



Red Hat
Ansible Automates
2021 Japan

2021年7月13日(火)



Exastro

Exastro & Ansibleで切り開く ～「攻めと守りの自動化」とは～

NEC サービス&プラットフォームSI事業部

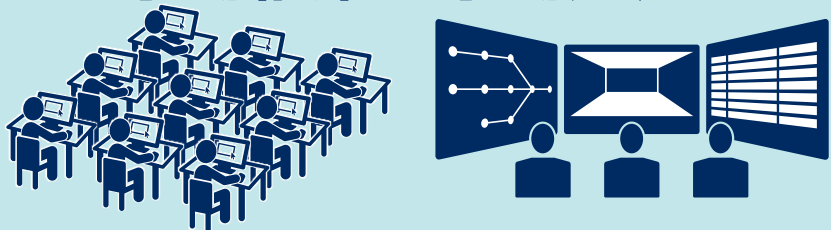
吉田 功一

新技術を取り込みにくい

密結合で保守しづらいアプリ

従来のスタティックな環境

手動構築・手動運用



モノリシック
(物理機器)

仮想化技術がまだ普及していなかった頃…

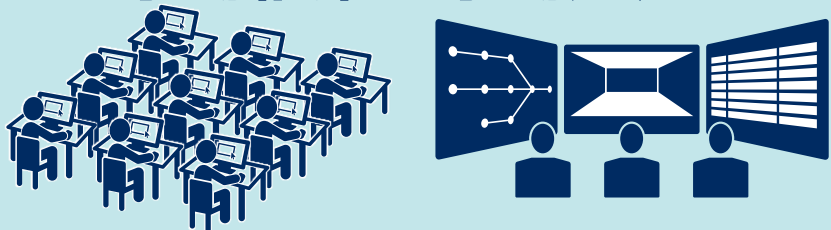
- ✓ サービス停止させないために
「**最繁トラヒック×バッファ×冗長2倍**」
のリソースを物理的に並べていた
- ✓ また「CPU、Mem、I/O」の選択肢が少
なく、**リソースを使いきれなかった**
- ✓ それでもなるべく少ない機器点数で多くの
サービスを捌けるように、**複数機能をうまく
混載(コンソリ)することが“正しい”**とされ
ていた
- ✓ **アプリケーションもこれらを前提**に作られ
ていた(それが正しかった)

新技術を取り込みにくい

密結合で保守しづらいアプリ

従来のスタティックな環境

手動構築・手動運用



モノリシック
(クラウドリフト)

仮想化技術という武器を手に入れた結果

- ✓ 「CPU、Mem、I/O」が**選択可能**に
- ✓ 「オートスケールWeb」くらいの**簡単な動的システム**は組めるようになった
- ✓ 結果、システムサイズはコンパクトになり**ハードウェア寿命から脱却**できた
- ✓ **複数機能を混載(コンソリ)**させないことが**“正しい”**と変わり始めた

しかし、以下の課題は解決されず...

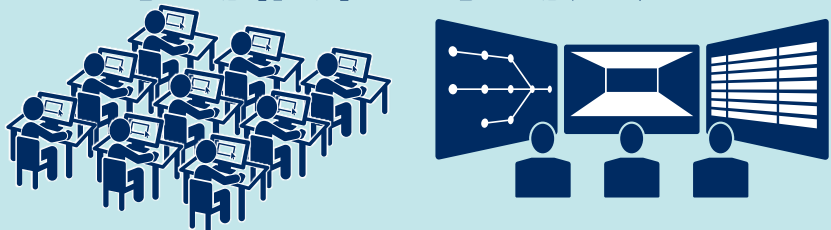
- ✓ アプリケーションを作り直さない限り**多くの機能で冗長2倍は継続**
→ クラウドシフトへの期待 ... ①
- ✓ 根本的にはシステム形状は変わらないので**運用の手間は大きくは変わらない**
→ 運用自動化への期待 ... ②

新技术を取り込みにくい

密結合で保守しづらいアプリ

従来のスタティックな環境

手動構築・手動運用



モノリシック
(クラウドリフト)

クラウドシフトへの期待(①)

OPEXの効率化

アプリ設計の段階から自動化を組み込むことで、必要な時に必要なリソースを使う「**本格的な動的システム**」
(リソースの冗長確保からの脱却)

ROI(投資利益率)の引き上げ

アプリ設計の段階から自動化を組み込むことで、運用しながら機能追加できる「**廃棄容易なシステム**」(DevOpsの実現)

クラウドシフトの問題点

- ✓ アプリの作り直しを伴う
- ✓ 初期構築時にカラクリを仕込むのが大変

⇒とにかく敷居が高い！

頻繁に改造したい部位を選択しなければ…

Introduction 「攻めの自動化」 「守りの自動化」 とは？



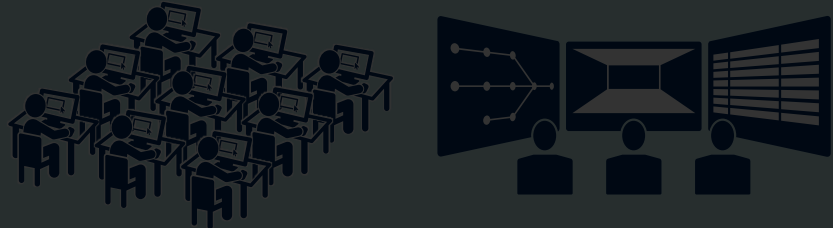
Introduction 「攻めの自動化」 「守りの自動化」 とは？

新技术を取り込みにくい

密結合で保守しづらいアプリ

従来のスタティックな環境

手動構築・手動運用



モノリシック

(物理機器クラウドリフト)

