

～DevOpsツールとWAS Liberty / PureApplicationの連携～

# 1. WAS LibertyとDevOpsツールの高い親和性



# アジェンダ

---

1. 新しいシステム構築・運用
2. 自動化ツールを使用したLibertyインフラ構築・運用
  1. Libertyプロファイル概要
  2. システムの構築・運用フロー
  3. 自動化ツールを使用したLiberty運用

# 1. 新しいシステム構築・運用

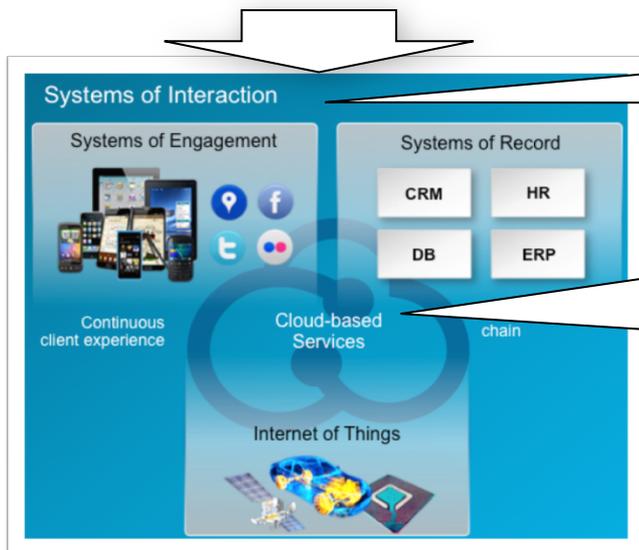
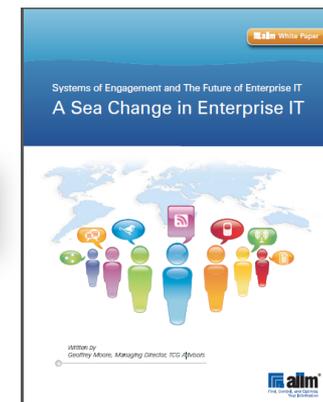
---

# 近年のシステム運用の潮流

- 従来の**Systems of Record**から**Systems of Engagement**へ  
フォーカスするアプリケーションのタイプが変わるとともにインフラのあり方も変化

- ◆ Systems of Engagement and The Future of Enterprise IT  
- Geoffrey Moore

- *But these systems of record are no longer a source of competitive differentiation for organizations. They are a necessary condition of doing business.*
- *This means they have to focus more resources on their core businesses, core competences, and core differentiation.*
- *What will enable this transformation are Systems of Engagement that will overlay and complement our deep investments in systems of record.*



1. ビジネス・ニーズに即応
2. スピーディーな機能追加・修正

## DevOps

1. インフラの状態を **コードとして記述** ⇒ 自動化

## Infrastructure as Code

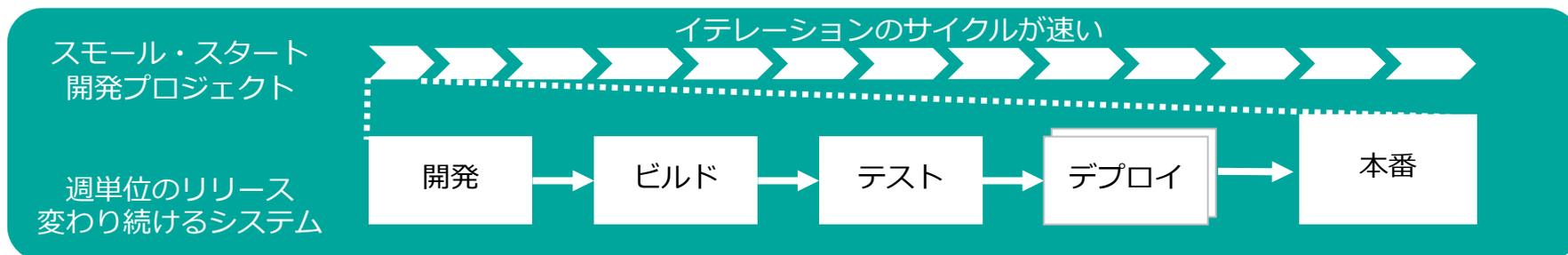
2. 自動化の恩恵により  
⇒ インフラは都度作り直す / 変更を加えない普遍・**不変のインフラ**

## Immutable Infrastructure

# イノベーションのスピードに合わせたシステム開発

## ■ 開発手法：アジャイル開発、DevOps

- ◆ 必要な機能から実装して小規模な開発を反復
- ◆ 開発と運用を一体化して実施



## ■ アプリケーションの側面

- ◆ 機能毎にアプリケーションを実装
  - マイクロサービス・アーキテクチャー
- ◆ サービスの組み合わせが前提
  - 変更に強くテストは最小単位での実施

## ■ インフラの側面

- ◆ クラウド環境の利用
- ◆ 最小構成でのサービス開始
- ◆ インフラ構築の自動化
  - **Infrastructure as code**
    - インフラ構築をコード化して何度も同じ構成を実施
  - **Immutable Infrastructure**
    - 一度作成した環境に手を入れず常に新しい環境を作成

# つまり・・・「新しいシステム構築・運用」の要素としては・・・

## DevOps

- システムの在り方が「業務効率化」から「新しい価値創造」へシフト
- より迅速にビジネスニーズをシステムに反映させる必要性
- アプリケーションへの機能追加・改修とそれに伴う**インフラの変更**をいかに効率よく行えるか？

