ITmedia DX Summit vol.7
プログラム -Day4- 3月11日　@IT（クラウドネイティブ）

クラウドネイティブの必需品「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

Introduction 「攻めの自動化」「守りの自動化」とは？

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)

Introduction 「攻めの自動化」「守りの自動化」とは？

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

Introduction 「攻めの自動化」「守りの自動化」とは？

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)

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？

新技術を取り込みに

密結合で保守しづらい

従来のスタティックな

手動構築・手動運用

クラウドネイティブに
切り出す！

大変な頑張り…
✓ アプリを作り直し
✓ 初期構築時に
カラクリを仕込む

頻繁に
改造する
部位

モノリシック
(クラウドリフト)

Mobile | Social | IoT | 5G — Tech

スケーラブルなアプリ
コンテナ
サービスメッシュ
イミュータブルインフラストラクチャ
宣言型API

AP

近代的でダイナミックな環境
パブリッククラウド
プライベートクラウド
ハイブリッドクラウド

PF

自動構築・自律運用

OP

クラウドネイティブ

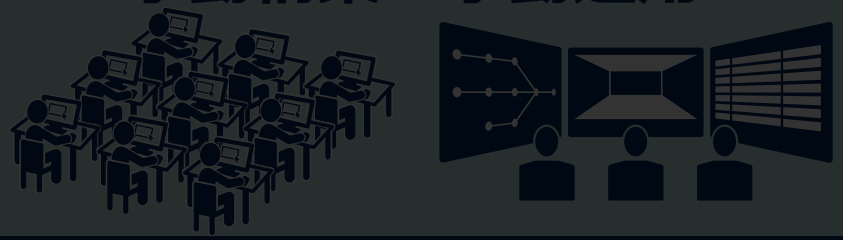
Introduction 「攻めの自動化」「守りの自動化」とは？

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

Exastro

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？

「攻めの自動化」の進め方

クラウドシフトで期待できること

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

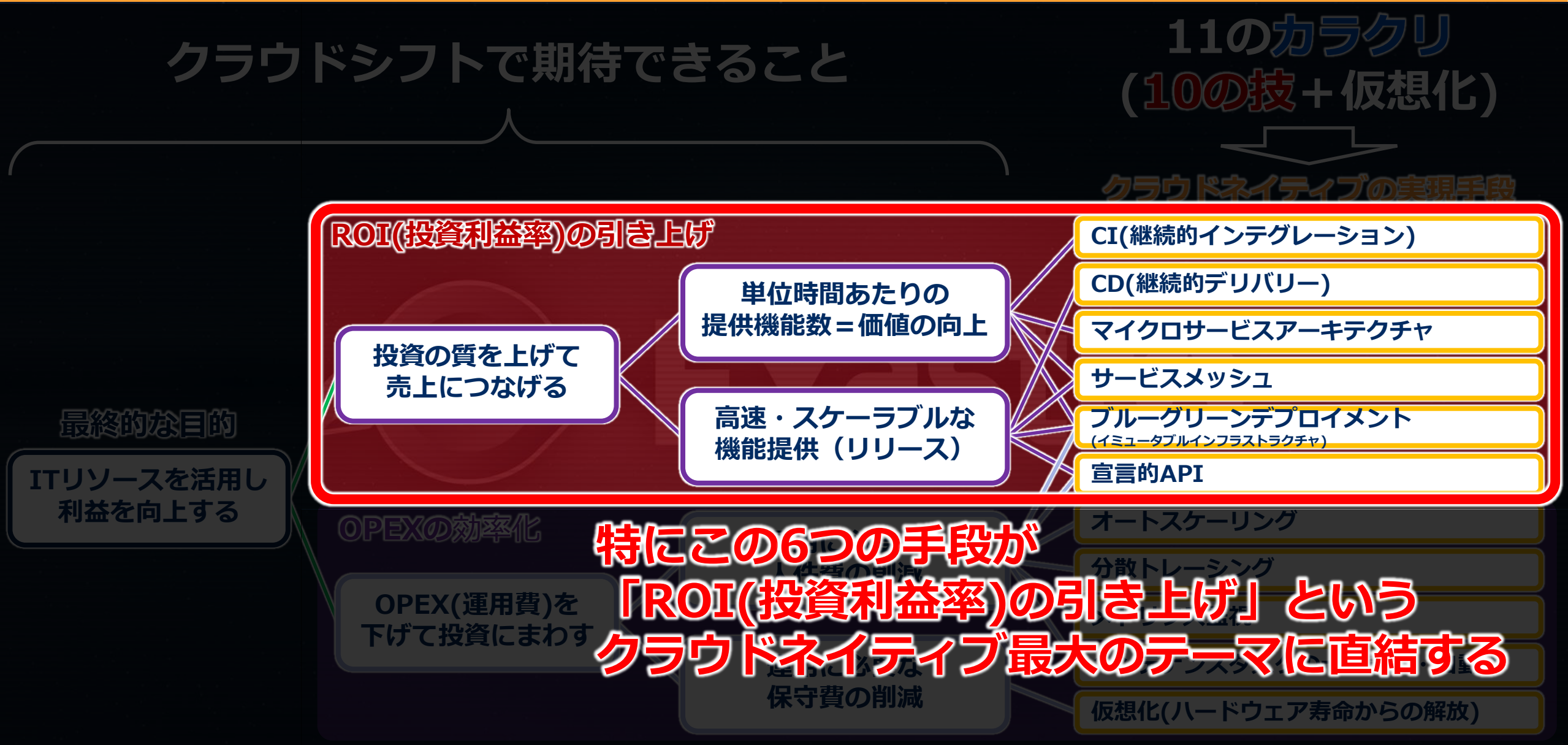
- 運用に必要な人件費の削減
- 運用に必要な保守費の削減

11のカラクリ（10の技＋仮想化）

クラウドネイティブの実現手段

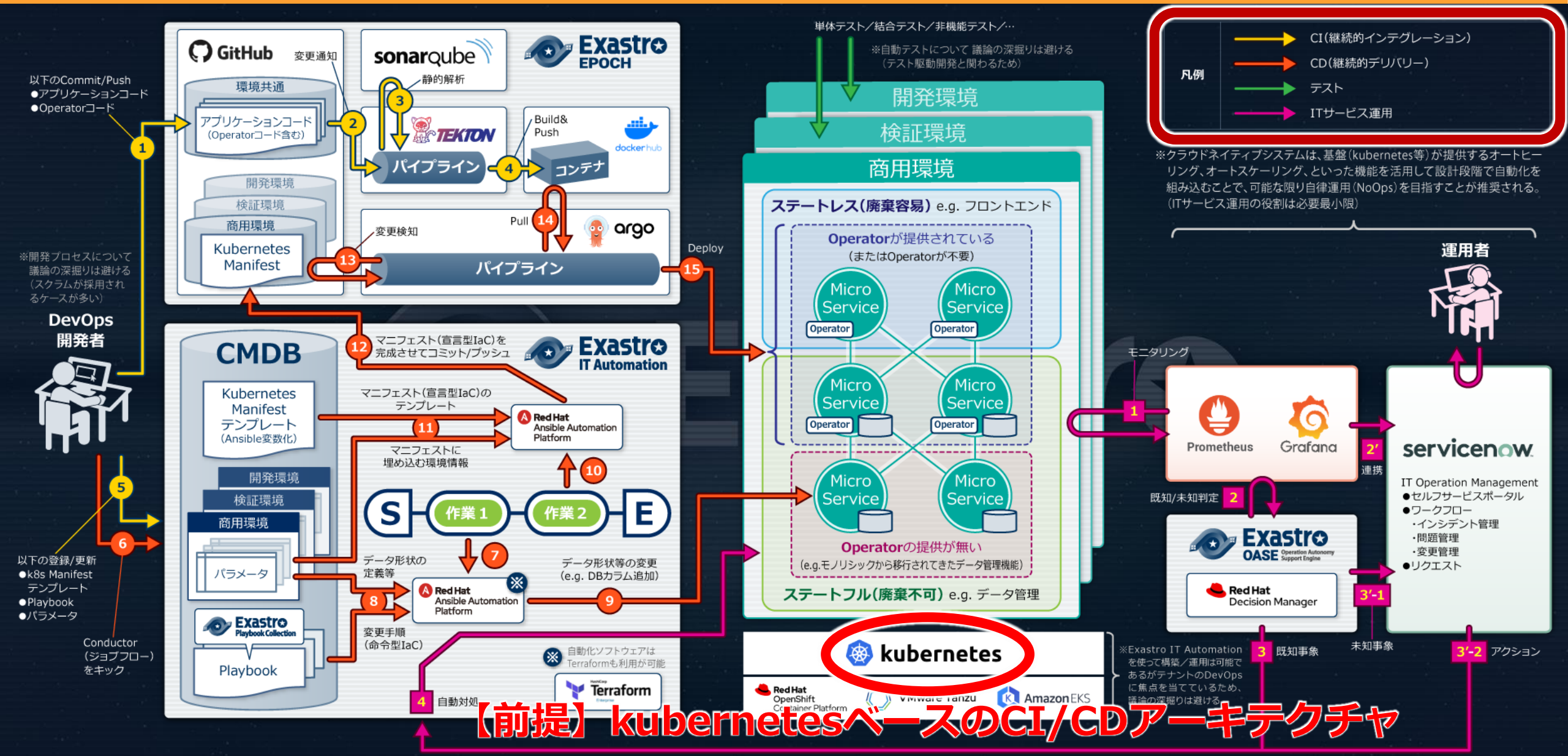
- CI(継続的インテグレーション)
- CD(継続的デリバリー)
- マイクロサービスアーキテクチャ
- サービスメッシュ
- ブルーグリーンデプロイメント
  (イミュータブルインフラストラクチャ)
- 宣言的API
- オートスケーリング
- 分散トレーシング
- メトリクス監視
- メンテナンスタスクの一元管理・自動化
- 仮想化(ハードウェア寿命からの解放)

「攻めの自動化」の進め方



特にこの6つの手段が