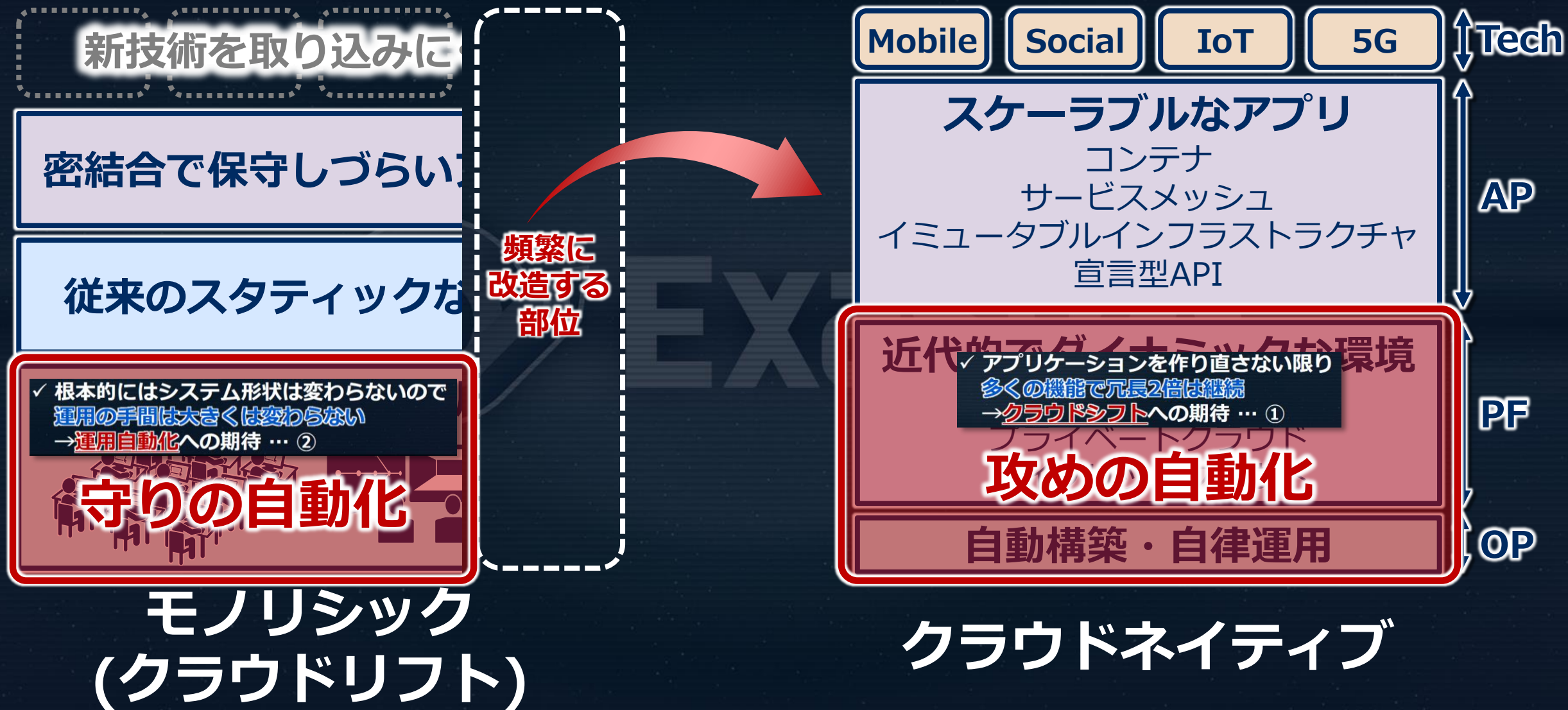
仮想化技術という武器を手に入れた結果

- ✓ 「CPU、Mem、I/O」が**選択可能**に
- ✓ 「オートスケールWeb」くらいの**簡単な動的システム**は組めるようになった
- ✓ 結果、システムサイズはコンパクトになり**ハードウェア寿命から脱却**できた
- ✓ **複数機能を混載(コンソリ)**させないことが**“正しい”**と変わり始めた

しかし、以下の課題は解決されず...

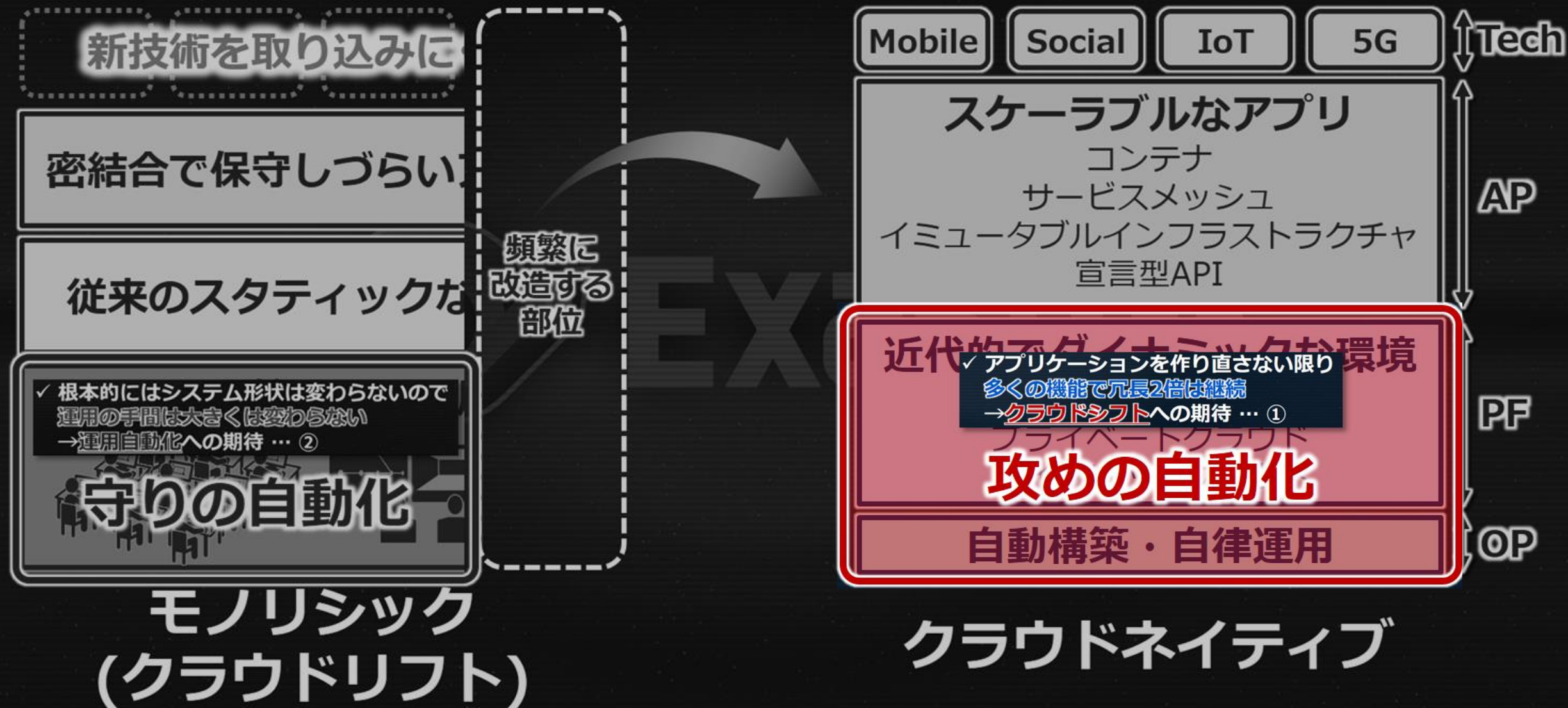
- ✓ アプリケーションを作り直さない限り**多くの機能で冗長2倍は継続**
→クラウドシフトへの期待 ... ①
- ✓ 根本的にはシステム形状は変わらないので**運用の手間は大きくは変わらない**
→運用自動化への期待 ... ②

Introduction 「攻めの自動化」 「守りの自動化」とは？



「攻めの自動化」の進め方

「攻めの自動化」の進め方



ところで...

