



自動化を支えるCI/CD パイプラインの世界

【4-B1-3】DevOps

Shingo.Kitayama

Introduction

 shkitayama

 spchildren



← KPP

Shingo Kitayama

日本ヒューレット・パカード株式会社 テクニカルアーキテクト
オープンソースソリューションの提案、コンサルティング、および構築デリバリーを担当

- 元楽天 国際市場インフラ担当
- Ansibleの人だった。気・が・す・る
- Mesos User Group Tokyo(MUGT)運営



openstack®

HPE

POINTNEXT

Now on Sale



DevOps/自動化セッションについて



- ❑ DevOps/自動化セッションでは、最新の自動化技術の基礎からDevOpsへの応用、そして実践までを取り上げ、運用にまつわるオペレーションの課題と解決について掘り下げていきます。

- ❑ プログラム
 - ❑ Day1
 - ❑ [4-B1-2] 進化を続ける運用自動化ツール
 - ❑ [4-B1-3] CI/CDパイプラインを支えるツールと組織
 - ❑ [4-B1-4] 実環境での運用自動化とその管理方法
 - ❑ [5-A1-5] Googleのインフラ技術に見る基盤標準化と DevOpsの真実
 - ❑ Day2
 - ❑ [5-A3-10] パネル: 運用自動化で実現しなければならないこと、そのために必要なもの

本日本日お伝えしたいこと

自動化を支えるCI/CDパイプラインの世界

- ・今あらためて考える、現在のOpenStackの価値とは
- ・Infrastructure as Codeに必要なこと
- ・CI/CDを運用することの注意

※お伝えしないこと

- ・Infrastructure as Codeの現場の裏テクニク
- ・各ツールの技術要素やその実装

意訳: 各コンポーネントの運用や現場の課題は、次のセッション見てね。

本日のAgenda

自動化を支えるCI/CDパイプラインの世界

- 自動化に必要なこと
- Infrastructure as Codeの運用自動化
- CI/CDパイプラインの運用
- まとめ

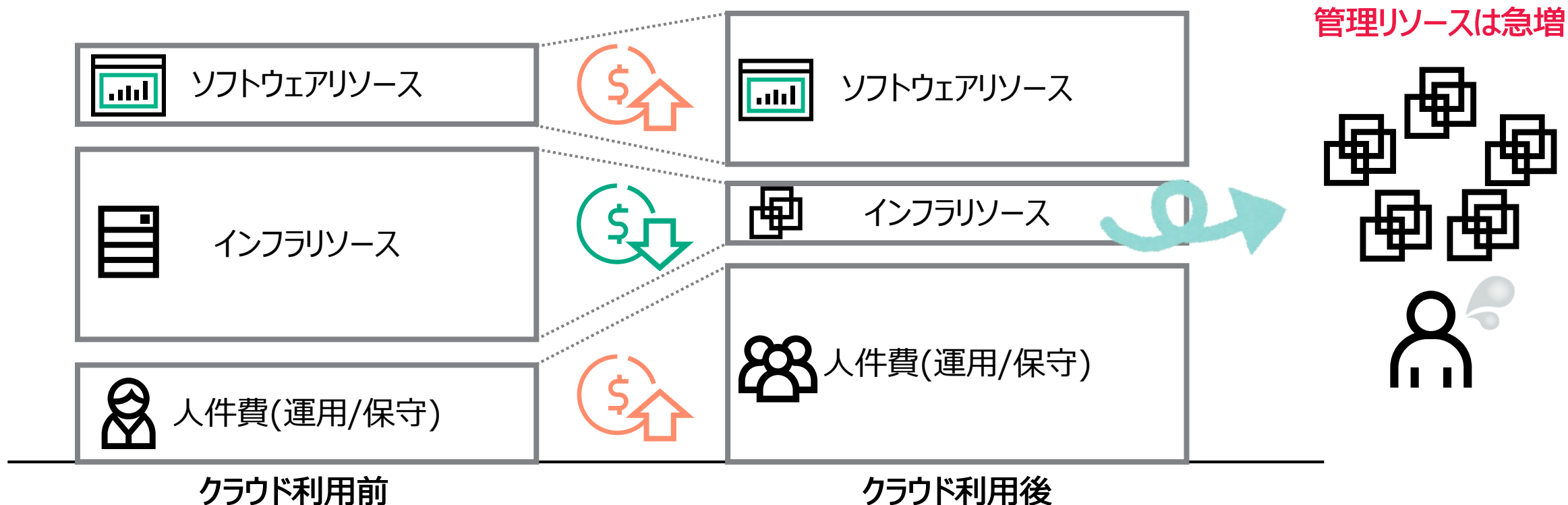
自動化に必要なこと

- ▣ Infrastructure as Codeの運用自動化
- ▣ CI/CDパイプラインの運用
- ▣ まとめ

インフラ自動化が必要な理由

インフラのコスト構造

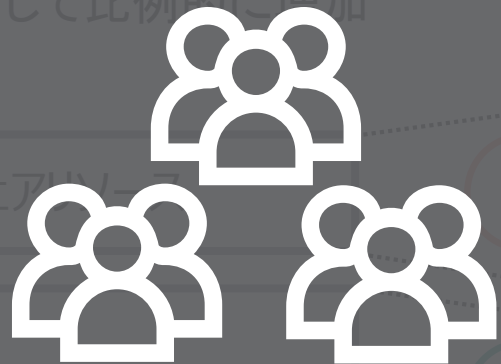
インフラリソースの単価が下がる一方で、リソースの量は増加
人件費はリソース量に対して比例的に増加





自動化して人件費を下げれば、コストが下がる!!

“Automation = Reduce Cost”



Automation
Tool



管理リソースは
増加

“Infrastructure as Code 導入!!”

クラウド利用前

クラウド利用後



自動化して人件費を下げれば、コストが下がる!!

~~“Automation = Reduce Cost”~~



Automation
Tool



管理リソースは
増加

“Infrastructure as Code 導入!!”

って...ベンダーに騙されてませんか!?

そんな簡単なわけないですよ。

Infrastructure as Codeのメリット

アジリティの高いサービスが提供出来るように”み・え・る”

オペレーション工数の削減

従来、手動で行ってきた作業をコード化、自動化することにより、オペレーション工数および納期の短縮が期待できる。

オペレーション品質の向上

作業をコード化して、自動化することにより、オペレーション品質を均一に保つ効果がある。

システム運用の標準化の促進

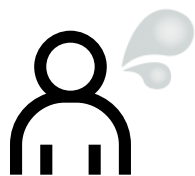
自動化やバージョン管理を適切に行うことで、システム運用のポリシーや業務標準化を形成できる。また、コードを再利用することにより無駄を排除し、継続的インテグレーションやデリバリを実現できる。