「ROI(投資利益率)の引き上げ」という
クラウドネイティブ最大のテーマに直結する

「攻めの自動化」の進め方


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
※クラウドネイティブシステムは、基盤(kubernetes等)が提供するオートヒーリング、オートスケーリング、といった機能を活用して設計段階で自動化を組み込むことで、可能な限り自律運用(NoOps)を目指すことが推奨される。(ITサービス運用の役割は必要最小限)
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るケースが多い)
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
GitHub
変更通知
環境共通
アプリケーションコード
(Operatorコード含む)
sonarqube
静的解析
Exastro
EPOCH
TEKTON
パイプライン
Build&
Push
docker hub
コンテナ
開発環境
検証環境
商用環境
Kubernetes
Manifest
変更検知
Pull
argo
パイプライン
Deploy
CMDB
Kubernetes
Manifest
テンプレート
(Ansible変数化)
マニフェスト(宣言型IaC)を
完成させてコミット/プッシュ
Exastro
IT Automation
マニフェスト(宣言型IaC)の
テンプレート
Red Hat
Ansible Automation
Platform
マニフェストに
埋め込む環境情報
開発環境
検証環境
商用環境
パラメータ
S
作業1
作業2
E
データ形状の
定義等
データ形状等の変更
(e.g. DBカラム追加)
Exastro
Playbook Collection
Playbook
変更手順
(命令型IaC)
※自動化ソフトウェアは
Terraformも利用が可能
Terraform
4 自動対処
単体テスト/結合テスト/非機能テスト/…
※自動テストについて 議論の深掘りは避ける
(テスト駆動開発と関わるため)
開発環境
検証環境
商用環境
ステートレス(廃棄容易) e.g. フロントエンド
Operatorが提供されている
(またはOperatorが不要)
Micro Service
Operator
Operatorの提供が無い
(e.g.モノリシックから移行されてきたデータ管理機能)
ステートフル(廃棄不可) e.g. データ管理
kubernetes
Red Hat
OpenShift
Container Platform
Amazon EKS
※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
運用者
モニタリング
Prometheus
Grafana
連携
既知/未知判定
Exastro
OASE
Red Hat
Decision Manager
既知事象
未知事象
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
アクション
【前提】kubernetesベースのCI/CDアーキテクチャ