クラウドネイティブの**カラクリ**とは？
初期構築時にどんな**カラクリ**を仕込んでおけば、
気持ちよくDevOpsをまわせる？

クラウドシフトで期待できること

11のクラウドリ (10の技 + 仮想化)

クラウドネイティブの実現手段

最終的な目的

ITリソースを活用し
利益を向上する

ROI(投資利益率)の引き上げ

投資の質を上げて
売上につなげる

単位時間あたりの
提供機能数 = 価値の向上

高速・スケーラブルな
機能提供 (リリース)

OPEXの効率化

OPEX(運用費)を
下げて投資にまわす

運用に必要な
人件費の削減

運用に必要な
保守費の削減

CI(継続的インテグレーション)

CD(継続的デリバリー)

マイクロサービスアーキテクチャ

サービスメッシュ

ブルーグリーンデプロイメント
(イミュータブルインフラストラクチャ)

宣言的API

オートスケーリング

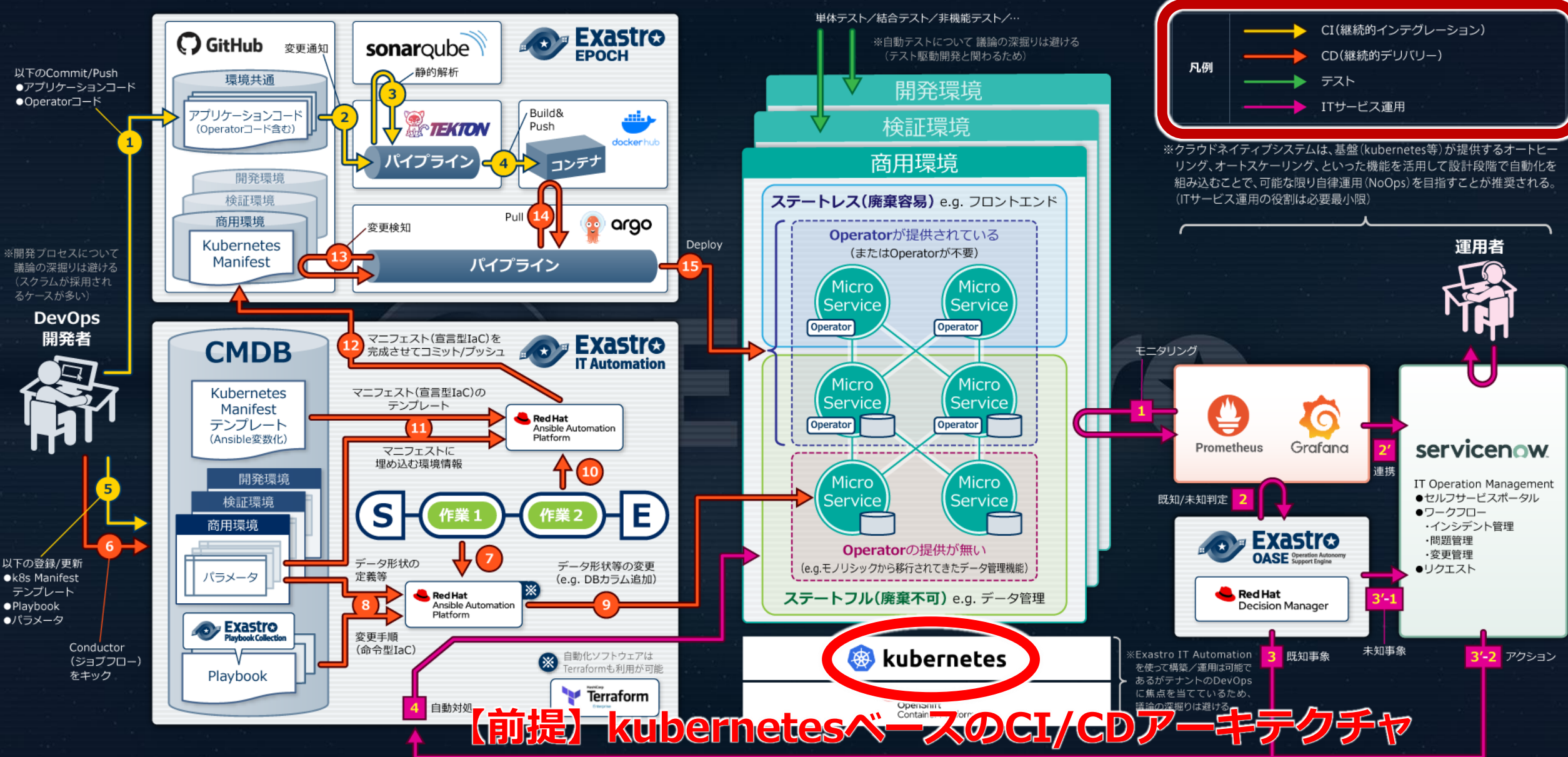
分散トレーシング

メトリクス監視

メンテナンスタスクの一元管理・自動化

仮想化(ハードウェア寿命からの解放)