## Infrastructure as Code

- インフラ構築も**自動化**
- 自動化のためにはインフラ構成や運用を**コード化**しておく必要性
- そして・・・冪等性の獲得ができる(再現性のある)インフラ構築が可能

## Immutable Infrastructure

- 「一度デプロイしたインフラは変えない」「コードを修正して再度デプロイして**作り直すインフラ**」
- 再現可能なインフラ構築によって**再現可能なシステム**を実現（インフラも含めたアプリ稼動が再現可能）

## 結果として・・・

- **再現可能なシステム** ⇒ テストが容易
- 開発～テスト～本番環境を一連の処理として実施可能（本番リリースへの過剰な神経や配慮は不要）

## そして・・・「新しいシステム構築・運用」への1つの回答

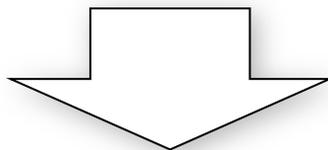
### ■ Liberty + a で実現する新しいシステム構築・運用

#### ◆ コード化

- インフラもコードとして扱えるように・・・
  - **Libertyプロファイル** ではサーバー構成を1つのファイルに記述(ファイル分割も可能)

#### ◆ 自動化

- 各種**自動化ツール**が充実化
  - IBM製品 : Rational 製品群 (UrbanCode、Test Workbench・・・)
  - OSS : Chef、Maven、Jenkins・・・



本日のセミナーでは・・・

Libertyプロファイルと自動化ツールを組み合わせ

新しい時代の新しいシステム運用を実現するためのシステム構築例を紹介

## 2. 自動化ツールを使用したLibertyインフラ構築・運用

1. Libertyプロフィール概要
2. システムの構築・運用フロー
3. 自動化ツールを使用したLiberty運用

# Libertyプロファイルの特長

## ① Java EE 7対応 & 新機能提供

Java EE 7 標準に準拠したアプリを完全サポート  
JAX-WS, JAX-RS, JMSもサポート  
新機能も継続的に提供

## ② 軽量ランタイム

メモリー使用量が小: 60MB程度～  
ディスク使用量も100MB以下  
起動が速い: 5秒程度

## ③ Unzipによる導入とデプロイ

パッケージをした  
サーバー + アプリ + 構成情報を  
Unzipでデプロイ可能

## ④ 簡単な構成

最低限必要な構成ファイルはserver.xmlひとつだけ  
デフォルトベースで簡単構成

## ⑤ 統合ツール (WDT)

高機能なEclipse用の連携ツール  
を無償で提供  
Eclipseから簡単に使用可能

## ⑥ 自動化ツールとの連携

多くのOSSツールに  
無償でプラグインを提供

## ⑦ 様々な環境で稼動

オンプレ、クラウド(IaaS、  
PaaS)で稼動可能

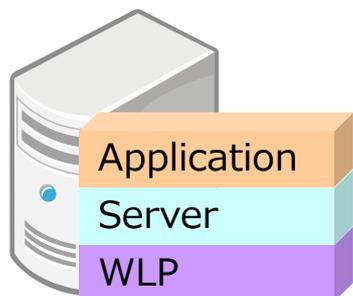
## ⑧ 構成の動的変更

構成変更やアプリケーションの更新  
は再起動なしに反映

WAS V8.5.5.x Liberty & WDT

# Libertyプロファイルのパッケージング機能概要

サーバー単位のためLiberty Collective  
環境全体のPackageは不可



Application アプリケーション情報  
Server サーバー構成  
WLP ランタイム環境

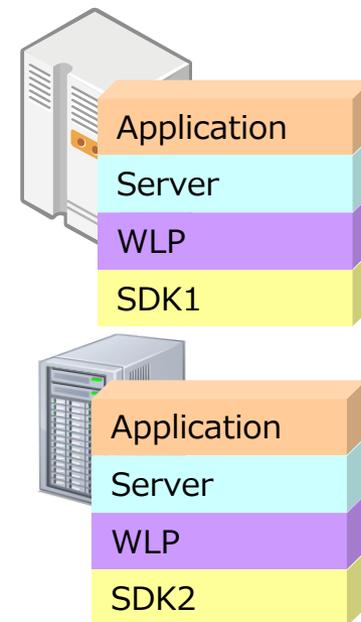
パッケージング



server.zip / server.jar

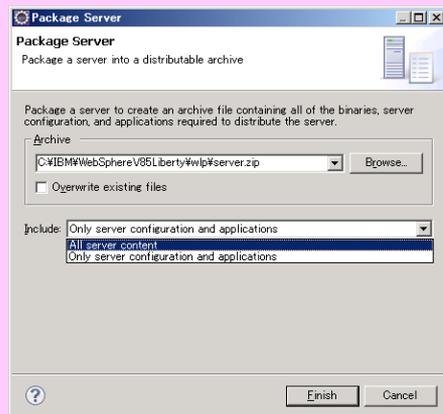
展開

展開



## WDTのパッケージ化ウィザードを利用した場合

[例]