作業統制の強化

作業オペレーションを自動化することにより、内部統制やセキュリティ対策面での効果が期待できる。

Infrastructure as Codeのデメリット

再利用化しなければ高コスト化するだけ



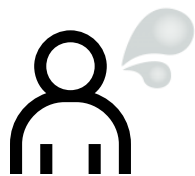
作業方法が増える

一部既存の手順運用のまま、一部自動化運用を適用。とすると、いままでよりもルールが増えることになり、自動化が推進されない。



適用に時間を要する

少しの変更のためだけにコード化すると、返って時間がかかる。また、例外処理など人間の判断を全てコード化することが難しい。



ツールの学習コストが必要

チームメンバー全員がそのツールに対する知識をもっていなければ、コードの変更ができない。学習コストの低いツールを検討することが重要。

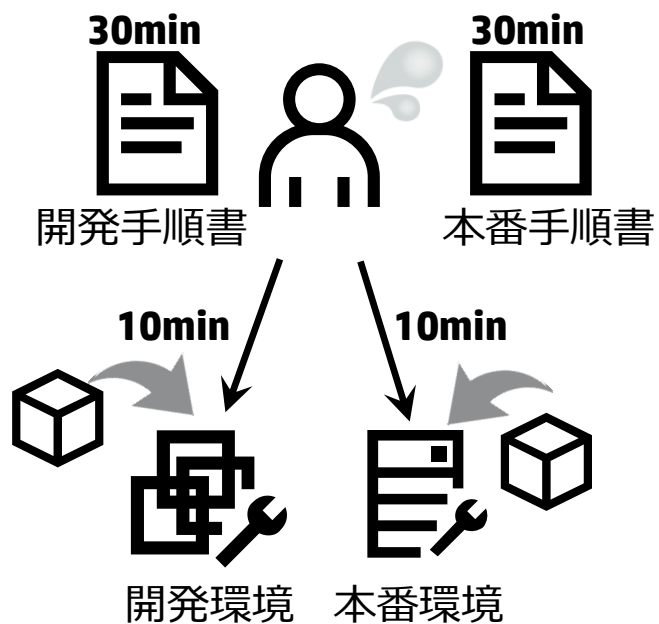
再利用化するためには、インフラリソースの標準化が必要。

インフラ自動化に必要なこと

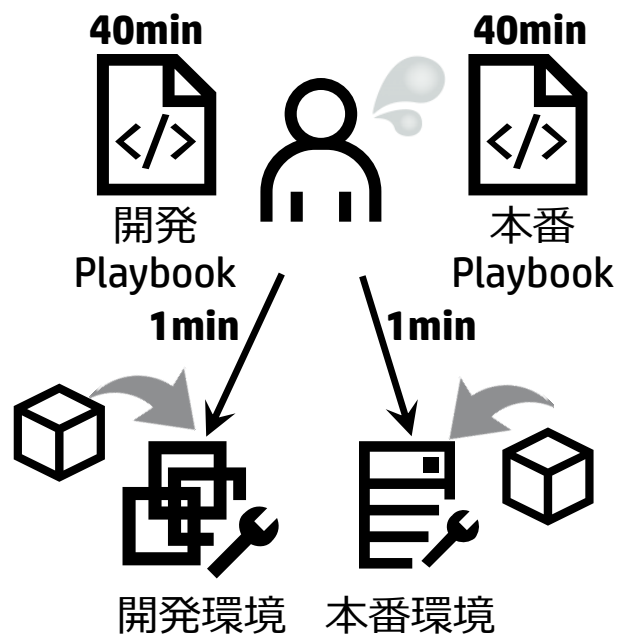
標準化を前提とした自動化を目指すことが重要

“Automation = Standardization and Reuse = Reduce Cost”

