クラウドネイティブへの加速を妨げているものとは？

ビジネス環境の激しい変化に対応するために、多くの企業で DX（デジタルトランスフォーメーション）の推進が求められている。DX を推進していく上で重要なカギとなる技術がクラウドネイティブだ。

既存のシステムに多く見られるモノリシックなアーキテクチャとは異なり、クラウドネイティブは、オープンかつスケーラブルなシステムを実現することを可能にする。

しかし、既存のモノリシックなシステムを運用しながら、クラウドネイティブアプリケーションを開発することは簡単ではない。多くの IT エンジニアが既存システムの構築、運用に張り付いており、IT エンジニアが不足しているためである。また、クラウドネイティブアプリケーションを開発するためにはコンテナやコンテナオーケストレーションといった技術を開発プロセスに組み込む必要があり、技術者の習熟や実践のための環境構築が必要である。つまり、片手間では難しいことがモダナイズの障壁となっている。

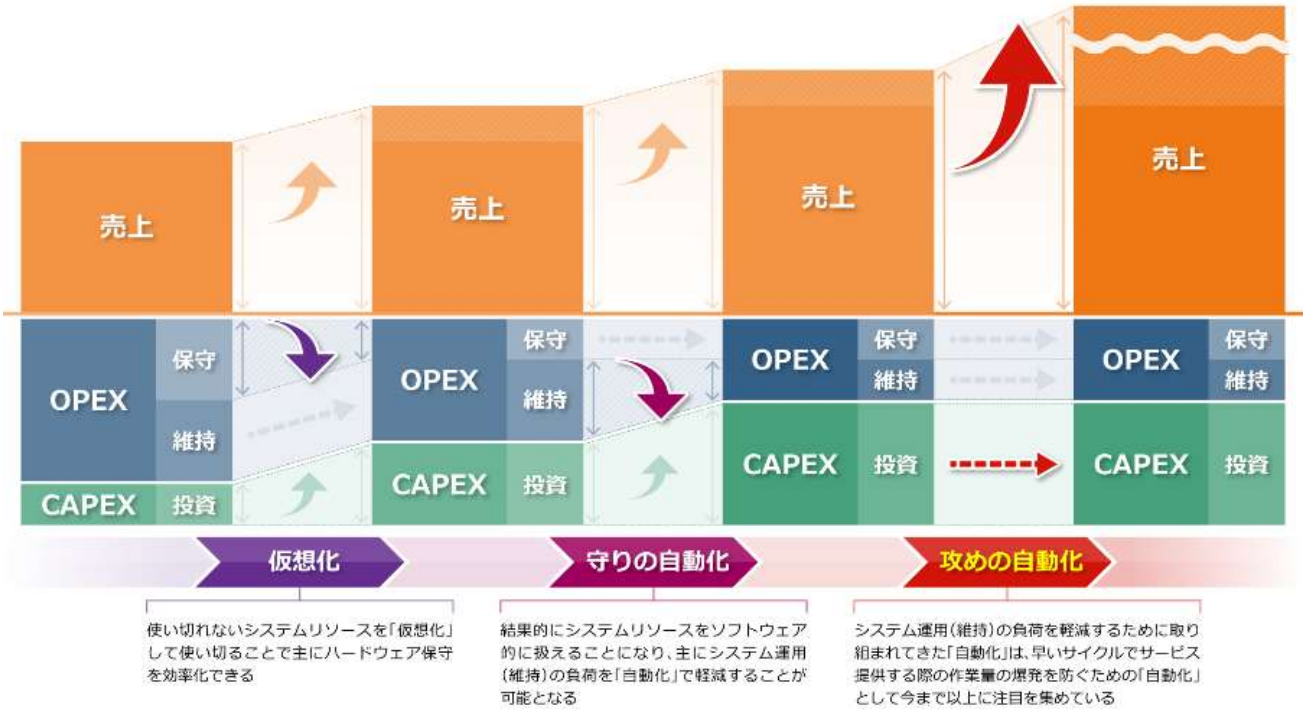
これらの問題を解決するためには、既存システムに張り付いている IT エンジニアを開放し、クラウドネイティブ化を実現する IT エンジニアとして確保する必要がある。しかし、多くの人員を割いているとはいえ、既に最適化されている人員を違う作業に割り当てられるほど効率化、省力化するのは容易ではない。