## serverコマンドのpackageアクションを利用した場合

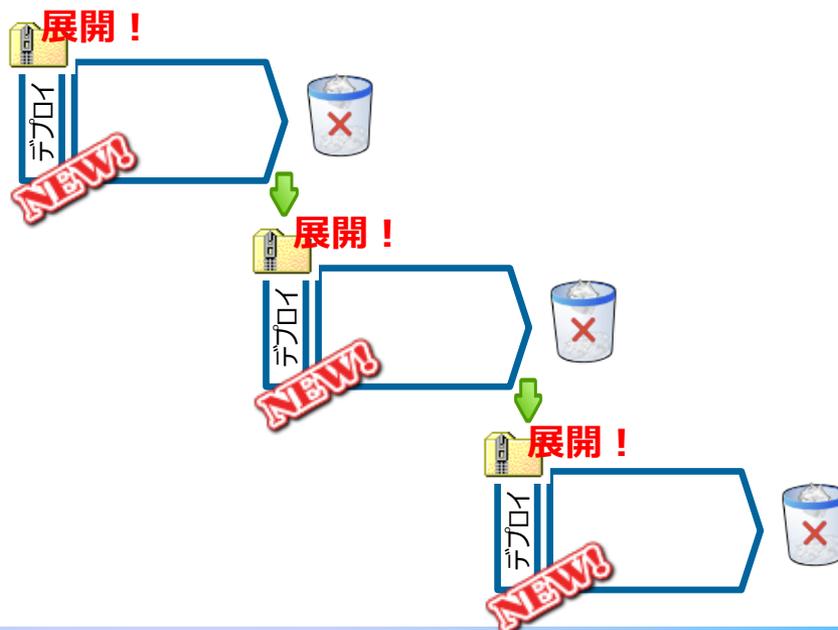
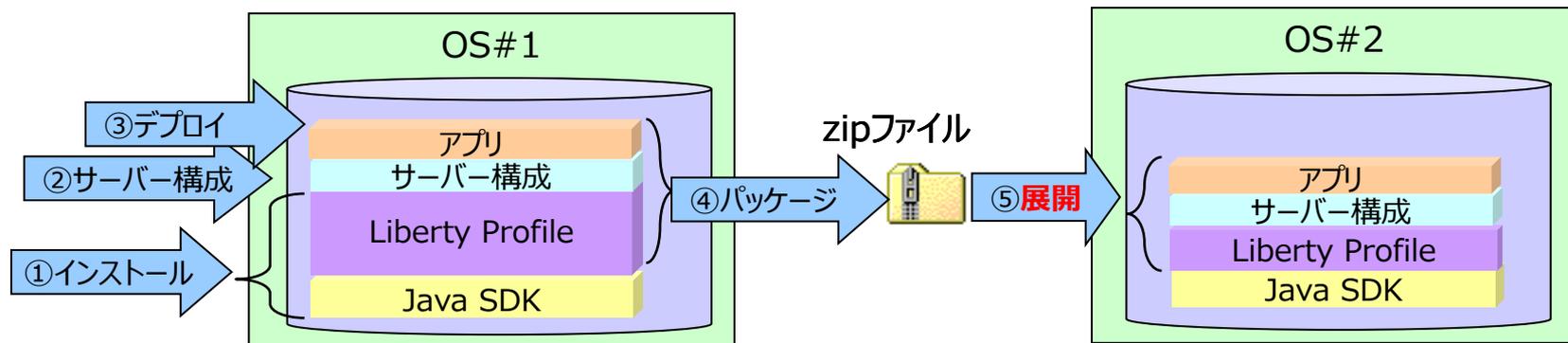
[例] # <WLP>/bin/server package <servername> --archive=server.zip

- 同一マシンの別ロケーション
- 別マシン
- 別プラットフォーム などに展開可能

ファイル名に「.jar」を付けることで  
Jar形式のパッケージングも可能

# Libertyプロフィールによる Immutable Infrastructure の実現

- Libertyプロフィールのパッケージング機能を使った Immutable Infrastructure



## 作っては壊すシステムへ

常に新たな環境を作り直すことで  
変化に強いシステム環境を維持する

# Libertyプロファイルの各種自動化ツールとの連携・親和性

- 各種CI・Buildツールとの連携機能を提供
- UrbanCode連携プラグインを提供
  - ◆ <https://developer.ibm.com/urbancode/plugin/websphere-liberty-ibmucd/>
- OSSツールは連携機能をGitHubで公開
  - ◆ 継続的インテグレーション: UrbanCode, Jenkins
  - ◆ ビルドツール: Ant, Maven, gradle
  - ◆ 構成管理ツール: CHEF, Puppet