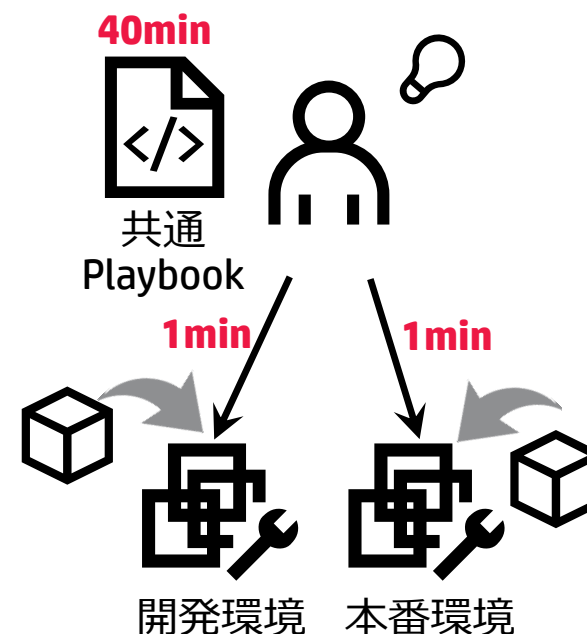
手動作業



手順書を自動化



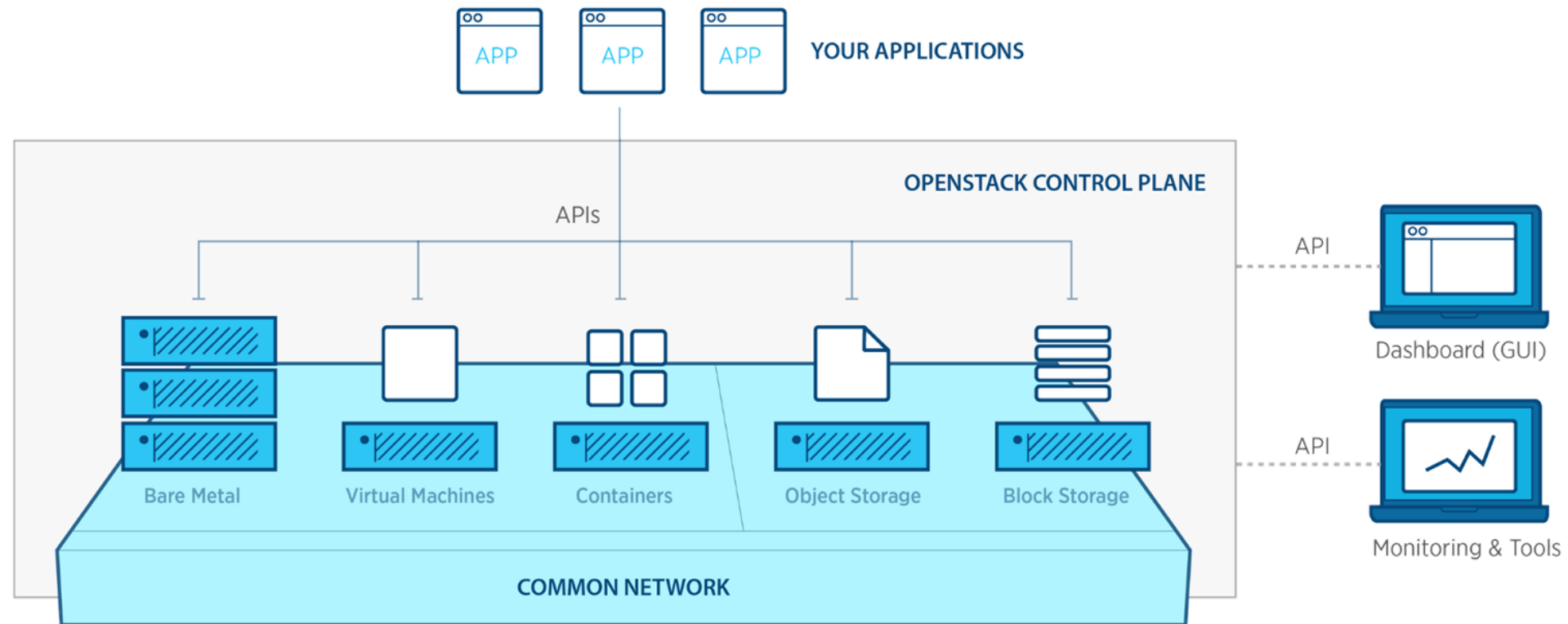
標準化された自動化



OpenStackが提供している機能的価値

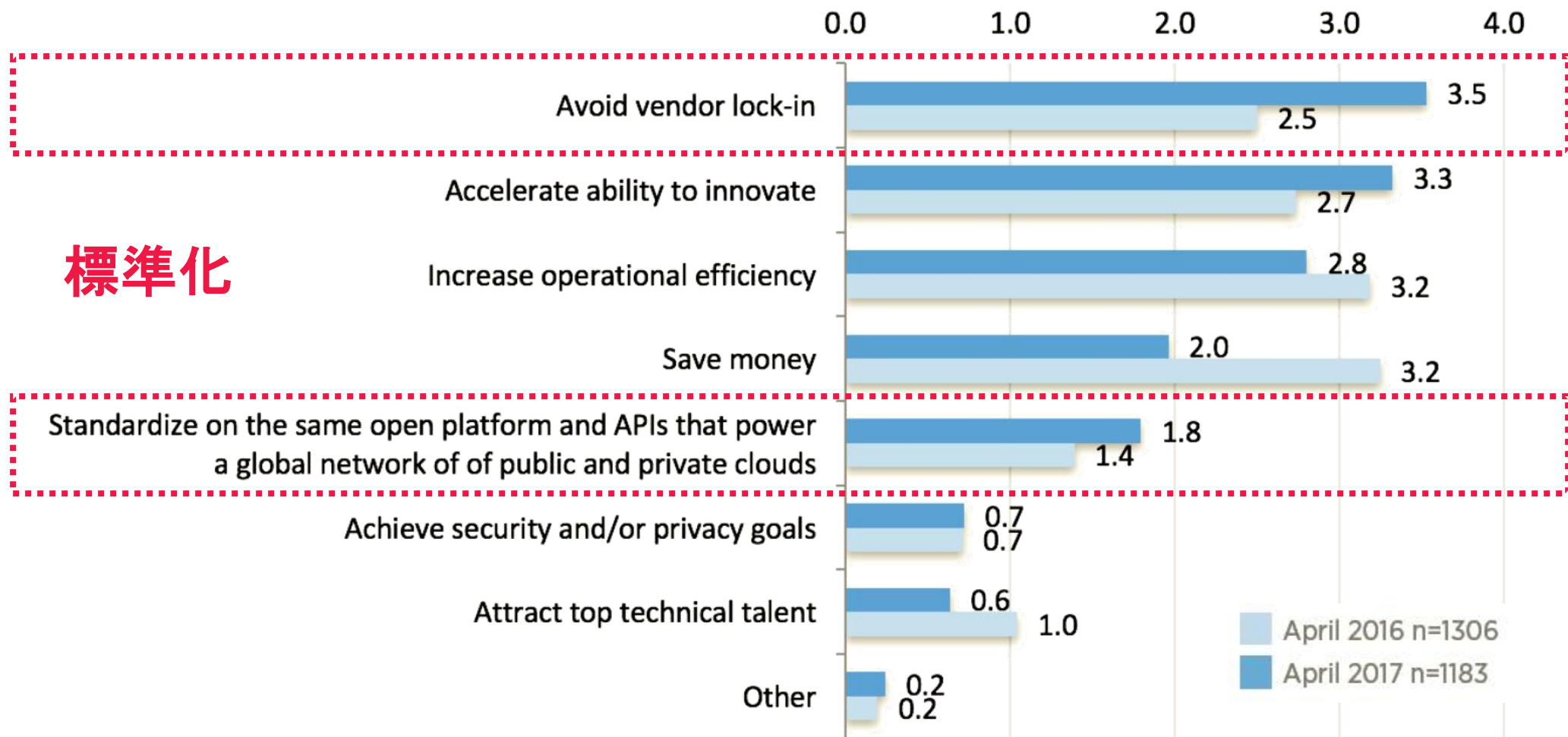
標準化を前提とした自動化プラットフォーム

OpenStackの価値はインフラストラクチャーの違いを吸収するためのオープンなAPI を提供しているという点
つまり、**標準化を推進したアジリティの高いインフラリソースを提供。**



企業がOpenStackを採用している理由

Why do organizations choose OpenStack? By OpenStack User Survey (April 2017)



現在のOpenStackの価値

IaaS基盤から「ビジネス価値の創出基盤」へ成長

ビジネス価値の創出基盤

ビジネス価値を創出するアプリレイヤに対して、IaaSレイヤを意識せず利用することが理想。

IaaSのブラックボックス化 ≡ 自動化

「自動化」に期待されていること

拡張性

Scalability



×

柔軟性

Flexibility



×

信頼性

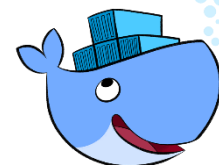
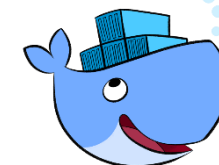
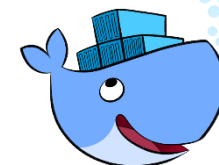
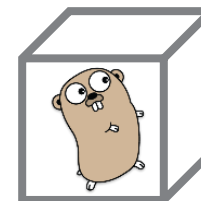
Reliability



Business



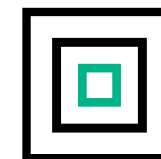
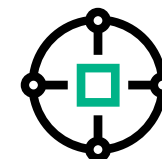
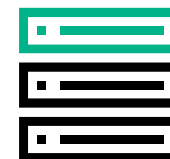
アプリレイヤ
(コンテナレイヤ)



IaaSレイヤ



openstack®



□ 自動化に必要なこと

☞ Infrastructure as Codeの運用自動化

□ CI/CDパイプラインの運用

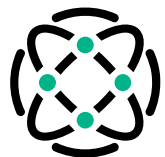
□ まとめ



openstack®

HPE
POINTNEXT

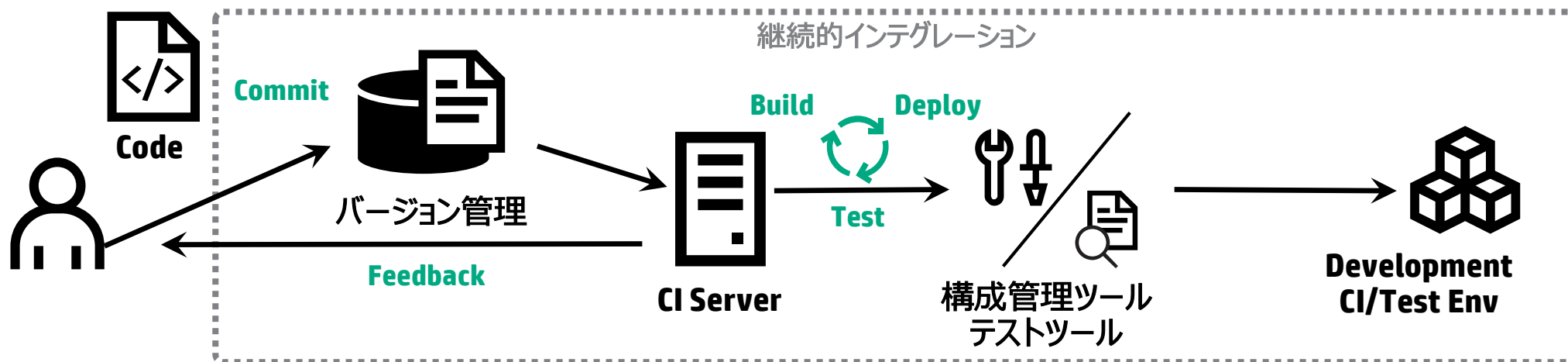
改めてInfrastructure as Codeとは



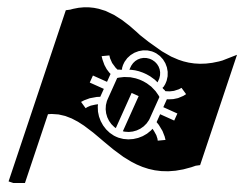
Infrastructure as Code(IaC)

手作業で行っていたインフラの構築や変更作業をコードで定義し、自動化すること。

ソフトウェア開発で実施されてきた開発プロセスをインフラシステム、アプリケーション、ミドルウェアのデプロイメントやコンフィグレーションの管理に適用。



自動化によって顕在化した弊害



オートメーション恐怖症

最終的には、自動化が恐怖であることを回避するプラクティスが必要。



サーバスプロール



構成ドリフト



スノーフレークサーバ



脆弱なインフラ

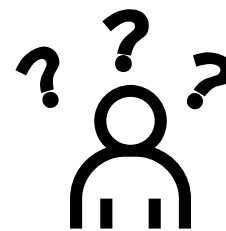


オートメーション恐怖症

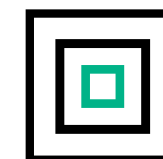
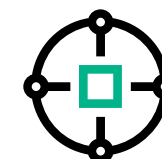