そこで注目されるのが「自動化」である。

システム構築、運用の自動化は既存のシステム管理からエンジニアを開放するだけでなく、スピーディなサービス展開に追従するためにも有効である。

目的が異なる2つの自動化

システムの世界では、数年ほど前から「自動化」がキーワードとして取り上げられてきた。まず、自動化の前進として行われてきたのが「仮想化」である。仮想化が行われたことによって、ハードウェアリソースをシステム的、効率的に扱うことができ、保守費削減の実現を可能とした。



その後、次のステップとして維持費の削減が注目された。そこで取り組まれ始めたのが、既存のモノリシックなシステムにおける構築、運用の自動化、すなわち、「守りの自動化」である。構築、運用業務の様々な作業をシステムに置き換え、自動的に実施することで、作業時間の短縮や人的リソースの効率化の実現を促進している。

そして、この守りの自動化によりクラウドネイティブ化が進んだ後に、次のステップとして登場するのが「攻めの自動化」である。

攻めの自動化は、クラウドネイティブシステムをスピーディに構築、運用する取り組みだ。ブルーグリーンデプロイメント等の手法を使い環境を再構築するなど、自動構築と自動運用が当然の前提となる。攻めの自動化の目的は、投資対効果（ROI）の引き上げにある。

守りの自動化、攻めの自動化、これら２つの領域は別々に議論され、それぞれ異なる技術で自動化を考える傾向にあった。しかし、実際には、クラウドネイティブ化が進んでも、モノリシックとクラウドネイティブがハイブリットな形で構築、運用されるケースが多い。

そのため、まずは守りの自動化の実現により、現状発生している運用コストを下げること。そして、その際に、攻めの自動化も見据えた実現手法を選択することが重要となる。

「守りの自動化」の実現で考えるべきポイントとは

「守りの自動化」には、構築の自動化、定常作業の自動化など、様々な自動化すべき作業があるが、最も自動化すべき作業は“障害対処の自動化”であると考える。

障害対処は、属人化や高負荷となりやすい作業であり、一瞬の遅れがサービスへの影響に繋がってしまう。そのため、作業時間の短縮や人的リソースの効率化が必要となってくる。

では、障害対応の自動化を行う上でどういった点を考慮すべきだろうか。障害対応で必要な次の５つのステップの観点から考えてみる。

１．障害検知
２．暫定対処
３．サービス影響確認

4．調査
5．本格対処

1. 障害検知

障害が発生したら、まずはその異常に気づかなくてはならない。この異常に気付くためのソフトウェアは監視ソフトウェアとして多く流通しており、最も自動化に着手しやすい作業と言えるだろう。

2. 暫定対処

障害を検知したら、すぐに影響を最小限に抑えるために対処する必要がある。対処内容は、サービスの利用者に障害を通知したり、代替手段によりサービスを継続する等、障害の内容によって様々である。そのため、実際の運用現場では、多くの場合、有識者が先導し、手動で対処にあたっている。この点は、属人化しやすく、スピードも求められることから、自動化に取り組むべきポイントとなる。

3. サービス影響確認

障害発生により、サービスへの影響範囲を確認するためのステップである。SLA（Service Level Agreement）との比較や、ステークホルダーへの報告のために必要なステップである。本ステップは、自動化のスコープ対象から外れる場合が多いが、サービスを運用する上で欠かせない作業であるため、自動化のフローに組み込むことで、運用負荷削減に繋げていきたいポイントとなる。

4. 調査

システムは暫定対処をして終わりではない。新規の事象の場合、何故障害が発生したのか、根本原因を突き止める必要がある。調査は、システムのリソース状況やシステムが出力するログを分析することが多い。ここでは、主に人による作業が中心となるが、情報収集の作業を自動化することが可能だ。部分的に自動化を取り入れることで、より“手動でしか実施できない作業”に集中することができ、人的リソースを効率化することが可能となる。