「攻めの自動化」の進め方



「攻めの自動化」の進め方

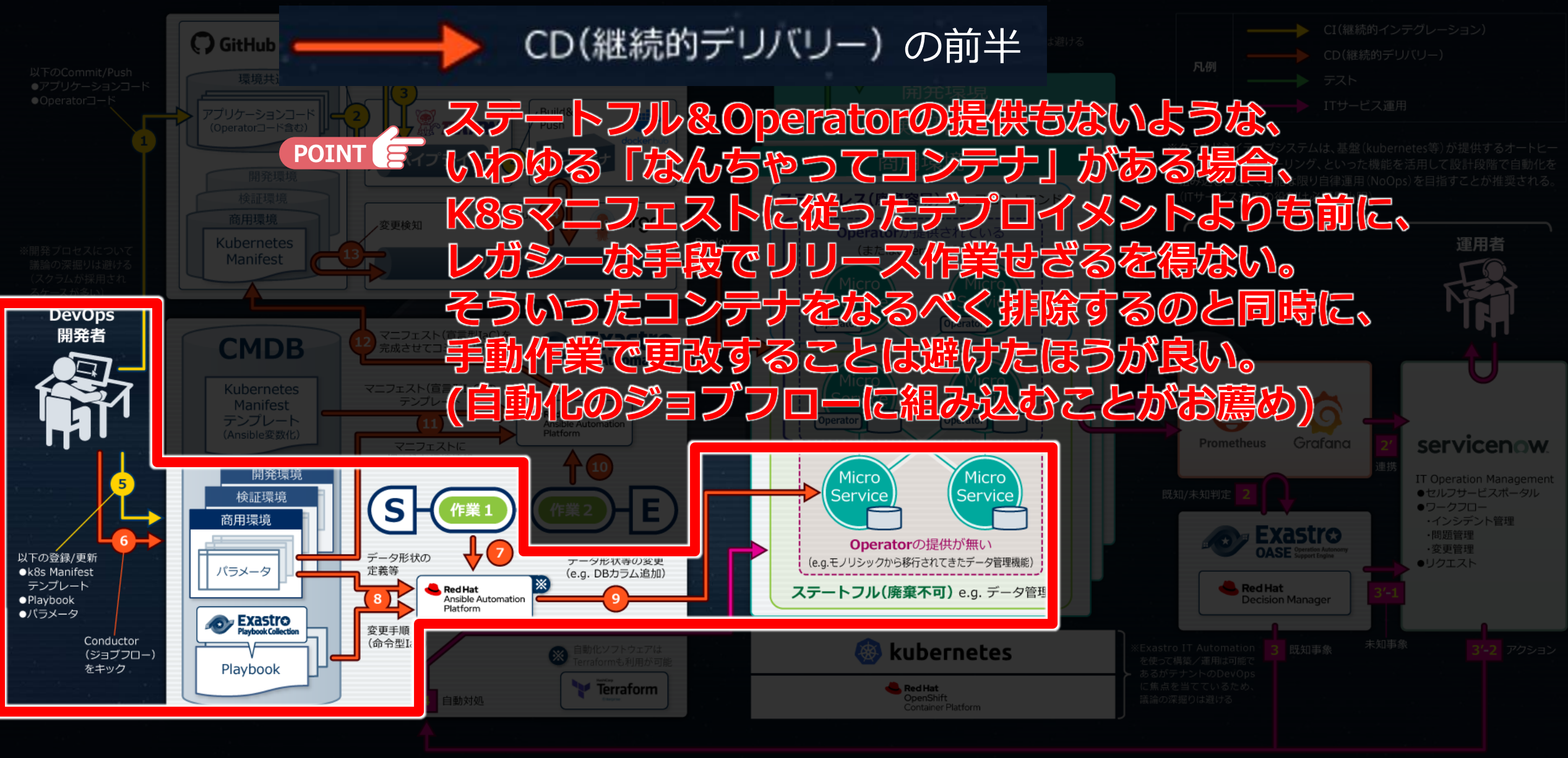


「攻めの自動化」の進め方

CD(継続的デリバリー)の前半

POINT

ステートフル&Operatorの提供もないような、いわゆる「なんちゃってコンテナ」がある場合、K8sマニフェストに従ったデプロイメントよりも前に、レガシーな手段でリリース作業せざるを得ない。そういったコンテナをなるべく排除するのと同時に、手動作業で更改することは避けたほうが良い。
(自動化のジョブフローに組み込むことがお薦め)



以下のCommit/Push
●アプリケーションコード
●Operatorコード

※開発プロセスについて議論の深掘りは避ける(スクラムが採用されれば)

DevOps 開発者

以下の登録/更新
●k8s Manifest テンプレート
●Playbook
●パラメータ

Conductor (ジョブフロー) をキック

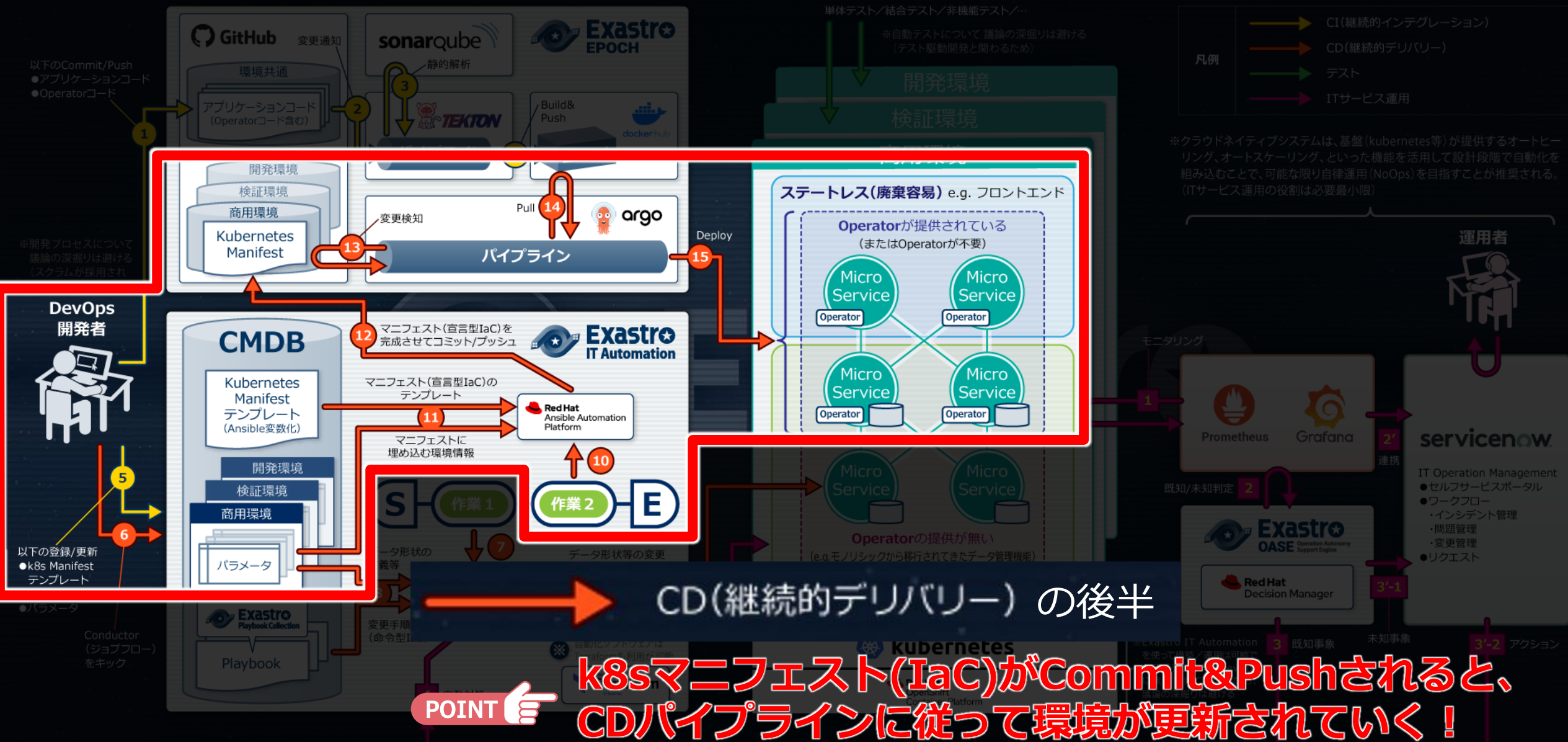
凡例
→ CI(継続的インテグレーション)
→ CD(継続的デリバリー)
→ テスト
→ ITサービス運用



servicenow
IT Operation Management
●セルフサービスポータル
●ワークフロー
●インシデント管理
●問題管理
●変更管理
●リクエスト