システム疲労

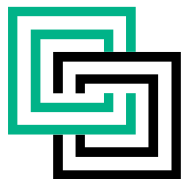
Code化すると、ブラックボックス化してしまうため、実行時の不安だけが残る。



openstack®

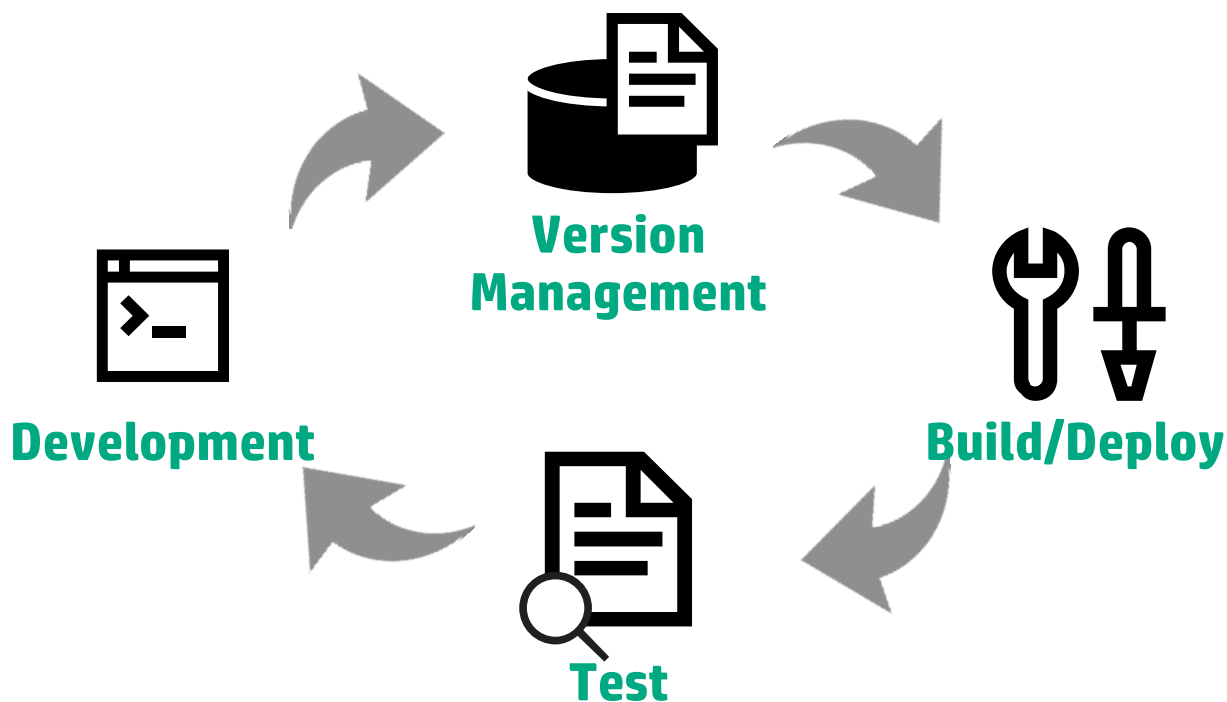


継続的インテグレーションとは



Continuous Integration(CI)

継続的インテグレーションとは、一日に何度もビルドを実行し、ソフトウェアをインテグレーションした時に、発生する様々な問題を早期に検出し、フィードバックサイクルを短くして、ソフトウェア開発の品質と生産性を向上させる仕組み



成果物(Artifact)による品質の保証

CIとは、ビルドやテストを実行して、安定に動く結果をどこかに保存する仕組み。

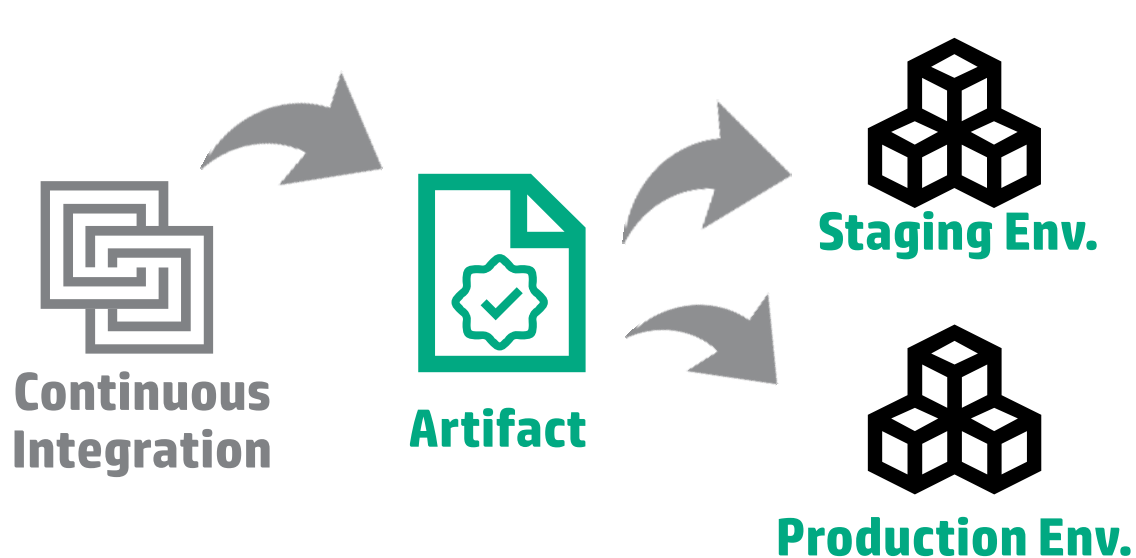
→ いわゆる、成功するAnsibleのコードや、VMテンプレートを構築すること。

継続的デリバリ(デプロイ)とは



Continuous Delivery(CD)

継続的デリバリとは、ソフトウェアのライフサイクル全体を通じて、常に本番環境にリリースできる状態に保ち、エンドユーザーへの提供を迅速に行う仕組み。



迅速なビジネス価値の提供

CIによって生成された成果物を、本番環境に自動的にデプロイする仕組み

→ いわゆる、迅速なデプロイによりユーザーからのフィードバックを回収し、ビジネス価値を提供すること。

Infrastructure as Codeの原則

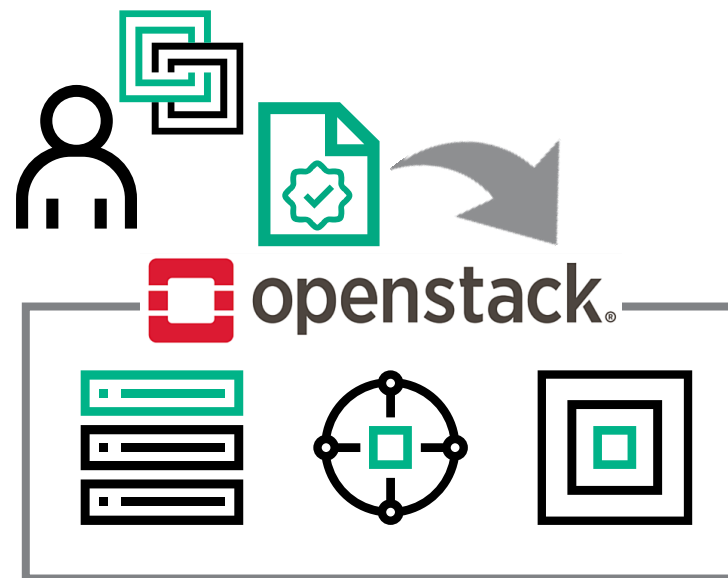


予防よりも「復旧」が重要

Infrastructure as Codeは変更管理が容易なインフラを作るためのアプローチ

- ✓ 簡単に再現できるシステム
- ✓ 使い捨てにできるシステム
- ✓ 標準化されたシステム
- ✓ 反復可能なシステム
- ✓ 変更しやすいデザイン

自動化の成果物に自信が持てること。
→不具合は早く見つけれられることが必要





Infrastructure as CodeにCI/CDなんて、 ほんとに必要??

