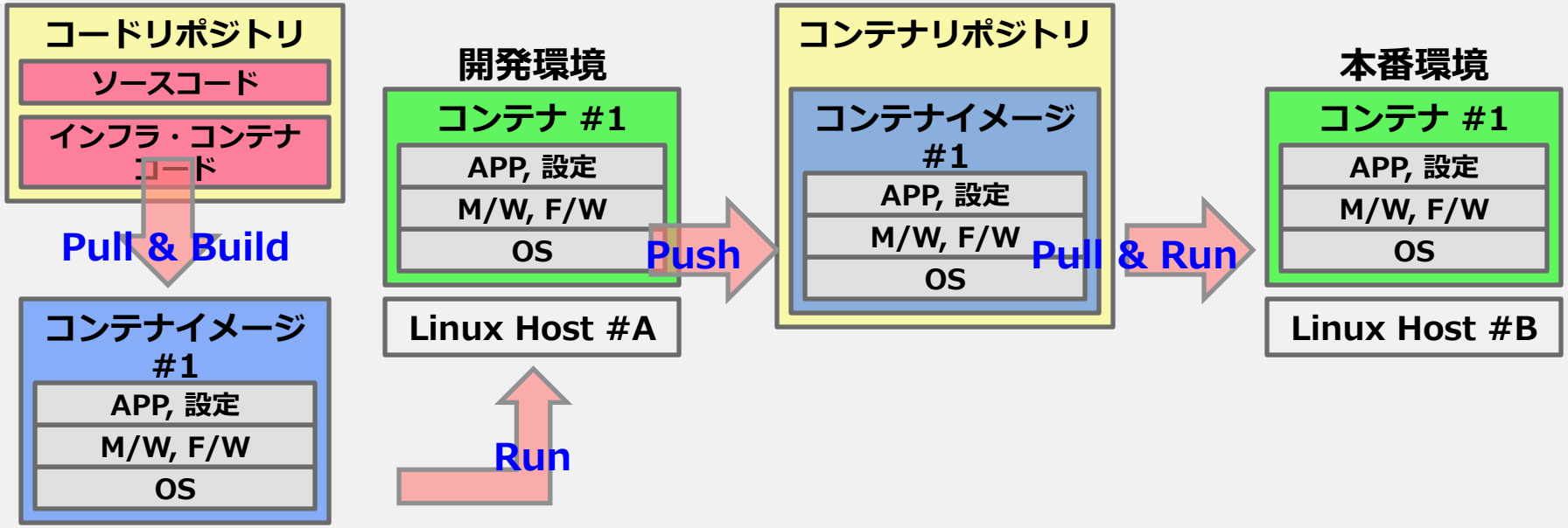
DevOps な 構成管理・ビルド

DevOps と 技術・ツール



redhat.

コンテナ技術の特徴



- システムをコンテナイメージとしてパッケージング
- コンテナイメージはコンテナリポジトリ上で 名前:タグ名 で管理可能
- ホストの異なる任意のコンテナ環境でコンテナイメージを実行可能

redhat.

DevOps なインフラの特徴

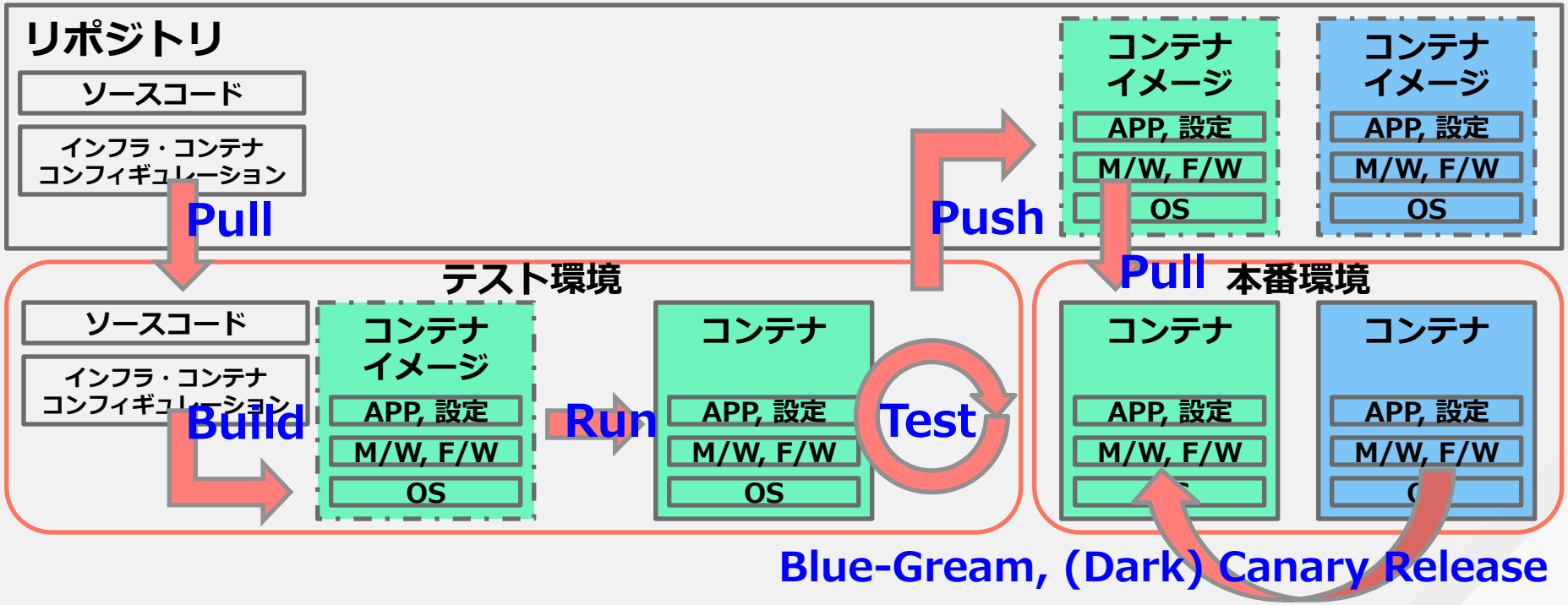
DevOps と 技術・ツール

- インフラ構築 の コード化・自動化
- インフラ要件 の コード化
  - 制御 API を使用した 非機能要件 の コード化
- インフラ要件 の テストの自動化
- システム の 高い移植性 による 保証性の高いテスト
  - コンテナイメージの push&pull via コンテナリポジトリ
- インフラ の 構成管理 の 効率化

redhat.

DevOps な CI/CD

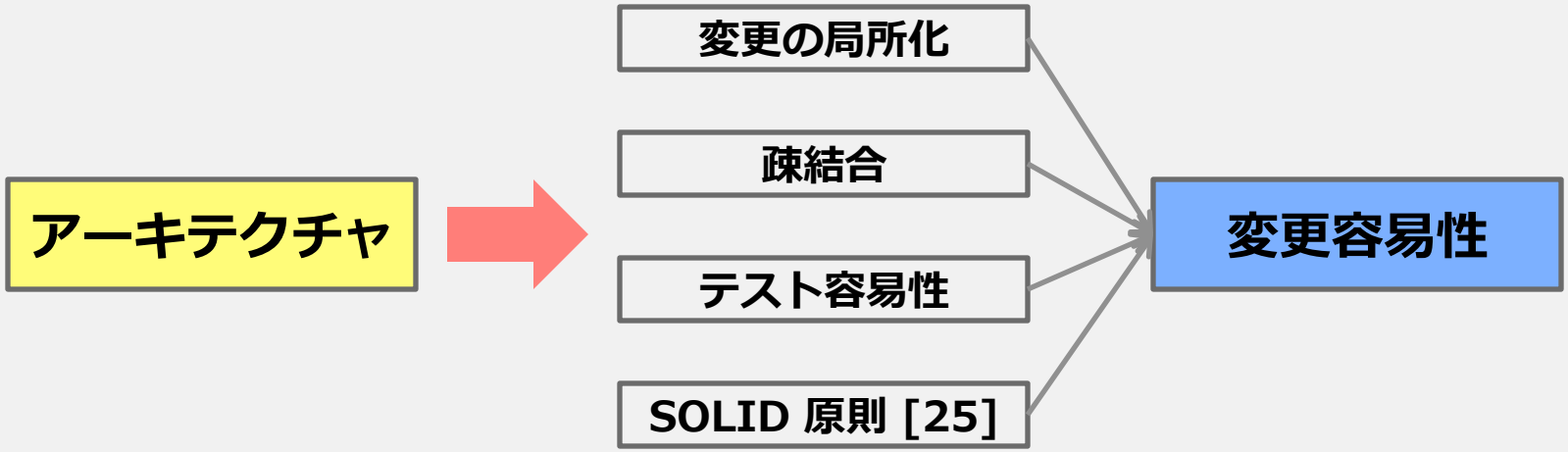
DevOps と 技術・ツール



redhat.

システムの “変更のしやすさ” とアーキテクチャ

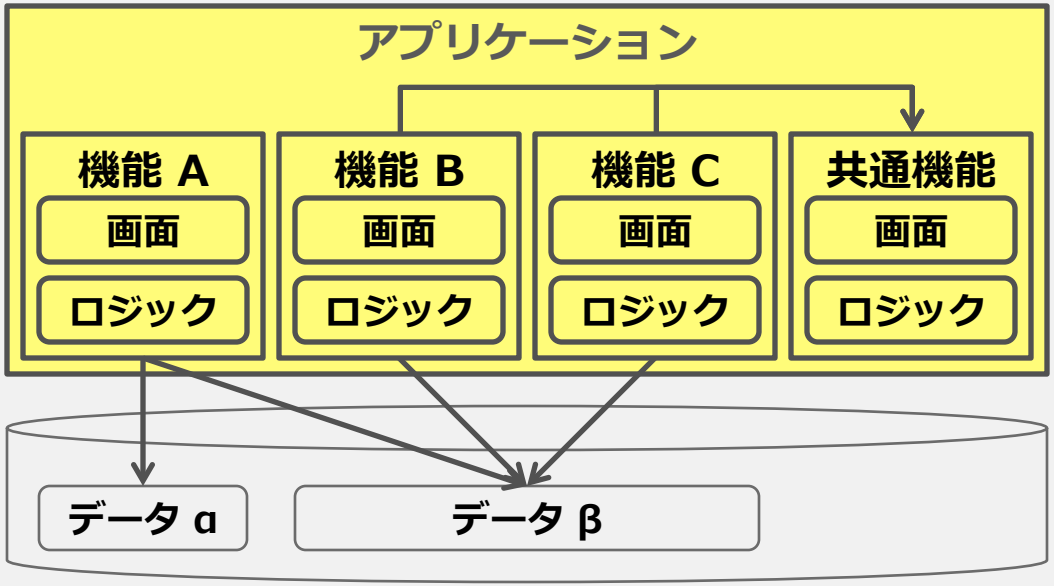
DevOps と アーキテクチャ



redhat.

DevOps と アーキテクチャ

Monolithic アーキテクチャ

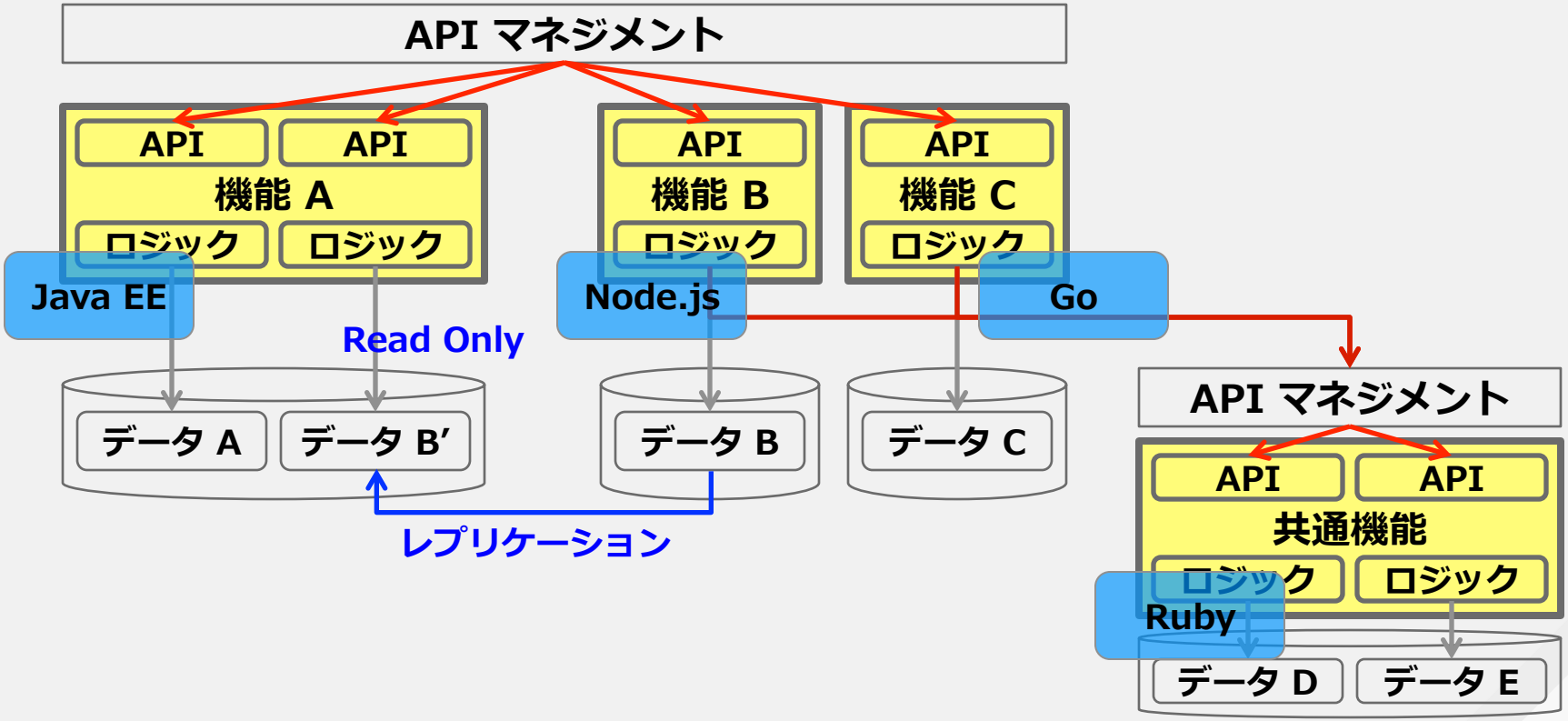


問題点

- 機能間の依存性
- 巨大な影響範囲
- 異なる機能間で以下を共有
  - リリースサイクル
  - インフラ

redhat.

Microservices アーキテクチャ



redhat.

Microservices アーキテクチャの利点

DevOps と アーキテクチャ

- 疎結合
- 小さな影響範囲 = 変更容易性
- 小さな関心事 = 変更スピード
- 個別のリリースサイクル
- 個別のインフラ = 個別に柔軟に非機能要件に対応
- チームにフィット (コンウェイの法則 [15])
- 技術異質性

redhat.

DevOps  Microservices

redhat.

Microservices アーキテクチャの課題

DevOps と アーキテクチャ

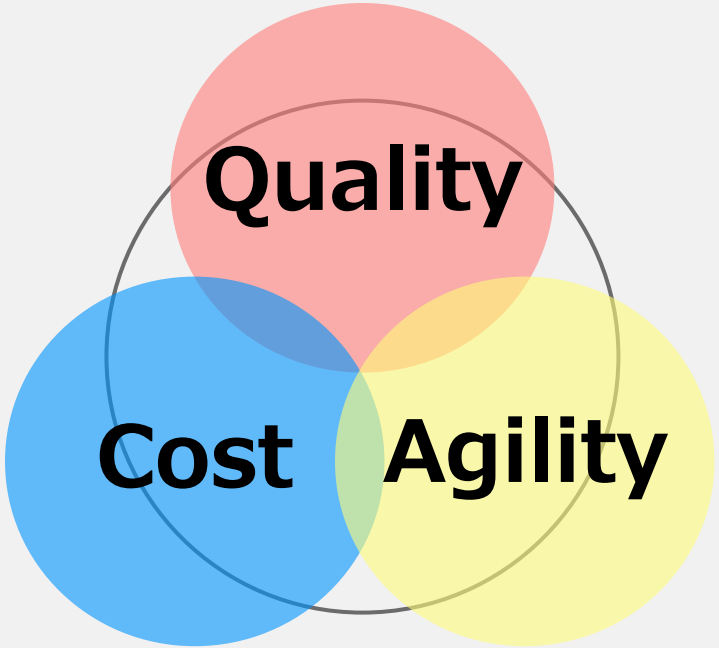
- Microservice の設計
- トランザクションの一貫性の保証
- パフォーマンス
- トレーサビリティ
  - ロギング
  - モニタリング (ヘルスチェック)
- サービスの構成設計
- 共通処理の切り出し

redhat.