5. 本格対処

最後に、同じ障害の発生を防ぐために、対処を行う。本ステップは、対処内容の難易度に応じて手動、自動を選択していくのが良い。対処内容を細かくモジュール化し、再利用性の高い作業については、自動化を進めていくといった手法をとることも可能だ。

以上が、障害対応におけるステップとなる。すべてのステップを一連のフローで自動化するのは少々難易度が高いが、「4．調査」の情報収集までを一連のフローとして自動化し、さらに「5．本格対処」も部分的に自動化を取り入れて行くことをお勧めしたい。

「攻めの自動化」の実現で考えるべきポイントとは

「攻めの自動化」には、スピーディに“価値を創出するための自動化”が最も重要であると考える。

スピーディに価値を創出するための開発手法としてアジャイル開発があるが、アジャイル開発ではユーザの要望に応じて通常1、2週間程度のサイクルで実装とリリースを行う。つまり、1、2週間ごとにテストやコンテナイメージのビルド、そしてリリースといった作業を頻繁に繰り返す必要があるのだ。

旧来型の開発と比較してテスト、ビルド、リリースといった作業回数が圧倒的に増えるため、こういった作業を自動化し、どれだけ作業者が開発に集中できるかがポイントとなってくる。

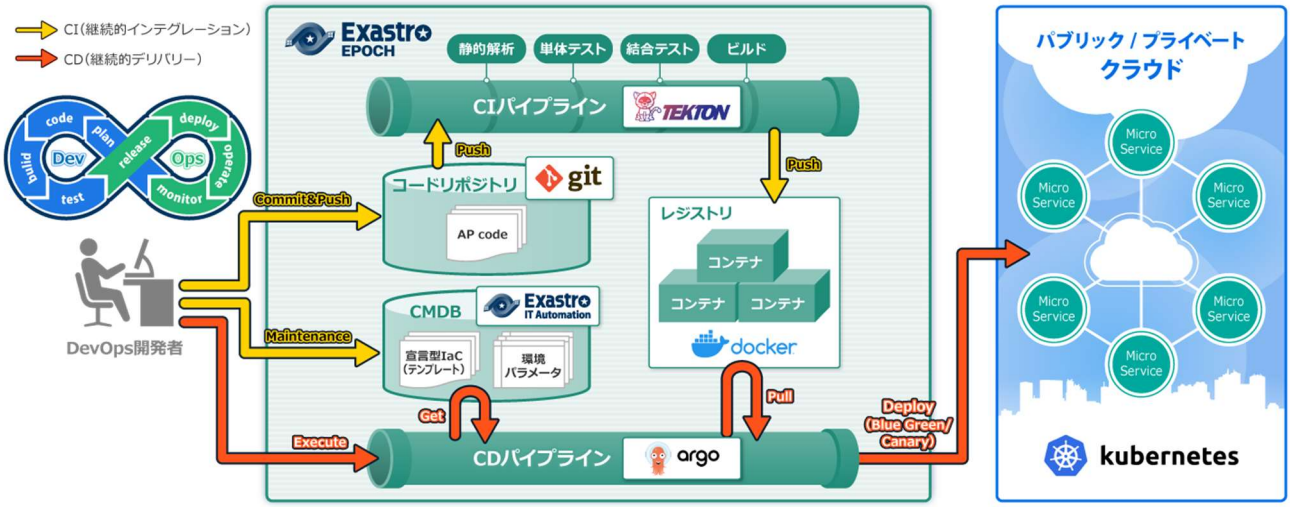
攻めの自動化の実現をサポートする

「Exastro EPOCH」

攻めの自動化のための環境立ち上げに有効なのは、オープンソースソフトウェア（OSS）の「Exastro EPOCH」だ。

Exastro EPOCH は、コンテナベースのアプリケーション開発を支援し、Kubernetes に代表されるコンテナオーケストレーションツールへのデプロイのプロセス及び環境をセットで提供している。