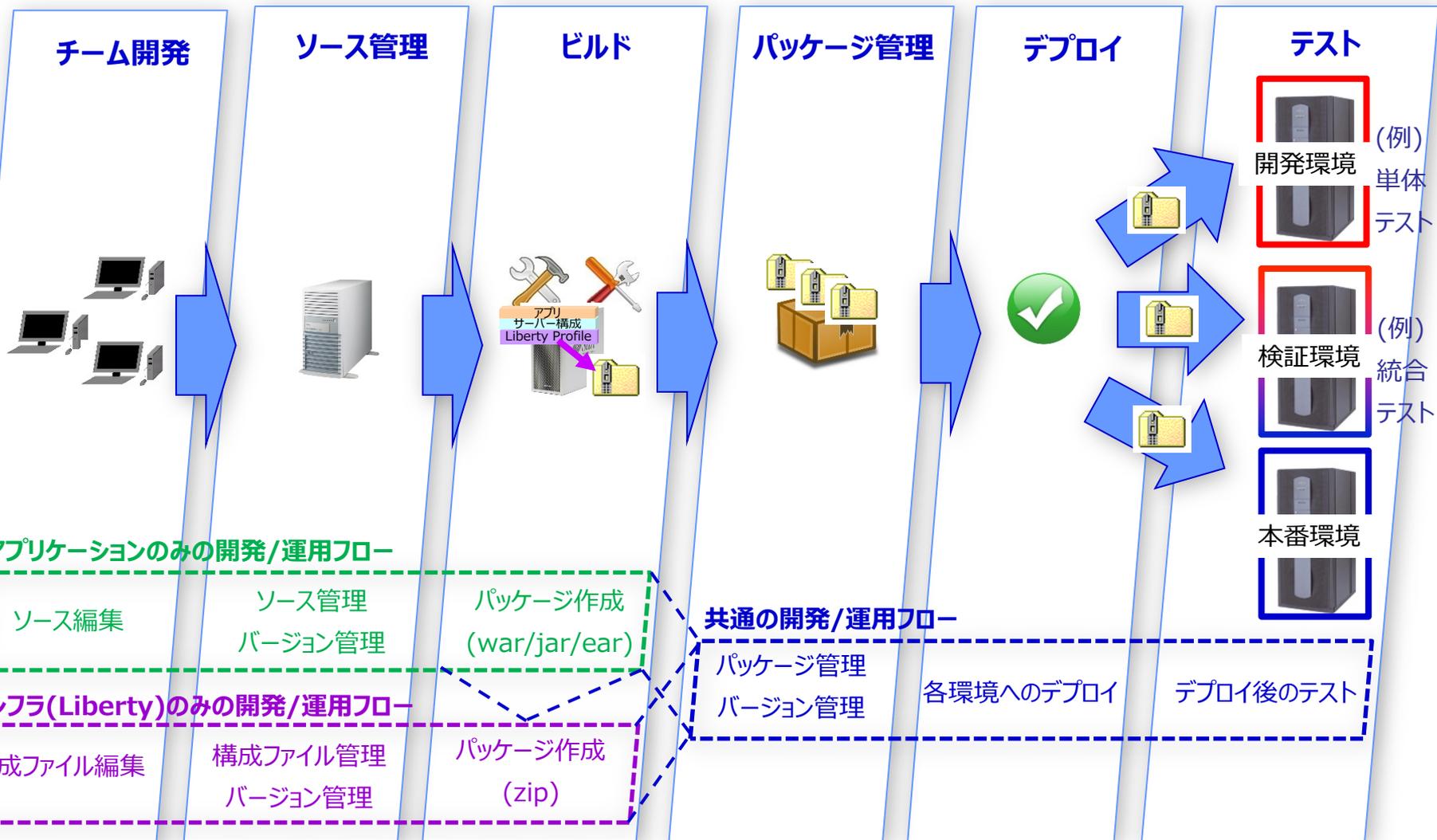
<https://github.com/wasdev>



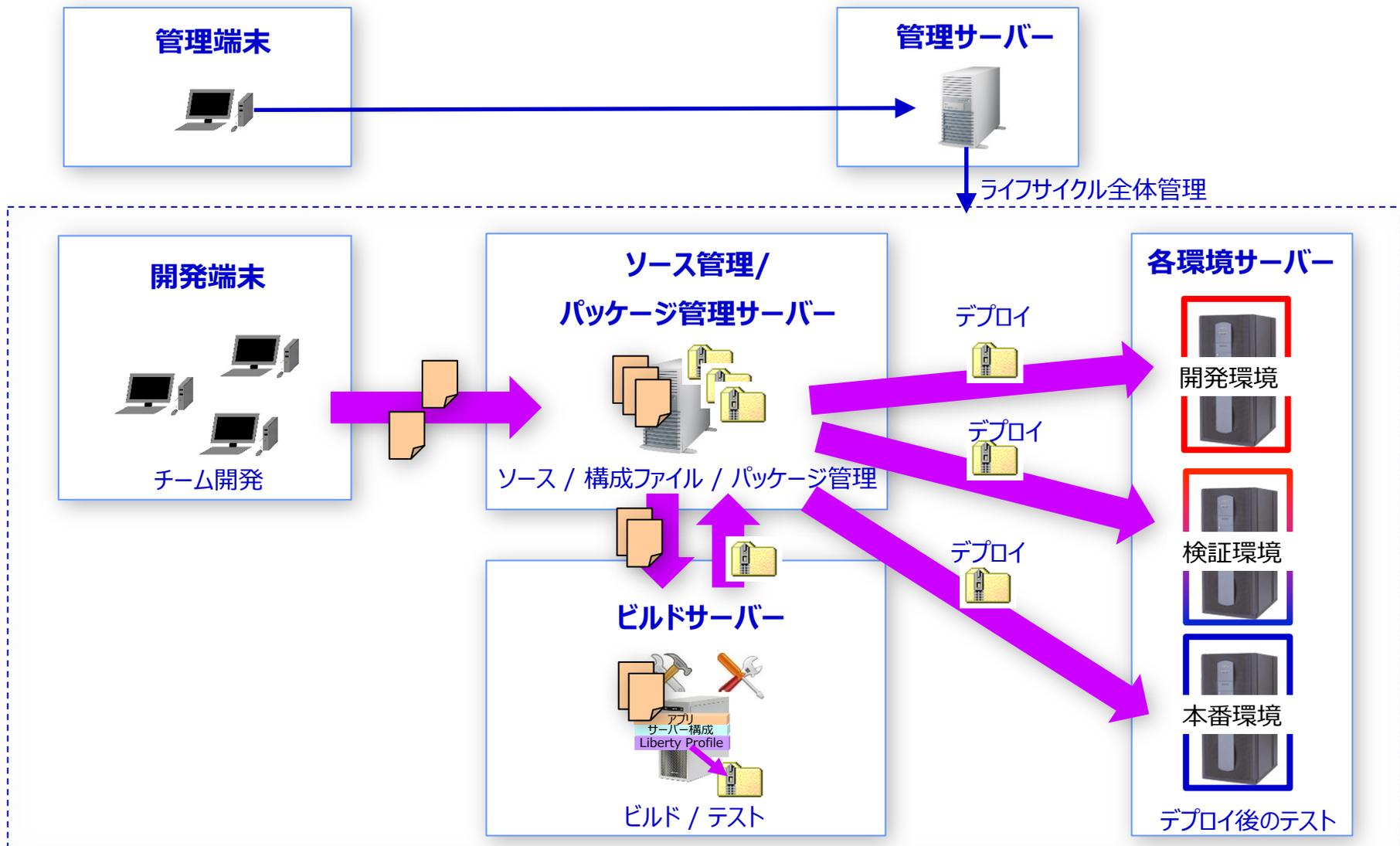
## 2. 自動化ツールを使用したLibertyインフラ構築・運用

1. Libertyプロファイル概要
2. システムの構築・運用フロー
3. 自動化ツールを使用したLiberty運用

# フェーズで見るシステムの構築・運用フロー



# トポロジーで見るシステムの構築・運用フロー



## 2. 自動化ツールを使用したLibertyインフラ構築・運用

1. Libertyプロファイル概要
2. システムの構築・運用フロー
3. 自動化ツールを使用したLiberty運用

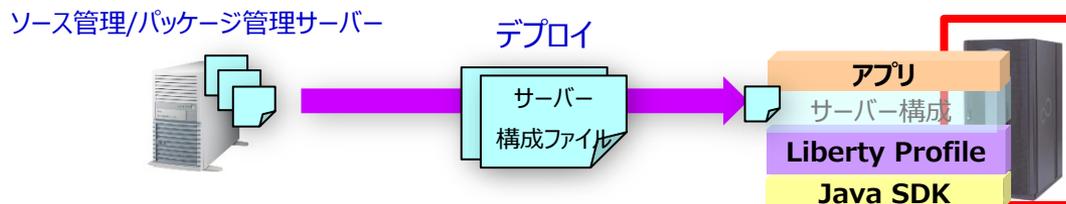
## 各フェーズにおいて利用可能な自動化ツール

フェーズ	IBMツール	OSSツール (オープンソース・ソフトウェア)
チーム開発	IBM Rational Team Concert	Git Subversion
ソース管理 パッケージ管理		
ビルド	Rational Build Forge	Ant Maven gradle
デプロイ	IBM UrbanCode Deploy IBM PureApplication	Puppet Chef Jenkins
ライフサイクル 全体管理		
テスト	IBM Rational Quality Manager IBM Rational Test Workbench IBM Rational Test Virtualization Server	JUnit Arquillian