DevOps の目的と主要成功要因

redhat.

DevOps の目的



Quality

- ビジネスファースト
- 継続・分析と持続的な変化・改善によるビジネス価値の向上

Cost

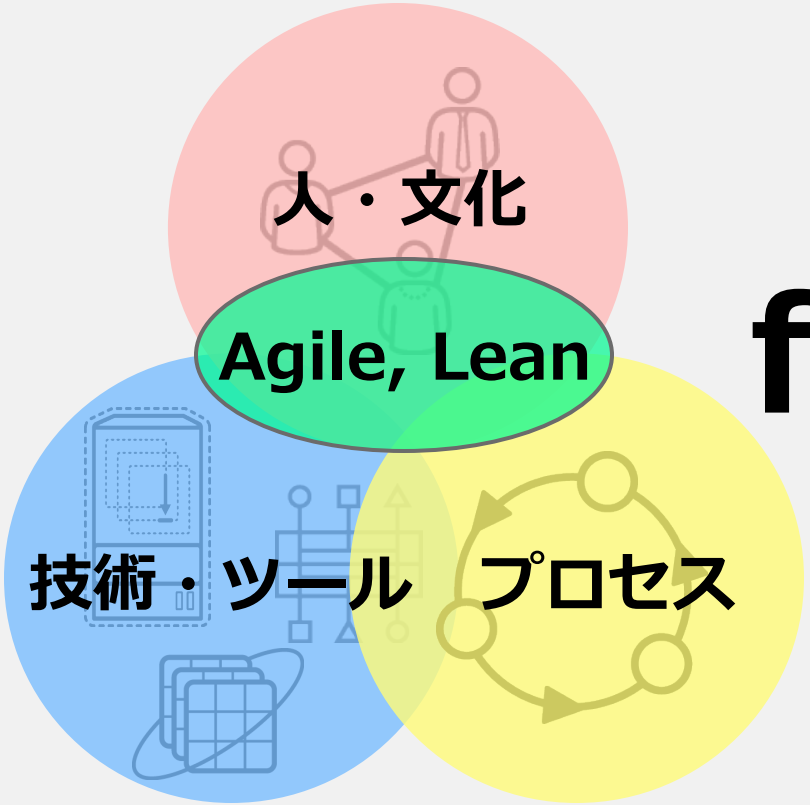
持続的な変化・改善を持続する妥当な対価

Agility

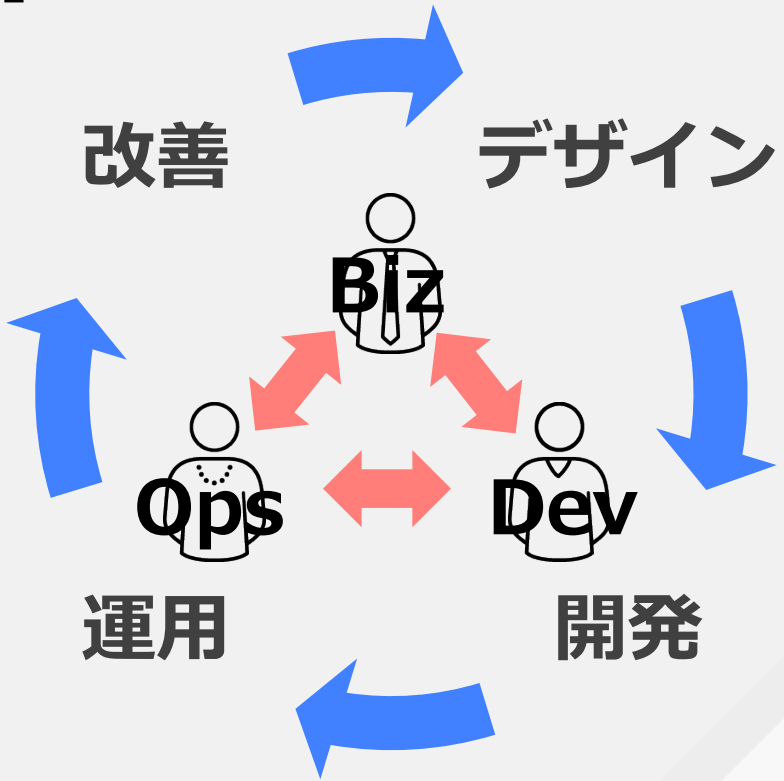
- リリースの早さ
- 変化・改善への機敏・柔軟な対応

redhat.

DevOps

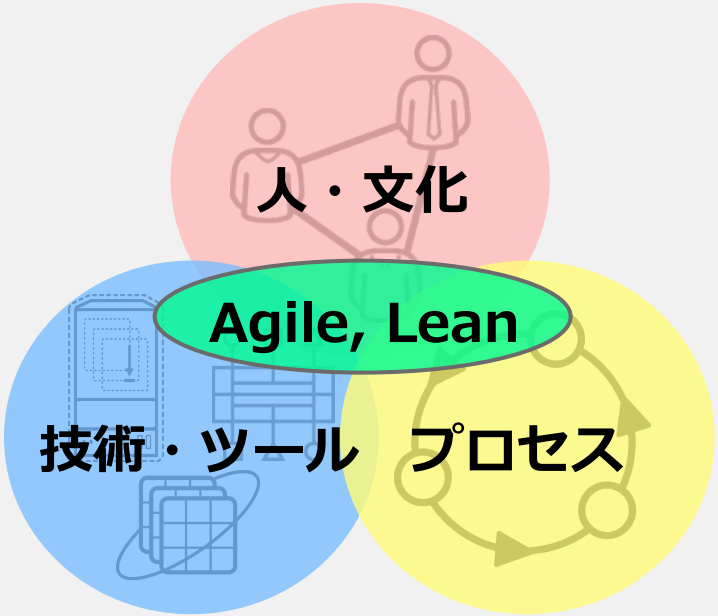


for



redhat.

DevOps の 主要成功要因



人、文化

- ビジネスファースト、チームセントリック、変化・改善に前向きなマインド
- ゴール・KPI の共有と計測・分析

技術、ツール、アーキテクチャ

- 継続的インテグレーション、継続的デリバリー
- 自動化・構成管理・バージョン管理
- コンテナ、仮想化
- Microservices (進化的) アーキテクチャ

プロセス

- Biz, Dev, Ops 横断的な Agile, Lean の実践
- フィードバックループ (継続的改善)

redhat.

II.実践編

redhat.

Agenda – 実践編

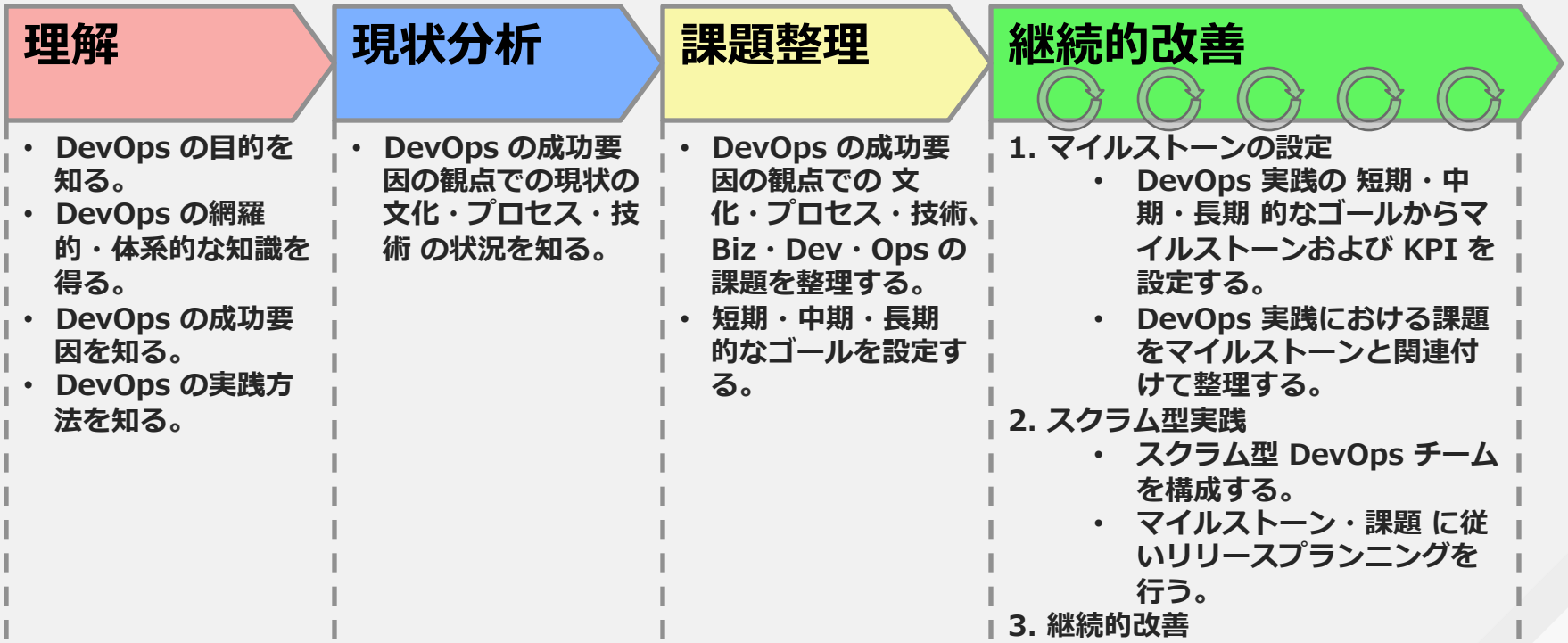
- DevOps の実践モデル
- DevOps 主要成功要因の 効果 と 難易度
- CI/CD パイプラインの実践 と テスト戦略
- 自動化
- アーキテクチャの改善 (技術的負債の排除)
- Agile, Lean の実践
- メトリクスの可視化と KPI による継続的改善
- チームビルディング (文化の改善)

redhat.

DevOps の実践モデル

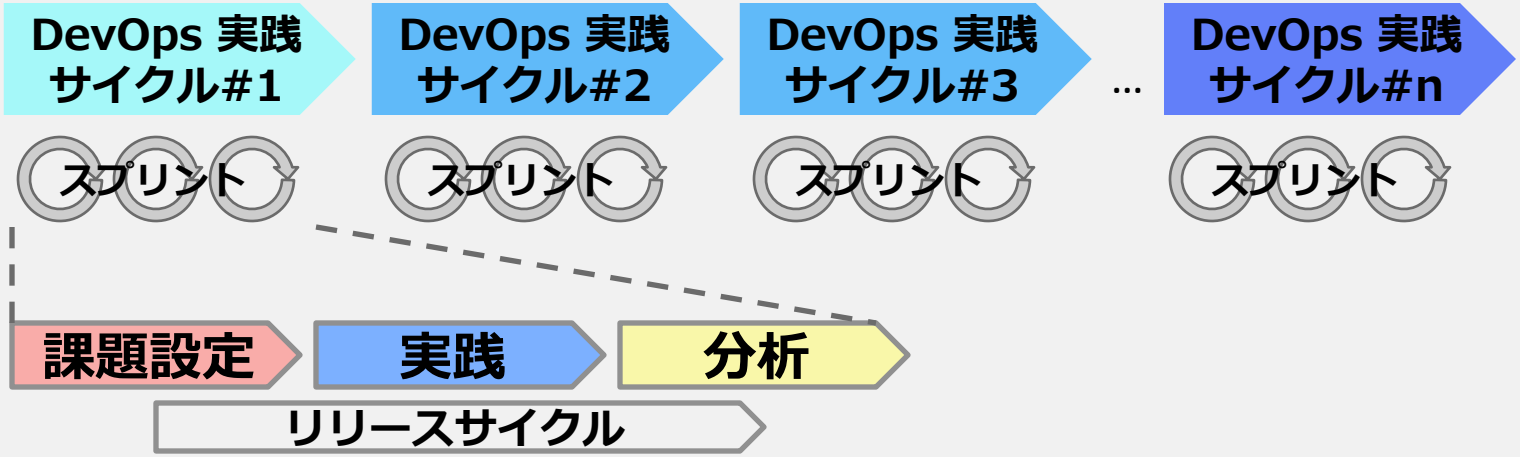
redhat.

DevOps の実践モデル



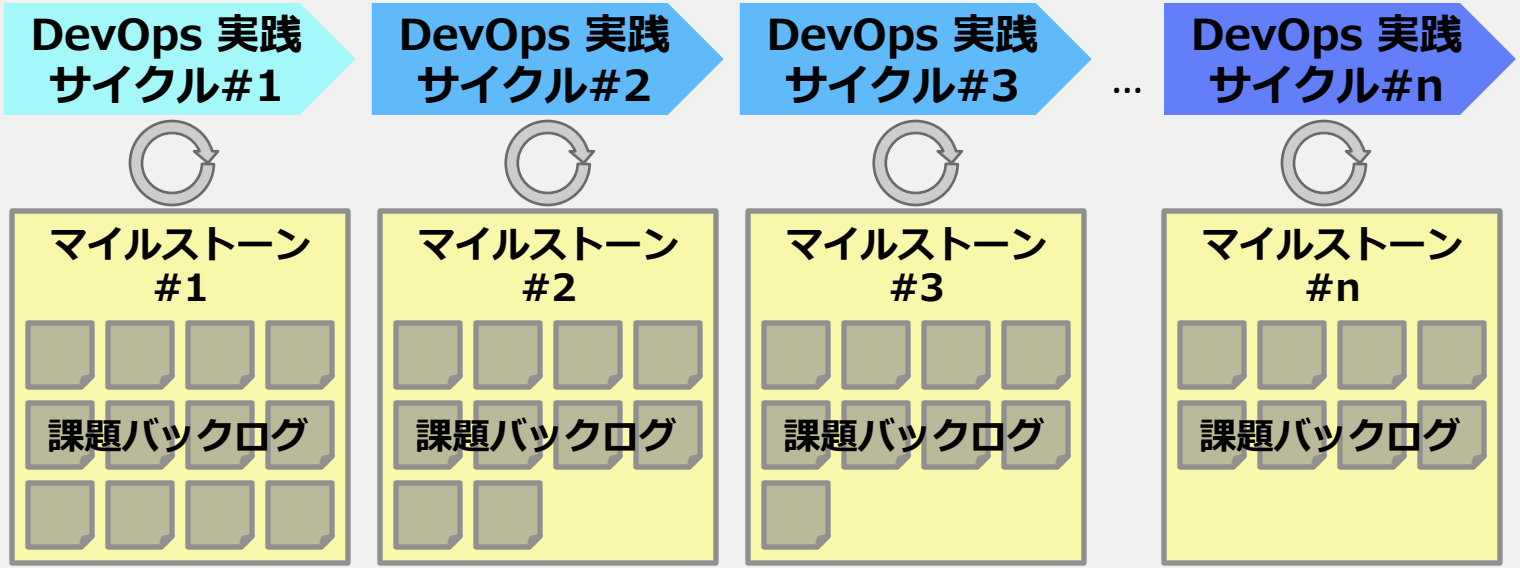
redhat.