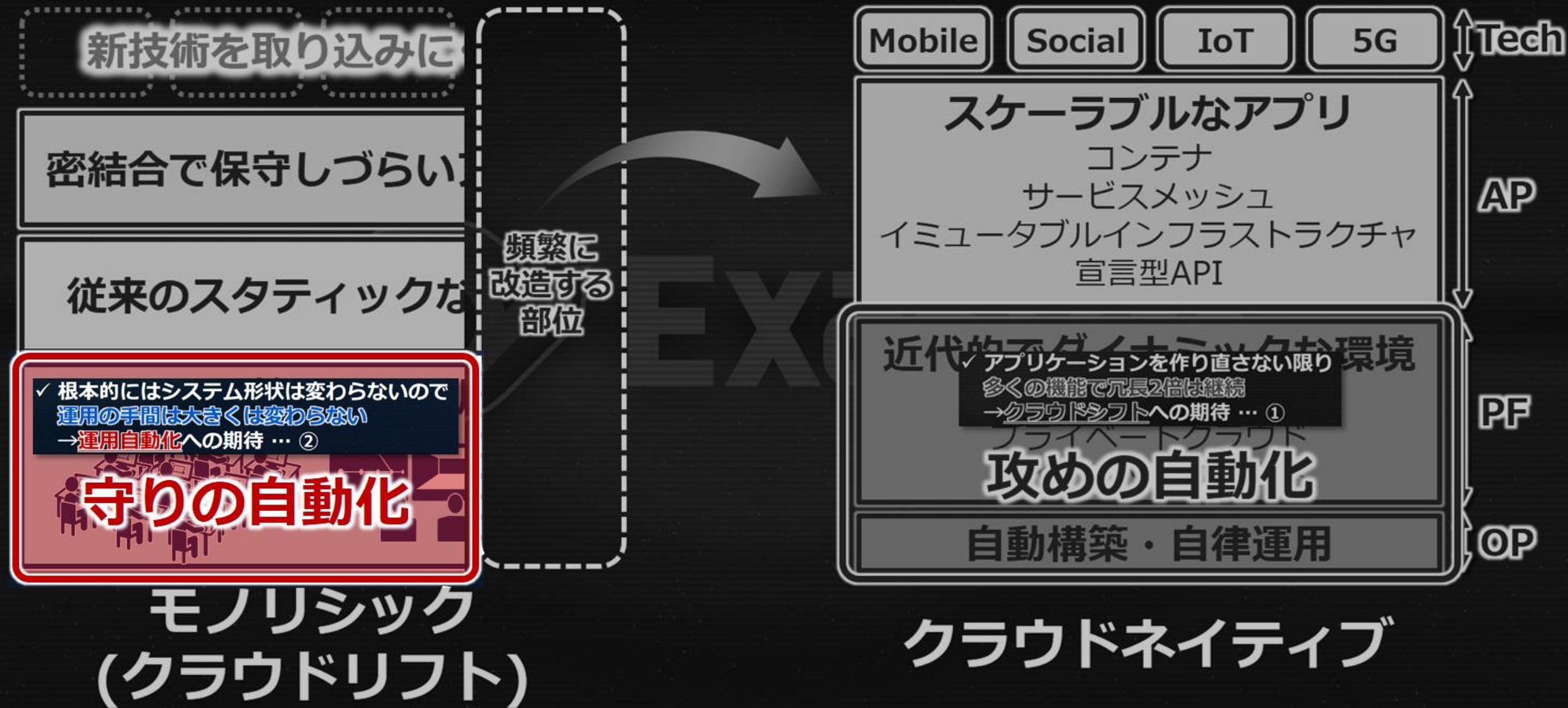
※Exastro IT Automation を使って構築/運用は可能であるがテナントのDevOpsに焦点を当てているため、議論の深掘りは避ける

「攻めの自動化」の進め方

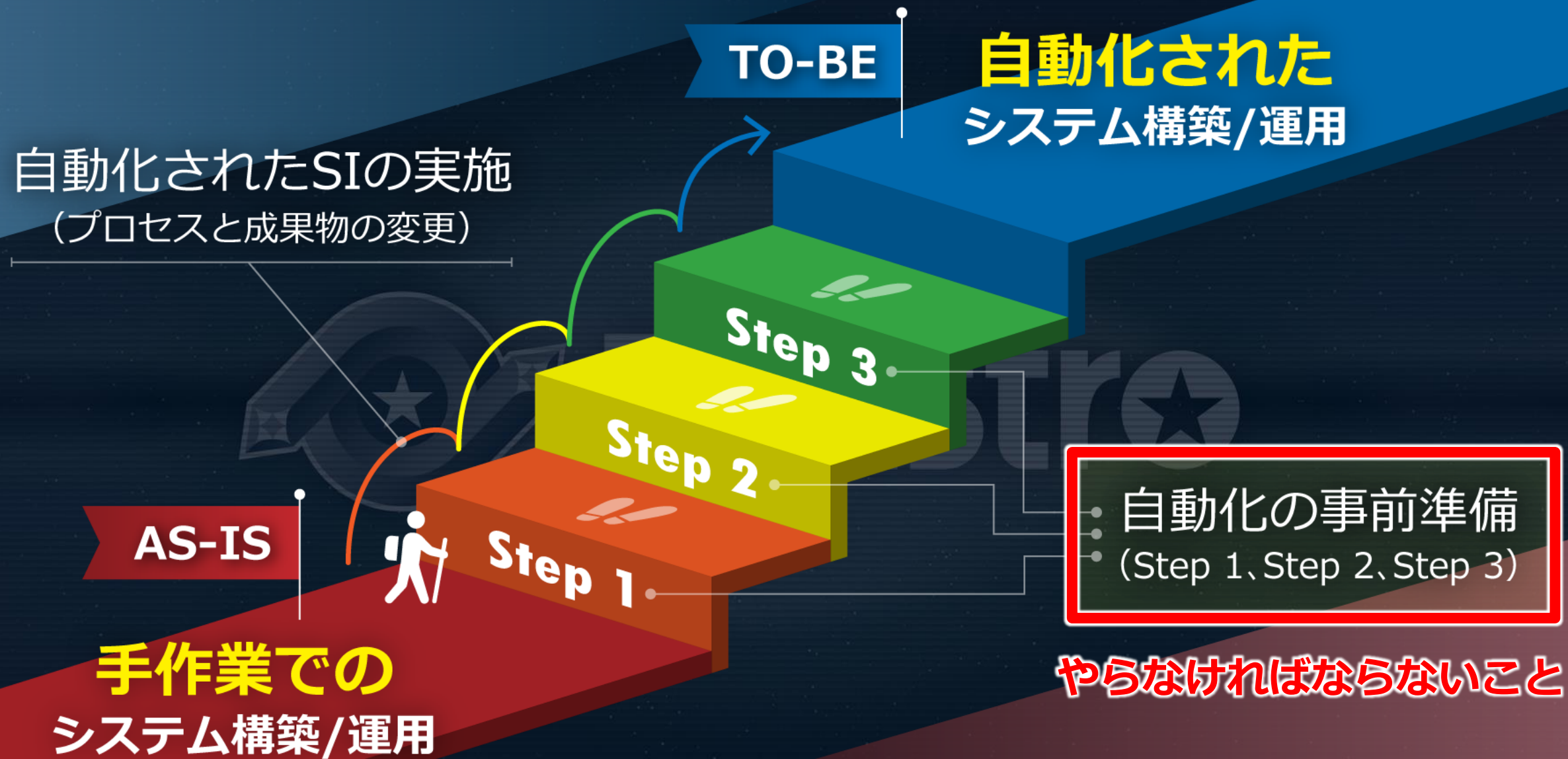


「守りの自動化」の進め方

「守りの自動化」の進め方



「守りの自動化」の進め方



モノリシックシステムの

構築・運用に携わるITエンジニアの現場の声

設計

- ✓ チーム間の情報伝達に遅延やミスが発生する
- ✓ データの二重管理や独自文言が設計ミスにつながる
- ✓ 多重開発により設計書(帳票)の管理が煩雑化する
- ✓ 結果として設定の前後性を確認できない

作業準備

- ✓ チーム間の作業順序が複雑で毎回タイムチャートを作成しては使い捨てる
- ✓ 作業ごとに手順書を作成/レビューしては使い捨てる
- ✓ 手順ごとにコンフィグを埋め込んでいて、新機種/新OSを追加するごとに手順書のパターンが増える(マルチベンダー対応の障壁)