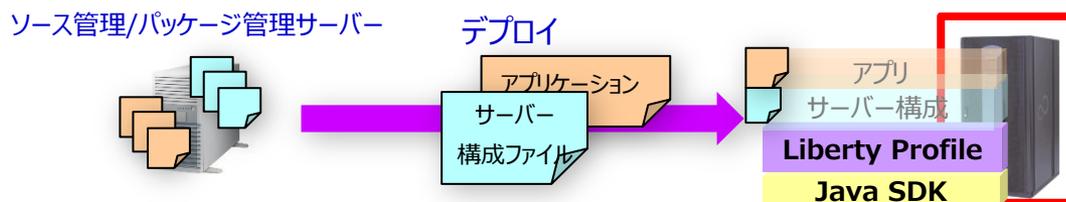
# 自動化ツールを使用したLiberty運用における検討ポイント（1/4）

## ■ デployする単位としてのどのようなパターンを想定しておくか？

- ◆ サーバー構成ファイル(server.xml等)のみをデプロイ



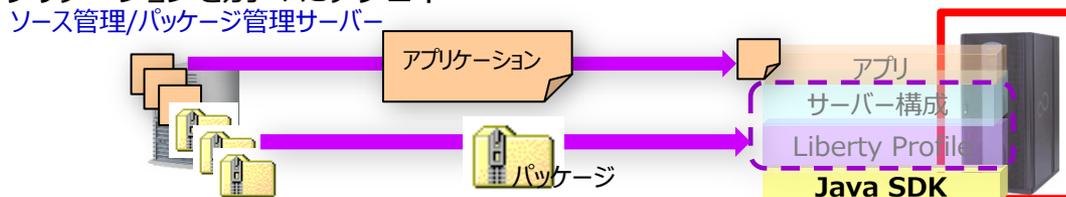
- ◆ サーバー構成ファイルとアプリケーションをデプロイ



- ◆ Libertyランタイムやアプリケーションをまとめたパッケージ(zip)をデプロイ（構成によりJDKを含めることも可能）



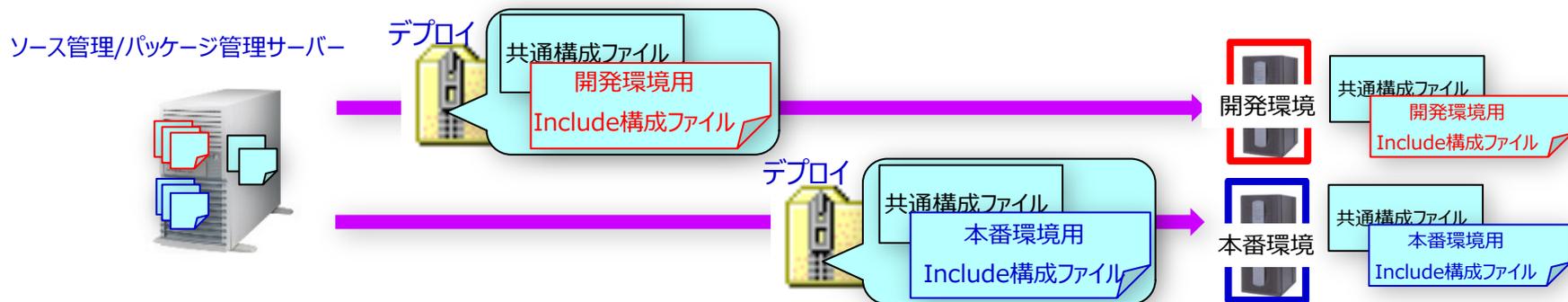
- ◆ Libertyランタイムとアプリケーションを別々にデプロイ



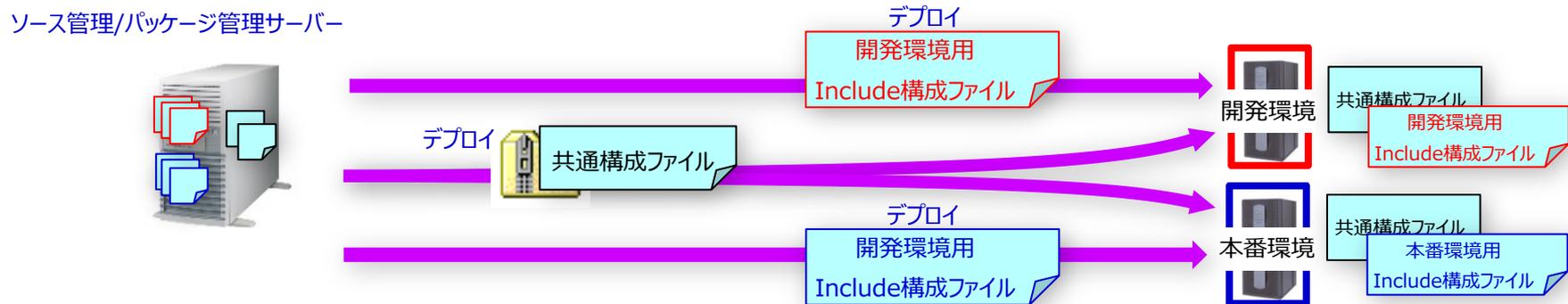
# 自動化ツールを使用したLiberty運用における検討ポイント（2/4）

- 環境固有の構成(DB接続先等)を反映する方法としてどのようなパターンを想定しておくか？

- ◆ 環境固有の構成ファイルと環境共通となる構成ファイルを一緒にパッケージして環境毎にまとめてデプロイ

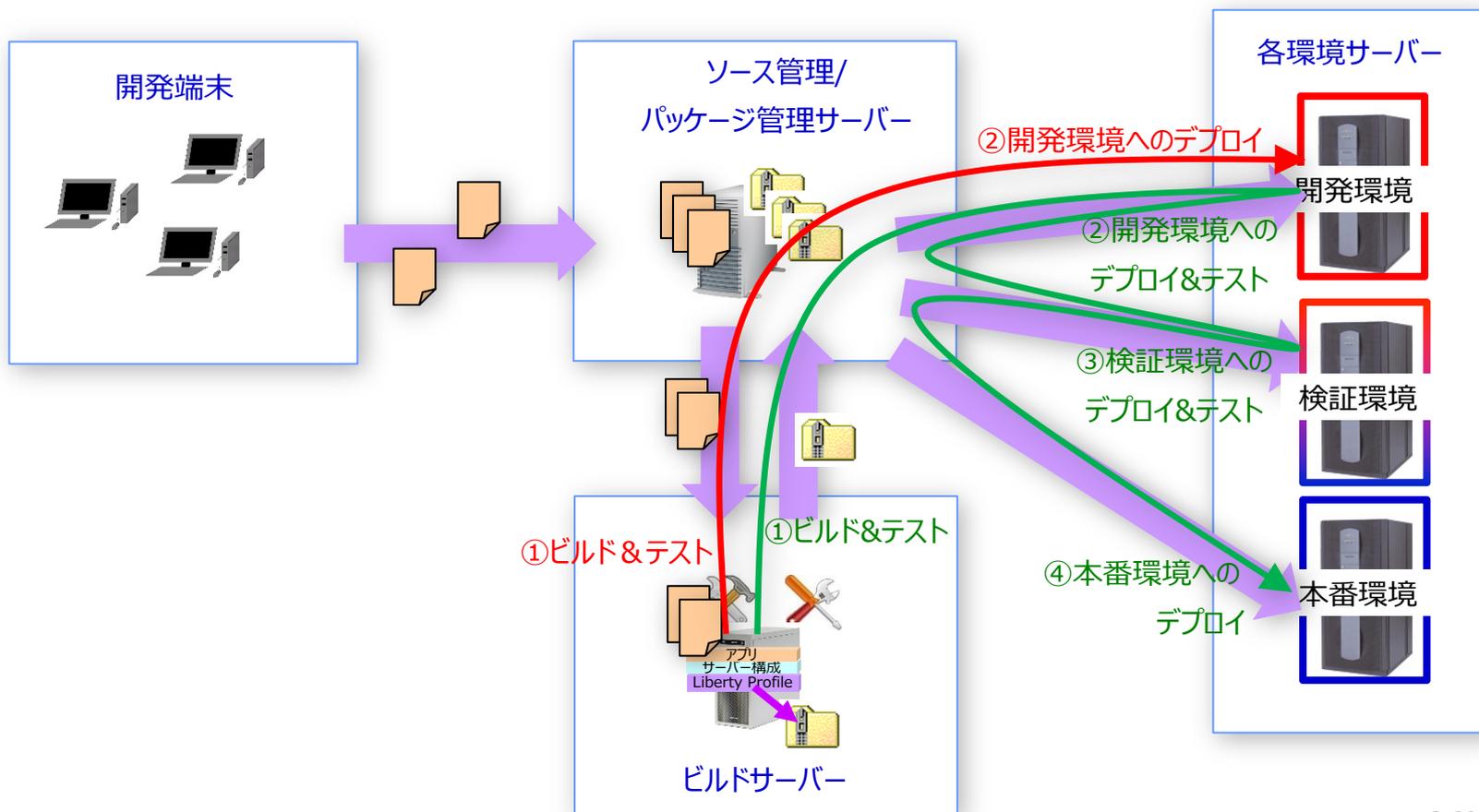