DevOps のスクラム型継続的改善



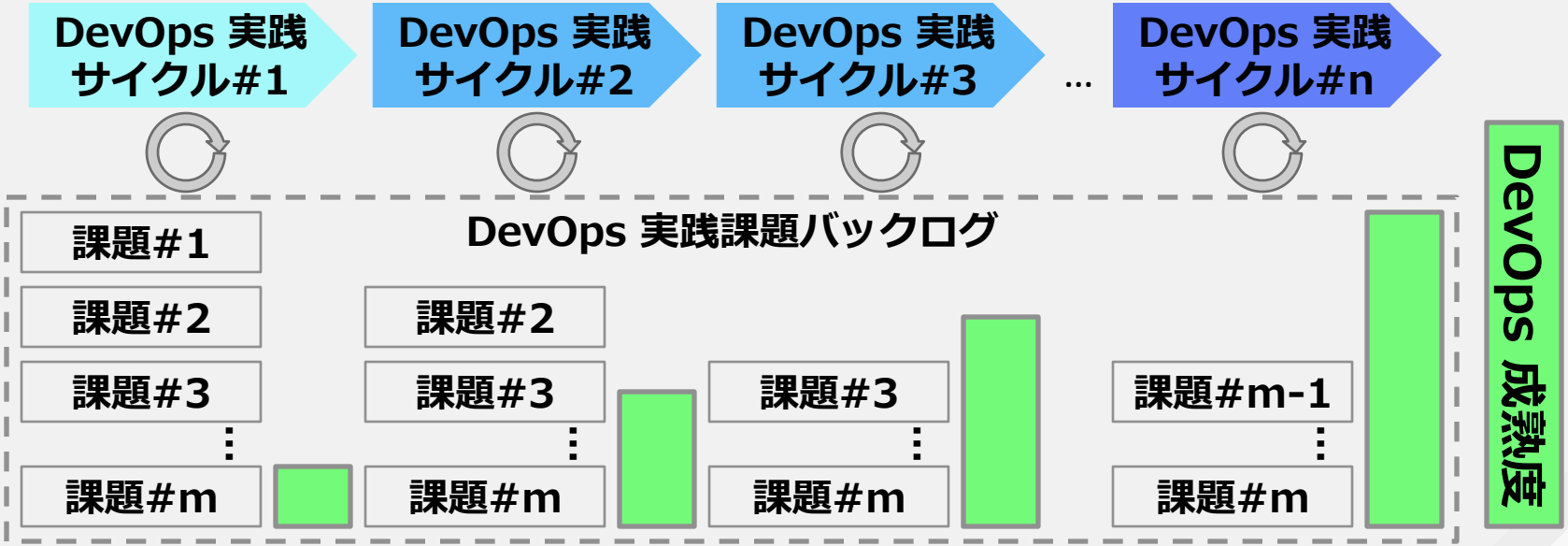
redhat.

スクラム型継続的改善とマイルストーン



redhat.

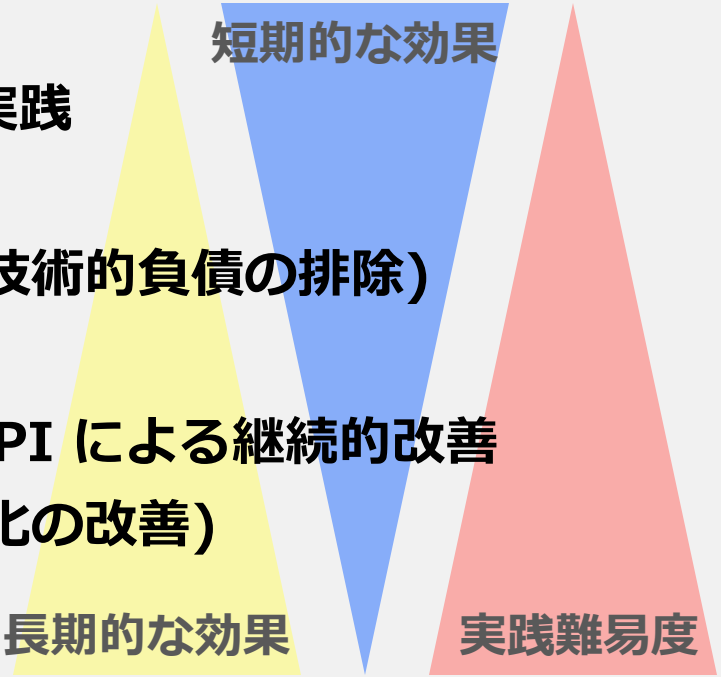
スクラム型継続的改善と成熟度



redhat.

DevOps 主要成功要因の 効果 と 難易度

1. CI/CD パイプラインの実践
2. 自動化
3. アーキテクチャの改善 (技術的負債の排除)
4. Agile, Lean の実践
5. メトリクスの可視化と KPI による継続的改善
6. チームビルディング (文化の改善)

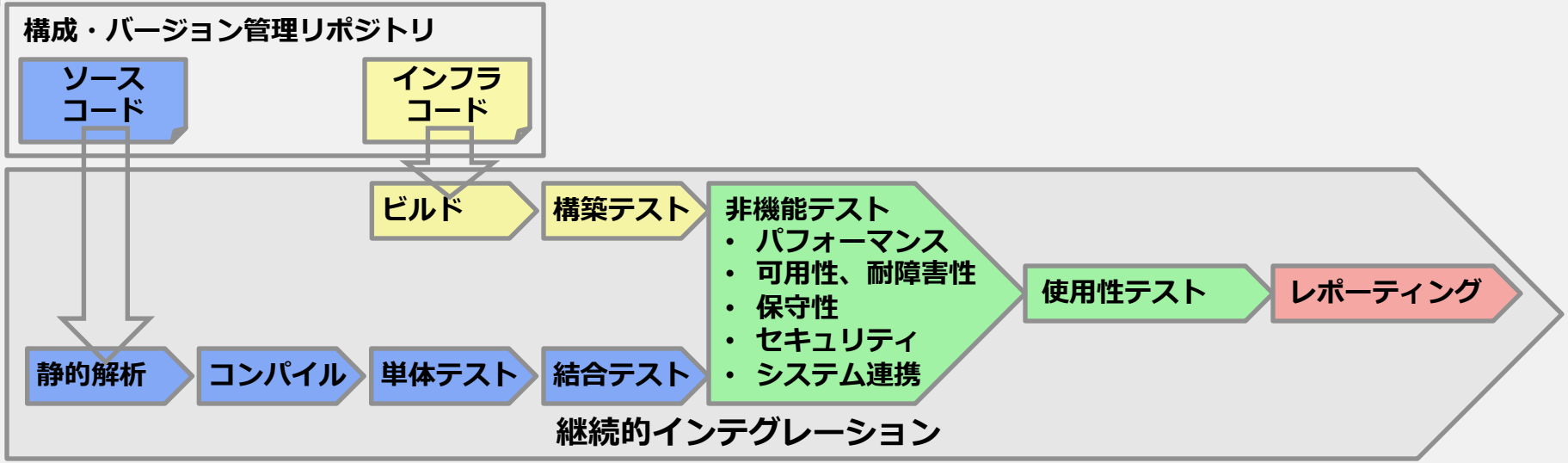


redhat.

CI/CD パイプラインの実践とテスト戦略

redhat.

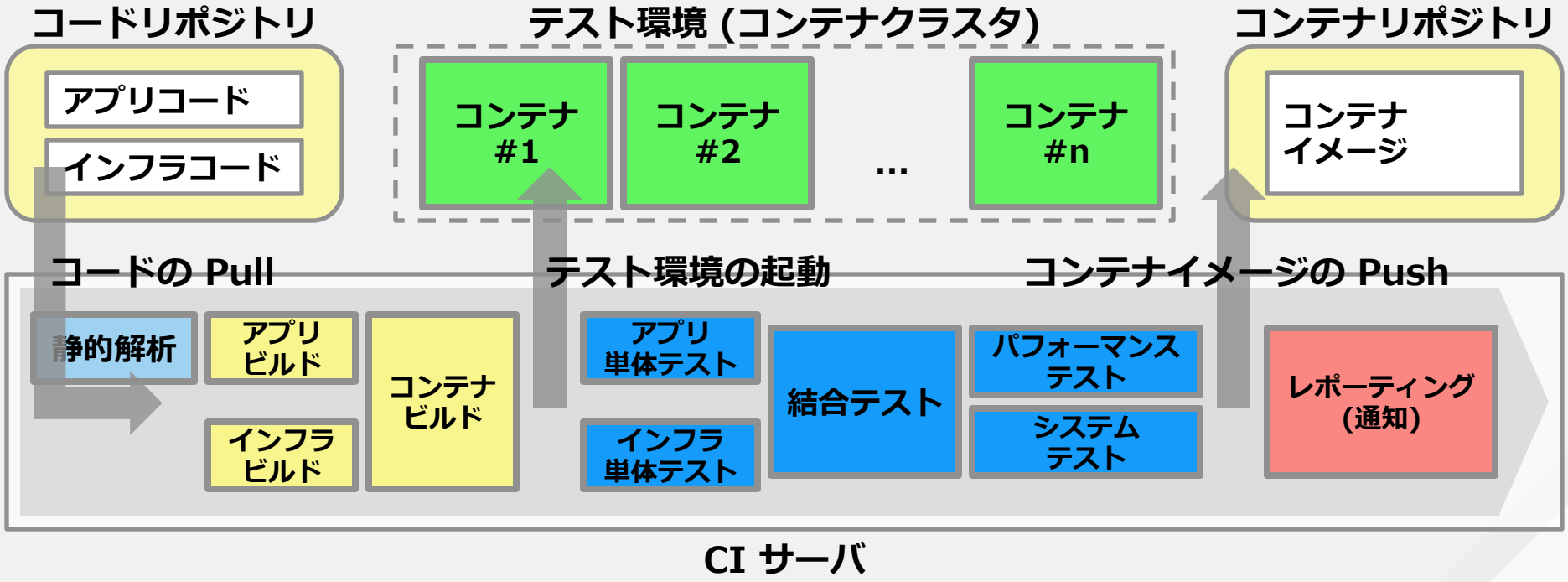
継続的インテグレーション概要



1. 小さな変更毎に ビルド&テスト を行うことによる不具合発生時の解析性向上 (影響範囲、経過時間)
2. 一貫性・可読性・簡潔性の低いコードの効率的な検知
3. テストメトリクスの継続的チェック
4. テストの効率化・高速化
5. 1.～4. による品質の効率的な維持、保守性・生産性の向上

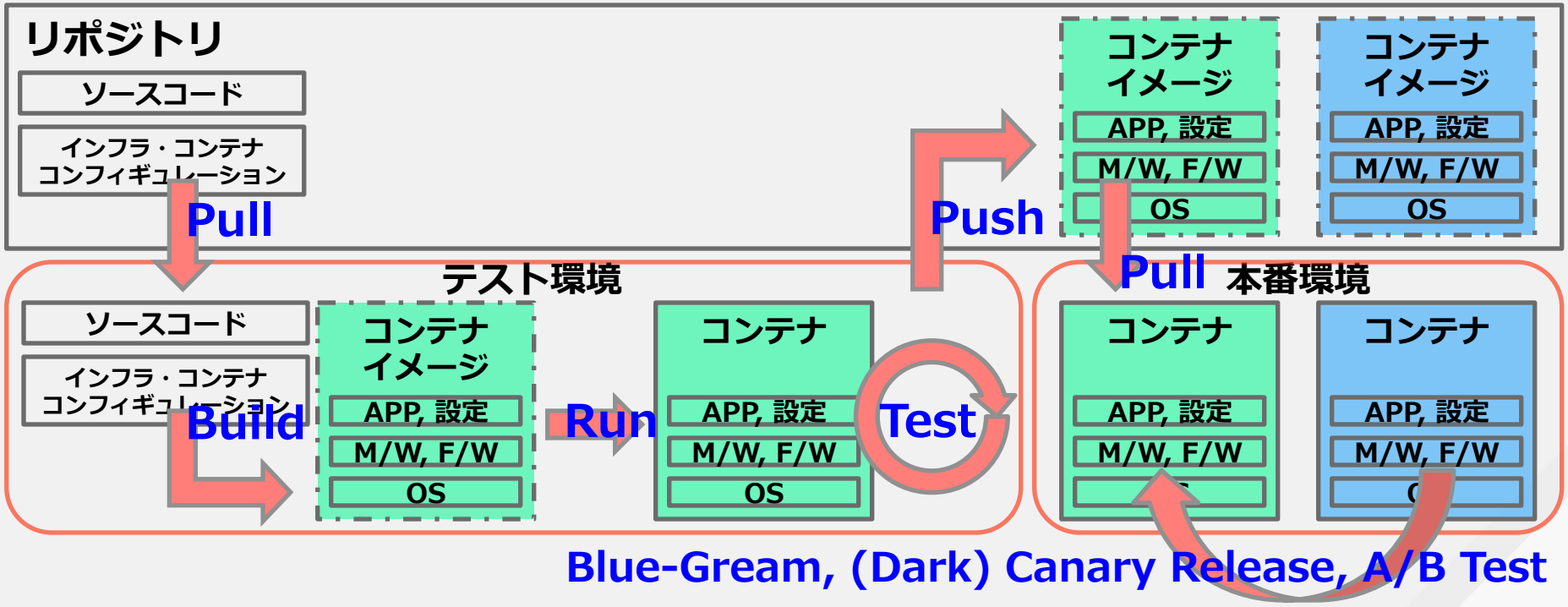
redhat.

コンテナ と 継続的インテグレーション



redhat.

コンテナ と 継続的デリバリ



redhat.

テストのプランニングの視点

テスト戦略

- テスト戦略検討のための 3 つの観点からテスト戦略の分類・整理を行う。
- テスト分類毎にテスト実装・実施のためのポリシー・ガイドを作成する。

テスト基盤の確立と開発プロセスへの適用

- テスト自体の品質特性の観点からテスト基盤 (システムアーキテクチャ・ツールセット) の検討・実装・改善を行う。
- テストの実装・実施を開発プロセスに組み込む。

KPI の可視化と継続的改善

- テスト関連の KPI を収集・可視化し継続的に改善する。

redhat.

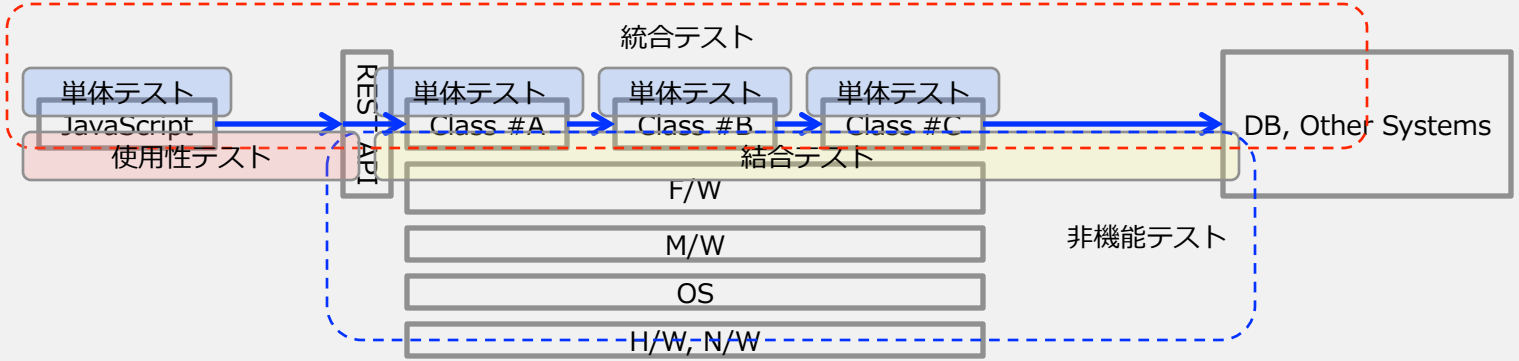
アジャイルテストの 4 象限

ポイント

- SLA、対象システムのビジネスとの関係等から象限毎のテストの重要度を検討する。
- 非機能要件をブレークダウンし品質特性毎の重要度を検討する。
  - 可用性、耐障害性、回復性
  - 保守性
  - 生産性 (開発生産性、運用効率)
- 以下の観点から自動化の範囲を検討する。
  - テストの重要度
  - テストの再利用性
  - 自動・手動のコスト

redhat.