CI パイプラインでは、コンテナアプリケーションのビルドをサポートし、CD パイプラインでは、コンテナオーケストレーション基盤を維持・管理することが可能だ。



クラウドネイティブなアプリケーションの CI/CD パイプラインを即時提供

アプリケーションの開発からサービス提供までのプロセスを、一元管理するための仕組みの一つに CI/CD(Continuous Integration/Continuous Delivery)がある。一方で、CI/CD を始めるためには、数多ある CI/CD ツールの中から必要なものを選定し、更にそれらを適切に連携させるために構築作業や設定の投入をする必要があり、そこには多大な時間とコストを必要とする。

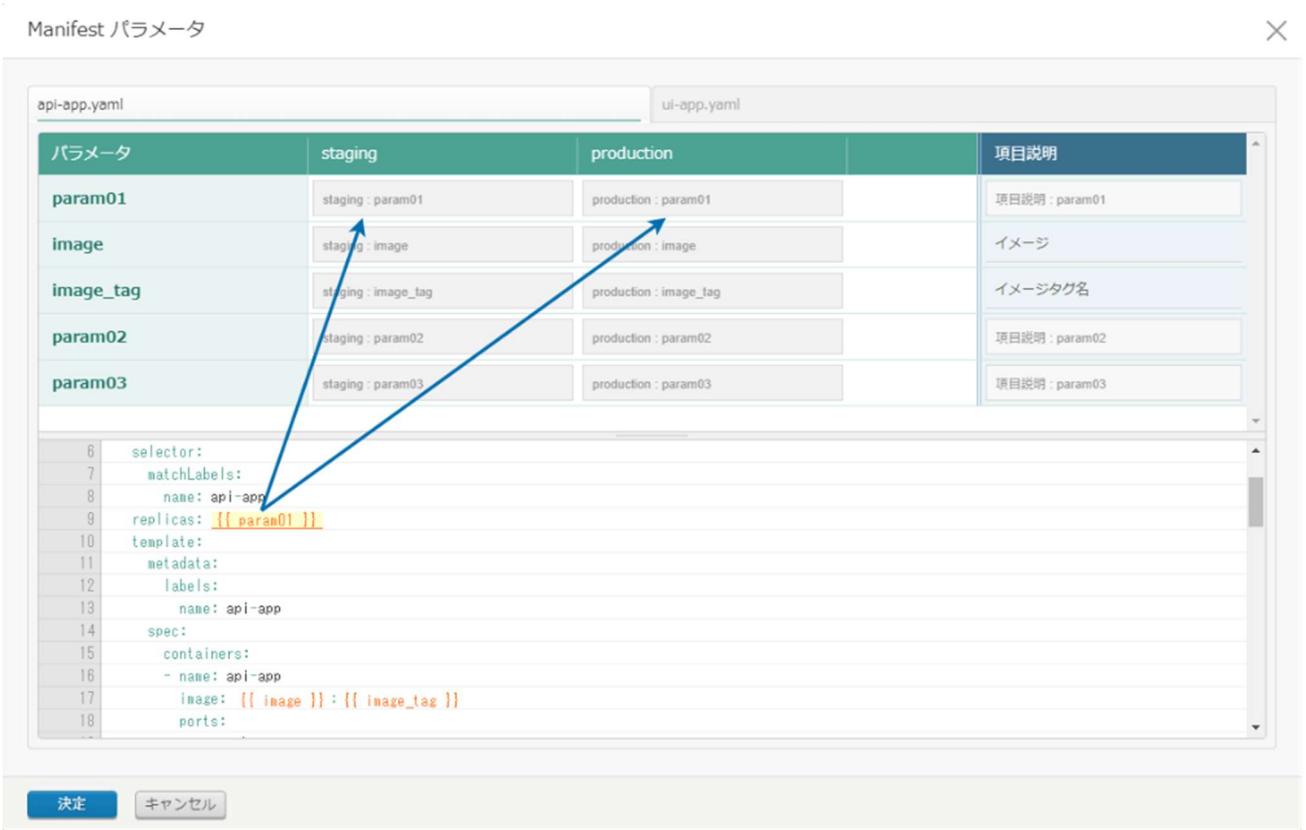
Exastro EPOCH は、CI/CD を始めるために必要なツールを内包・連携した状態で提供しており、CI/CD の各ツールの使用方法を理解することなく、すぐに、簡単に取り掛かることができる。