作業実施

- ✓ 人手作業なので作業時間が一定でない
⇒チーム間で作業待ちが発生
- ✓ 人手作業なので人為ミスの懸念から逃れられない

これらの課題は大まかには3つのステップで解決できます



設計



作業準備

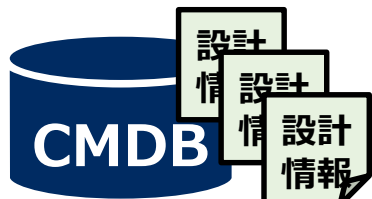


作業実施

解決策

解決策

Step 1
設計情報の一元管理

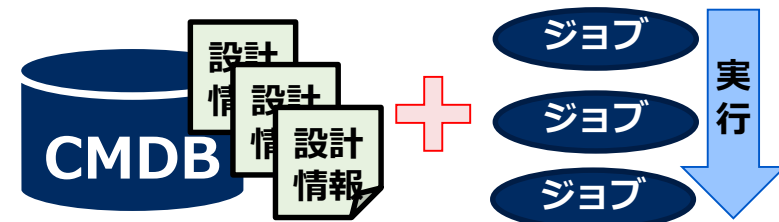


Step 2
自動実行の実現



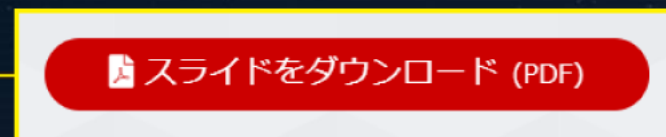
連携

Step 3
一元管理と自動実行の連携

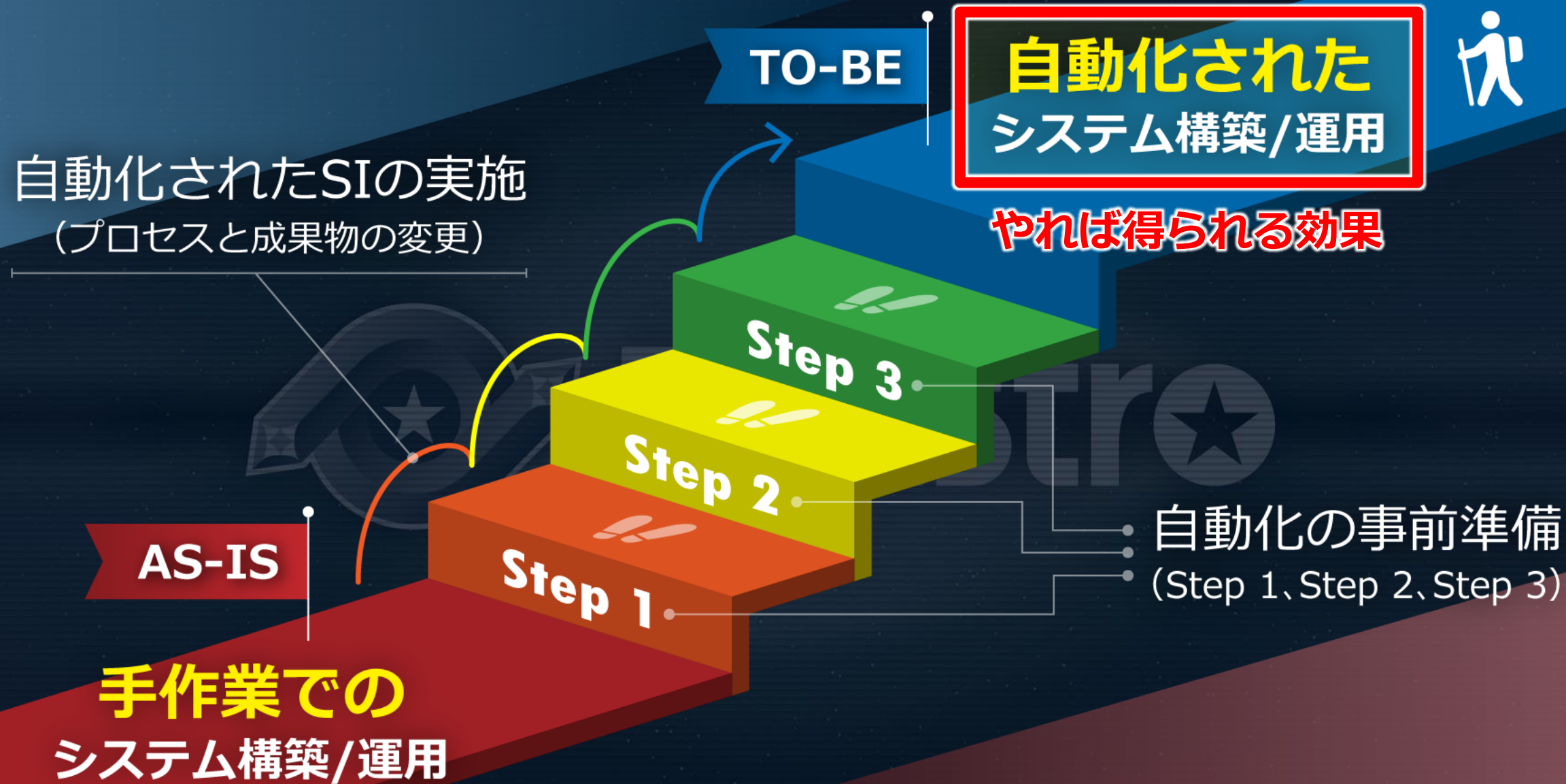


ガイドはOSSコミュニティサイトからダウンロードできます

Exastro IT Automation コミュニティサイト https://exastro-suite.github.io/it-automation-docs/index_ja.html



「守りの自動化」の進め方



活用事例



【事例①】NTTドコモ様 ISPサービスを含む10,000機器以上を自動化

課題



システムのパラメータが時系列で構成管理できておらず、またPlaybookの部品化ができておらず、構築および運用における作業の自動化/効率化が実現できていなかった。