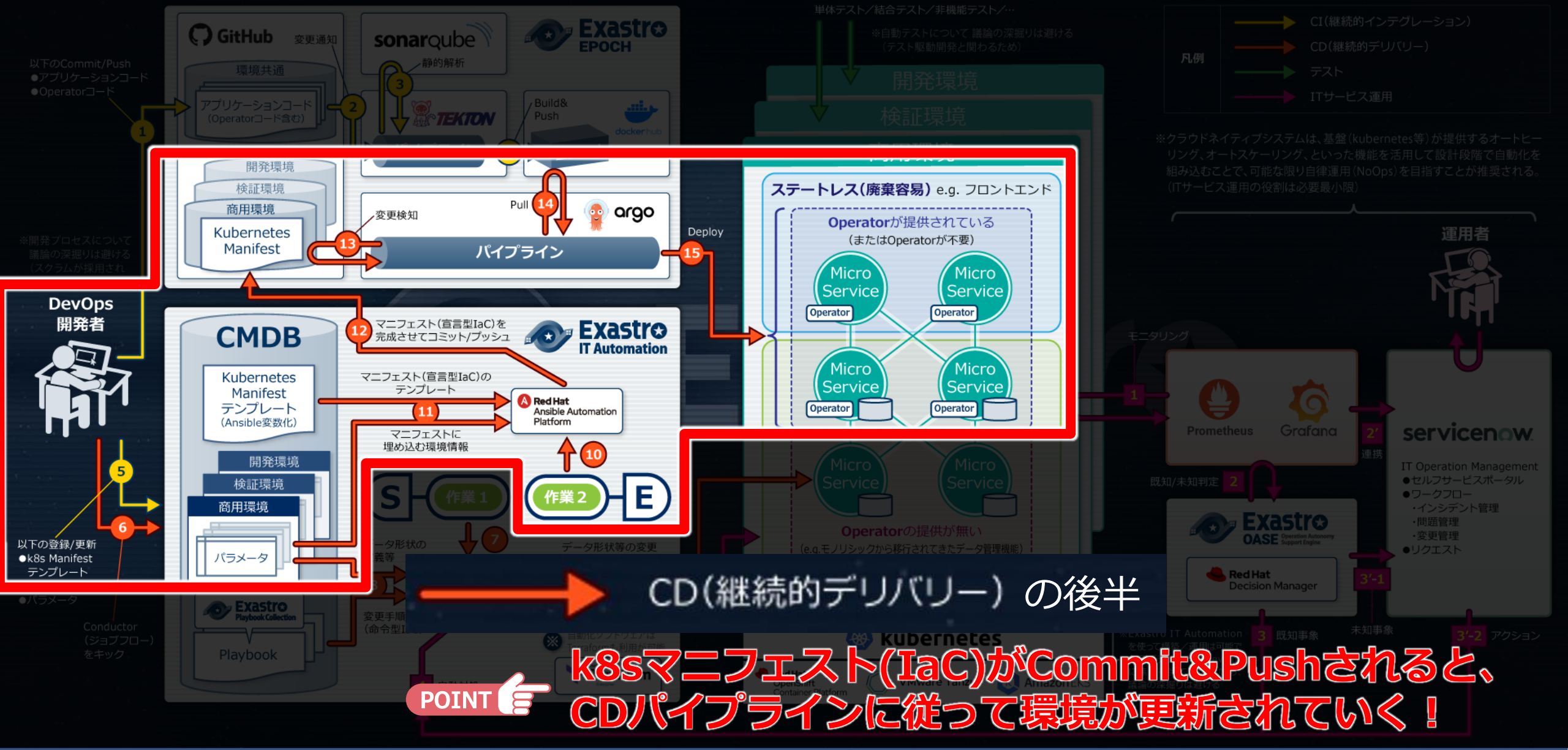
「攻めの自動化」の進め方


GitHub
変更通知
sonarqube
静的解析
Exastro EPOCH
環境共通
アプリケーションコード
(Operatorコード含む)
開発環境
TEKTON
パイプライン
Build&
Push
コンテナ
docker hub
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るケースが多い)
DevOps
開発者
CMDB
Kubernetes
Manifest
テンプレート
(Ansible変数化)
開発環境
検証環境
商用環境
パラメータ
以下の登録/更新
●k8s Manifest
テンプレート
Conductor
(ジョブフロー)
をキック
Exastro
Playbook Collection
Playbook
商用環境
Kubernetes
Manifest
変更検知
Pull
argo
パイプライン
Deploy
マニフェストに
埋め込む環境情報
Ansible Automation
Platform
作業1
作業2
データ形状の
定義等
データ形状等の変更
(e.g. DBカラム追加)
Red Hat
Ansible Automation
Platform
変更手順
(命令型IaC)
自動化ソフトウェアは
Terraformも利用が可能
Terraform
自動対処
単体テスト/結合テスト/非機能テスト/…
※自動テストについて 議論の深掘りは避ける
(テスト駆動開発と関わるため)
開発環境
検証環境
商用環境
ステートレス(廃棄容易) e.g. フロントエンド
Operatorが提供されている
(またはOperatorが不要)
Micro Service
Operator
Operatorの提供が無い
(e.g.モノリシックから移行されてきたデータ管理機能)
ステートフル(廃棄不可) e.g. データ管理
kubernetes
Red Hat OpenShift Container Platform
VMware Tanzu
Amazon EKS
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
※クラウドネイティブシステムは、基盤(kubernetes等)が提供するオートヒーリング、オートスケーリング、といった機能を活用して設計段階で自動化を組み込むことで、可能な限り自律運用(NoOps)を目指すことが推奨される。
(ITサービス運用の役割は必要最小限)
運用者
Exastro OASE
Red Hat Decision Manager
IT Operation Management
・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
既知事象
未知事象
アクション
CI(継続的インテグレーション)
POINT
k8sマニフェスト(IaC)もテンプレート管理しよう！
(環境毎にマニフェストを持つのはバグの温床)