CI パイプラインでは、開発したアプリケーションのソースコードを静的解析し、単体・内部結合テストを経て、コンテナイメージのビルドまでをノンストップで行える。

CD パイプラインでは、CI パイプラインでビルドしたイメージを選択するだけで、コンテナオーケストレーション環境にすぐにリリースできる。

宣言的 IaC の環境依存パラメータをシステム上で管理

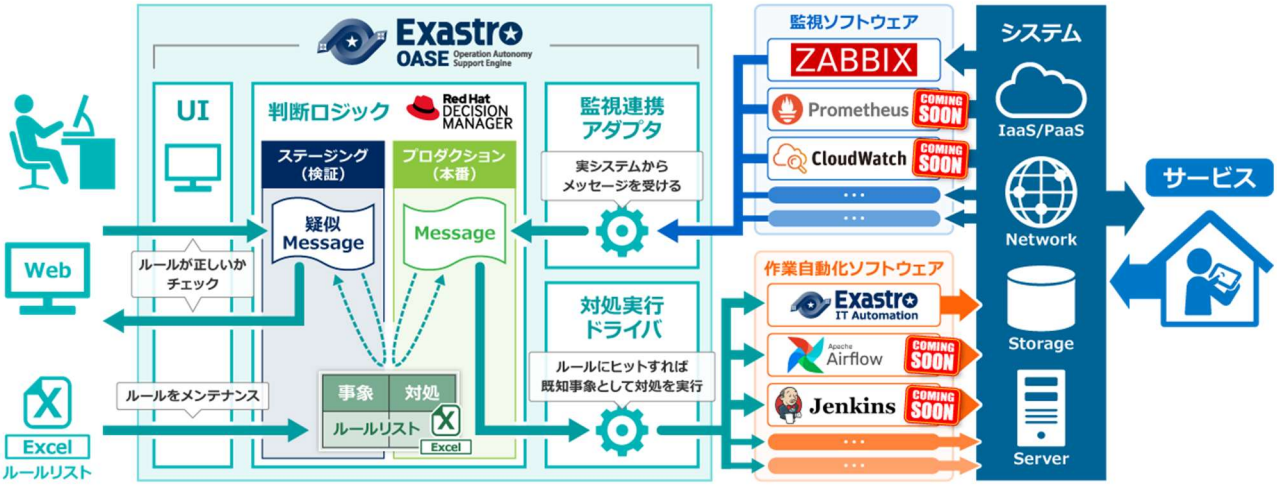
プロダクション環境とステージング環境はなるべく同じ状態にしておくことが望ましいが、Kubernetes のようにマニフェストファイル(宣言型 IaC)を管理する場合、各環境ごとに同じ項目と異なる項目を適切に管理する必要がある。なぜなら、管理が適切に行われない場合、誤ったマニフェストファイルを Kubernetes クラスタに適用することで、サービス障害を発生させてしまう可能性があるからだ。例えば、開発、検証、本番環境といった複数の環境を持っている場合、テストの状況に応じて、環境ごとに異なるコンテナイメージ名とタグ名を指定する必要がある。Exastro EPOCH では、マニフェストファイル(宣言型 IaC)の共通する部分はテンプレートとして管理し、異なる部分をパラメータとして管理することで適切に宣言型 IaC 管理する機能を持っている。 この機能を使うことでマニフェストファイルの共通部分はテンプレートとして管理し、コンテナイメージ名やタグ名のような変更が頻繁に発生するものはパラメータとして管理することが簡単に実現できる。



守りの自動化の実現をサポートする
「Exastro Operation Autonomy Support Engine」(Exastro OASE)

障害対応の切り分けで有効となる手段が、オープンソースソフトウェア（OSS）の「Exastro OASE」だ。

Exastro OASE は、監視ソフトウェアや作業自動化ソフトウェアと連携し、障害対処における既知未知の判断と、既知事象における対処内容の判断を行う。



Exastro OASE の特徴は 6 つある。

・ルールリストをステージングで確認する
・監視ソフトウェアや作業自動化ソフトウェアとマルチに連携する
・コーディングレスなルールを定義する
・アクション実行回数を制御する
・複数のルールリストを管理する
・グループ毎にアクセスを制御する