解決策

Exastro IT Automationの機能で

- システムのパラメータを構成管理
- Ansible Playbookを部品管理

することで設計～構築、および運用のフェーズを繋げて、作業の自動化を実現した。

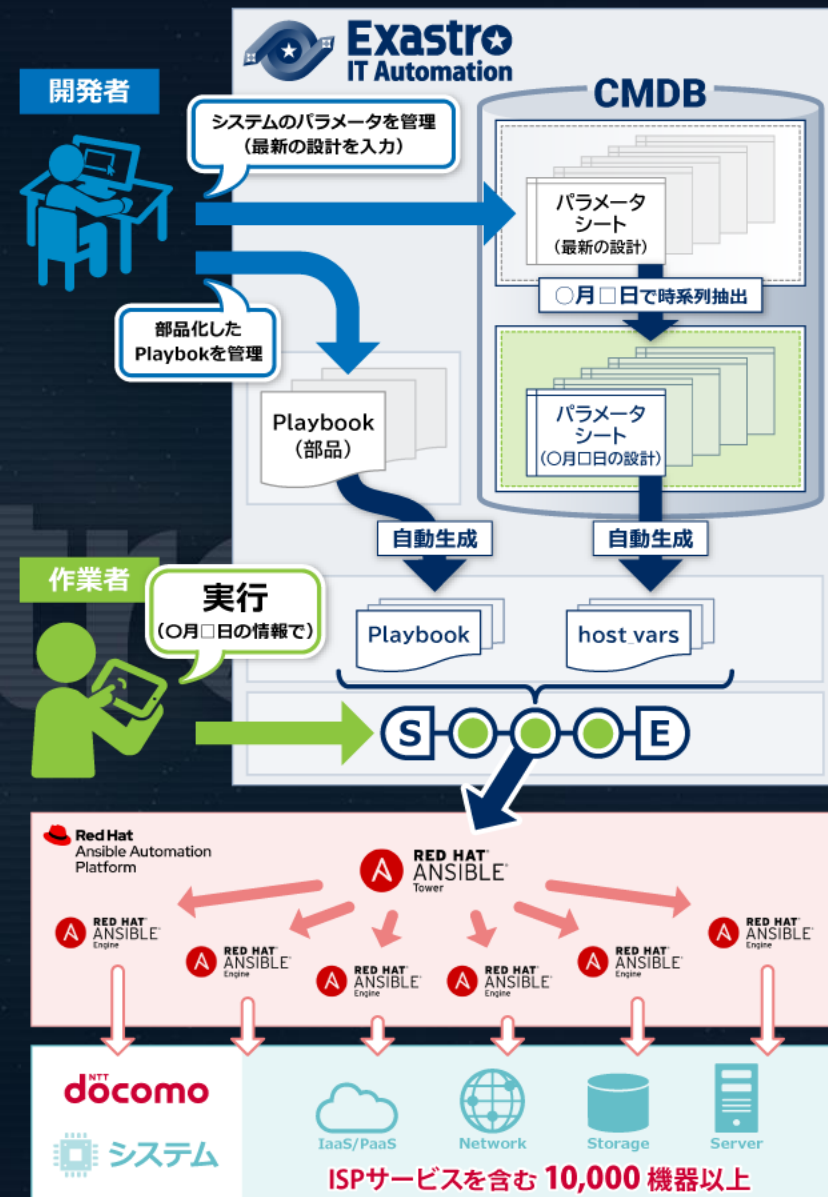


効果



例:運用監視設定作業

		設計工数	作業工数
構成管理 自動化前	年間239の作業工数で実施していた 239/年	86	153
構成管理 自動化後	100/年 ← 58%削減 年間100の作業工数で実施できるようになった	86	14