Infrastructure as Codeは変更管理が容易なインフラを作るためのアプローチ

- ・VMのイメージとかHWってそんなに作り直す!?
- ・Immutable Infrastructureなんて都市伝説!?
- ・とりあえずAnsibleで構築できればよくない!?

「構築の自動化」と「運用の自動化」

Infrastructure as Codeで言うCI/CDとは



Dev 構築の自動化 Development Automation

構築の拡張性や、柔軟性を捉えた自動化

- ・標準化することでコストダウンに繋がる
- ・ビジネス要求の解決

Ops 運用の自動化 Operating Automation

インフラの信頼性や、柔軟性を捉えた自動化

- ・標準化することで品質の向上に繋がる
- ・サービスの維持継続

「構築の自動化」と「運用の自動化」

Infrastructure as Codeで言うCI/CDとは



Infrastructure as CodeのCI/CDは、主に「運用の自動化」を支援

構築の拡張性や、柔軟性を捉えた自動化
標準化することでコストダウンに繋がる

Ops 運用の自動化
Operating Automation

インフラの信頼性や、柔軟性を捉えた自動化
・標準化することで品質の向上に繋がる
・サービスの維持継続

運用の自動化とは

自動化の課題を改善するためには、CI/CDが必要

システム運用

System Operating

サービスを安定的に提供する為に必要となる日々の作業、または発生した事象に迅速に対応すること



運用の自動化

Operating Automation

“

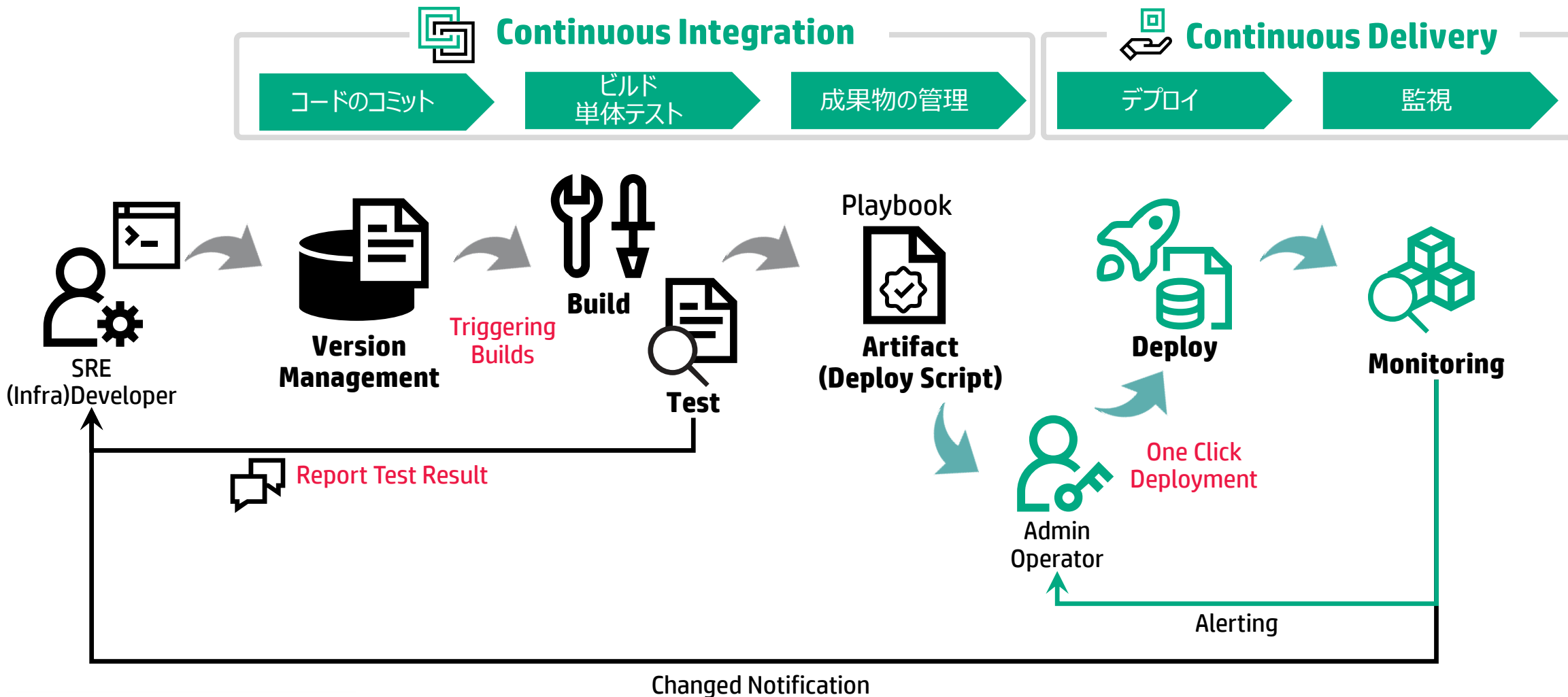
信頼のある成果物を日常的にデプロイし続けること
(CI/CD)

”