システムスタック観点でのテスト戦略



テスト分類	概要
単体テスト	コンポーネント単位のホワイトボックステスト。API の挙動、境界値分析、限界値分析、エラーハンドリング、網羅率 (命令網羅、判定条件網羅、条件網羅、複数条件網羅、経路組み合わせ網羅) 等の観点でテストを実装する。コンポーネント連携部分にはモックオブジェクトを使用。
結合テスト	コンポーネント間連携、F/W, M/W 連携箇所に関するホワイトボックステスト。ストーリーテスト、一部の非機能テスト (トランザクション、ロギング、エラーハンドリング 等) 等の観点でテストを実装する。外部システムにはスタブを使用。
非機能テスト	非機能要件に関連する品質特性 (後述) 単位のテスト
統合テスト	システム間連携テスト
使用性テスト	UI の使用性テスト

redhat.

品質特性 (一部) 観点でのテスト戦略とアーキテクチャ

	品質特性		実現手段
保守性	モジュール性	変更の影響範囲が局所化されている度合いやモジュール間の依存性の度合	• アーキテクチャ設計、レビュー • 静的解析
	一貫性	記法・用語・概念が一貫していること	• ユビキタス言語、レビュー • 静的解析
	再利用性	ソフトウェアコンポーネントが他のシステムを構築する際に利用できる度合	• アーキテクチャ設計、レビュー
	解析性	障害・変更箇所の識別のし易さや変更の影響範囲の特定のし易さの度合	• アーキテクチャ設計、レビュー • 結合テスト、非機能テスト
生産性	変更容易性	欠陥や品質の低下なく変更が効果的・効率的に行える度合	• アーキテクチャ設計、レビュー
	テスト容易性	テストポリシー・テスト評価・テスト実装の効果性・効率性の度合	• アーキテクチャ設計 • テスト戦略、テストプロセス、テスト基盤
	可読性	ソースコードを読む際の、その目的や処理の流れの理解のし易さの度合	• 実装ポリシー、レビュー • 静的解析
	簡潔性	実行されない・冗長性・複雑性の少なさの度合	• 実装ポリシー、レビュー • 静的解析

redhat.

テストに求められる非機能品質特性

品質特性		実現手段
効率性 **移植性** **再現性**	• 小さな変更単位でシステムの全てのスタックのテストを継続的に実行可能なこと • システム全体のバージョン管理 • 明確な手順による自動化された環境構築手段 • 明確な手順による自動化されたテスト実行手段 • テストの実行が高速であること • テスト済環境の移植性 • テスト環境と本番環境の差異が少ないこと	• アーキテクチャ設計、レビュー • バージョン管理・ビルド ツール • CI パイプライン (CI ツール、テストツール) • コンテナ
一貫性	記法・用語・ポリシーが一貫していること	• アーキテクチャ設計、レビュー • テストポリシー
再利用性	テストが再利用可能であること	• アーキテクチャ設計、レビュー • テストポリシー
明瞭性	テストの意図が明瞭であること	• アーキテクチャ設計、レビュー • テストポリシー
解析性	各種メトリクスが可視化されていること	• テストポリシー • メトリクス収集・可視化ツール
可読性	テストコードを読む際の、その目的や処理の流れの理解のし易さの度合	• アーキテクチャ設計、レビュー • テストポリシー

redhat.

テスト関連のメトリクス (一部)

テスト網羅率
- ソースコード網羅率 等

ソースコード静的解析
- LoC (ステップ数)、サイクロマティック複雑度、重複コード、コメント率、規約違反率 等

バグ
- バグ数、バグ発生率、バグ解決時間 等

アジャイル
- Velocity、サイクルタイム、リリース時間 等

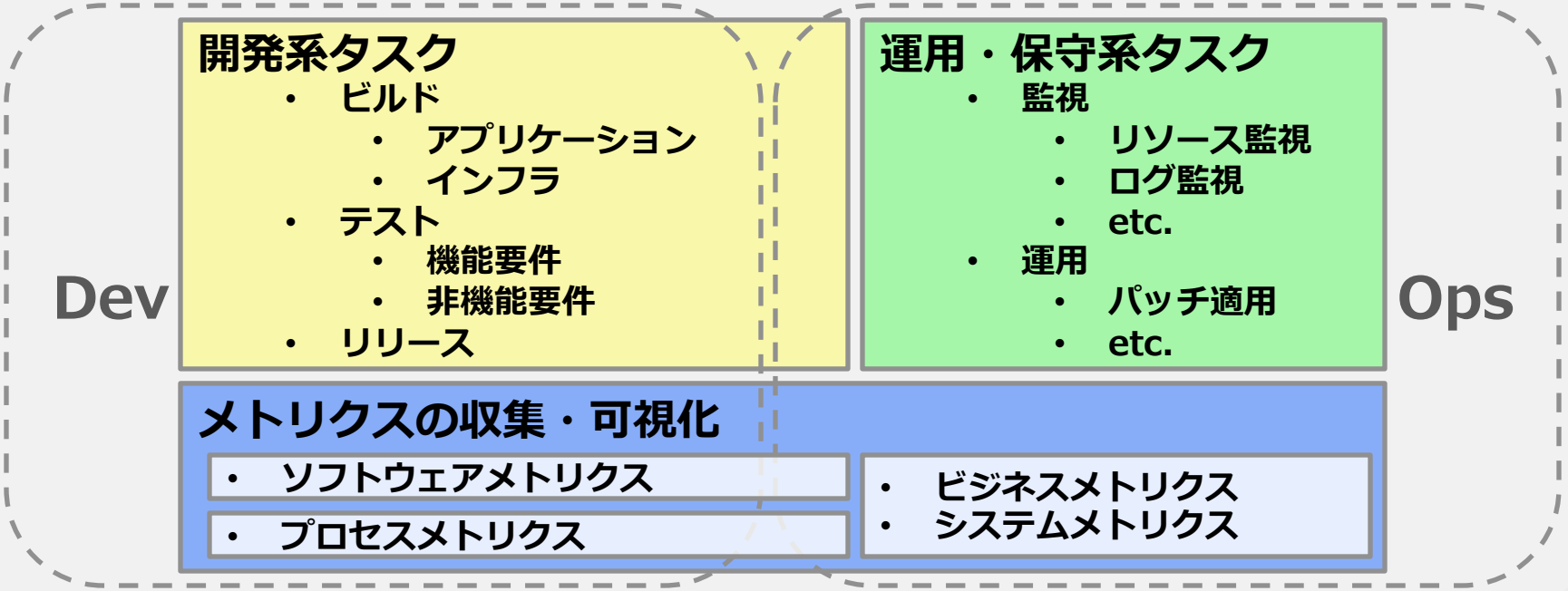
各種メトリクスを計測することが重要ではなく、メトリクスの背後にある品質・コスト・スピード に影響を与える要因・事実を検証し改善に繋げるアクションをスプリント単位で継続的に行うことが重要

redhat.

自動化

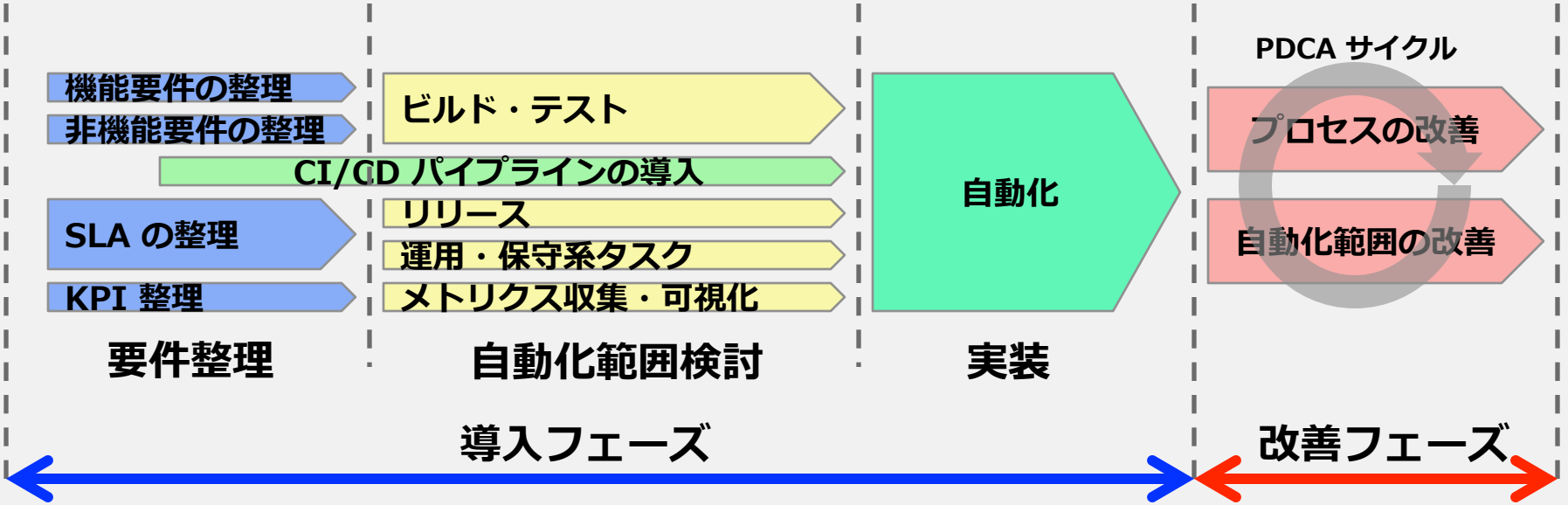
redhat.

自動化の範囲



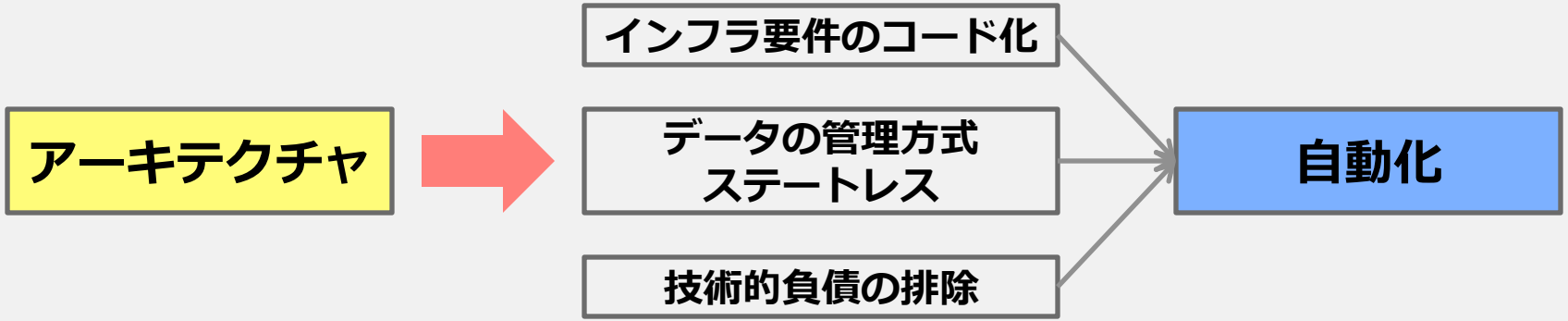
redhat.

自動化のステップ



redhat.

システムの “自動化” とアーキテクチャ



redhat.

アーキテクチャの改善
(技術的負債の排除)

redhat.

品質特性の定義

redhat.

品質特性を担保するためのアプローチ

品質特性に対するモジュール化のアプローチ

アーキテクチャ

- デザインパターン (GoF, etc.)
- DRY 原則
- オブジェクト指向設計 (SOLID 原則)
- サービス指向設計
- レイヤ指向設計
- フレームワーク (Java EE, etc.)
- ドメイン駆動設計
- Microservices アーキテクチャ

凝集度・結合度の高いモジュール分割による保守性・生産性の向上

凝集度
モジュール内の関連性の高さの度合

結合度
モジュール間の関連性の低さの度合

redhat.

ドメイン駆動設計

redhat.

ドメイン駆動設計の概要

コンセプト

- ビジネスのドメイン (概念・業務) を可能な限りシステムに導入

方法論

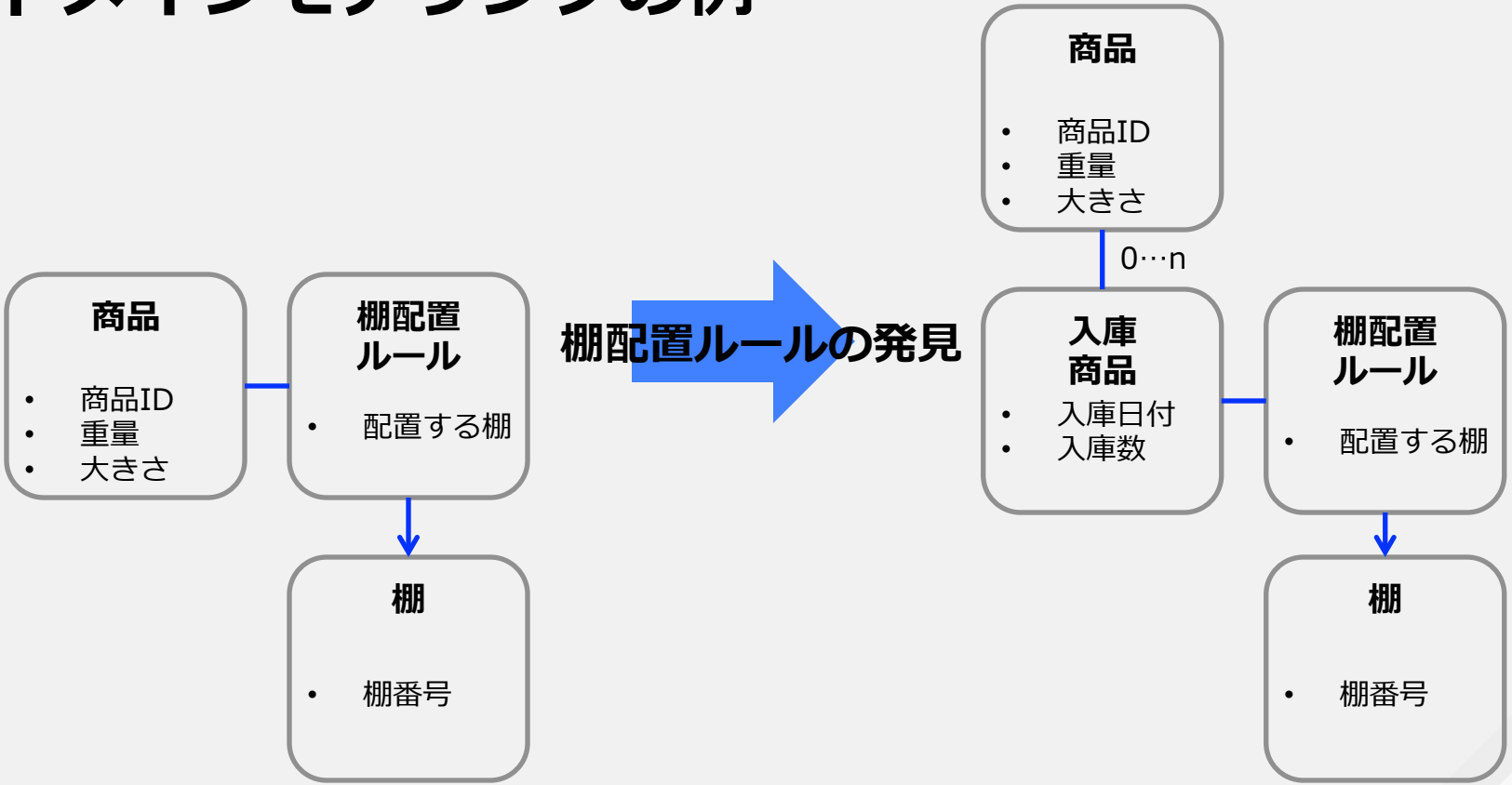
- 業務に関する 用語・知識・作業 をまとめたもの
- ドメインを実装するための様々な要素とパターン
- 初期段階からモデルをコード化し、ドメインエキスパート (Biz) と 開発者 (Dev) でインクリメンタルに論理整合性を検証