【事例②】 KDDI様 構成管理を活用した大規模キャリア統合運用部門の作業自動化

課題

手順改版のタイムラグ

S-in後にシステムの構成変更に伴う手順引継ぎに数日以上を要している。

使用言語の乱立による保守性の悪化

プログラミング基準がなく、使用言語も多様で資材も散在しているため、保守性が低く、管理工数が高止まっている。

作業場所に制約がある

ソース、仕様書が特定のサーバやセキュリティルームの専用端末に散在し、作業場所に制限が発生している。

解決策

システム構成と手順の一元管理

システム構成と手順をCMDBに一元管理することで、構成変更による手順の改版を極小化しリスクを低減。

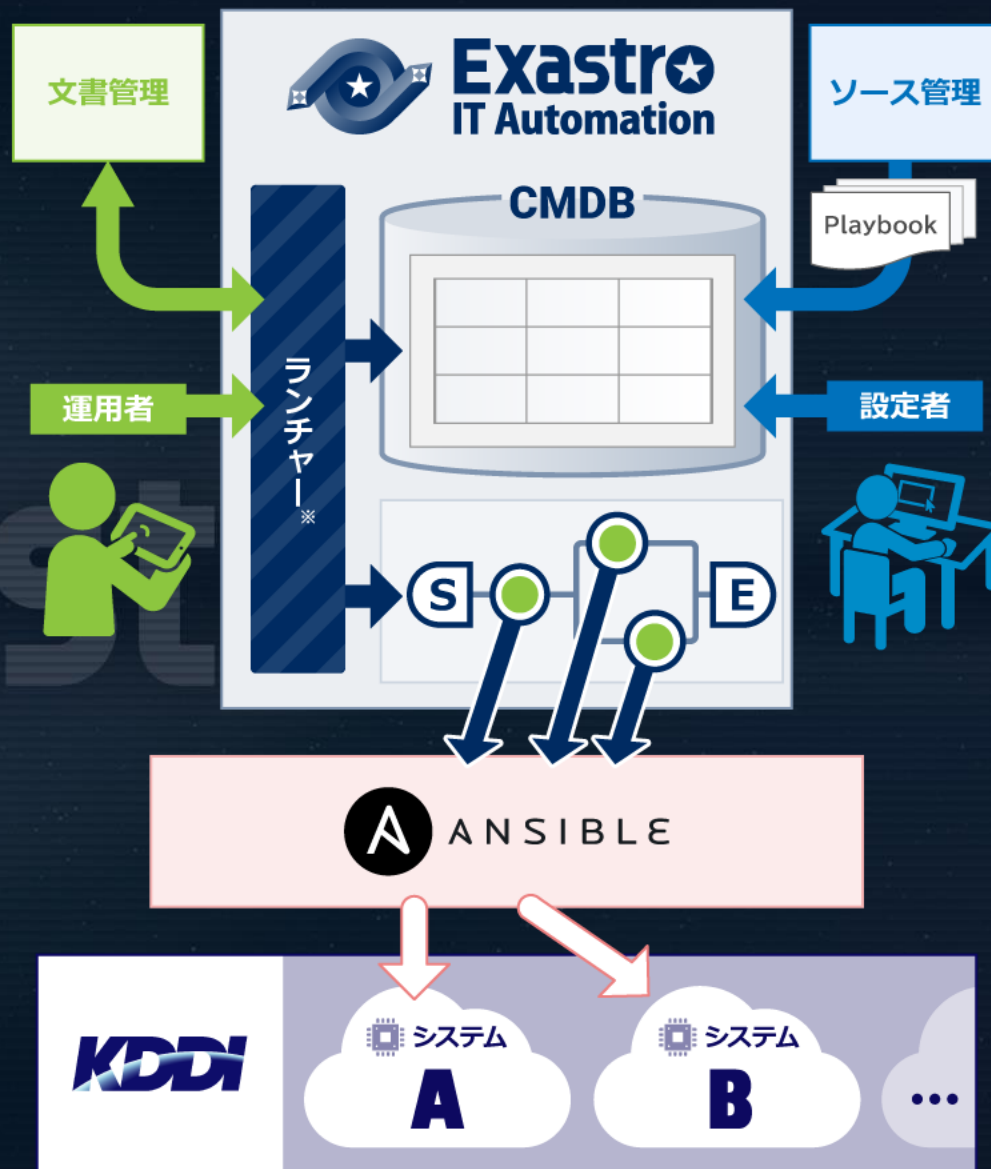
使用言語の統一

手順書とプログラム・ソースを一元管理することで保守性向上と管理工数を低減。

作業時のセキュリティと機動性の両立

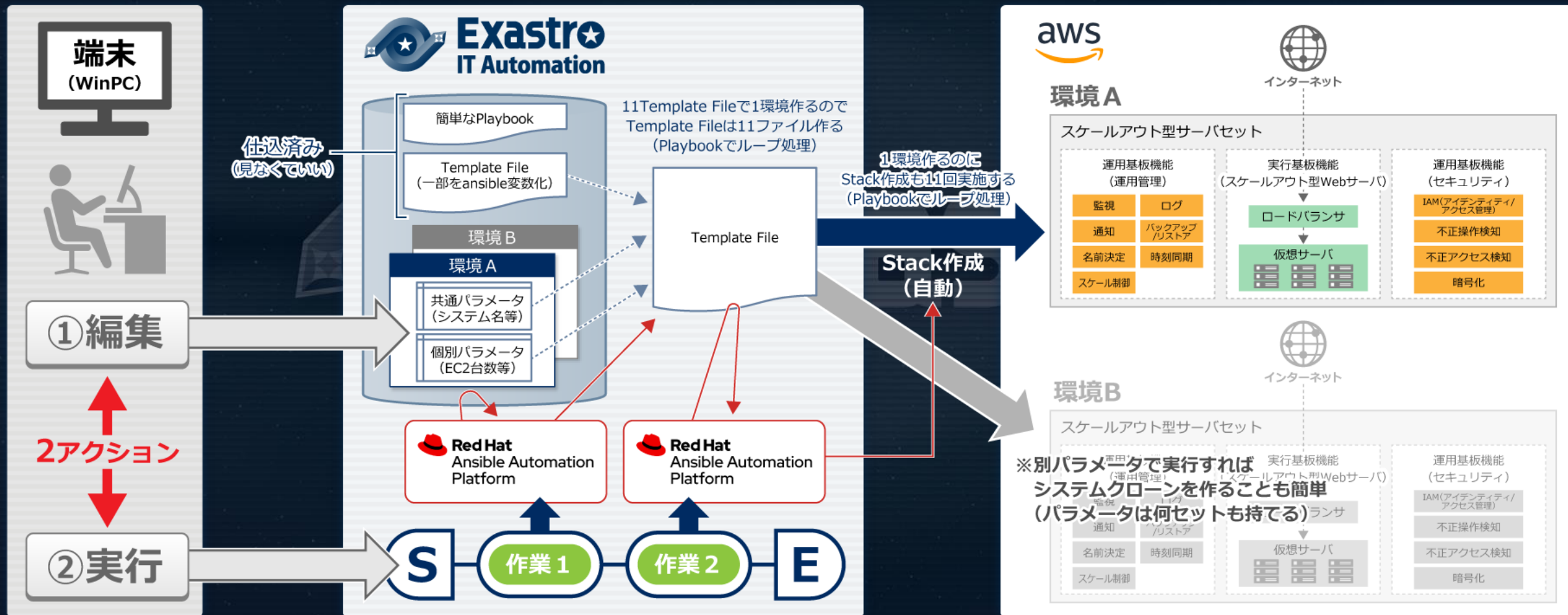
運用向け作業実行指示画面(ランチャー)*を開発し、作業場所に捕らわれずセキュリティと機動性を両立。

*ランチャー画面はITAが提供する画面フレームワークを使ってカスタム開発。



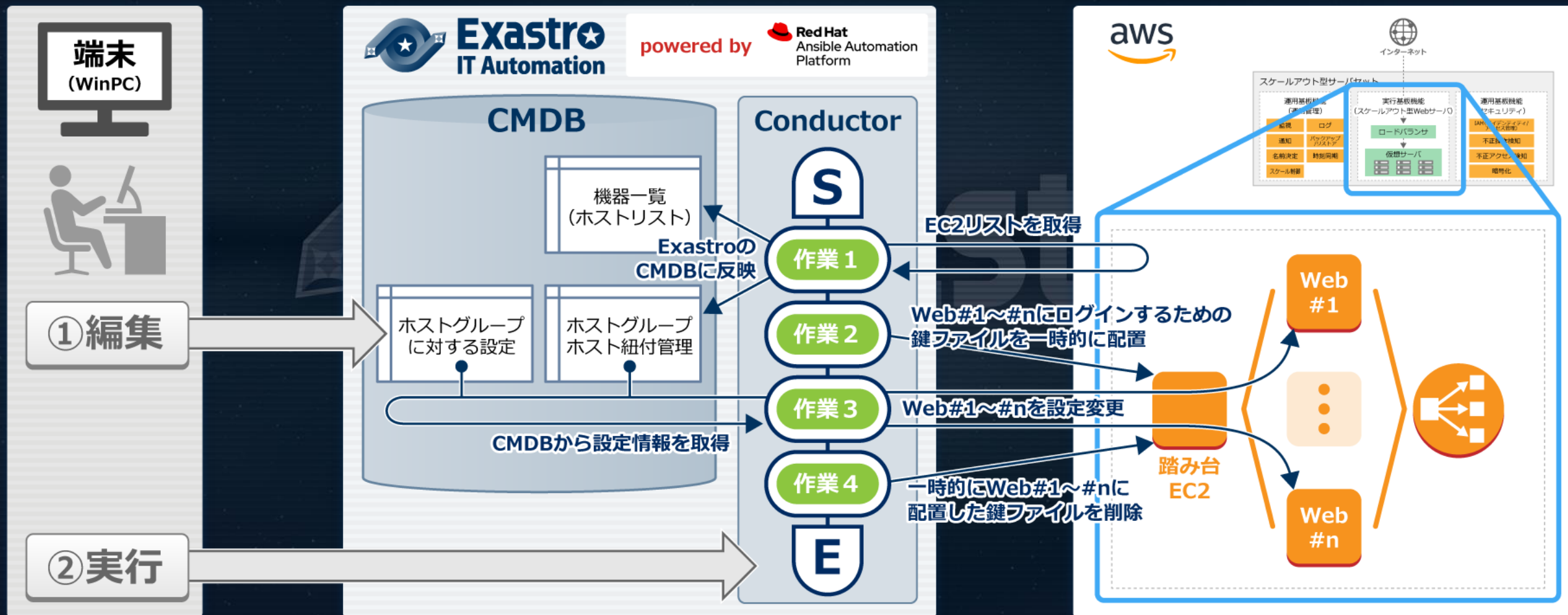
【事例③】 企業の運用部門が各部門のAWSシステムを統制する仕組み (1/2)

Exastro IT Automationで「CloudFormationテンプレート」を管理し、各部門にガバナンスの効いたAWS環境を払い出す仕組みを提供しました。



【事例③】 企業の運用部門が各部門のAWSシステムを統制する仕組み (2/2)

稼働中のEC2(オートスケール)に緊急でパッチ適用する、
といった運用シナリオにも対応しました



まとめ

Summary 「攻めの自動化」 「守りの自動化」とは？





Red Hat Decision Manager



Red Hat Ansible Automation Platform



Red Hat OpenShift Container Platform



✓ 根本的にはシステム形状は変わらないので
運用の手間は大きくは変わらない
→運用自動化への期待 ... ②

守りの自動化

モノリシック
(クラウドリフト)

✓ アプリケーションを作り直さない限り
多くの機能で冗長2倍は継続
→クラウドシフトへの期待 ... ①

攻めの自動化

自動構築・自律運用

クラウドネイティブ



Exastroの自動化
OASE Operation Autonomy Support Engine



Exastro
IT Automation

with **Exastro**
Playbook Collection



Exastro
EPOCH



Red Hat
Decision Manager



Red Hat
Ansible Automation Platform



Red Hat
OpenShift Container Platform



✓ 根本的にはシステム形状は変わらないので
運用の手間は大きくは変わらない
→運用自動化への期待 ... ②

守りの自動化

モノリシック
(クラウドリフト)

近代的なデザインミックス環境

✓ アプリケーションを作り直さない限り
多くの機能で冗長2倍は継続
→クラウドシフトへの期待 ... ①

攻めの自動化

自動構築・自律運用

クラウドネイティブ



Exastro