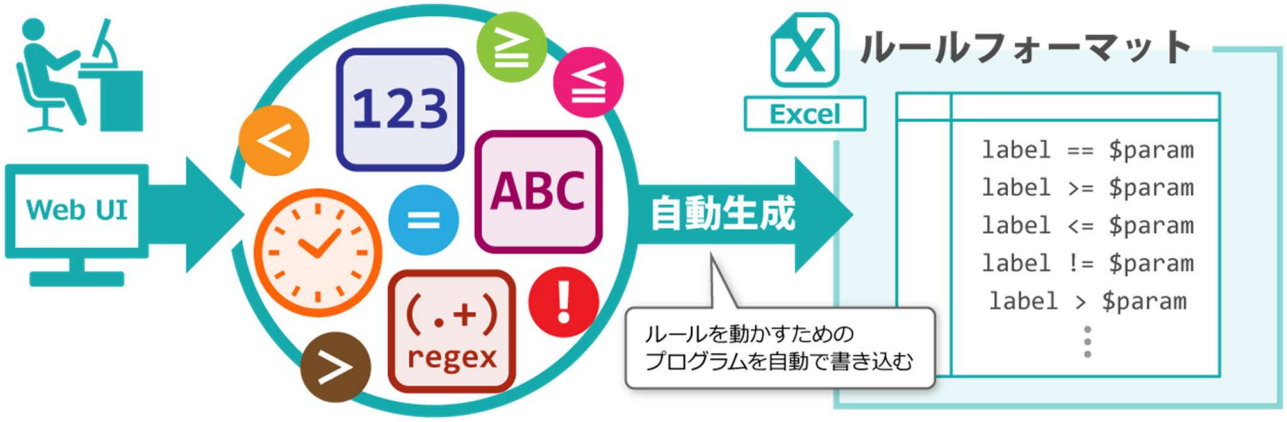
今回は、「コーディングレスなルールを定義する」と「複数のルールリストを管理する」について、詳しく説明する。

コーディングレスなルールを定義する

障害発生時の暫定対処は監視ソフトウェア上でも実現可能ではないかと考える方もいるであろう。確かに、Zabbix など、対処を登録し、自動実行するソフトウェアはいくつか存在する。しかし、そういった機能は、対処内容をコード化して記述しなくてはならない場合が多く、普段コーディングを行わない運用者にとっては大きな障壁となる。

そこで、Exastro OASE では、Red Hat Decision Manager をシステム内に組み込むことで、障害メッセージに紐づける対処の定義、すなわち、ルール定義を Excel で実施することを可能とした。Excel は、運用の様々な作業で利用されることが多く、運用者にとっても、馴染みの深いツールである。

また、Excel を利用するにあたり、様々な定義をコード化し、仕込んでおく必要があるが、Exastro OASE では、必要なコーディングは全て自動で実施を行う。そのため、運用者は、「対処を行う条件」と「対処方法」を、自然言語で記載することが可能となる。（正規表現の利用も可能）