※SRE(Site Reliability Engineering)の皆様のお仕事

CI/CDのパイプライン

信頼のある成果物をデプロイすること



■ 自動化に必要なこと

■ Infrastructure as Codeの運用自動化

■ CI/CDパイプラインの運用

■ まとめ

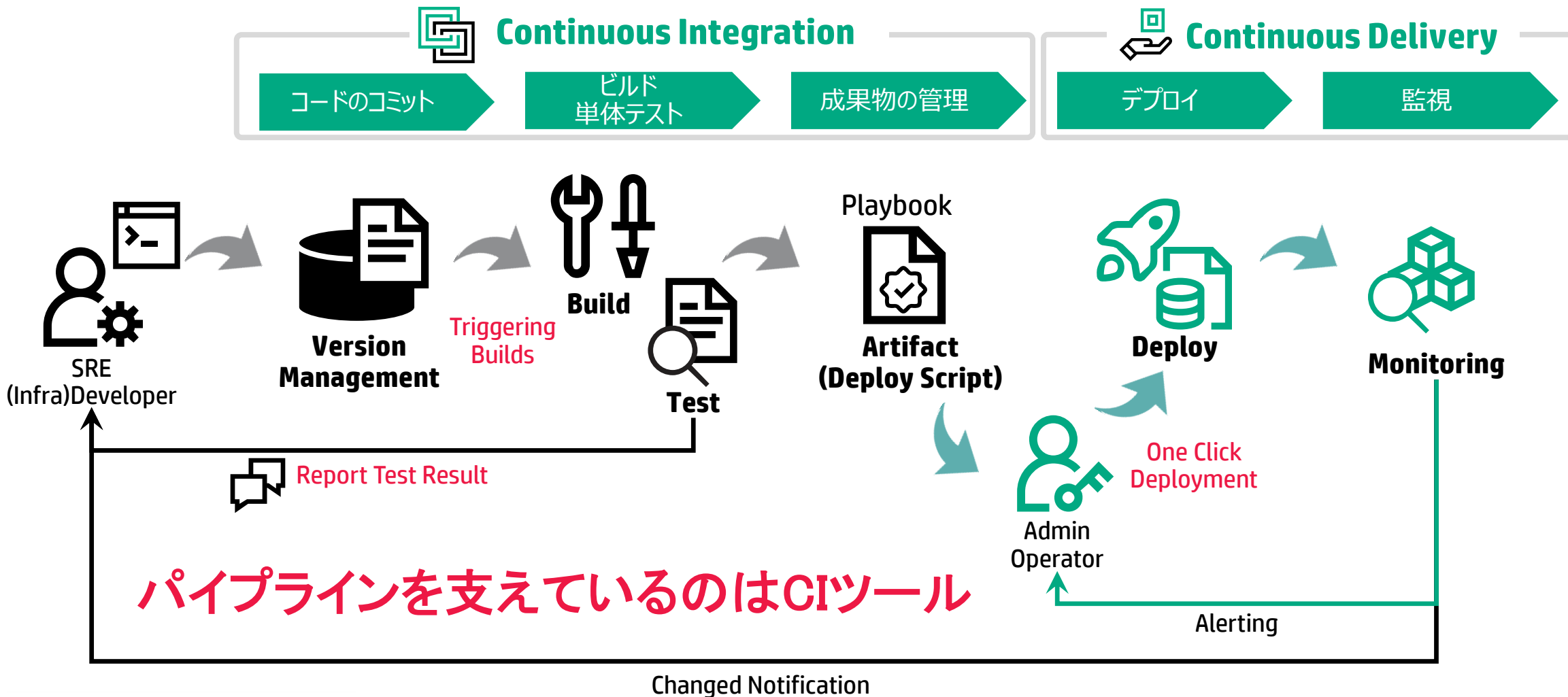


openstack®

HPE
POINTNEXT

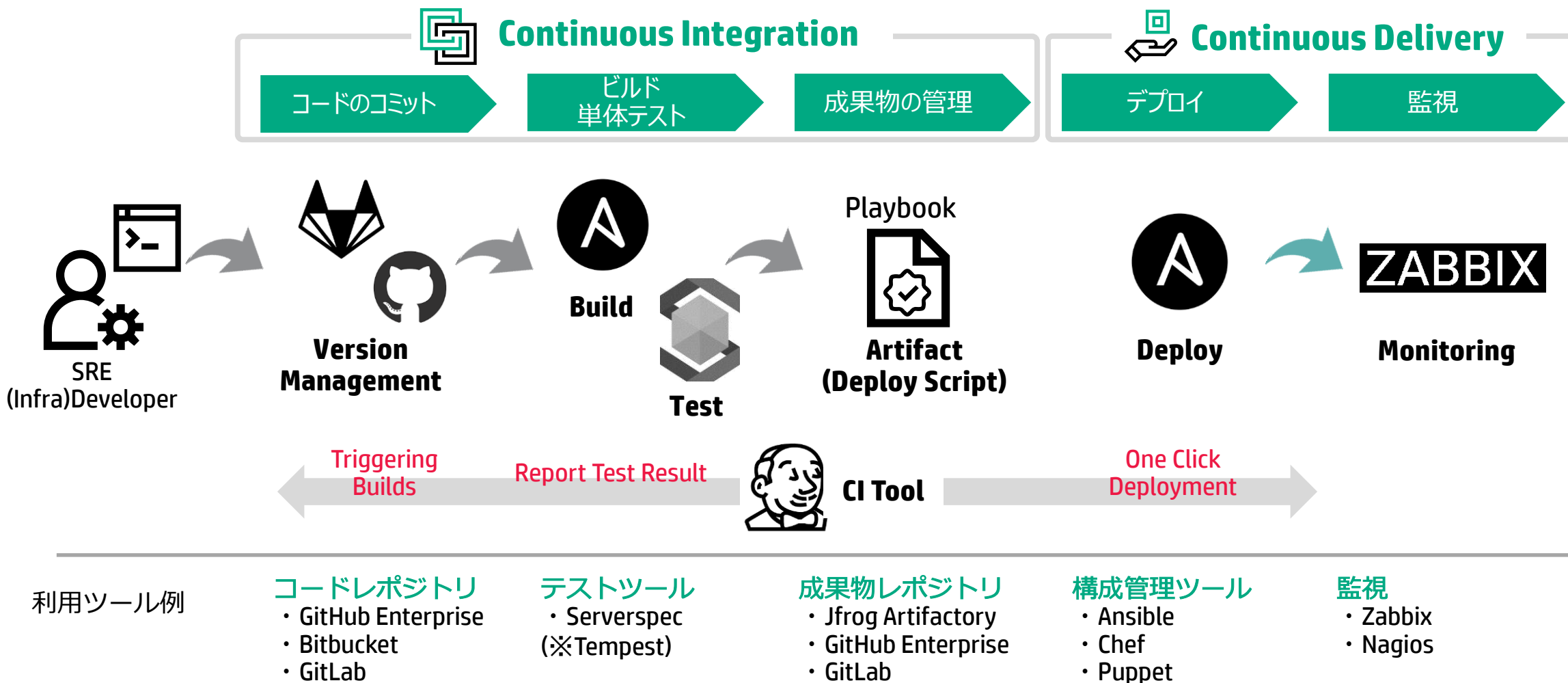
(再掲)CI/CDのパイプライン

信頼のある成果物をデプロイすること



CI/CDのパイプライン

インフラCI/CDのツール群



CIツールに求められること

CIツールの役割りは広範囲



※CIツールはJenkinsだけでなく、
世の中には数多くの素晴らしいツールが存在します。

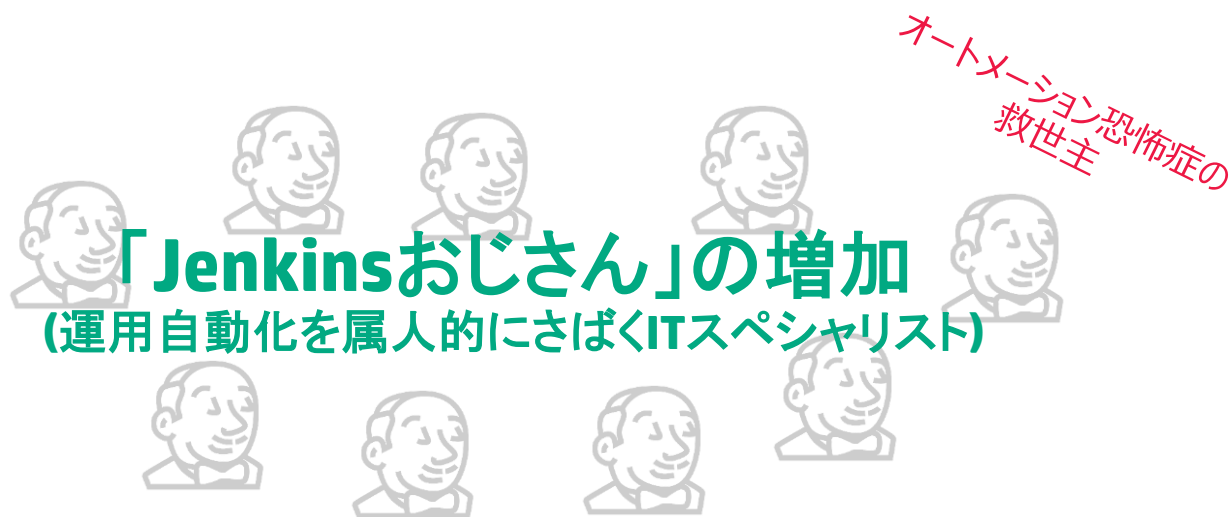


成果物(Artifact)による品質の保証

- ・パイプライン上の他のツールとの連携
- ・CIパイプラインの標準化
- ・CIツールの可用性、運用容易性
- ・効率的なJobの管理(共通リソースの管理)