- ◆ 各環境毎にIncludeされる構成ファイルを作成して共通部分パッケージのデプロイ後に別途デプロイ



# 自動化ツールを使用したLiberty運用における検討ポイント（3/4）

- フェーズ全体の中でどこまでのスコープを一連のジョブで実行するか？
  - ◆ ビルドサーバー上でビルド後にテスト実行して成功したら開発環境にデプロイ（下図赤色の矢印）
  - ◆ 上記に加えて開発環境と検証環境へのデプロイとテスト実行に成功したら本番環境にデプロイ（下図緑色の矢印）



# 自動化ツールを使用したLiberty運用における検討ポイント（4/4）

## ■ その他の検討ポイント

- ◆ どのツールを組み合わせで使用するか？
- ◆ どのようなトポロジーを構成するか？
- ◆ どこまで自動化してどこから手動運用とするか？
- ◆ 自動化の中で承認プロセスはどうするか？
- ◆ 誰が各プロセスの主体となって処理をキックするか？  
（コード修正の自動検出, 各開発者, リリース担当者 etc）
- ◆ どのようなリリースを対象とするか？（通常リリース, 緊急リリース, Fix適用 etc）
- ◆ 各サーバー上でどこまでのテストを実施するか？
- ◆ 一連ジョブのチェックポイントやエラー時対応はどうするか？
- ◆ 各ツール自体のメンテナンスや可用性の確保をどうするか？