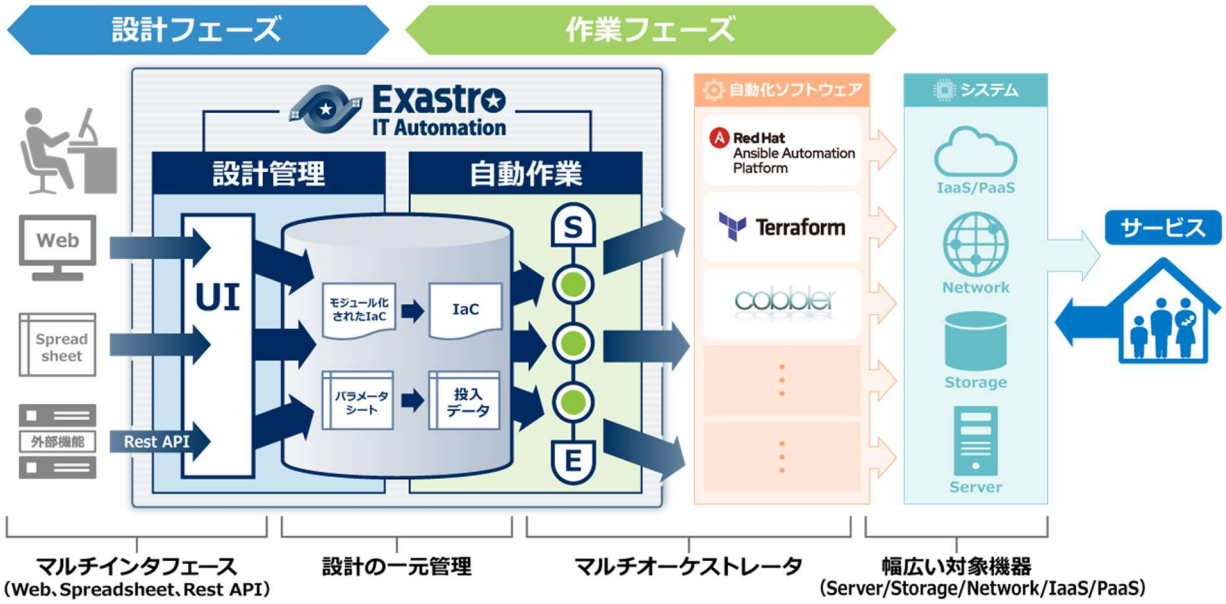
複数のルールリストを管理する

ルールリストは複数作成することが可能だ。そのため、サービス毎や、インフラ等、メンテナンスし易い状態にカスタマイズできる。発生する障害メッセージはルールリスト毎に設定でき、自動で振り分けが行われるため、他のルールリストを考慮する必要もない。さらに、「グループ毎にアクセスを制御する」の特徴とも関連するが、ルールリスト毎に、アクセス制御の設定が可能なため、管理者を分けてメンテナンスしていくことも可能である。

守りの自動化を実現し、
攻めの自動化の武器にもなる
「Exastro IT Automation」(Exastro ITA)

作業実行において有効な手段が、同じくオープンソースソフトウェア（OSS）の「Exastro ITA」だ。

Exastro ITA は、作業に必要な IaC（Infrastructure as Code）や、IaC を実行する上で必要となるパラメータ情報を一元管理し、作業自動化ソフトウェアに連携を行うことで、システムに対して、自動実行を行う。Red Hat Ansible Automation Platform（Ansible）や Terraform Enterprise（Terraform）といった自動化ソフトウェアを実行管理することが可能で、従来のモノリシックなシステムだけでなく、クラウドネイティブなシステムも対象となる。そのため、攻めと守りの自動化、どちらにも対応できるソフトウェアと言える。



Exastro ITA の特徴は 7 つある。

・マルチインタフェースと RBAC（ロールベースアクセス制御）
・パラメータをグルーピング／履歴管理する
・IaC を解析して変数を刈り取る
・IaC をモジュール管理して再利用性を高める
・複数の自動化ソフトウェアを繋げて実行する
・自動化をやめない最後の切り札 Pioneer モード
・実行状況をリアルタイムで監視する