利点

- ドメインの分割はビジネスに従う
- 業務の変更への対応性

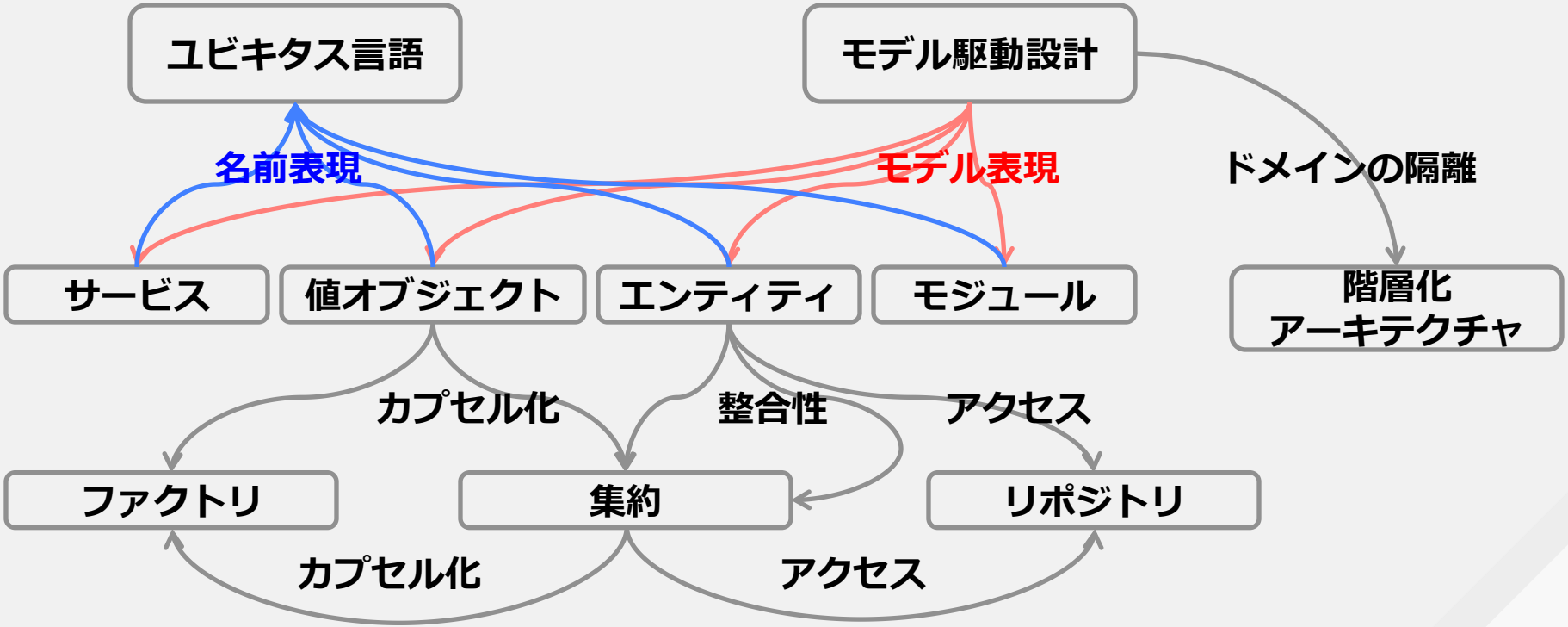
redhat.

ドメインモデリングの例



redhat.

ドメイン駆動設計の要素とパターン



redhat.

ドメインモデルの構成要素

- **エンティティ**
  - 何らかの識別子で一意に識別されるオブジェクト
  - 属性が異なっても識別子が同じなら同一視される
  - 識別子は外部の仕組みで計画的に割り振られたものや、システムで自動生成したもの
  - 例：Customer, Group
- **値オブジェクト**
  - 識別子を持たず、値そのものに意味があるオブジェクト
  - 問題をシンプルにするため immutable とするのが望ましい
    - オブジェクトの生成や共有にコストがかかる場合は mutable としてもよい
  - 例：Address, Color
- **サービス**
  - モデル設計（あるいはユースケース定義）上で、動詞に相当する語彙をモデル化するために使用
  - ステートレスが好ましい
- **モジュール**
  - 高凝縮と疎結合
  - 関連しあう概念は１つにまとめ、関連の薄い概念同士は別のモジュールに分割

redhat.

階層化アーキテクチャ

階層化アーキテクチャ：

UI レイヤ	ユーザとの相互作用の境界となる層。
アプリケーションレイヤ	アプリケーションの振る舞いをコーディネートする薄い層。ビジネスロジックは書かない。
ドメインレイヤ	ドメインモデルを含むシステムの中核となる層。
インフラレイヤ	他の層のライブラリとしての層。エンティティの永続化やUIのライブラリも含む。

Java EE の階層化アーキテクチャとの違い

- アプリケーション層が挿入
  - Interface Segregation 原則 (ISP) や SOA のインテグレーション層に相当
- ビジネスロジックとエンティティを分離せず、合わせてドメインモデルとして設計
  - エンティティにロジックを書くのをためらわない

redhat.

Agile, Lean の実践

redhat.

Agile, Lean の原則

リーン思考の Biz・Dev・Ops での共通認識化

原則1：ムダをなくす

原則2：品質を作り込む

原則3：知識を作り出す

原則4：決定を遅らせる

原則5：速く提供する

原則6：人を尊重する

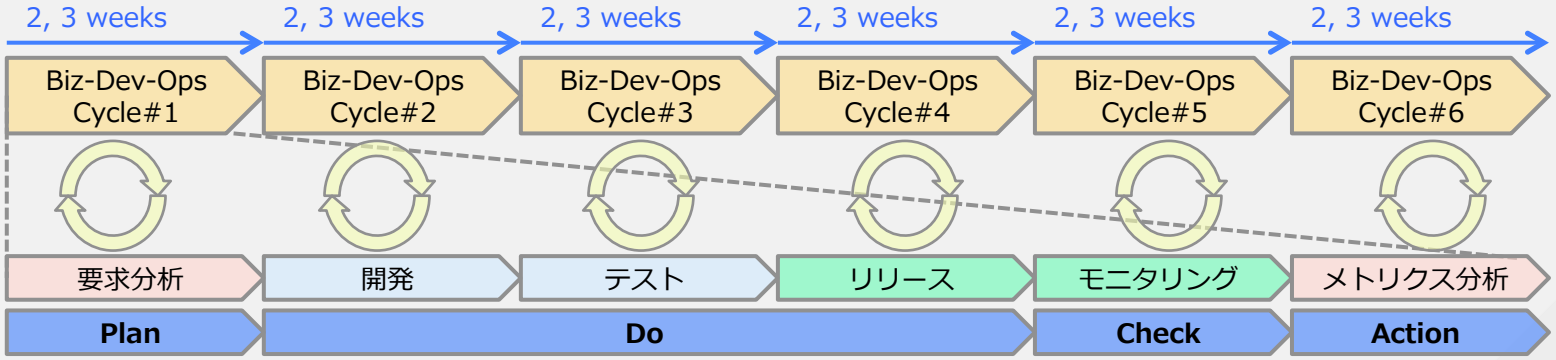
原則7：全体を最適化する

リリースにおける変更数 (バッチサイズ) を小さく

- リードタイムを短く, フィードバックを早く
- 問題の局所化, リスクの低減
- オーバーヘッドの低減

短期サイクルのリリース

- 学びの機会
- Minimal Valuable Product (MVP)



redhat.

Agile, Lean の手法

Scrum

ソフトウェア工学において迅速かつ適応的にソフトウェア開発を行う軽量な開発手法群の総称

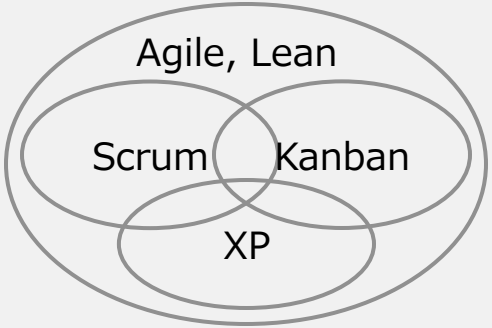
- コミュニケーション (Wiki)
- バージョン管理
- ユニットテスト
- ビルドの自動化

Kanban

製造業を中心に展開されているリーン生産方式の考え方（リーン思考）を、ソフトウェア開発に応用した Agile の一手法

Kanban

5つの価値と19の具体的なプラクティス（実践）を定義。ドキュメントよりもソースコードを、組織的開発の歯車となることよりも、個人の責任と勇気を重んじる人間中心の開発プロセス



redhat.

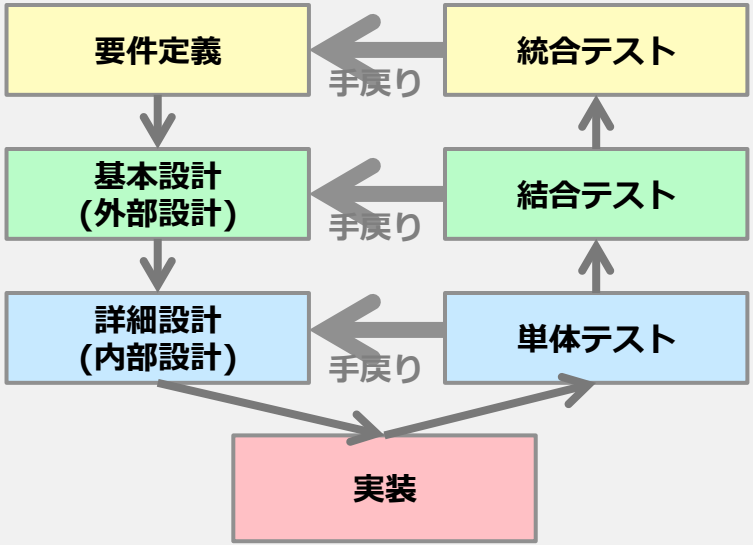
ウォーターフォール型開発における欠陥の 検知と修正

欠陥が検知される工程

- 結合テスト工程以降に欠陥が頻出
- アセンブル (OS, M/W, 各種設定ファイル, F/W, アプリケーション をビルド) + 環境構築 に工数・工期を費やす

欠陥の原因究明・修正と影響範囲

- 対応する設計フェーズ (下図参照) への手戻り
- 複数の機能に跨る広範囲な原因調査
- 複数の機能に跨る広範囲なリグレッションテスト



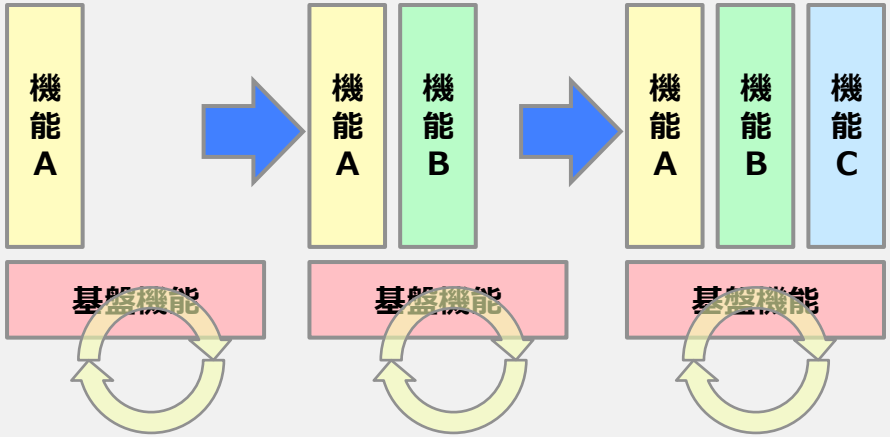
欠陥の 検知と修正 に関するプラクティス

反復型開発

- 小規模な機能のリリースを繰り返すことにより影響範囲を極小化
- 実装 ～ ビルド ～ テスト を繰り返すことによりフィードバックの機会を得る

自動化

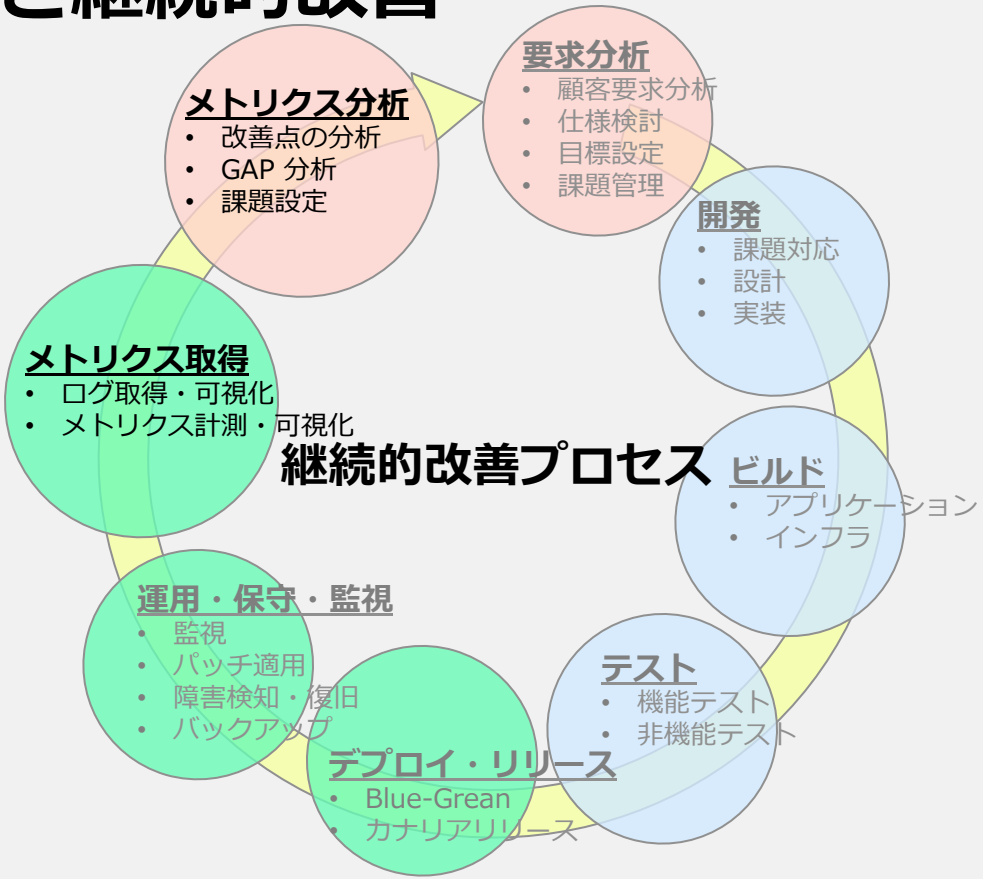
- 環境構築の容易性・環境の再現性 (環境間の差異の排除) 向上
- テストの自動化によるコードの保守性の向上
- ソースコードの静的解析によりソースコードの可読性が向上



メトリクスと継続的改善

redhat.

メトリクスと継続的改善

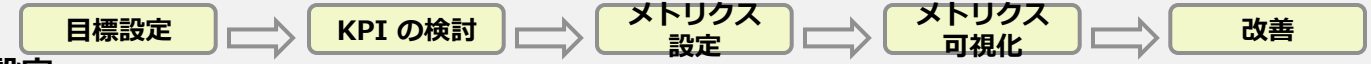


redhat.