CIツールの運用コスト

CIツールの管理は属人的になりがち

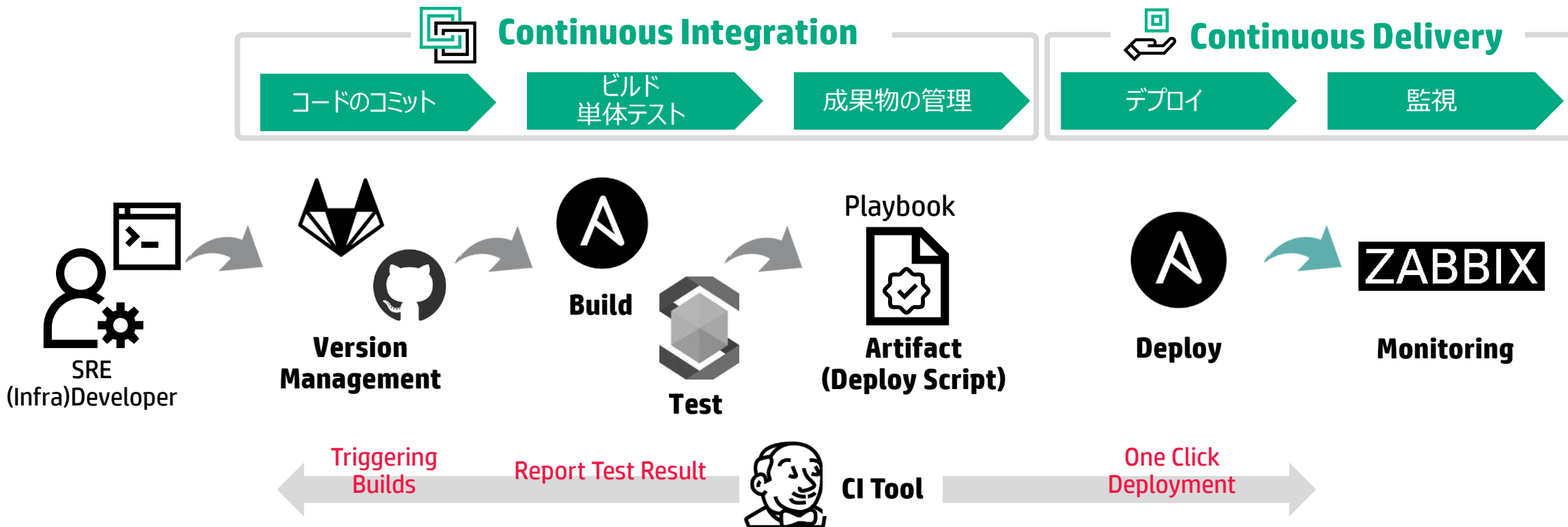


- ・複数のジョブの依存関係が複雑化(パイプライン制御)
- ・その場しのぎのビルドの設定
- ・ジョブ設定の再利用ができない
などなど

CIツールの運用を減らさないと、運用自動化にはならない

Pipeline as Codeに注目

部分の自動化から流れの自動化へ



 流れの自動化: ジョブをスクリプトで定義する

流れの自動化によるメリット



Pipeline as Code

CIツールで行っていたジョブの登録をコードで定義し、自動化すること。

ジョブの修正履歴を管理

GUIの設定に比べて、どこをいつ修正したのかの履歴をコードで示すことができる。

ジョブの標準化、再利用が可能

コード化することにより、ジョブ同士の依存関係を含めて再利用することが可能になる。

属人的な専門性をなくし、標準化、自動化を目指す
(※運用の運用を自動化)

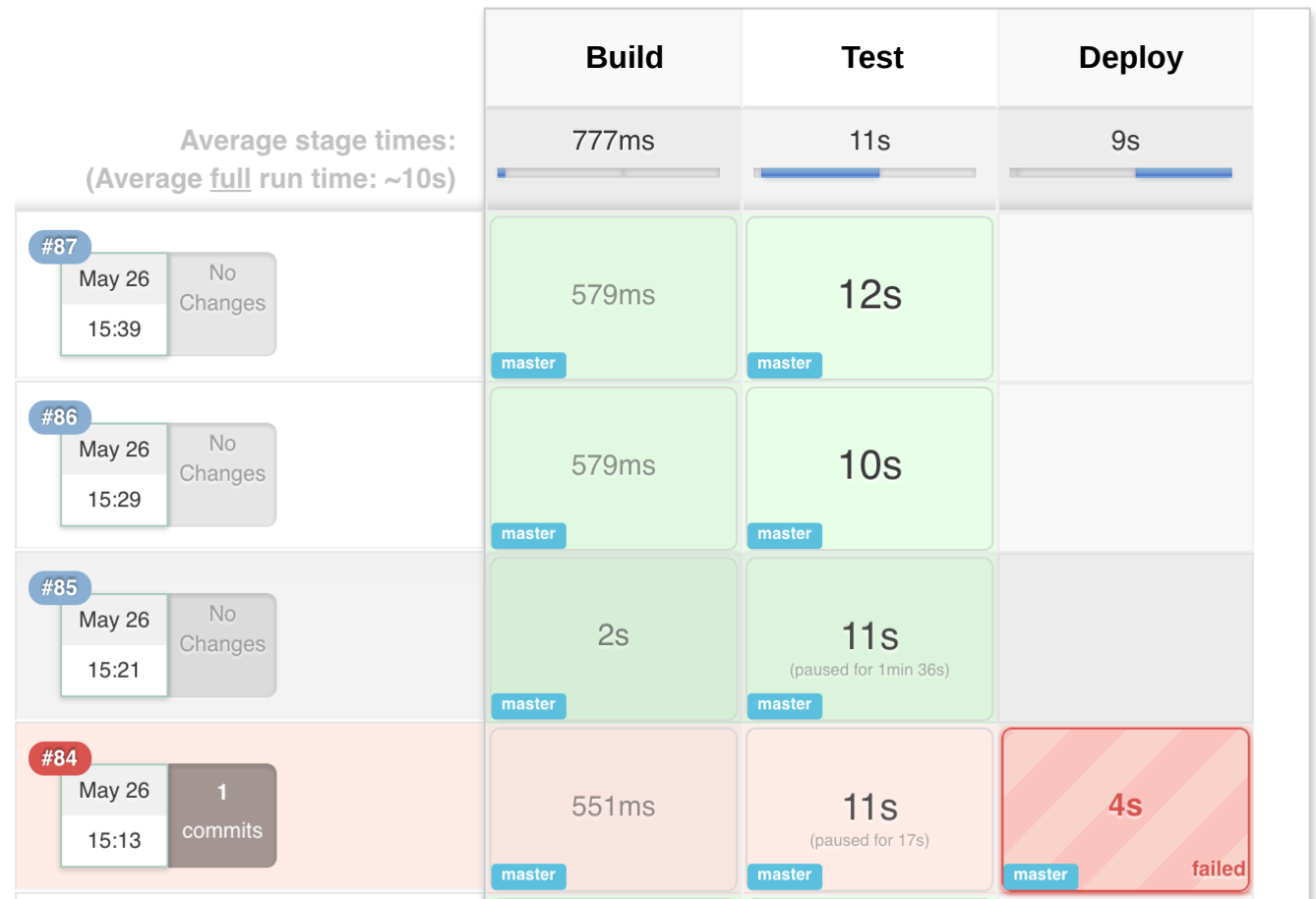
Pipeline as Code 例 (Jenkins)

Groovy DSL

```
pipeline {
  agent any

  stages {
    stage('Build') {
      steps {
        sh 'ansible-playbook setup.yml'
      }
    }
    stage('Test'){
      steps {
        sh 'rake spec:ci-built'
      }
    }
    stage('Deploy') {
      steps {
        input message: "Do you want to co
        sh 'ansible-playbook --tag deploy '
      }
    }
  }
}
```

Stage View



Pipeline as Code例(GitLab CI)