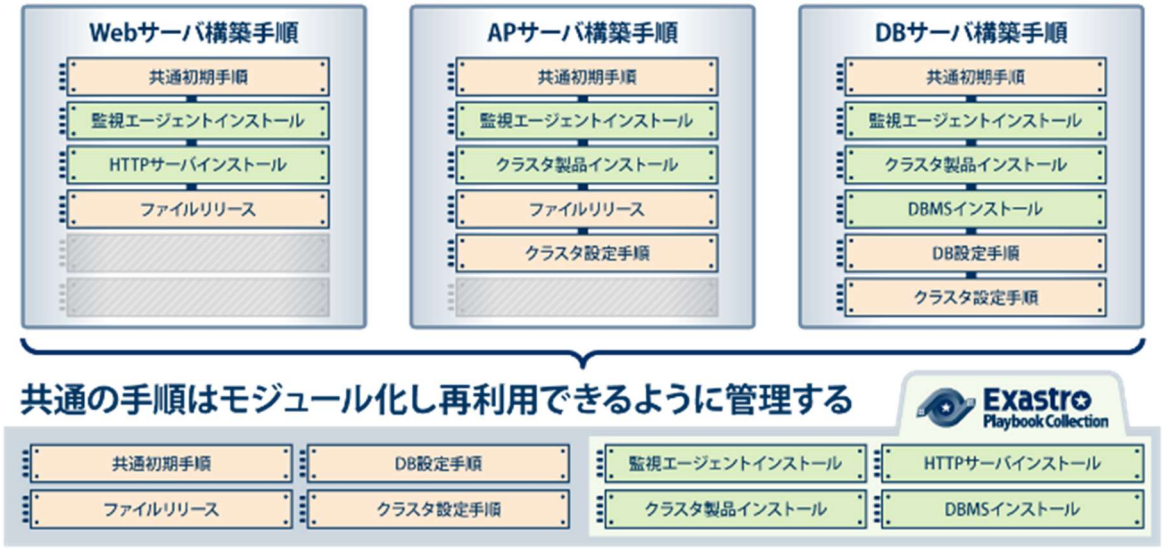
今回は、「IaC をモジュール管理して再利用性を高める」と「複数の自動化ソフトウェアを繋げて実行する」について、詳しく説明する。

IaC をモジュール管理して再利用性を高める

IaC（Playbook 等）は、一度のみの利用で使い捨ててしまうのは非常に勿体ない。できる限り再利用し、メンテナンスを容易にしていくことが必要だ。Exastro ITA では、再利用性を高めることができるよう、IaC をモジュール化し、作業時に組み立てる仕組みをもつ。

例えば、「図ｘｘ」のような状況を考えてみる。図に示している３つの構築手順は、一見すると全く異なる作業であるが、モジュール化してみるとどうだろうか。初期手順や、ファイルリリース、クラスタ設定手順等、共通している手順があることが確認できる。予めそれらを細かくモジュール化し、管理しておくことで、再利用を行うことが可能となるのである。

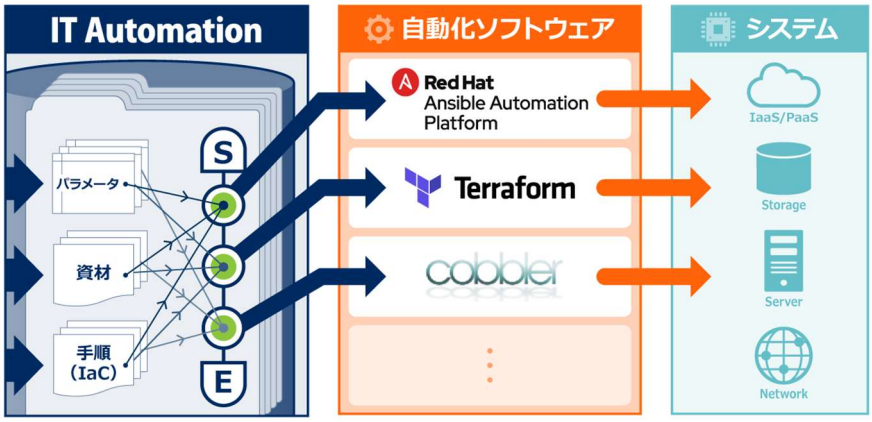
Exastro ITA を利用すると、それらモジュール化した IaC の管理、実行が容易となる。



複数の自動化ソフトウェアを繋げて実行する

Exastro ITA では、IaC やパラメータを管理し、作業自動化ソフトウェアを単に実行するだけのソフトウェアではない。自動化ソフトウェアを動かすためには、その自動化ソフトウェアが求める状態に情報を加工する必要があるが、Exastro ITA では、その加工も自動的に行っている。

例えば、Red Hat Ansible Automation Platform では、実行する対象のノード毎に host_vars の作成が必要となるが、Exastro ITA では自動的に生成を行うため、意識する必要がなく、さらに何十台、何百台といったノードに対しての実行も容易に行うことができる。



Exastro コミュニティサイト

メモ


Exastro