メトリクス概要

メトリクスの目的

ゴール・課題に対する目標達成 (ゴール到達・課題解決) と現在の状況の定量的・客観的な把握と改善



メトリクスの設定

1. ゴール・課題に対する KPI (目標達成に影響を与える指標とその因果関係の明確化) の Biz・Dev・Ops の認識共有
2. KPI に対応するメトリクスの設定

DevOps プロセス評価のためのメトリクスの例 (プロセスメトリクス)

サイクルタイム：プロセスの一部 (開発 ～ デプロイ) にかかる時間

リードタイム：プロセス全体にかかる時間

スループット：単位時間あたりに解決した課題数

納期順守率：期日までに解決した課題数

品質：バグ数等からサービスが顧客要求仕様を適切満たすかを計測

価値要求・失敗要求：仕様の不適切さ Biz・Dev・Ops 間の認識の齟齬等から要求マネジメントの質 (適切なものを作っているか) を計測

廃棄項目数：要求分析・開発サイクルで破棄された課題数から革新性・課題選定の妥当性を計測

品質評価のためのメトリクスの例 (ソフトウェアメトリクス)

- バグ数、コードの静的解析 (コード行数、メソッドの擬集度、複雑度、結合度)、テスト網羅率
- システムの品質は 変更容易性・保守性・テスト容易性・回復可能性 等の評価軸とアーキテクチャ上の評価対象範囲 (レイヤ) を整理しメトリクスを選定する。

ビジネスゴールに対するサービス評価のためのメトリクスの例 (ビジネスメトリクス)

- CVR (Conversion Rate)
- CTR (Click Through Rate)
- TAT (Turn Around Time)
- 離脱率

サービス稼働状況評価のためのメトリクスの例 (システムメトリクス)

- パフォーマンス (レスポンスタイム、スループット)
- リソース (CPU、Memory、Disk) 使用率
- I/O (Network、Disk) 待機率
- 平均故障間隔 (Mean Time Between Failure：MTBF)
- 平均復旧時間 (Mean Time To Repair：MTTR)
- 稼働率

チームビルディング
(文化の改善)

redhat.

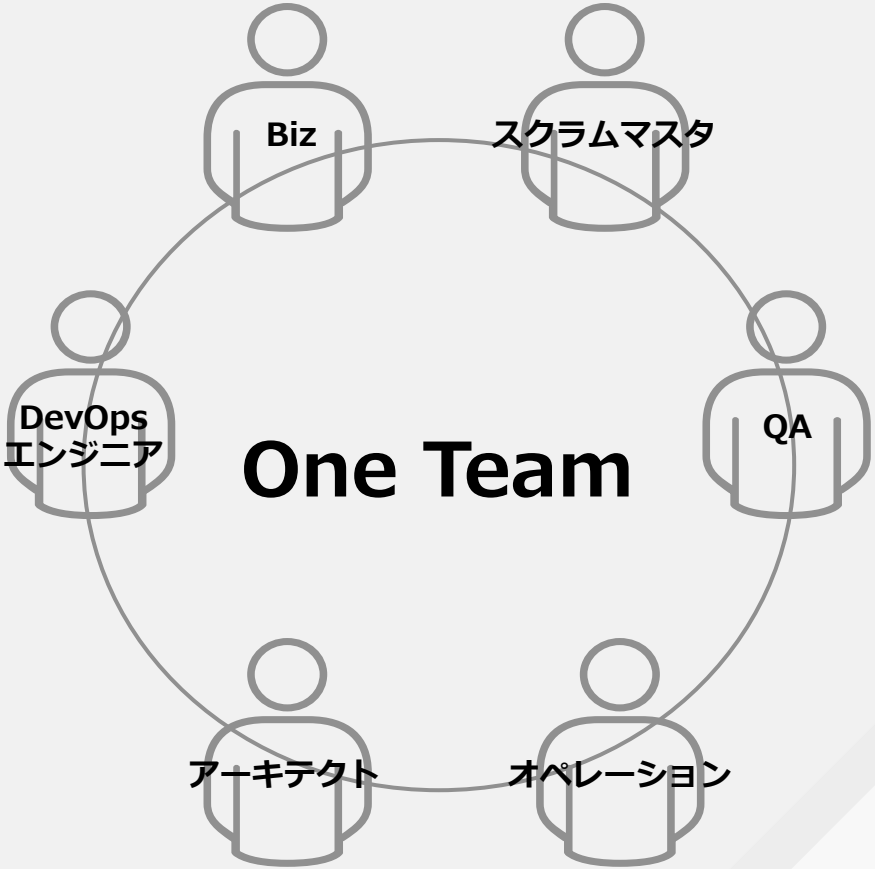
DevOps 実践で直面する課題

- システムの 開発・運用 を外部委託しているため DevOps を手動しにくい。(契約形態についても課題)
- 失敗を避け避難する・チャレンジを評価しない文化
- リードタイム短縮の必要性に関する認識の低さ
- 組織構造に起因する責任の所在の不明確
- 開発と運用の担当が異なる。

redhat.

Biz-Dev-Ops チーム

ポジション	主な関心事・役割
Biz (サービスオーナー)	• 要求分析 (ニーズとの GAP 分析) • KPI の分析 (ゴールとの GAP 分析) ・定義 • 要件・機能の意思決定
スクラムマスタ	• プロセス (Agile) 管理 • 課題・進捗の管理 • ファシリテーション
アーキテクト	• アーキテクチャ定義 • 技術支援 • 非機能要求の担保 • CI/CD パイプラインの設計 • SLA の定義
DevOps エンジニア	• アプリケーション・インフラ実装 • テスト実装
QA	• テスト妥当性の検証 • レビュー • 受入テスト
オペレーション	• 運用・監視・保守 • KPI の取得・可視化 • ユーザマニュアル



redhat.

Biz-Dev-Ops チームのコレボレーション

見える化と共有

- ビジネスゴール
- 課題と解決状況
- 定量的目標・KPI とメトリクス

文化

1. **Respect (尊重)**
2. **Trust (信頼)**
3. **Healthy attitude about failure (失敗に対する態度と学び)**
4. **Avoiding Blame (非難しない)**
5. **変化に対する態度**

4. **責任・権限・評価**

   Biz・Dev・Ops で責任を適切に分配し、達成度を定量的に評価する。

環境・ツール

成果物共有・コミュニケーション・課題管理・進捗管理・メトリクスの定量的な測定・評価を効率化するための環境・ツールを整備する。

コミュニケーション：WiKi, Chat, Bot ツール etc.

課題・進捗管理：プロジェクト管理ツール etc.

メトリクスの測定・評価：ログ解析・可視化ツール etc.

Column HRT という考え方 (“Team Geek” より)

Humanity (謙虚)	•世界の中心は君ではない。君は全知全能ではないし、絶対に正しいわけでもない。常に自分を改善していこう。
Respect (尊敬)	•一緒に働く人のことを心から思いやろう。相手を 1 人の人間として扱い、その能力や功績を高く評価しよう。
Trust (信頼)	•自分以外の人は有能であり、正しいことをすると信じよう。そうすれば、仕事を任せることができる。

redhat.

III.テクニカル編

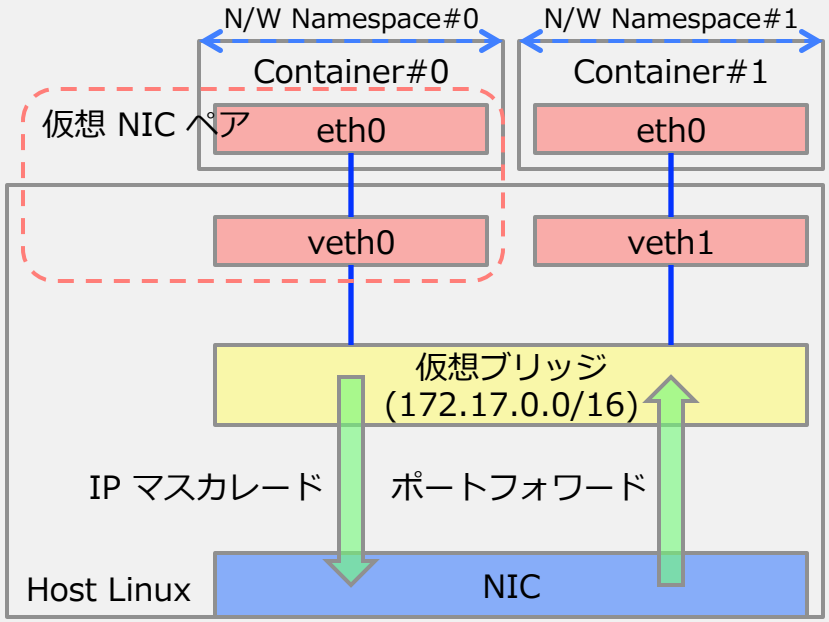
redhat.

Agenda – テクニカル編

- コンテナ (+ コンテナオーケストレーション) 技術 概要
- コンテナ技術を使用した CI/CD
- ツールセット

redhat.

コンテナの N/W の仕組み



仮想 NIC ペア

コンテナ内部に設定された仮想 NIC (eth0) とコンテナ毎に設定された仮想 NIC (veth) のペアで veth は仮想ブリッジに接続

仮想ブリッジ

Linux が提供する仮想 N/W スイッチ (デフォルト 172.17.42.1) で Host Linux 上で稼働する異なるプロセス間の通信に使用

デフォルトで 172.17.0.0/16 のサブネットが割り当てられコンテナ内部の仮想 NIC (eth0) には同サブネットの IP を割り当て

IP マスカレード

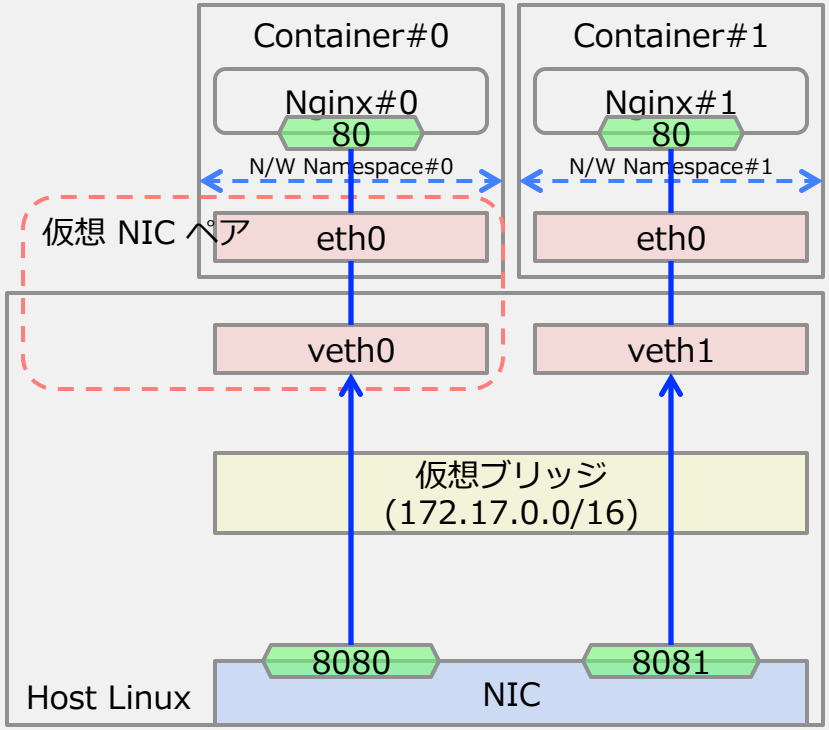
Host Linux の iptables によるコンテナから Host Linux 外部への通信

ポートフォワード

Host Linux の iptables による Host Linux からコンテナへの通信

redhat.

単一 Linux Host で稼働するコンテナの N/W 構成例

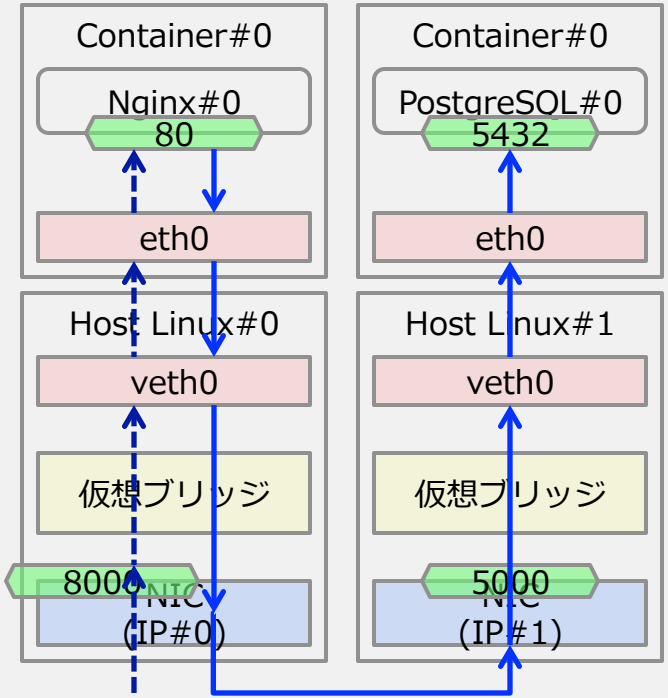


単一の Host Linux 上で同一のポート番号にバインドされたコンテナを複数起動する場合、コンテナ毎に異なる転送元ポート番号を設定することで対応可能。

Host Linux の外部からは Host Linux の IP アドレス：転送元ポート番号 でアクセスし、対応するコンテナに転送される。

redhat.

複数 Linux Host で稼働するコンテナの N/W 構成例

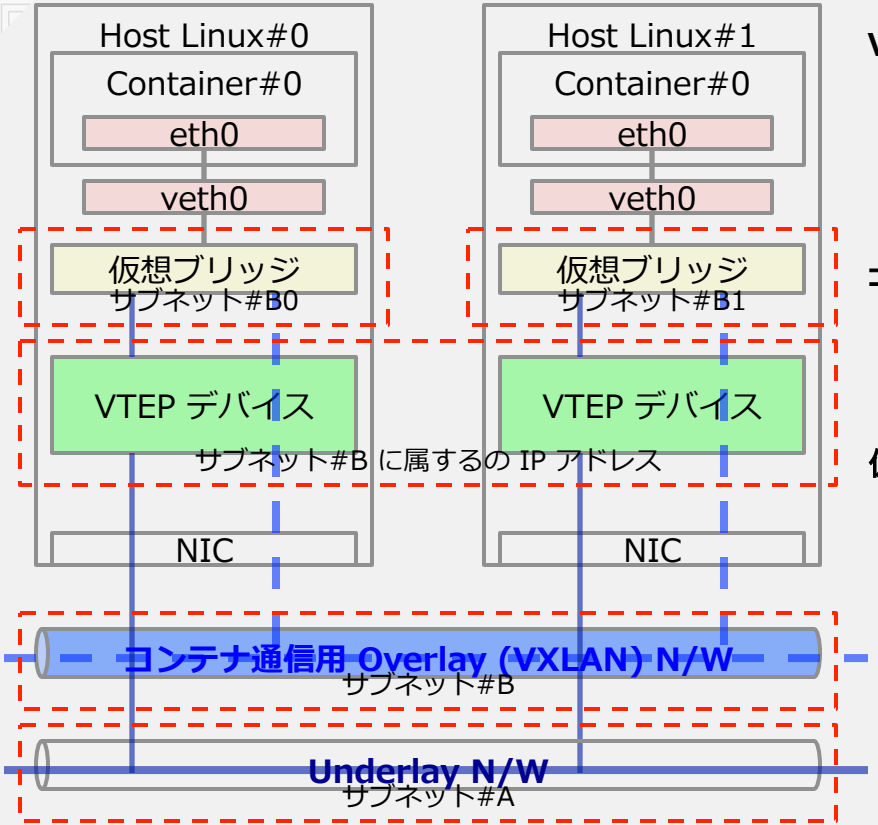


異なる Linux Host で稼働するコンテナを連携する場合、接続元コンテナは接続先コンテナが稼働する Linux Host の IP アドレス:転送元ポート番号 (※ 左記の例では IP#1:5000) に対して接続を行う。

接続元コンテナでは接続先の具体的な IP アドレス:ポート番号 を管理 (通常は環境変数で管理) する必要があり、接続先コンテナの IP アドレス が変更された場合 (フェイルオーバ 等) には別途対応する必要がある。

redhat.