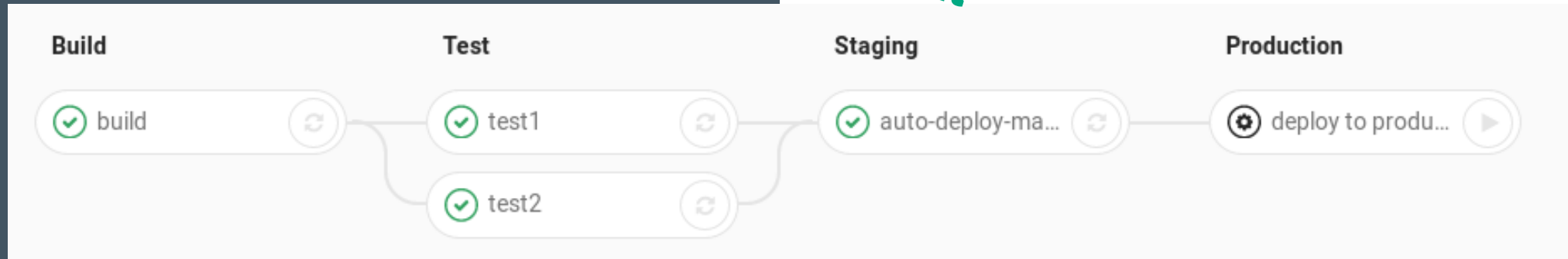
Yaml

```
...
stages:
  - Build
  - Test
  - Staging
  - Production
```

```
build:
  stage: Build
  script:
    - ansible-playbook ./setup.yml
```

```
test1:
  stage: Test
  script:
    - rake spec:ci-built
```

```
test2:
  stage: Test
  script:
    - ansible-playbook ./test.yml
```



All 2 Pending 0 Running 0 Finished 2 Branches Tags							Run Pipeline	CI Lint
Status	Pipeline	Commit	Stages					
passed	#7927123 by latest	pipeline-graph d4de4a5c Update .gitlab-ci.yml	✓✓✓→	00:00:55	1 hour ago			
passed	#7562143 by latest	master 4a2f619e Update .gitlab-ci.yml	✓✓✓→	00:00:57	2 weeks ago			

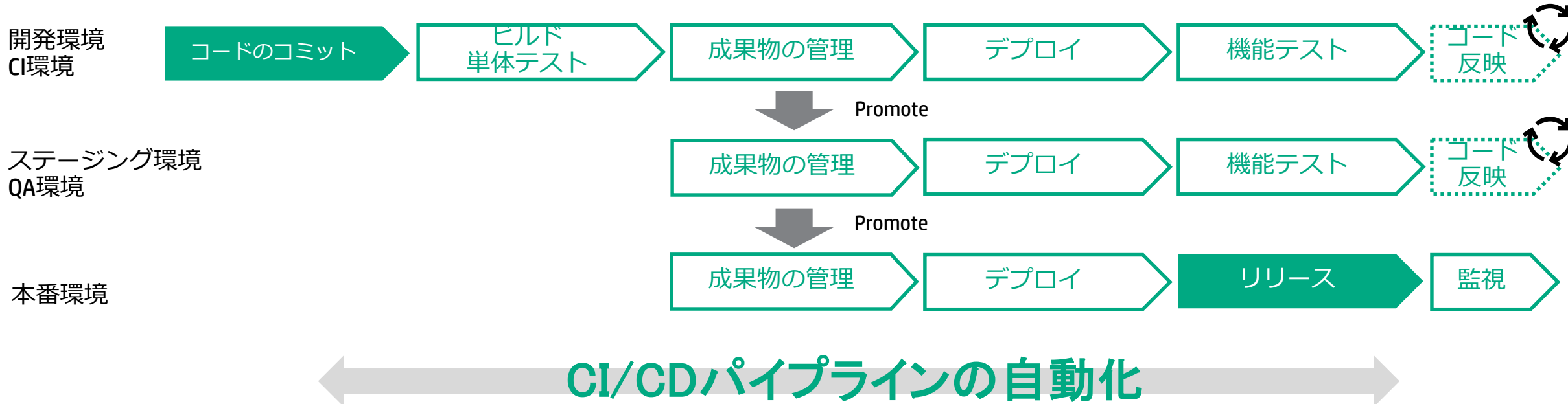
Pipeline as CodeによるCI/CDの自動化

より信頼のある自動化を目指すこと

※凡例

手動タスク

自動化タスク



“

信頼のある成果物を日常的にデプロイし続けること
(CI/CD)

”

■ 自動化に必要なこと

■ Infrastructure as Codeの運用自動化

■ CI/CDパイプラインの運用

☞ まとめ



openstack®

HPE
POINTNEXT

まとめ

自動化を支えるCI/CDパイプラインの世界



標準化を前提とした自動化を目指すことにより、
インフラを意識しないプラットフォームをつくりだすこと。



信頼のある成果物をいかに低コストで作るかがCIのメリット



部分の自動化だけでなく、流れの自動化を取り入れることによって、
より信頼性のある環境を作り出すことができる。



さいごに
自動化は1日にしてならず

正しい設計は、継続的に変化していくので属人化しない運用の自動化をっ!!

Appendix



openstack®

HPE
POINTNEXT

OpenStackのCI/CD

OpenStackの開発でもCI/CDが実装されている

16:55-17:35 / 5F Hall A-1

5-A1-4 コミュニティ ※同時通訳講演

zuul: a project gating system

<http://status.openstack.org/zuul/>



Monty Taylor

Chief Architect CI/CD, Red Hat Office of Technology

OpenStackは、世界でも最も大規模で最も複雑なオープンソースの一つであり、開発自体も同様に大規模で世界中に分散した人々によって行われています。そのため、OpenStackの開発では最初からCI/CDのソフトウェアとシステムが重要な役割を担っています。その中でも、

「gating」という、テストに合格した変更のみを自動的に組み込む仕組みが、必要不可欠なものとなっています。大規模な開発環境でそのようなことを行うためにOpenStackのインフラチームは「Zuul」という特別なソフトウェアを開発しました。Zuulは「Optimistic Branch Prediction

技術」を用いて並列にテストを実施し、テストを実施した状態がそのまま組み込まれることを保証します。近々リリースされるZuul v3では、ユーザが自身のプロジェクトや組織内で簡単にZuulを利用できるようになります。本講演では、Zuulとはなにか、どのような価値を提供するか、またどのように使うかについて説明します。

Promotion

Containerのデプロイについて知りたい!? Mesos User Group Tokyoに参加を!!

7月
25 Mesos Meetup Tokyo #2
カンカンいこうぜ! 第2回!

<https://mesos.connpass.com/event/58545/>



Mesos User Group Tokyo

Program against your datacenter
like it's a single pool of resources

Mesos with AWS SpotFleet (kotatsu360さん)

VASILYでクローラーシステムの下回りとして運用しているMesosクラスタは、AWS SpotFleet上でスケールさせています。その舞台裏について。

Mesos Frameworkの作り方 レベル2 (kuenishiさん)

実運用を見据えてMesos Frameworkを作るときに気をつけることなどをRetzの経験を踏まえて解説します。[「Mesos Frameworkの作り方」](#)の続編です。

Mesosでカナリアリリースだと? (tetsuyasdさん)

Mesos+Marathonでのアプリケーションデプロイに、カナリアリリースやオートスケールといった強力な機能を付加するVampを紹介します。

ハッシュタグ: #MUGT

募集内容	一般枠 無料
	LT発表者枠 無料
	ブログ記載枠 無料

Thank you!

Enjoy Automation, Build Simple Platform!!



本資料に関するお問い合わせ



Mailto: shingo.kitayama@hpe.com

本資料に関しては、お気軽にお問い合わせ下さい。
また、内容に関しては個人の意見に基づくものであり、十分考慮の上ですが、
所属組織団体の公式見解とは異なる場合がございます。何卒、ご了承下さい。



OpenStackは、米国およびその他の国において登録されたOpenStack Foundationの商標です。
その他、本資料で記載されているロゴ、システム名、製品名は各社及び商標権者の登録商標あるいは商標です。