などなど

# 自動化ツールを使用したLiberty運用パターン例

- この章では次ページより下記運用パターンを紹介

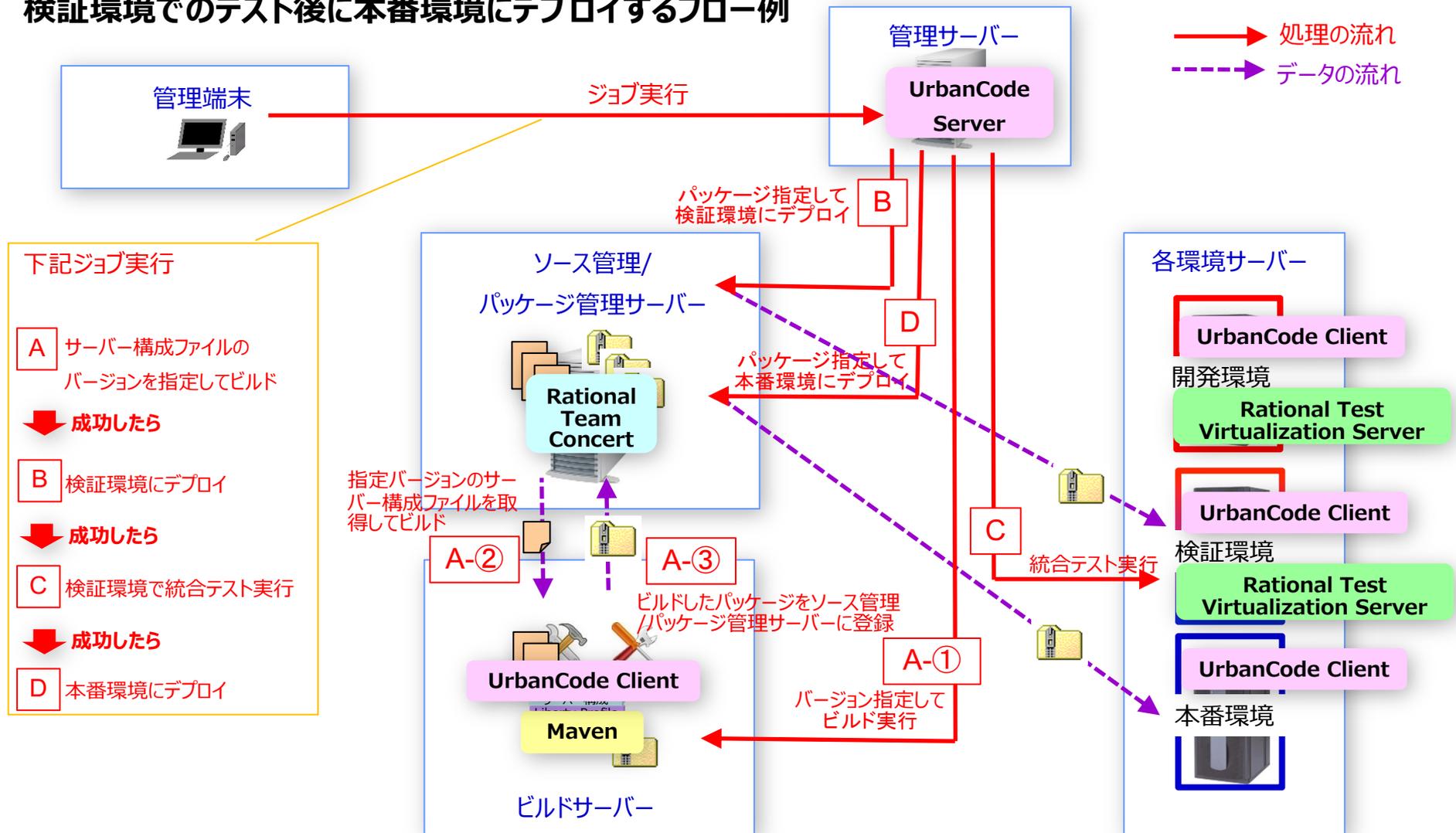
- ① IBMツール構成

- ② OSSツール構成

- ③ 最小トポロジー構成

# 自動化ツールを使用したLiberty運用パターン例 ①IBMツール構成

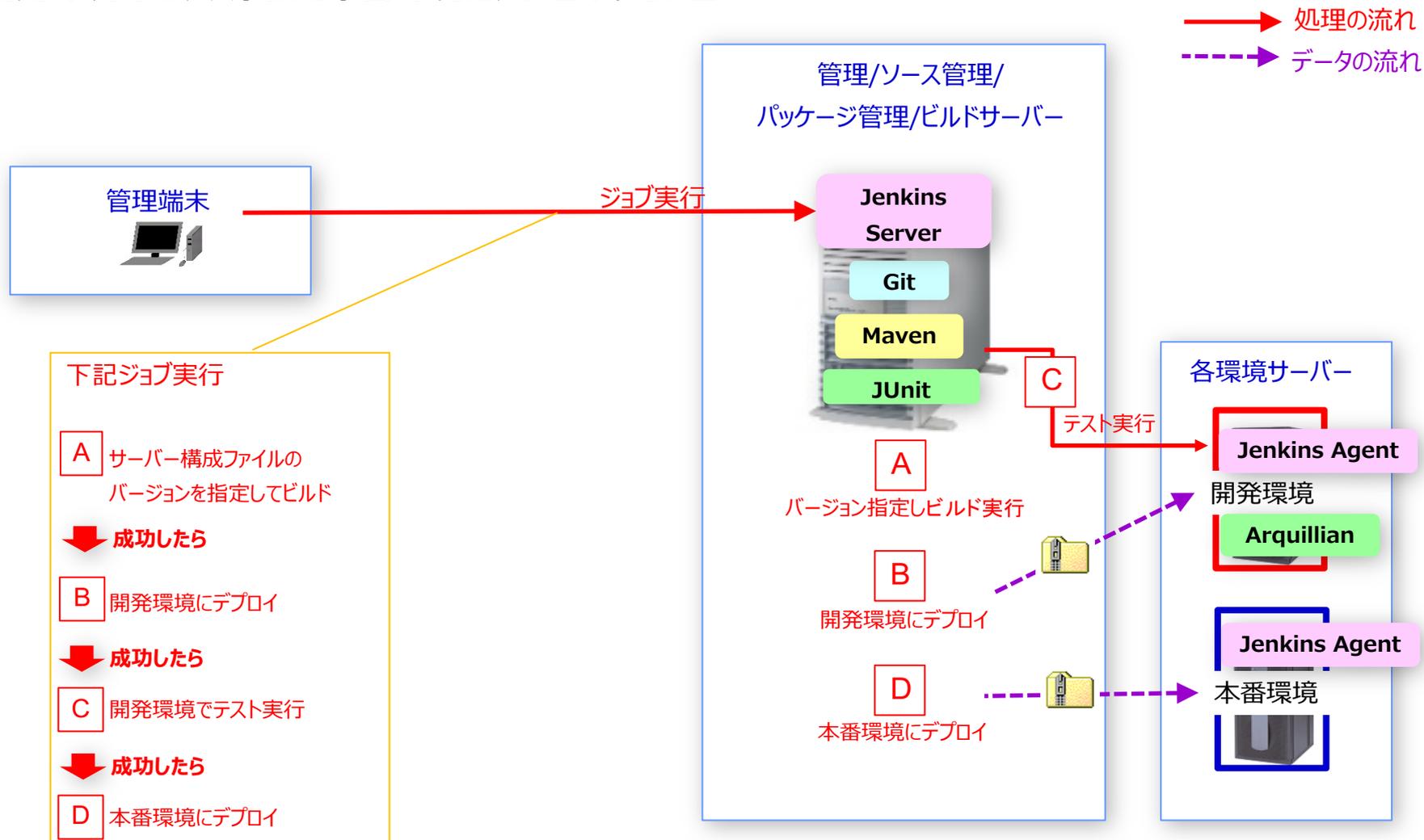
## 検証環境でのテスト後に本番環境にデプロイするフロー例





# 自動化ツールを使用したLiberty運用パターン例 ③最小構成トポロジー

## 開発環境でのテスト後に本番環境にデプロイするフロー



---

**END**