コンテナ N/W 概要



VTEP

Flannel/OVS が設定する VXLANデバイス でコンテナ通信用オーバレイ　N/W に接続される。

同仮想 NIC はコンテナ通信用オーバレイ N/W に割り当てられたサブネット中の IP アドレスが割り当てられる。

コンテナ通信用オーバレイ N/W

Flannel/OVS により構築された VXLAN によるオーバレイ N/W で、Host Linux が接続するサブネットとは別のサブネットが割り当てられる。

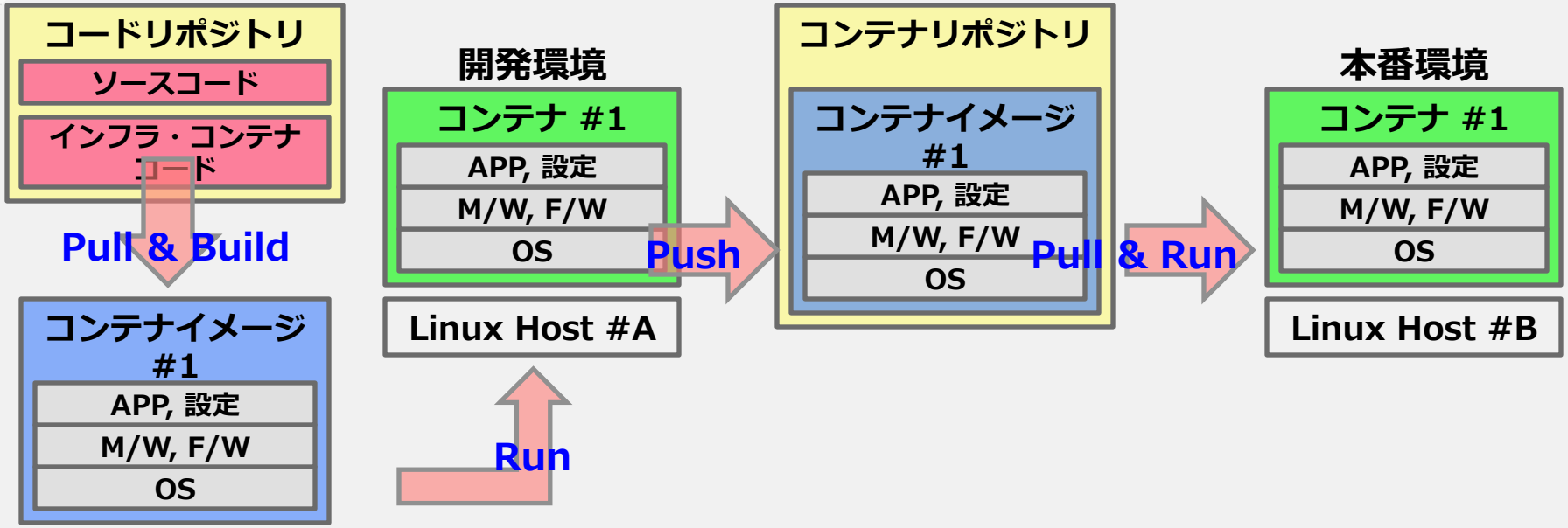
仮想ブリッジ

Flannel により構築されたコンテナ通信用オーバレイ N/W に割り当てられたサブネットを分割したサブネットが割り当てられる。

仮想 NIC (flannel.1) とのパケット転送は Linux のパケットフォワーディングが使用される。

redhat.

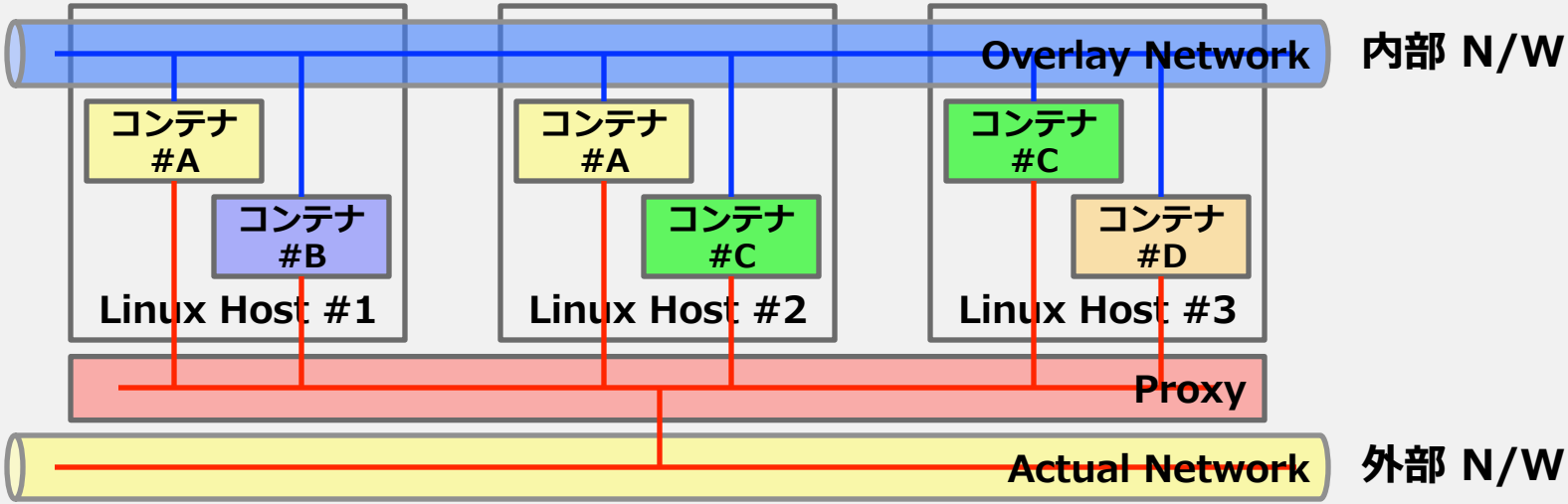
コンテナ技術の特徴



- システムをコンテナイメージとしてパッケージング
- コンテナイメージはコンテナリポジトリ上で 名前:タグ名 で管理可能
- ホストの異なる任意のコンテナ環境でコンテナイメージを実行可能

redhat.

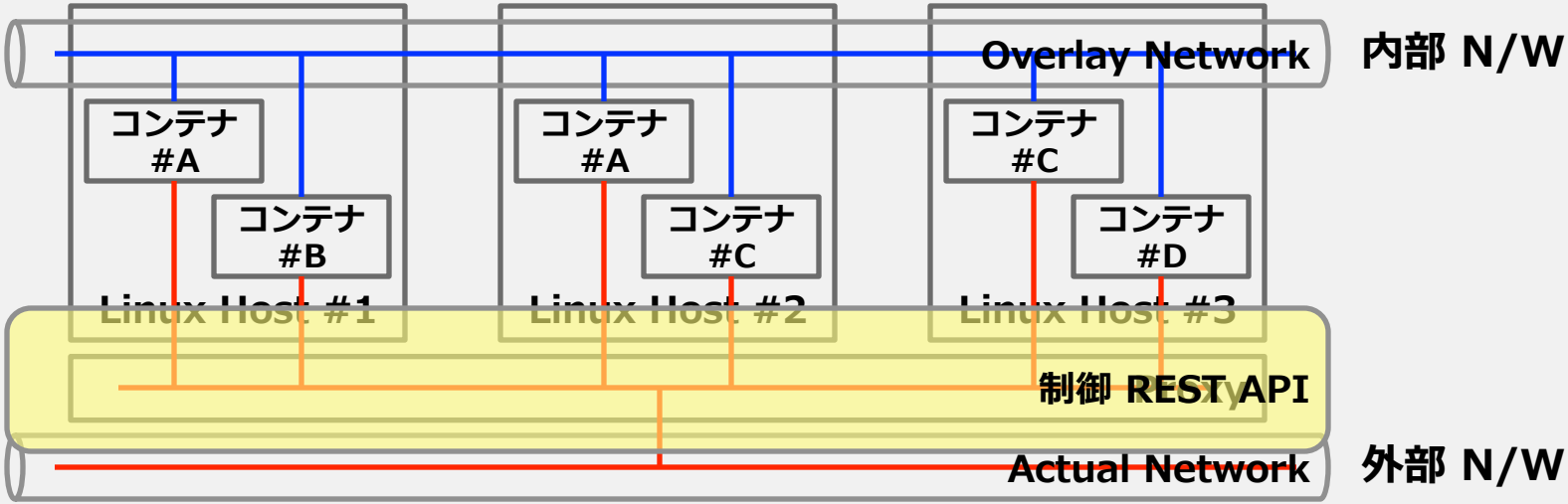
コンテナオーケストレーション – 基本機能



- コンテナのブートストラップ
- コンテナ連携用内部ネットワークの制御
- 外部ネットワークからコンテナへのアクセスの ルーティング, ロードバランシング

redhat.

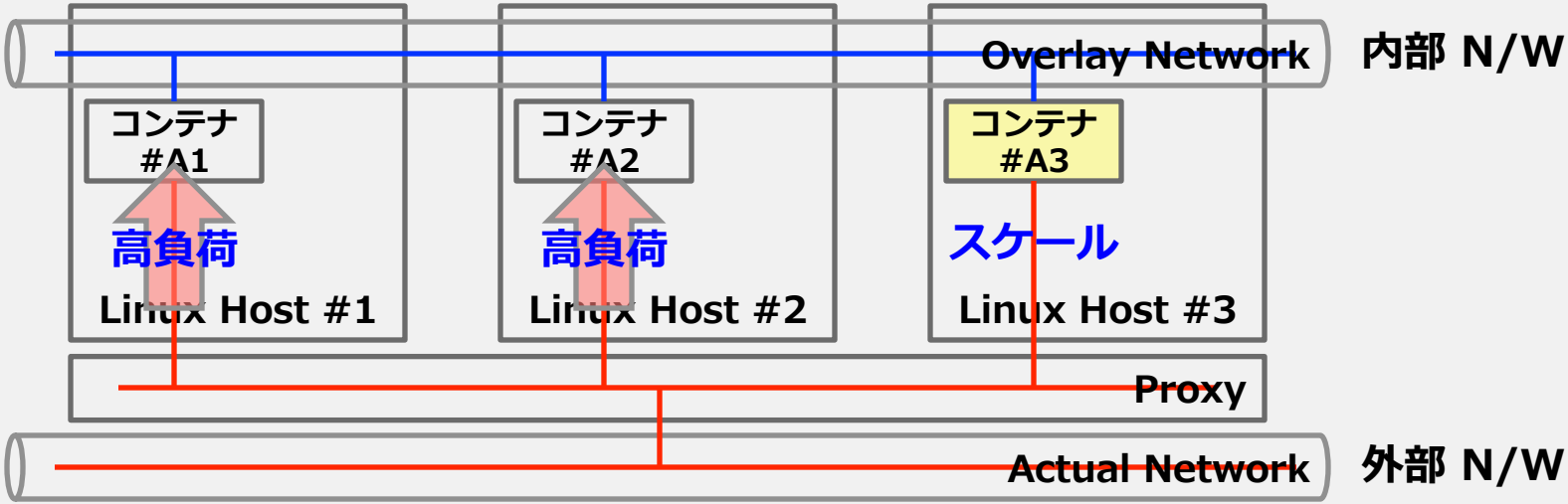
コンテナオーケストレーション – 制御 API



- コンテナの情報の取得
- コンテナのブートストラッピング
- etc.

redhat.

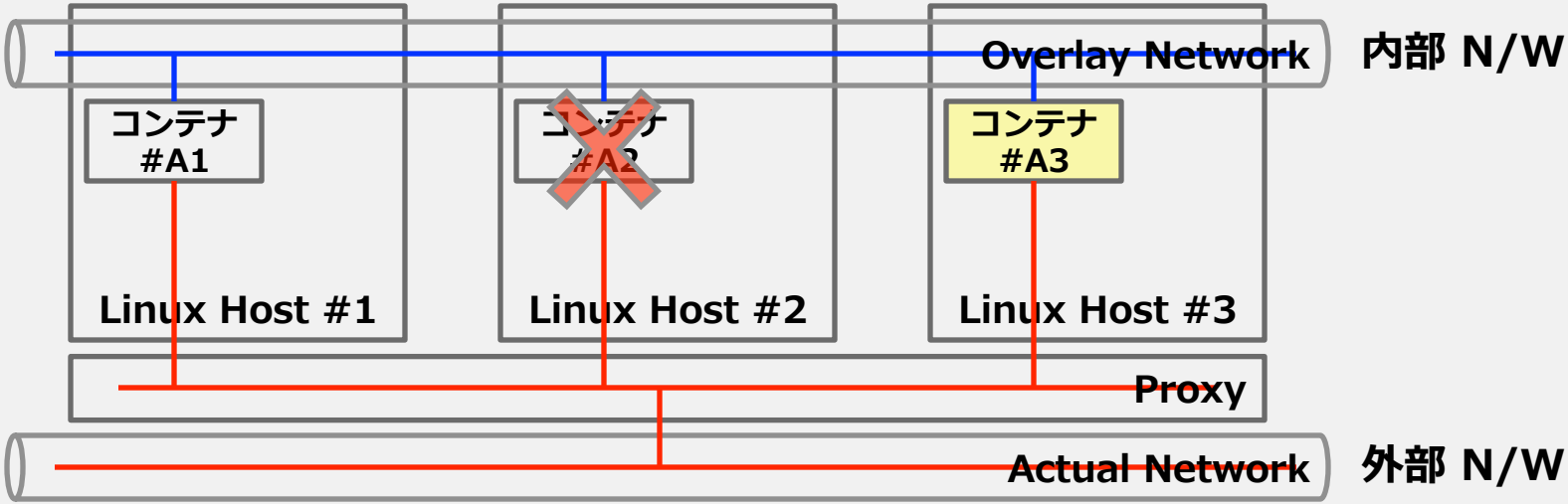
コンテナオーケストレーション – オートスケール



- 負荷の閾値 (任意) に従いスケール
- 新たなコンテナへのロードバランス
- 新たなコンテナを起動するホストのルール

redhat.

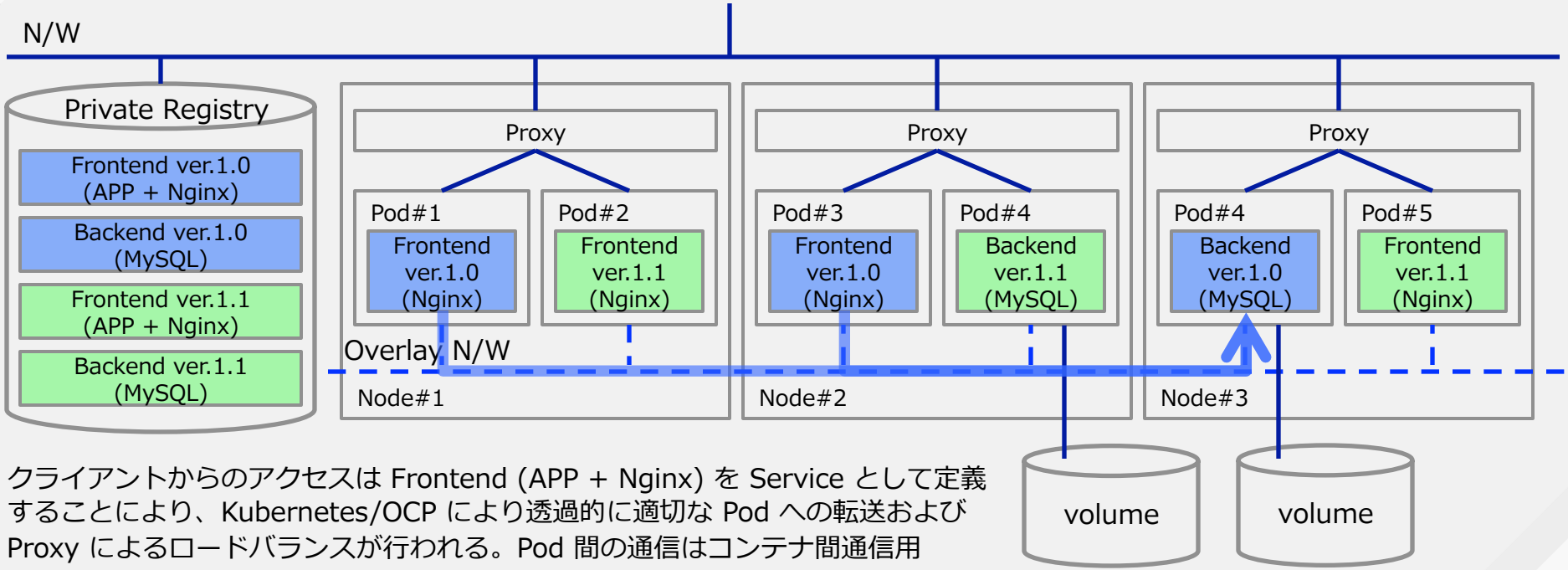
コンテナオーケストレーション – セルフヒーリング



- コンテナ障害発生の検知と新たなコンテナの起動
- 新たなコンテナへのロードバランス
- コンテナの最小稼働数に関するルール

redhat.