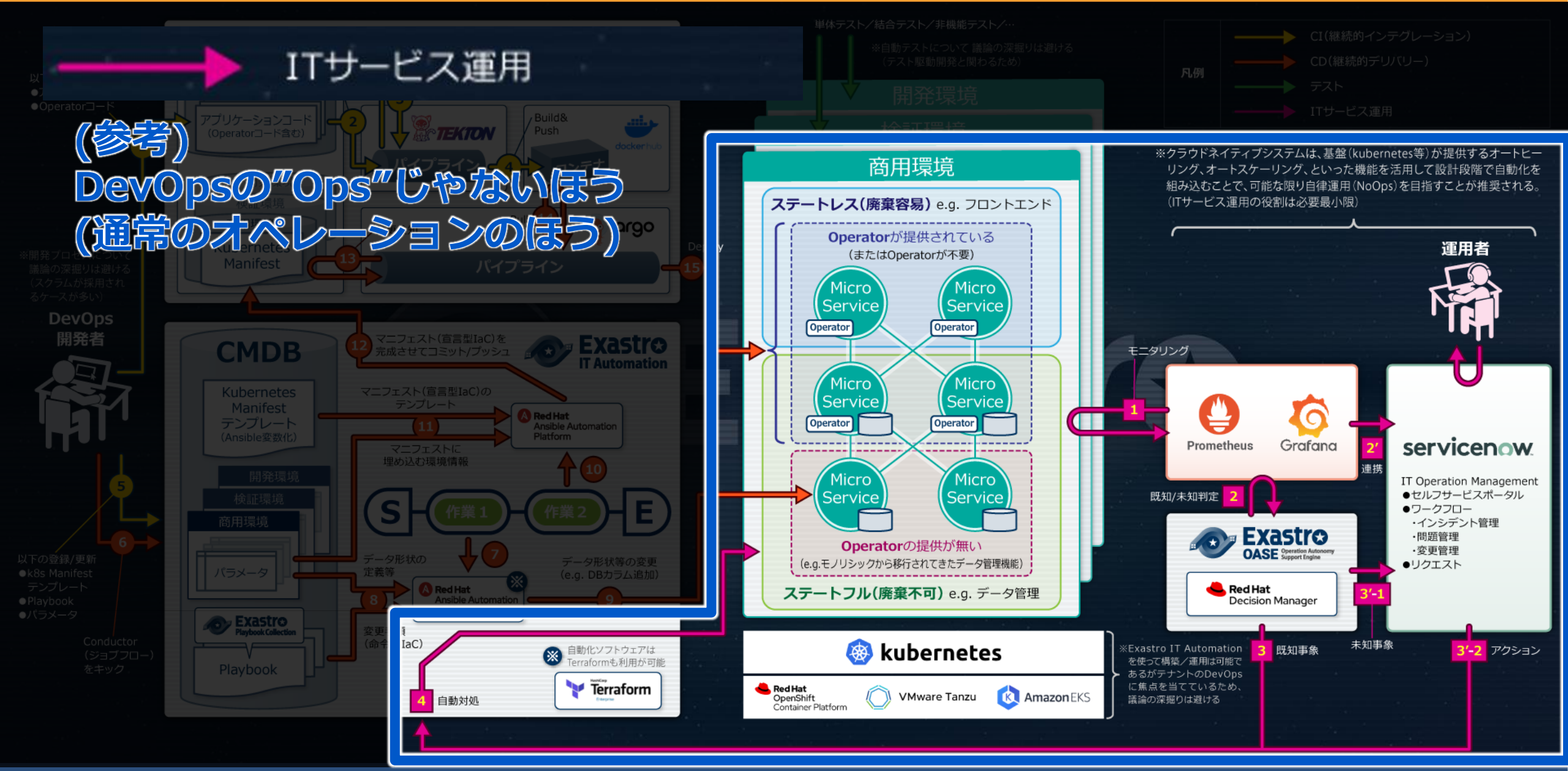
「攻めの自動化」の進め方


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)
GitHub
アプリケーションコード
(Operatorコード含む)
開発環境
検証環境
商用環境
Kubernetes Manifest
変更検知
CMDB
Kubernetes Manifest テンプレート
(Ansible変数化)
DevOps 開発者
Ansible Automation Platform
パラメータ
Exastro Playbook Collection
Playbook
S
作業 1
作業 2
E
Red Hat Ansible Automation Platform
データ形状の定義等
変更手順
(命令型I
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
Micro Service
Micro Service
Operatorの提供が無い
(e.g.モノリシックから移行されてきたデータ管理機能)
ステートフル(廃棄不可) e.g. データ管理
(e.g. DBカラム追加)
kubernetes
Red Hat OpenShift Container Platform
VMware Tanzu
Amazon EKS
Terraform
自動対処
運用者
Prometheus
Grafana
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 OASE
Red Hat Decision Manager
既知/未知判定
既知事象
未知事象
アクション
連携
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

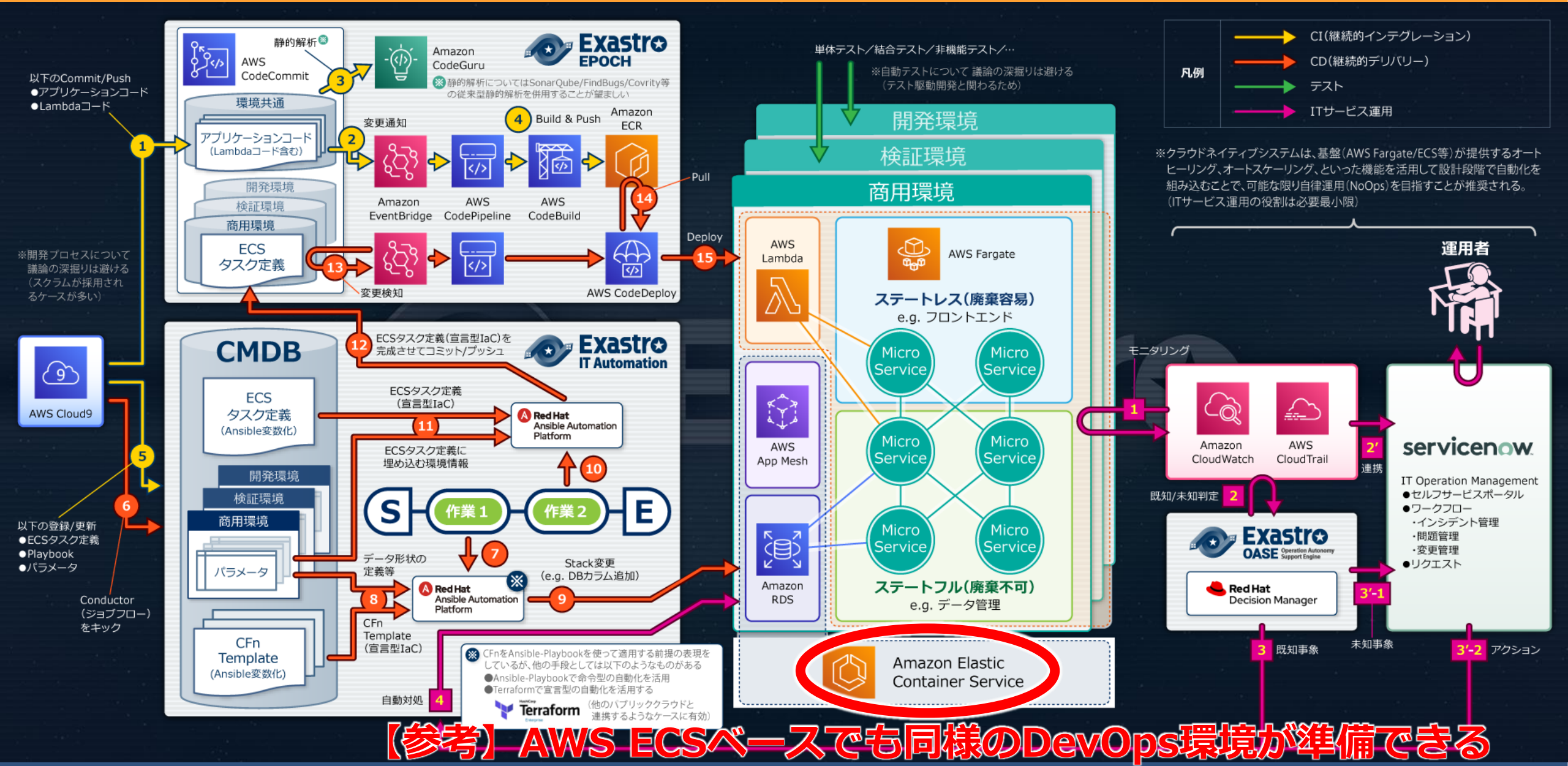
「攻めの自動化」の進め方


GitHub
変更通知
sonarqube
静的解析
Exastro EPOCH
環境共通
アプリケーションコード
(Operatorコード含む)
以下のCommit/Push
●アプリケーションコード
●Operatorコード
TEKTON
Build&
Push
dockerhub
開発環境
検証環境
商用環境
Kubernetes
Manifest
変更検知
Pull
argo
パイプライン
Deploy
DevOps
開発者
CMDB
Kubernetes
Manifest
テンプレート
(Ansible変数化)
マニフェスト(宣言型IaC)を
完成させてコミット/プッシュ
Exastro
IT Automation
マニフェスト(宣言型IaC)の
テンプレート
Red Hat
Ansible Automation
Platform
マニフェストに
埋め込む環境情報
作業 1
作業 2
パラメータ
以下の登録/更新
●k8s Manifest
テンプレート
●パラメータ
Conductor
(ジョブフロー)
をキック
Exastro
Playbook Collection
Playbook
ステートレス(廃棄容易) e.g. フロントエンド
Operatorが提供されている
(またはOperatorが不要)
Micro
Service
Operator
Operatorの提供が無い
(e.g.モノリシックから移行されてきたデータ管理機能)
単体テスト/結合テスト/非機能テスト/…
※自動テストについて 議論の深掘りは避ける
(テスト駆動開発と関わるため)
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
※クラウドネイティブシステムは、基盤(kubernetes等)が提供するオートヒーリング、オートスケーリング、といった機能を活用して設計段階で自動化を組み込むことで、可能な限り自律運用(NoOps)を目指すことが推奨される。
(ITサービス運用の役割は必要最小限)
運用者
モニタリング
Prometheus
Grafana
連携
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
既知/未知判定
Exastro
OASE
Red Hat
Decision Manager
既知事象
未知事象
アクション
kubernetes
CD(継続的デリバリー) の後半
POINT
k8sマニフェスト(IaC)がCommit&Pushされると、
CDパイプラインに従って環境が更新されていく！

「攻めの自動化」の進め方


ITサービス運用
(参考)
DevOpsの"Ops"じゃないほう
(通常のオペレーションのほう)
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
単体テスト/結合テスト/非機能テスト/…
※自動テストについて 議論の深掘りは避ける
(テスト駆動開発と関わるため)
開発環境
DevOps
開発者
●Operatorコード
アプリケーションコード
(Operatorコード含む)
TEKTON
パイプライン
Build&
Push
コンテナ
Kubernetes
Manifest
argo
パイプライン
CMDB
Kubernetes
Manifest
テンプレート
(Ansible変数化)
マニフェスト(宣言型IaC)を
完成させてコミット/プッシュ
マニフェスト(宣言型IaC)の
テンプレート
Red Hat
Ansible Automation
Platform
マニフェストに
埋め込む環境情報
Exastro
IT Automation
開発環境
検証環境
商用環境
パラメータ
S
作業1
作業2
E
データ形状の
定義等
データ形状等の変更
(e.g. DBカラム追加)
Red Hat
Ansible Automation
Exastro
Playbook Collection
Playbook
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
IaC)
自動化ソフトウェアは
Terraformも利用が可能
Terraform
4 自動対処
商用環境
ステートレス(廃棄容易) e.g. フロントエンド
Operatorが提供されている
(またはOperatorが不要)
Micro Service
Operator
Operatorの提供が無い
(e.g.モノリシックから移行されてきたデータ管理機能)
ステートフル(廃棄不可) e.g. データ管理
kubernetes
Red Hat
OpenShift
Container Platform
VMware Tanzu
Amazon EKS
※クラウドネイティブシステムは、基盤(kubernetes等)が提供するオートヒーリング、オートスケーリング、といった機能を活用して設計段階で自動化を組み込むことで、可能な限り自律運用(NoOps)を目指すことが推奨される。
(ITサービス運用の役割は必要最小限)
運用者
モニタリング
1
Prometheus
Grafana
2'
連携
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
既知/未知判定 2
Exastro
OASE
Red Hat
Decision Manager
3'-1
3 既知事象
未知事象
3'-2 アクション
※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



【参考】AWS ECSベースでも同様のDevOps環境が準備できる

「守りの自動化」の進め方

Exastro

「守りの自動化」の進め方


TO-BE
自動化された
システム構築/運用
自動化されたSIの実施
（プロセスと成果物の変更）
Step 3
Step 2
Step 1
AS-IS
手作業での
システム構築/運用
自動化の事前準備
（Step 1、Step 2、Step 3）

「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

- ☑ 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

- ☑ 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)

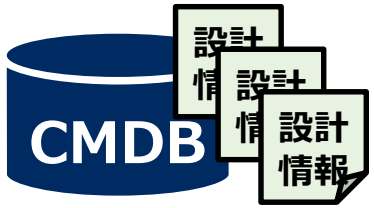
- ☑ 人手作業なので作業時間が一定でない
  ⇒チーム間で作業待ちが発生
- ☑ 人手作業なので人為ミスの懸念から逃れられない

「守りの自動化」の進め方

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

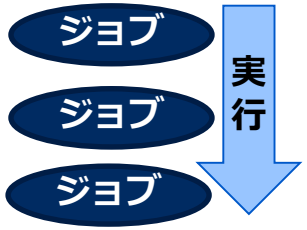
解決策

Step 1
設計情報の一元管理



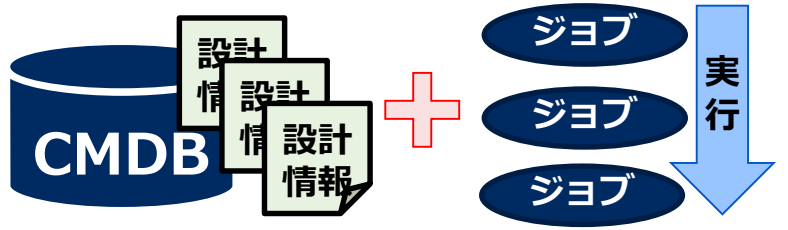
解決策

Step 2
自動実行の実現



連携

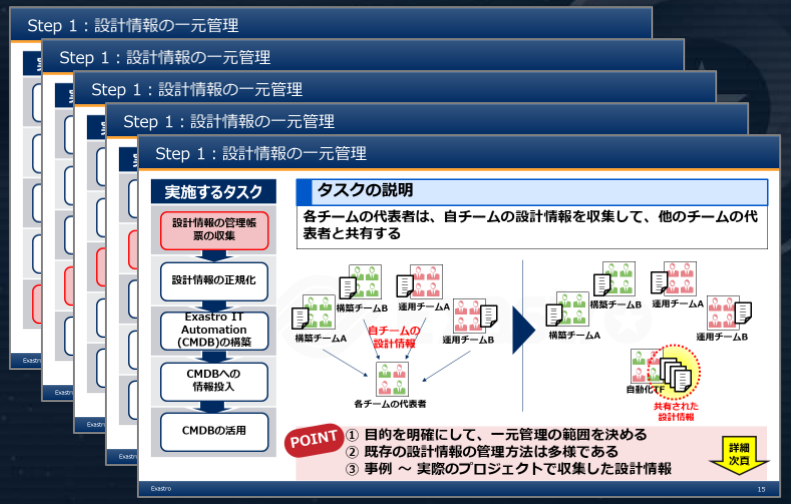
Step 3
一元管理と自動実行の連携



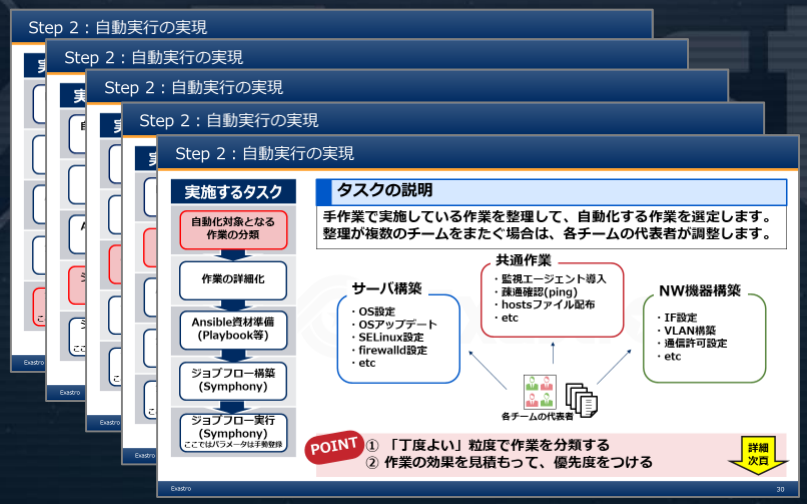
「守りの自動化」の進め方

細かく刻めば12つのタスクで解決できます

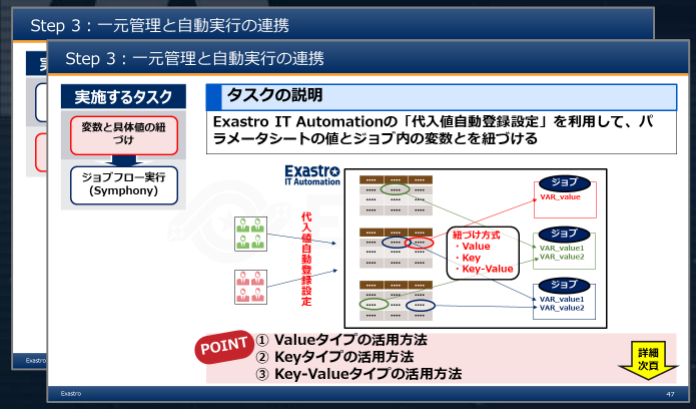
Step 1 ➡ Step 2 ➡ Step 3



5つのタスク



5つのタスク



2つのタスク

「守りの自動化」の進め方


TO-BE
自動化された
システム構築/運用
自動化されたSIの実施
（プロセスと成果物の変更）
Step 3
Step 2
Step 1
AS-IS
手作業での
システム構築/運用
自動化の事前準備
（Step 1、Step 2、Step 3）

「守りの自動化」の進め方

フェーズにおけるQCDの変化

凡例：変化なし　改善　追加の検討項目あり

	要件定義			基本設計			詳細設計			運用設計			製造			テスト			リリース		
	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q	C	D
自動化前	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺
自動化後	☺	追加	追加	☺	☺	☺	改善	改善	改善	☺	☺	☺	改善	改善	改善	☺	☺	☺	改善	改善	改善

プロセスと成果物の変化

凡例：変更なし 作業名　変更あり 作業名　追加 作業名　消滅 作業名

自動化前

- プロセス：→ 要件の確認 → 要件定義書の作成 →
- 成果物：・要件のリスト　・要件定義書

自動化後

- プロセス：→ <変更あり> 要件の確認 (自動化適用の検討) → <変更なし> 要件定義書の作成 →
- 成果物：・要件のリスト　・自動化適用の検討結果　・要件定義書

解説

要件の確認の段階で、自動化の適用範囲等について検討を行い、合意を取る必要がある。その検討分としてCとDが増加する。

「守りの自動化」の進め方

フェーズにおけるQCDの変化

凡例：変化なし　改善　追加の検討項目あり

	要件定義			基本設計			詳細設計			運用設計			製造			テスト			リリース		
	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q	C	D
自動化前	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺
自動化後	☺	追加	追加	☺	☺	☺	改善	改善	改善	☺	☺	☺	改善	改善	改善	☺	☺	☺	改善	改善	改善

プロセスと成果物の変化

凡例：変更なし 作業名　変更あり 作業名　追加 作業名　消滅 作業名

自動化前

プロセス：→ 基本設計書の作成 →

成果物：・基本設計書

自動化後

プロセス：<変更なし>
→ 基本設計書の作成 →

成果物：・基本設計書

解説

基本設計に取り込むべき内容は、自動化の事前準備の段階で検討済みであるため、ここでは追加の作業はない。

「守りの自動化」の進め方

フェーズにおけるQCDの変化

凡例：変化なし　改善　追加の検討項目あり

	要件定義			基本設計			詳細設計			運用設計			製造			テスト			リリース		
	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q	C	D
自動化前	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺
自動化後	☺	追加	追加	☺	☺	☺	改善	改善	改善	☺	☺	☺	改善	改善	改善	☺	☺	☺	改善	改善	改善

プロセスと成果物の変化

凡例：変更なし 作業名　変更あり 作業名　追加 作業名　消滅 作業名

自動化前

プロセス：パラメータ設計 → パラメータシート作成 → 作業手順書の作成 →

成果物：・パラメータ設計書　・パラメータシート　・作業手順書

自動化後

プロセス：<変更なし> パラメータ設計 → <変更あり> CMDBへのパラメータ登録 → <消滅> ~~作業手順書の作成~~ →

成果物：・パラメータ設計書　・CMDB　・~~作業手順書~~

解説

パラメータ設計で作成された各パラメータは、CMDBに登録していく。これによりパラメータの形式化が進み、表記ゆれや曖昧さが提言するため、Qが改善する。

また、パラメータの適用順序などの作業手順は、自動化の事前準備の段階で作成したジョブフローに置き換わるため、作業手順書の作成のタスクが消滅する。これにより、CとDも改善する。

フェーズにおけるQCDの変化

凡例：変化なし　改善　追加の検討項目あり

	要件定義			基本設計			詳細設計			運用設計			製造			テスト			リリース		
	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q	C	D
自動化前	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺
自動化後	☺	追加	追加	☺	☺	☺	改善	改善	改善	☺	☺	☺	改善	改善	改善	☺	☺	☺	改善	改善	改善

プロセスと成果物の変化

凡例：変更なし 作業名　変更あり 作業名　追加 作業名　消滅 作業名

自動化前

プロセス：→ 監視設計書の作成 → 運用手順書の作成 →

成果物：・監視設計書　・運用手順書

自動化後

プロセス：→ <変更なし> 監視設計書の作成 → <変更なし> 運用手順書の作成 →

成果物：・監視設計書　・運用手順書

解説

本書は構築の自動化に焦点を当てているため、運用の自動化は範囲外とする。

運用の自動化を実施する場合は、QCDおよびプロセスの変化が発生する。

「守りの自動化」の進め方

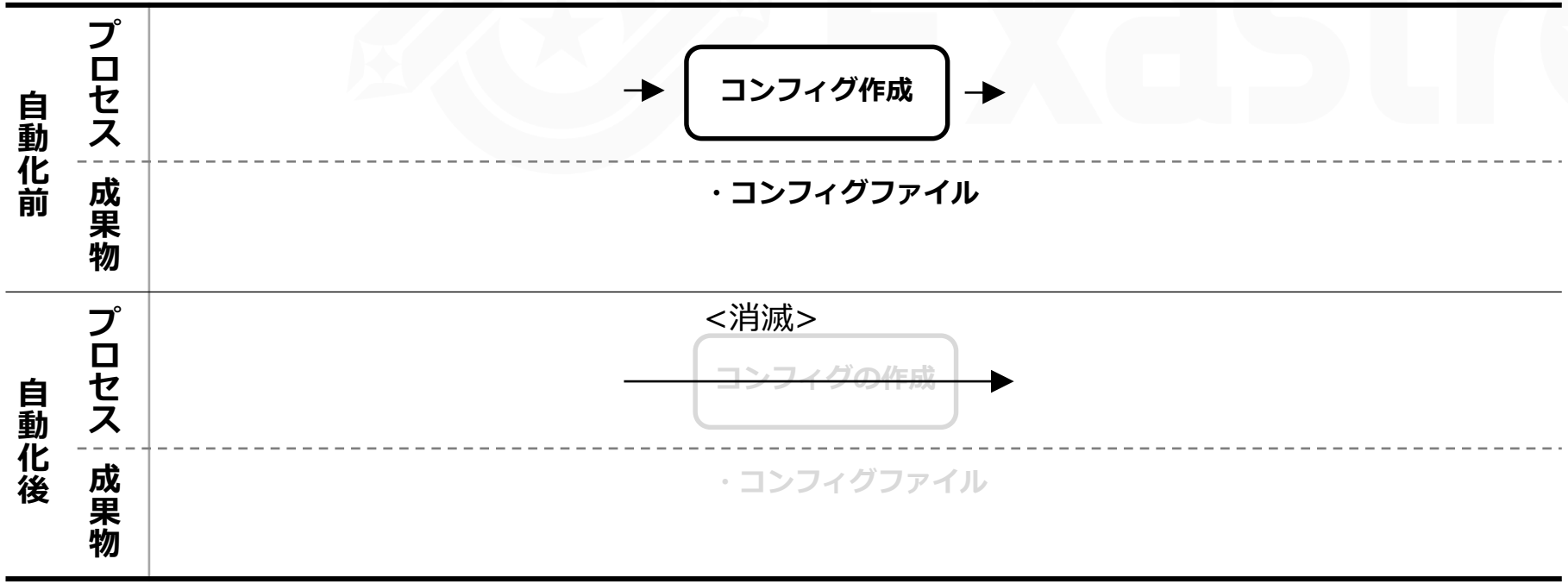
フェーズにおけるQCDの変化

凡例：変化なし　改善　追加の検討項目あり

	要件定義			基本設計			詳細設計			運用設計			製造			テスト			リリース		
	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q	C	D
自動化前	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺
自動化後	☺	😅	😅	☺	☺	☺	😄	😄	😄	☺	☺	☺	😄	😄	😄	☺	☺	☺	😄	😄	😄

プロセスと成果物の変化

凡例：変更なし 作業名　変更あり 作業名　追加 作業名　消滅 作業名



解説

詳細設計に基づき作成されるコンフィグファイルは、IaCとCMDBにより自動生成されるため、コンフィグ作成のタスクは消滅する。

「守りの自動化」の進め方

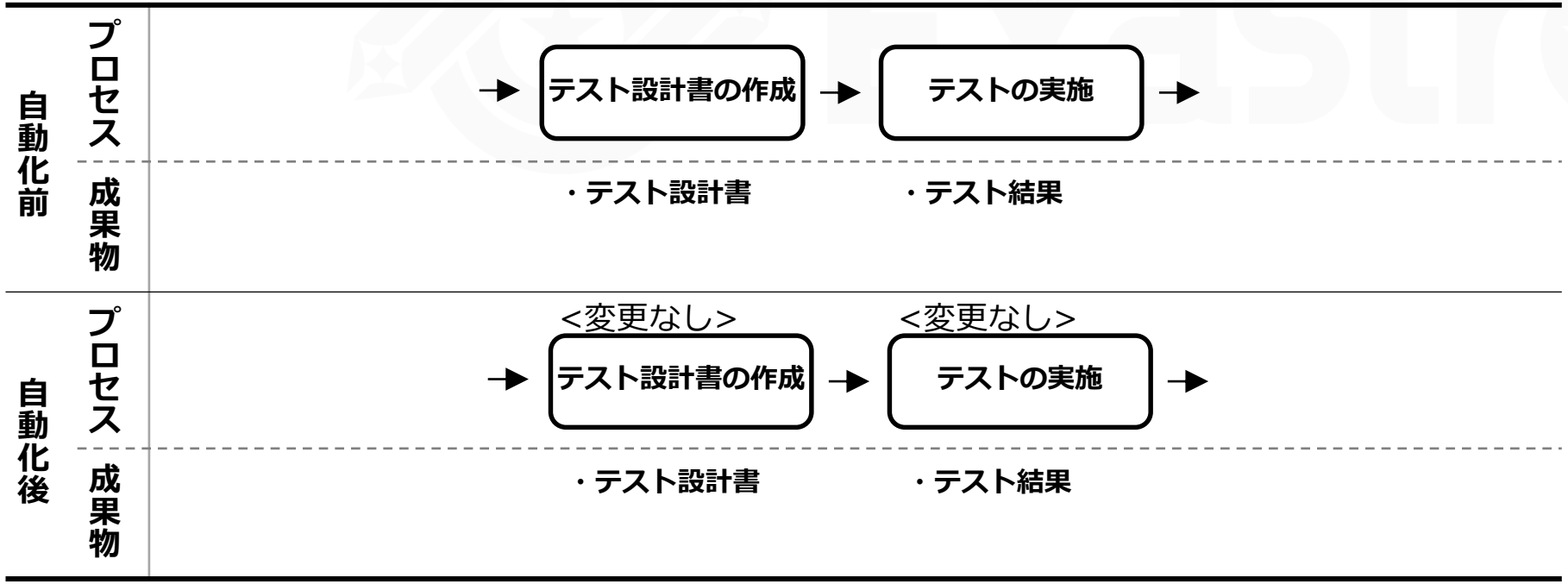
フェーズにおけるQCDの変化

凡例：変化なし　改善　追加の検討項目あり

	要件定義			基本設計			詳細設計			運用設計			製造			テスト			リリース		
	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q	C	D
自動化前	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺
自動化後	☺	☺	☺	☺	☺	☺	😄	😄	😄	☺	☺	☺	😄	😄	😄	☺	☺	☺	😄	😄	😄

プロセスと成果物の変化

凡例：変更なし 作業名　変更あり 作業名　追加 作業名　消滅 作業名



解説

本書は構築の自動化に焦点を当てているため、テスト自動化は範囲外とする。

テストの自動化を実施する場合は、QCDおよびプロセスの変化が発生する。

「守りの自動化」の進め方

フェーズにおけるQCDの変化

凡例： 変化なし　改善　追加の検討項目あり

	要件定義			基本設計			詳細設計			運用設計			製造			テスト			リリース		
	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q	C	D
自動化前	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺
自動化後	☺	💦	💦	☺	☺	☺	😄	😄	😄	☺	☺	☺	😄	😄	😄	☺	☺	☺	😄	😄	😄

プロセスと成果物の変化

凡例： 変更なし 作業名　変更あり 作業名　追加 作業名　消滅 作業名

自動化前

- プロセス：タイムチャート作成 → 作業の実施 → エビデンス確認 →
- 成果物：
  - タイムチャート作成：・タイムチャート
  - 作業の実施：・更新されたシステム ・エビデンス
  - エビデンス確認：・エビデンス確認結果

自動化後

- プロセス：<消滅> タイムチャート作成 → <変更あり> ジョブフロー実行 → <消滅> エビデンス確認 → <追加> ジョブフローの実行結果の確認 →
- 成果物：
  - タイムチャート作成：・タイムチャート
  - ジョブフロー実行：・更新されたシステム ・エビデンス ・エビデンス確認結果
  - エビデンス確認：・エビデンス確認結果
  - ジョブフローの実行結果の確認：・確認結果

解説

詳細設計で作成したジョブフローの実行がメインの作業になる。

タイムチャート作成がジョブフロー実行に置き換わるため消滅。

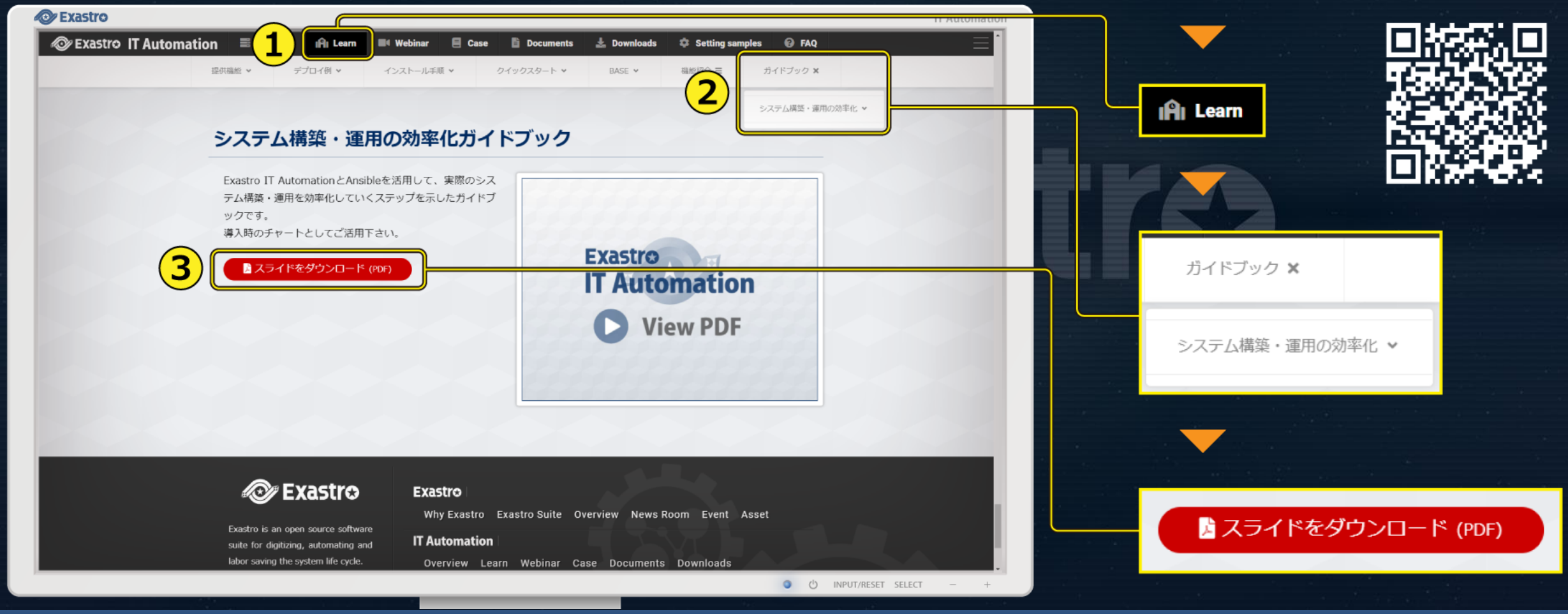
またジョブフロー中でエビデンスの確認も実施するため、タスクとしてのエビデンス確認も消滅。

よって、ジョブフローの実行と結果の確認のみとなり、QCDともに改善する

「守りの自動化」の進め方

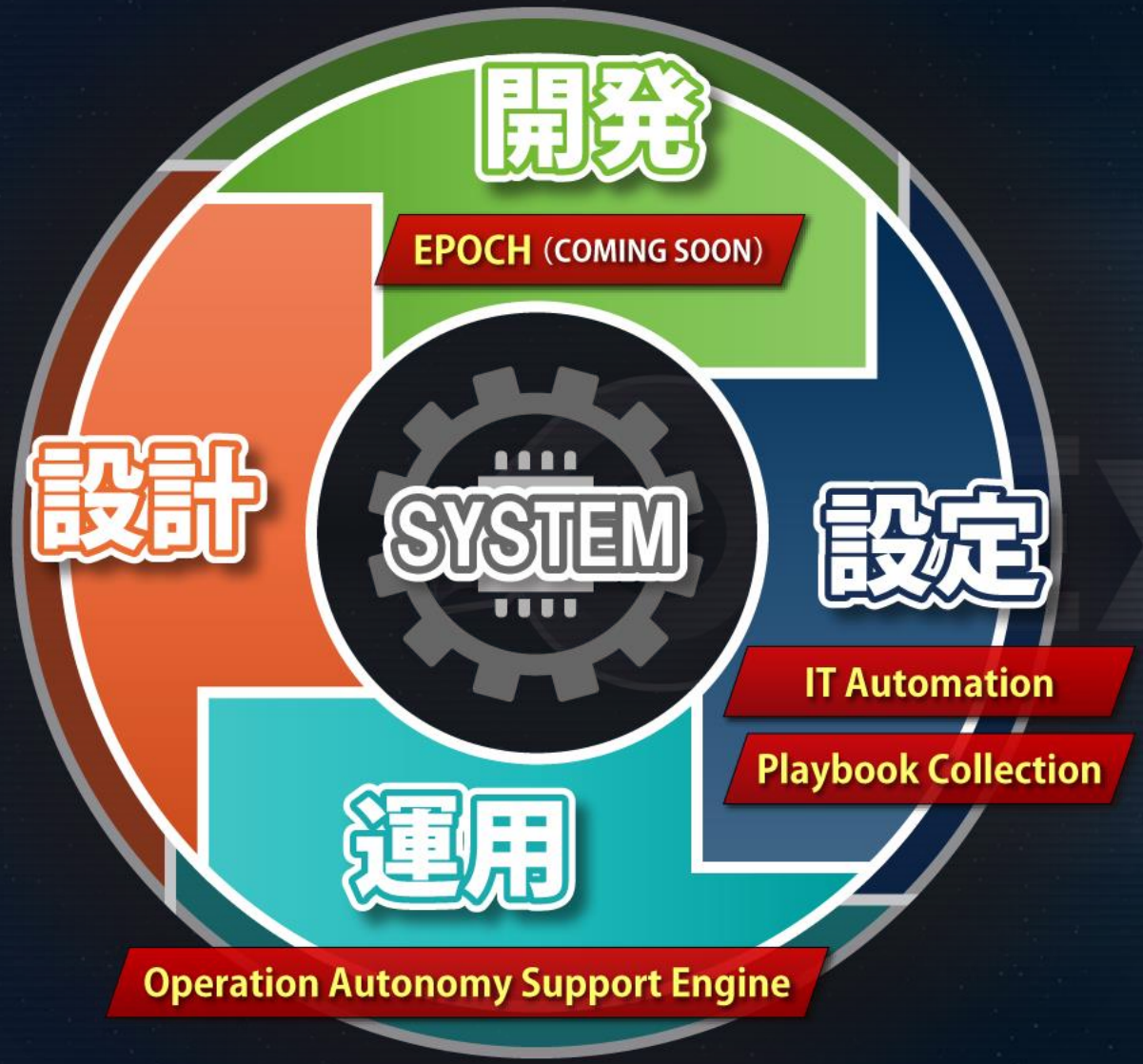
ガイドはOSSコミュニティーサイトからダウンロードできます

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



Exastro Suite
- 「攻めと守りの自動化」の実現手段 -

Exastro Suiteとは？



Exastro

Exastro は システムライフサイクル

設計 design	開発 development	設定 setting	運用 operation

を

デジタル化・自動化・省力化することを目的とした
オープンソースのソフトウェアスイートです。

詳しくはコミュニティーサイトへ！

Exastro

Search Exastro

https://exastro-suite.github.io/docs/index_ja.html

お問い合わせはコチラへ

contact@exastro.jp.nec.com

- クラウドネイティブのデモをお見せできます。
- ガイドブックを詳しく解説できます。

お問い合わせいただければと存じます


Exastro