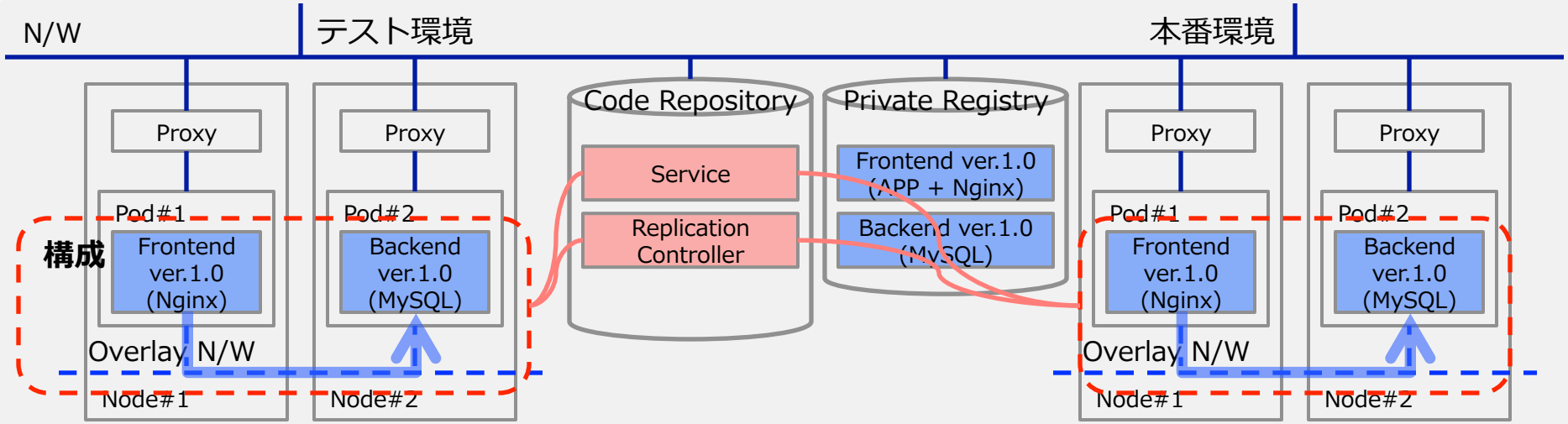
コンテナ環境における一般的な web システムの構成



クライアントからのアクセスは Frontend (APP + Nginx) を Service として定義することにより、Kubernetes/OCP により透過的に適切な Pod への転送および Proxy によるロードバランスが行われる。Pod 間の通信はコンテナ間通信用 Overlay N/W を使用して行われる。

redhat.

コンテナ環境におけるシステム構成の管理



Service, Replication Controller 等の定義により Frontend → Backend のシステム構成を構成管理可能

redhat.

コンテナの効果

概要	解説	効果
システム構成全体の構成管理	Frontend + Backend 全体を設定ファイル (Pod, Service, ReplicationController) で管理可能	・システム構築の高速化 ・構成管理範囲の拡大に伴う開発・運用の効率化
コンテナイメージによる構成管理	OS + M/W + APP を軽量なイメージとして管理可能 ※ コンテナ内の状態 (ファイル 等) についてもイメージ化して管理可能	・システム構築の高速化
コンテナイメージの移植性	コンテナイメージは軽量かつ環境依存設定 (IP 等) が含まれないため環境間 (開発、テスト、本番 等) での移植性が高い。テスト済環境をそのまま本番環境に移植可能。	・テスト済環境の移植によるテスト～リリースの高速化
Immutable Infrastructure	環境に変更を加えるのではなくリリースの度にシステム全体をゼロから構築することで、“認識忘れ”・“想定外” の状態の環境となる危険性を排除する。また、障害発生時の解析容易性を向上させる。	• インフラの明瞭性の向上、不具合発生率の低減、リリース効率の向上 • 障害の解析性の向上
リリース (Blue-Green Deployment)・切り戻しの効率化	コンテナは軽量で稼働するため、既存の H/W リソース上で Blue, Green 両環境を稼働させることによる効率的なリリースが可能。 また、コンテナイメージのバージョニングにより高速な切り戻し作業が可能。	・リリース・切り戻しの高速化 ・無停止リリース
障害の解析	障害が発生した状態のコンテナを開発環境で稼働させることにより効率的・再現性の高い解析が可能	・障害の解析性の向上
テストの網羅性・効率性・自動化	非機能要件のコード化による自動化テストによる網羅率の向上、負荷テスト等の分散処理等によるテストの効率化	・テスト品質の向上 ・テストの高速化

redhat.

非コンテナ環境との比較

概要	効果	評価	解説
システム構成全体の構成管理	・システム構築の高速化 ・構成管理範囲の拡大に伴う開発・運用の効率化	△	・システムのランタイム構成自体を管理することは現実的に難 ・環境依存情報 (IP 等) が設定ファイル上に記載
コンテナイメージによる構成管理	・システム構築の高速化	×	・仮想イメージのバージョニングは現実的に難
コンテナイメージの移植性	・テスト済環境の移植によるテスト～リリースの高速化	×	・仮想イメージの移植性は低い
Immutable Infrastructure	• インフラの明瞭性の向上、不具合発生率の低減、リリース効率の向上 • 障害の解析性の向上	△	・ツール (Ansible, Chef 等) の管理ルールに依存
リリース (Blue-Green Deployment)・切り戻しの効率化	・リリース・切り戻しの高速化 ・無停止リリース	×	・仮想マシンを複数稼働させるのは H/W, N/W リソースの観点で非効率
障害の解析	・障害の解析性の向上	×	
テストの網羅性・効率性・自動化	・テスト品質の向上 ・テストの高速化	×	

redhat.

Kubernetes/OpenShift の関係

Kubernetes :

Google がスタートさせたコンテナオーケストレーションのための OSS、同社のコンテナベースのサービス運用において 10 年以上の実績をもつ Borg と Omega がベースとなっている。※ Borg/Omega は Google Search, Google Mail 等 のサービスの管理に使用させている。Kubernetes プロジェクトは Google, Red Hat, Microsoft, IBM 等のベエンダーによるアップデートが頻繁に行われている。

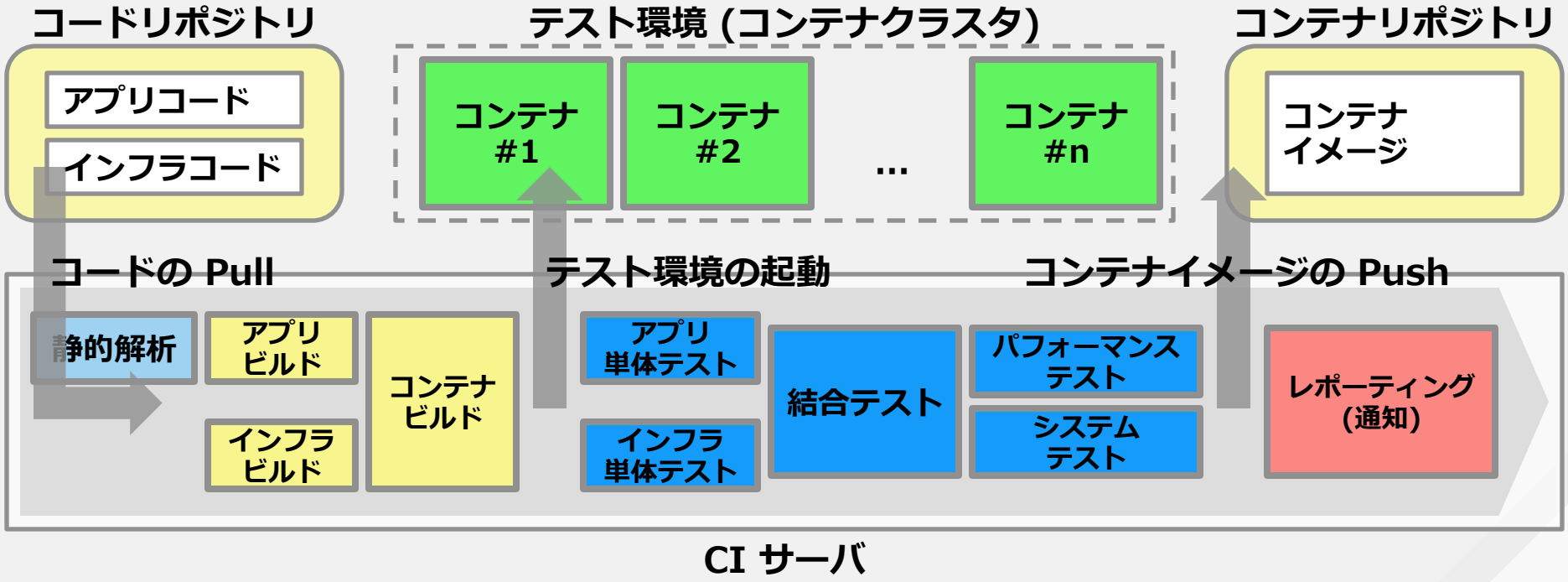
OpenShift :

Kubernetes ベースに 認証・認可、コンテナのビルド、Jenkins コンテナによる CI/CD フロー、管理 GUI/CUL 等の機能を付加したオーケストレーションツール。

※ OpenShift Origin : Red Hat Container Application Platform のオープンソースアップストリーム

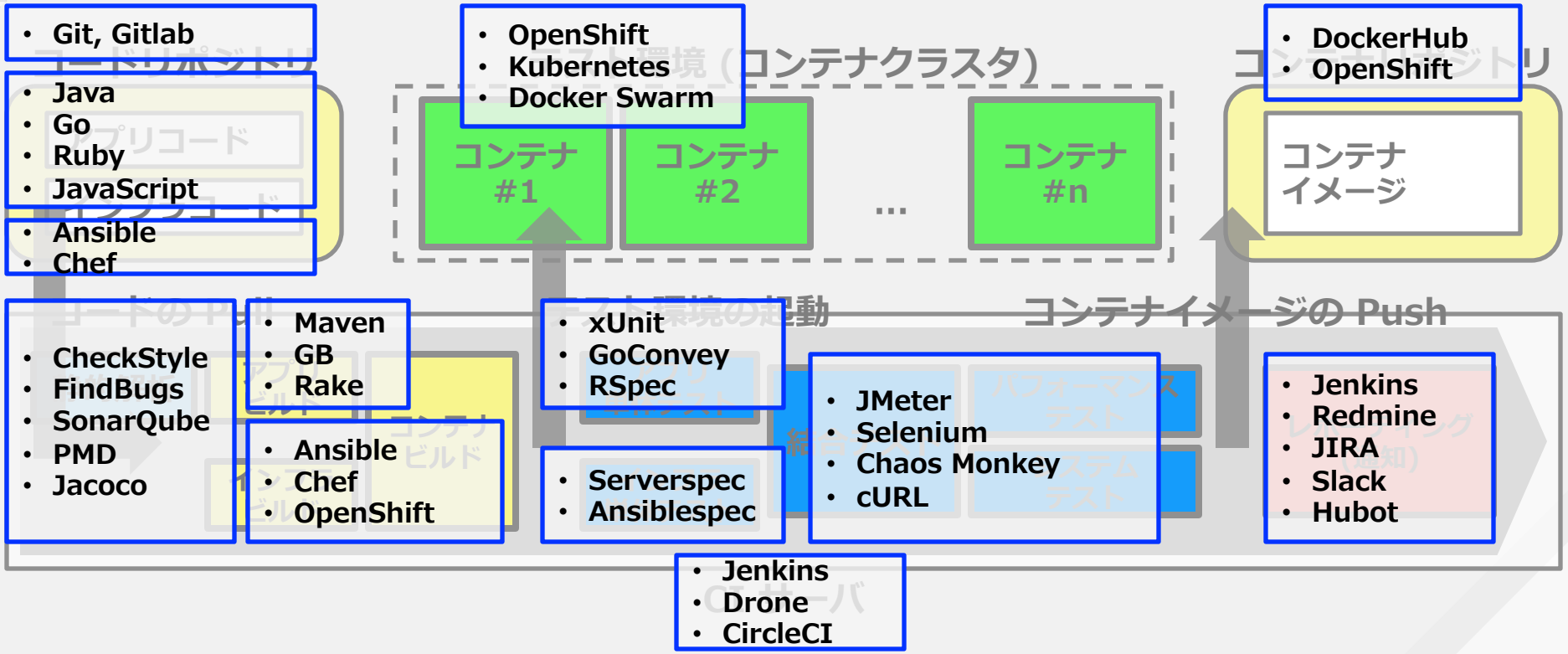
redhat.

CI (継続的インテグレーション)



redhat.

CI のツールセット



redhat.

References

redhat.

References – Agile, Lean, DevOps

[1] “10+ Deploys Per Day: Dev and Ops Cooperation at Flickr” (http://www.slideshare.net/jallspaw/10-deploys-per-day-dev-and-ops-cooperation-at-flickr)
[2] “リーン開発の現場” ISBN-13: 978-4274069321
[3] “アジャイルサムライ” ISBN-13: 978-4274068560
[4] “リーン開発の本質” ISBN-13: 978-4822283506
[5] “アジャイルプラクティス” ISBN-13: 978-4274066948
[6] “ウェブオペレーション” ISBN-13: 978-4873114934
[7] “継続的デリバリー“ ISBN-13: 978-4048707879
[8] ”アジャイルソフトウェア開発の 12 の原則” (http://agilemanifesto.org/principles.html)
[9] “カンバン” ISBN-13: 978-4897979571
[10] “リーン・スタートアップ” ISBN-13: 978-4822248970
[11] “Running Lean” ISBN-13: 978-4873115917
[12] “Effective DevOps” ISBN-13: 978-1491926307
[13] ”カンバン仕事術” ISBN-13: 978-4873117645
[14] “リーンソフトウェア開発” ISBN-13: 978-4822281939
[15] “Comway‘s law“ (https://en.wikipedia.org/wiki/Conway%27s_law)
[16] “エッセンシャルスクラム”
[17] “実践アジャイルテスト”

redhat.

References – Technology, Tool

[16] “マイクロサービスアーキテクチャ” ISBN-13: 978-4873117607
[17] “Infrastructure as Code” ISBN-13: 978-1491924358
[18] “Kubernetes Cookbook“ ISBN-13: 978-1785880063
[19] “DevOps Automation Cookbook” ISBN-13: 978-1784392826
[20] “Practical Devops” ISBN-13: 978-1785882876
[21] “システムテスト自動化 標準ガイド” ISBN-13: 978-4798139210
[22] “チーム開発実践入門” ISBN-13: 978-4774164281
[23] “ドメイン駆動設計” ISBN-13: 978-4798121963
[24] “実践ドメイン駆動設計“ ISBN-13: 978-4798131610
[25] “SOLID“ (https://en.wikipedia.org/wiki/SOLID_(object-oriented_design))
[26] “WildFly Swarm“ (http://wildfly-swarm.io)
[27] “MicroProfile“ (http://microprofile.io)
[28] “レガシーコード改善ガイド” TODO
[29] “リファクタリング” TODO
[30] “kubernetes” (http://kubernetes.io)
[31] “CoreOS” (https://coreos.com)
[32] “flannel” (https://github.com/coreos/flannel/)
[34] “etcd” (https://coreos.com/etcd/)
[35] “docker” (https://www.docker.com)
[36] “OpenShift” (https://www.openshift.org)
[37] “Docker Cookbook”
[38] “Kubernetes Cookbook”
[39] “Mastering CoreOS”

redhat.

redhat.

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos