

セイコーソリューションズ アラクサラネットワークス レッドハット SB C&S エーピーコミュニケーションズ共催

SmartCS x ALAXALA x Ansible ハンズオン

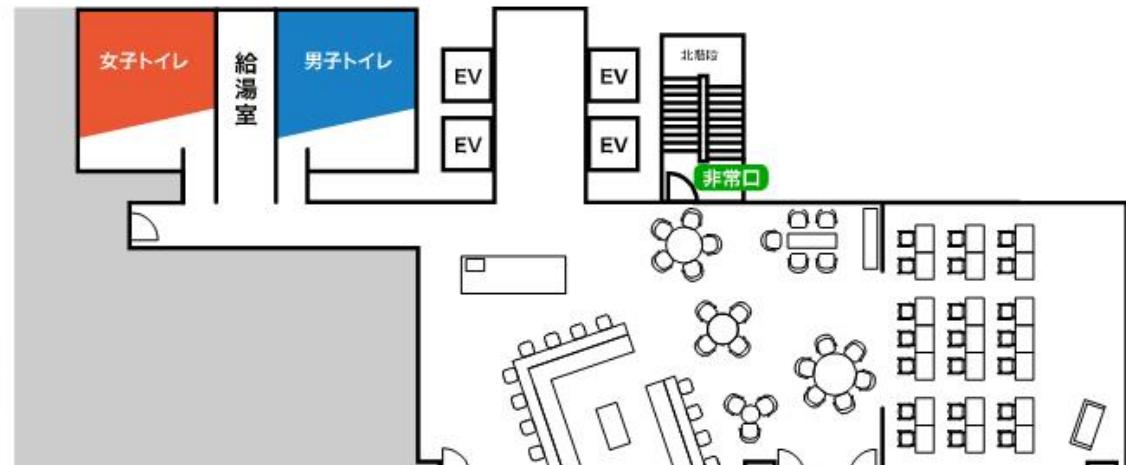
2020/1/31

エーピーコミュニケーションズ 設備のご案内

喫煙室について

エレベーターで1階へ、エレベーターホールを背にして、左手の通路にある**奥の扉**に入り、階段にて**B1F**へ
B1の扉から入り**奥の扉**を開けた**左手**が喫煙室です

※灰皿は設置されていませんので
給湯室にある携帯灰皿をお持ちください



SmartCS × ALAXALA ×



Red Hat

Ansible Automation
Platform

SmartCS × ALAXALA × Ansible ハンズオン

時間：1月31日（金）13時～18時（受付12時30分より）

会場：エーピーコミュニケーションズ

| 内容 | 担当 |
|--|------------------|
| はじめに | SB C&S 社 |
| NW自動化とAnsible | レッドハット 社 |
| コンソールサーバー SmartCSの説明 | セイコーソリューションズ |
| ■ハンズオン 演習1：ハンズオン環境の確認 演習2：SmartCSの基本動作(手動編) | セイコーソリューションズ |
| Ansible×SmartCSについて | セイコーソリューションズ |
| ALAXALA ネットワークス 様の機器紹介 Ansibleの取り組み | アラクサラネットワークス 社 |
| ■ハンズオン 演習3：Ansible×SmartCS×Alaxalaの連携 (基礎編) 演習4：Ansible×SmartCS×Alaxalaの連携 (応用編) | セイコーソリューションズ |
| Ansible Towerの説明 | レッドハット 社 |
| ■ハンズオン 演習5：Ansible Tower×SmartCS×Alaxalaの連携 ユースケースの紹介 | セイコーソリューションズ |
| 本日のまとめ | エーピーコミュニケーションズ 社 |

【はじめに】
SB C&S社

【NW自動化とAnsible】

Red Hat社



Red Hat
Ansible Automation
Platform

今、自動化が求められている背景

ITシステムは複雑化し作業は煩雑化しているが、 ITに対する周囲からの期待値は上がり続けている

- IT環境は刻々と変化
ex. UNIX→x86→仮想化→クラウド→コンテナ
 - システムごとに異なるツール・異なる運用
 - 結果、ITに関わる作業は刻々と煩雑化
 - IT運用者の**負担が増大**
- ⋮
- 周囲からのITに対する期待は、システムが増えるごとに上がり続けている

IT人材の不足が今後さらに深刻化

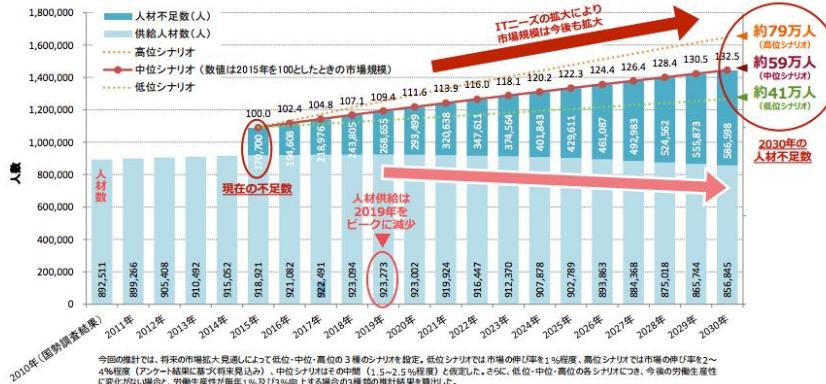
IT人材の「不足規模」に関する推計結果

- IT関連産業の産業人口に関する将来推計（マクロ推計）の一環として、人材の不足状況や今後の見通しに関するアンケート調査結果に基づき、現在及び将来の人材不足数に関する推計も実施。
- マクロ推計によれば、**2015年時点で約17万人のIT人材が不足している**という結果になった。さらに、前頁で示されたとおり、今後IT人材の供給力が低下するにもかかわらず、ITニーズの拡大によってIT市場は今後も拡大を続けることが見込まれるため、IT人材不足は今後ますます深刻化し、**2030年には、(中位シナリオの場合で)約59万人程度まで人材の不足規模が拡大する**との推計結果が得られた。

2 今後のIT人材の不足規模

IT人材の不足規模に関する予測

- 2015年の人材不足規模：約17万人
- 2030年の人材不足規模：約59万人（中位シナリオ）
- ⇒ IT人材不足は、今後ますます深刻化



IT人材の不足は、

2030年までに**79万人**不足が予測されている

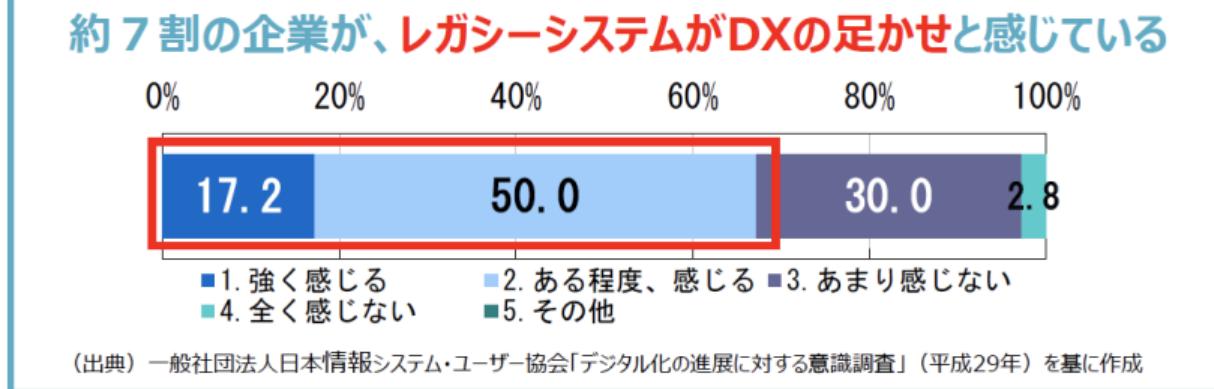
2019年4月、
働き方改革法案施行

時間外労働の上限について、
月45時間、年360時間を原則とし...*

*厚労省による概要資料より抜粋

ITシステム 2025年の崖 by 経済産業省

- 2025年以降、最大12兆円/年の経済損失が生じる可能性
 - システムの維持管理費がIT予算の9割以上に(技術的負債)
 - 保守運用担当者の不在によるシステム障害発生リスクが高まる
 - 市場の変化に迅速に対応できず、世界とのデジタル競争の敗者に



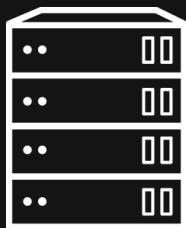
IT投資をイノベーションへシフトする

～自動化によって余力を確保し、DXを実現する～

EFFICIENCY

AGILITY

SPEED



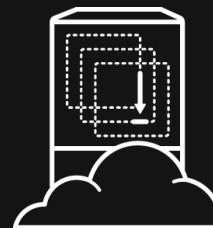
Optimize
existing IT



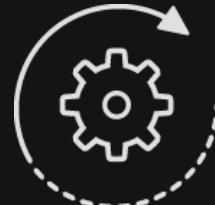
Integrate apps,
data,
& processes



Add & manage
hybrid cloud
infrastructure



Develop cloud-
native applications



Automate &
manage
IT

失敗する自動化の例

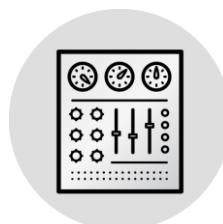


特定の人に依存した自動化

スクリプトやTeraTermマクロによる実装

個人に依存した秘伝のタレ化

属人化され、組織内の普及や
引き継ぎが困難

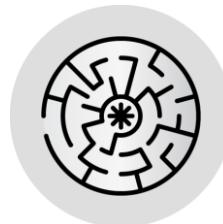


社外製品依存の自動化

他社が作り込んだ運用・自動化製品を活用

自社でのシステムの改修が難しい/ベンダー依存のため改修に時間と費用が掛かる

組織・システムの変化に対応できない



複雑すぎる自動化システム

複数の自動化ツール・方法を組み合わせて実装

学習コストが高く自動化手法の普及が課題、
それぞれのメンテナンスが手間に
局所最適な自動化

組織・システムの変化への対応にコストが掛かる

これまでの自動化の失敗理由

- 属人化された自動化は、組織に広まらず、人と共に衰退=手作業へ戻る
- 自社で変更できない仕組みは、組織・プロセス・文化の変化に追従できず、維持できない。
- 複雑なシステムは初期学習コストが大きくなり、属人化を加速させ、組織への浸透を阻害

特に、ネットワークで自動化が広まらない理由

- これまでの自動化はプログラムに近いものが主流だったが、
ネットワークエンジニアには、高いプログラミングスキルを求められていなかった
- 多くのユーザー企業はマルチベンダーでネットワークを形成しているが、
NW自動化ツールは、マルチベンダーへの対応が乏しい特定の
ネットワークベンダーに依存した製品が多くた
- 同じ値でもネットワークベンダーによって意味が異なる場合があるため、
マルチベンダー環境の自動化（特に特定の製品を担当するエンジニアが多い場合）には、
それぞれの製品知識をもったエンジニアが共通して作業することが望まれるが、
マルチベンダー環境に適した共通言語や環境がなかった

NW自動化に適した、マルチベンダーに対応し、NWエンジニアでも使いやすい自動化がなかった



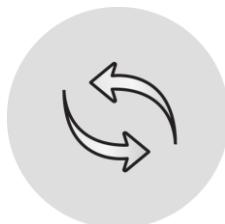
失敗しない自動化のための Ansible Automation

Ansible の特徴



Simple

ソースコードではなく、
技術者以外にも易しい、
統一された一つの言語/書式(YAML形式)で
手順書やパラメータシートのように
簡潔に記述できる



Powerful

様々な対象、ユースケースに利用でき、
幂等性を備えている



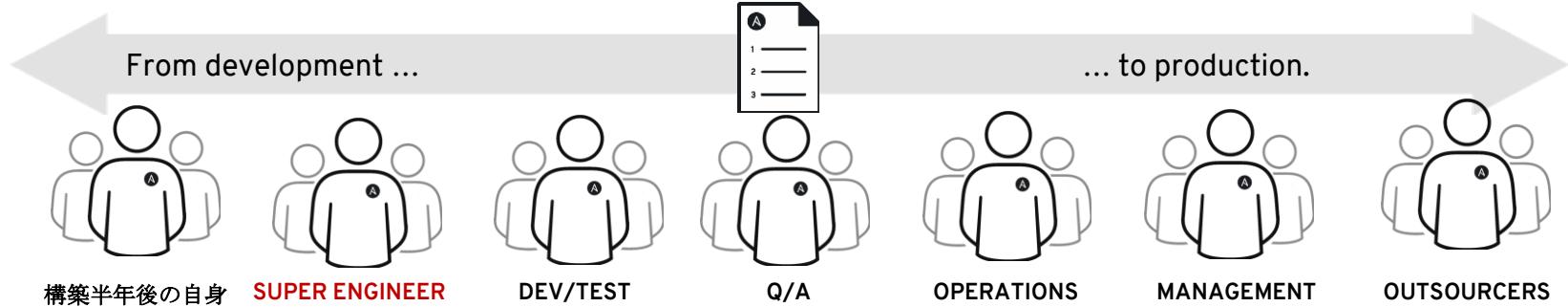
Agentless

管理対象へエージェントの
インストールが不要で、
対象に合わせた
基本的なプロトコルを使って、
すぐに利用できる

Simple

～組織のあらゆる人が理解しやすく、使いやすい～

Ansible Playbook → 実行可能な手順書プログラミング言語



AnsibleのSimpleな特徴

シンプルな自動化言語

: 誰でも読み書き可能な自動化のための標準言語

Not Programming

: プログラムではないため、特別なコーディングスキルは不要

実行可能な手順書

: 手順書と同様に上から下へひとつひとつ順番に実行

コンピュータにも

: 人だけではなくコンピュータにも易しい

簡単で効率的

: すぐに自動化を構築でき、すぐに生産性を向上



Powerful

～あらゆるユースケース、あらゆる対象に～

Use Case



設定管理

アプリケーションやOSなどの設定と設定手順の実行とバージョン管理



インスタンス等の プロビジョニング

サーバ仮想化やクラウド環境におけるインスタンスの作成



ポリシーチェックと 適応の自動化

予め設定した項目を定期的に確認し問題点を発見し修正する



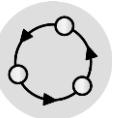
アプリケーションの デプロイ/管理

アプリケーションのインストールや状態の確認、更新などの処理



タスクフロー管理 (オーケストレーション)

複数の製品やアプリケーションへの処理を予め設定された順序に従って実行



CI/CD, DevOps

アプリケーションライフサイクル全般に渡るデプロイと管理により継続的デリバリーを実現

管理対象



OS



Storage



NW



サーバ仮想化



Cloud



Container



Application



DB



Monitoring

Red Hat

Ansibleの自動化対象例

| CLOUD | VIRT & CONTAINER | WINDOWS | NETWORK | DEVOPS | MONITORING |
|---|--|---|---|---|--|
| AWS Azure Digital Ocean Google OpenStack Rackspace +more | Docker VMware RHV OpenStack OpenShift +more | ACLs Files Packages IIS Regedits Shares Services Configs Users Domains +more | Arista A10 Cumulus Bigswitch Cisco Cumulus Dell F5 Juniper Palo Alto OpenSwitch +more | Jira GitHub Vagrant Jenkins Bamboo Atlassian Subversion Slack Hipchat +more | Dynatrace Airbrake BigPanda Datadog LogicMonitor Nagios New Relic PagerDuty Sensu StackDriver Zabbix +more |
| OPERATING SYSTEMS RHEL and Linux UNIX Windows +more | STORAGE NetApp Red Hat Storage Infinidat +more | | | | |

Agentless

～管理対象にインストール不要で、すぐに始められる～

Ansibleと管理対象間の通信プロトコル

SSH や WinRM、REST APIなど、各対象に最適な方式を使用します

エージェントを使わない設計のメリット

既存環境にも簡単に導入できる

管理対象に配布するエージェントのバージョンアップが不要

エージェントのセキュリティホールを考慮する必要がない

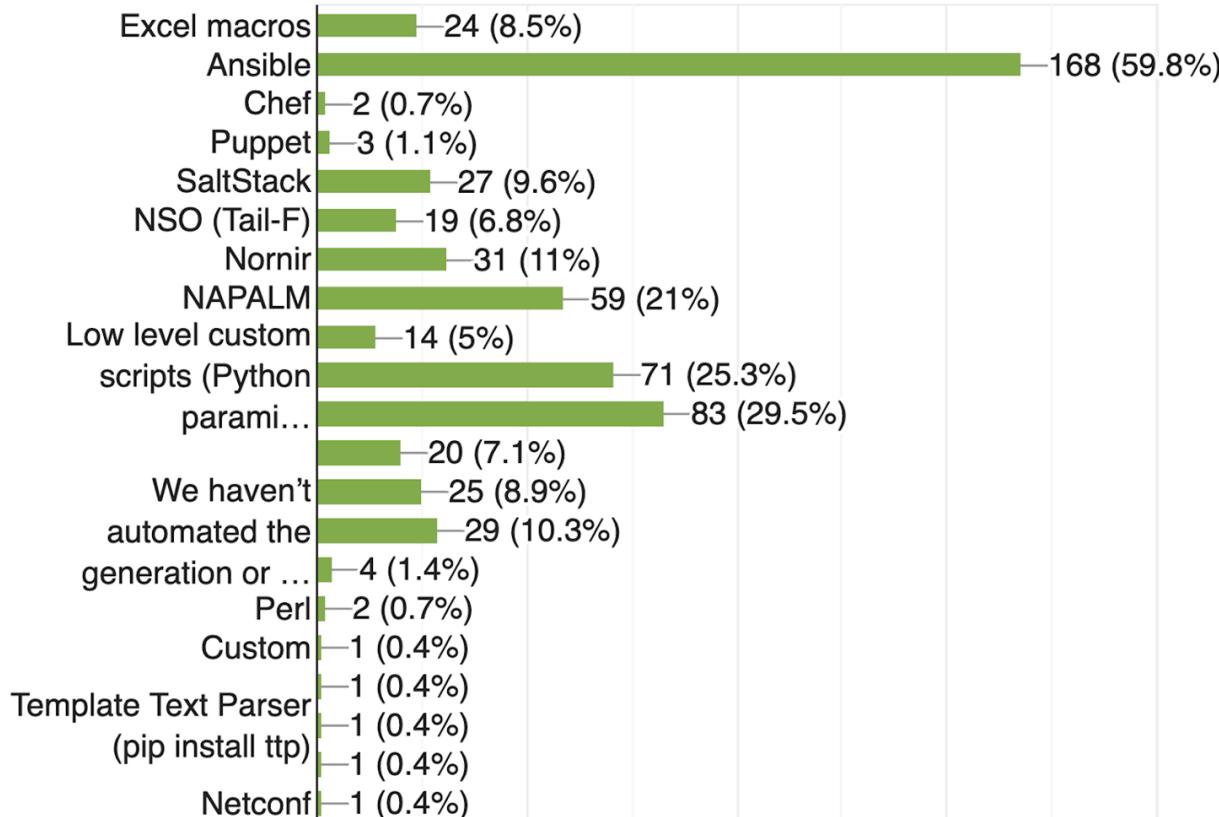
エージェントと他のプロセスの競合無し

エージェントのインストールができない対象の不安がない

ネットワーク自動化とAnsible Automation

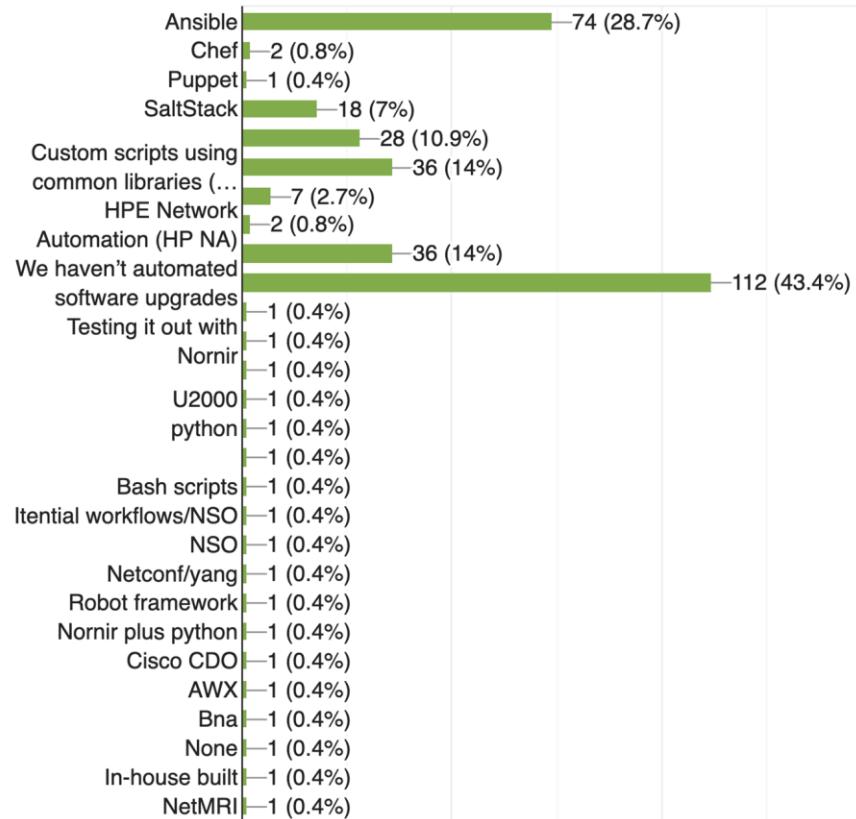
NW機器のコンフィグの生成 / 設定 / Deployに使用するツールは？

NetDevOps Survey 2019より



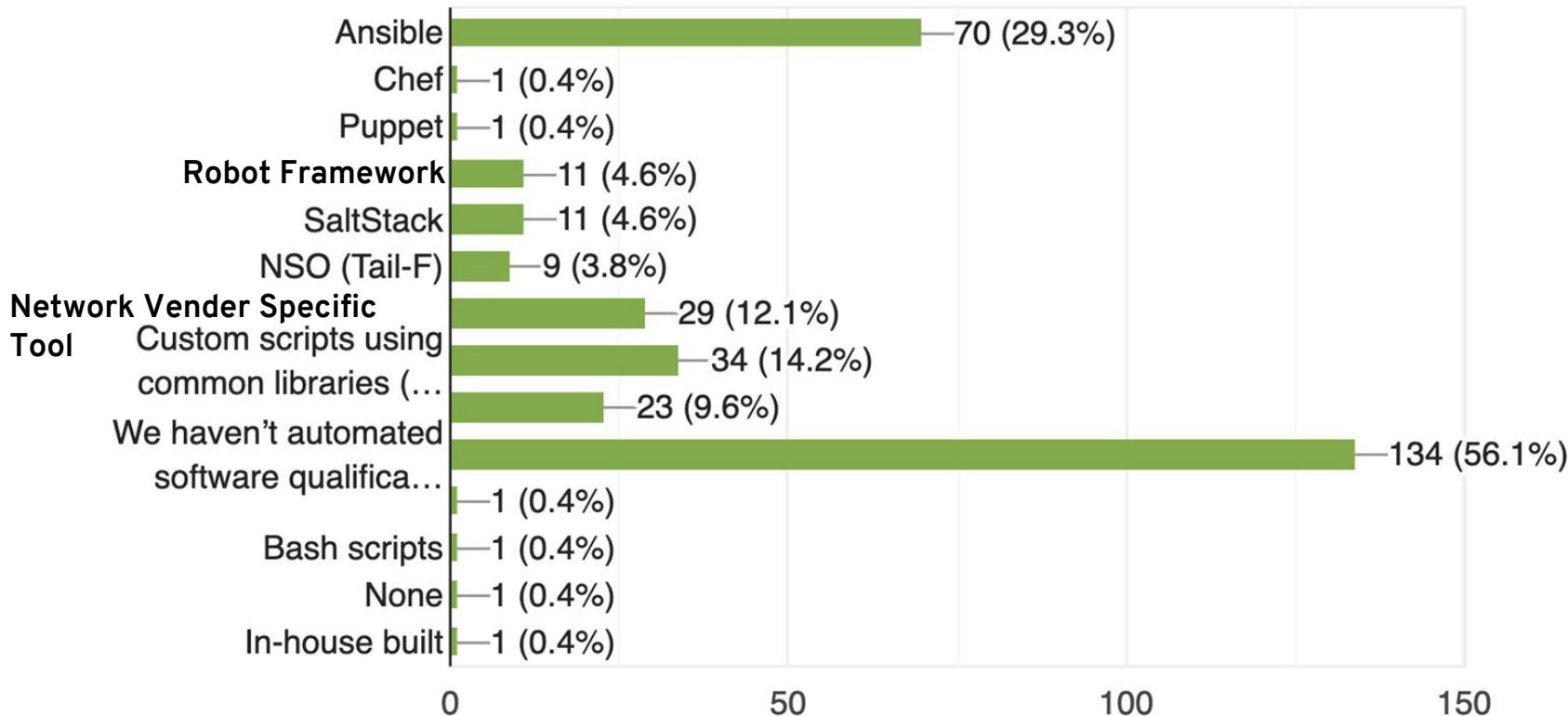
NW機器のソフトウェアアップグレードに使用する自動化ツールは？

NetDevOps Survey 2019より



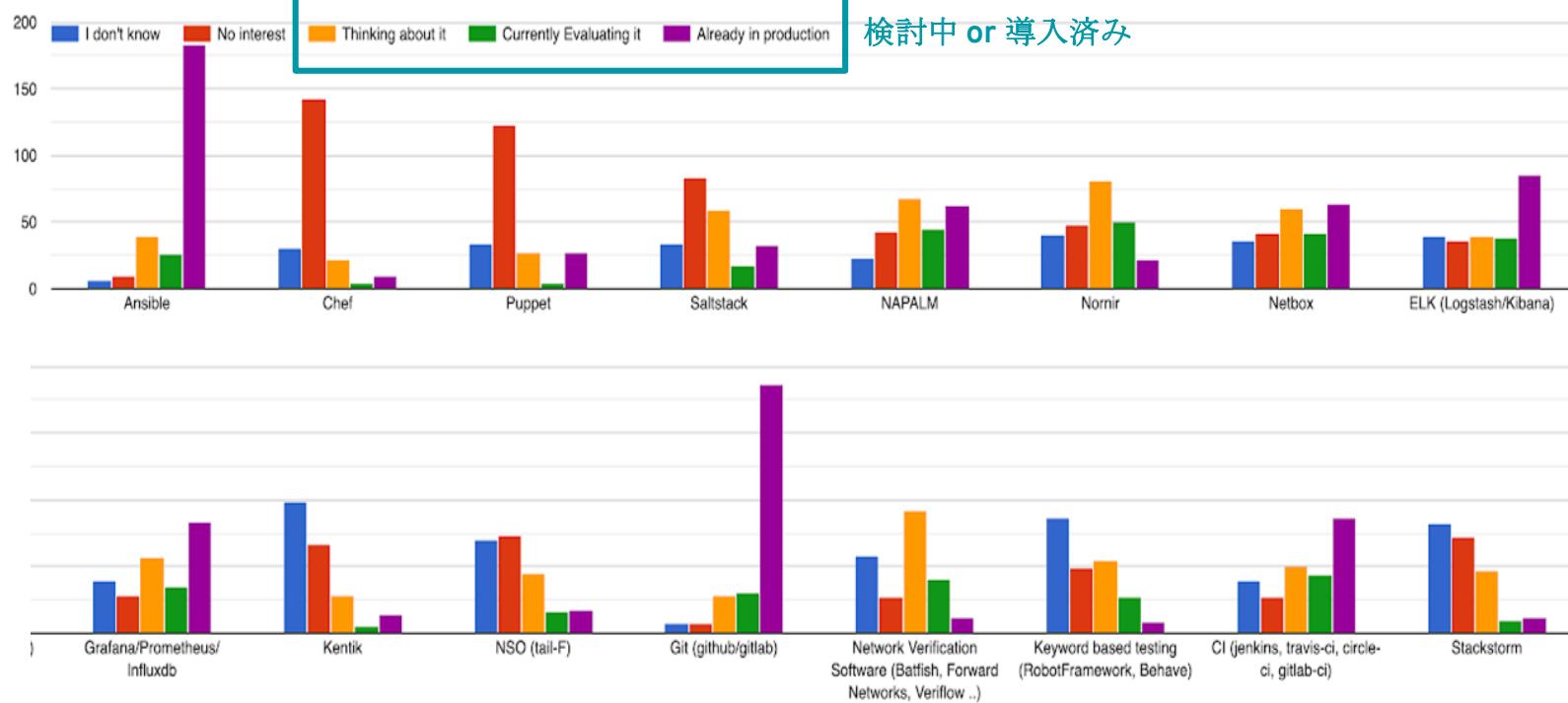
NW機器設定の品質チェック / 検証に使用するソフトウェアは？

NetDevOps Survey 2019より



興味がある / 導入しているツールはどれですか？

NetDevOps Survey 2019より



ネットワーク自動化の活用例

効果を出し易いネットワーク自動化の活用例



作業自動化

入力ミス削減

品質向上

複数人での指差し確認解消



スケジュール実行

夜間作業の自動化

夜間業務の縮小

働き方の改善

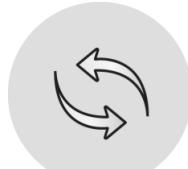


テスト自動化

テストの品質安定

同じ失敗の繰り返し防止

品質向上



バージョン管理

問題発覚時の対処の迅速化

Trial & Errorを簡単に

アップデートのハードル低減



サービス化

サーバアップデートに伴うLB操作等の付き添い作業から脱却

特定の人が任意のタイミングで
実行可能に



Ansibleにおけるネットワーク自動化の課題

ネットワーク自動化の課題

ITにおけるネットワークの役割

- 独立した端末で完結しないシステムにおいて端末間を繋ぐ役割

ネットワーク障害の特徴

- ネットワーク機器の設定変更の結果、疎通ができないくなる危険性を孕んでいる
- ネットワーク機器自体にもアクセスできなくなる場合を含む**
- ネットワーク障害は、影響範囲が非常に大きい

ネットワーク機器の設定変更・操作の危険性

- ネットワーク機器に対して疎通ができない=リモートからの操作・復旧ができない
 - データセンターに行き、NW機器にコンソール接続して対処する

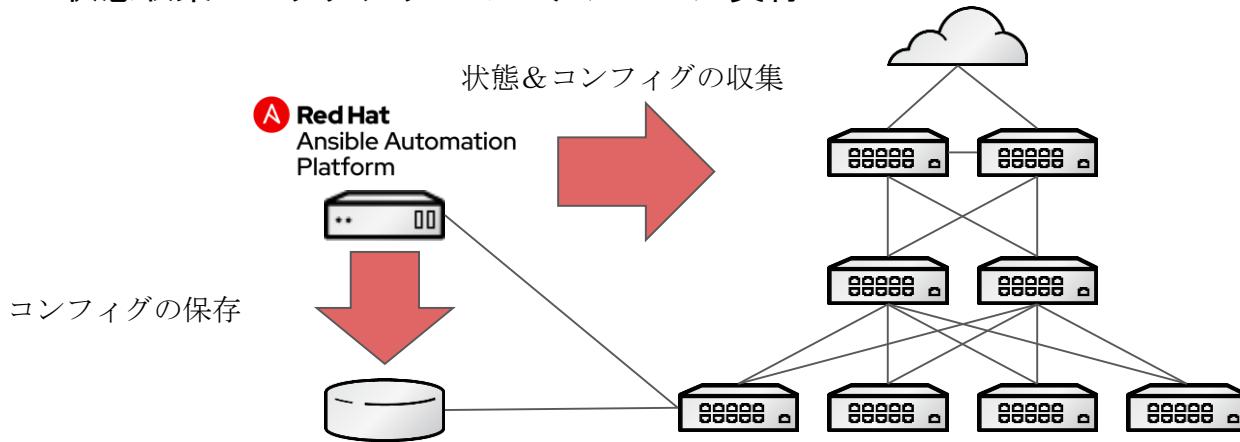


SmartCS
Intelligent console server

備考

NW機器のステータス / コンフィグの収集

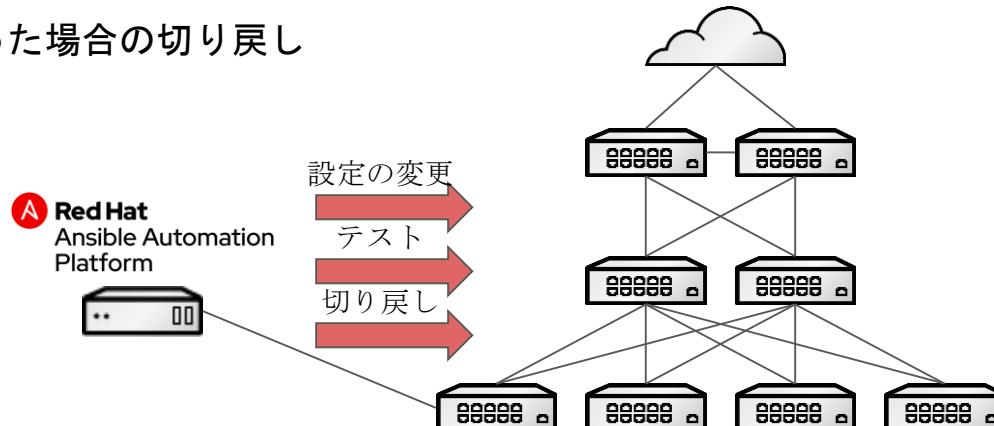
- 多様なネットワーク機器のコンフィグや状態の収集 / バックアップを一括化
 - 複数の機種 / 複数のベンダーにも対応できる
- 状態収集 / バックアップのスケジュール実行



- リスクなく始めることができ、もっとも始めやすい用途

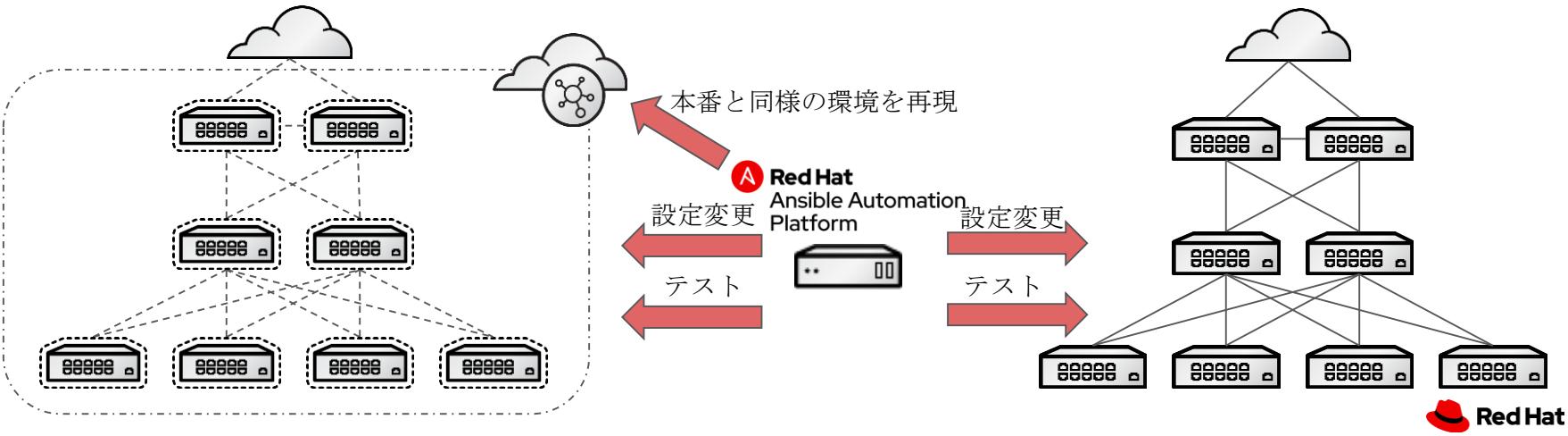
NW機器の設定変更と確認作業

- ネットワークの設定変更：経路変更 / IPの付け替え / 各種パラメータ変更
- 多段で構成されるネットワークに対する変更の順序実行
- 複数の機種 / 複数のベンダーの変更を一括で実施
- 設定変更後の疎通確認 / テスト
- 問題があった場合の切り戻し



仮想環境を利用したNW設定変更の事前チェック

- Cloudとバーチャルアプライアンスを利用して、本番環境と同じ構成のテスト環境を作成
- 設定変更を仮想環境に導入し、疎通確認
- 問題が無いことを確認してから、本番環境へ投入



コンソールサーバー SmartCSとは

- ・コンソールサーバ SmartCS の説明
- ・SmartCSのアクセス方法
- ・SmartCSのその他機能について

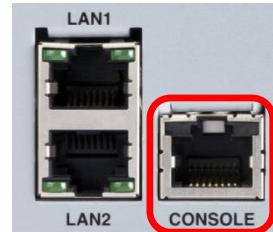
コンソールサーバーについて

SEIKO

コンソールポート とは

- NW機器のコンソールポートは
通常以下のような用途で使われます

① IP設定等の**初期構築**作業



RJ45



DB9

② 緊急時のオペレーション

LANインターフェース障害など、直接**IPリーチ出来ない場合**の「最後のアクセス手段」

とはいえ

- 監視対象装置全てのコンソールポートに対してそれぞれ監視端末を用意できない
- 緊急時に現地まですぐに行くことができない



コンソールサーバーの出番です！

コンソールサーバー **SmartCS**

- コンソールポートへのアクセスをリモートから行えるようにする装置
- DC内でToR等に設置され、監視対象装置に接続してNOCからの操作を可能にします。

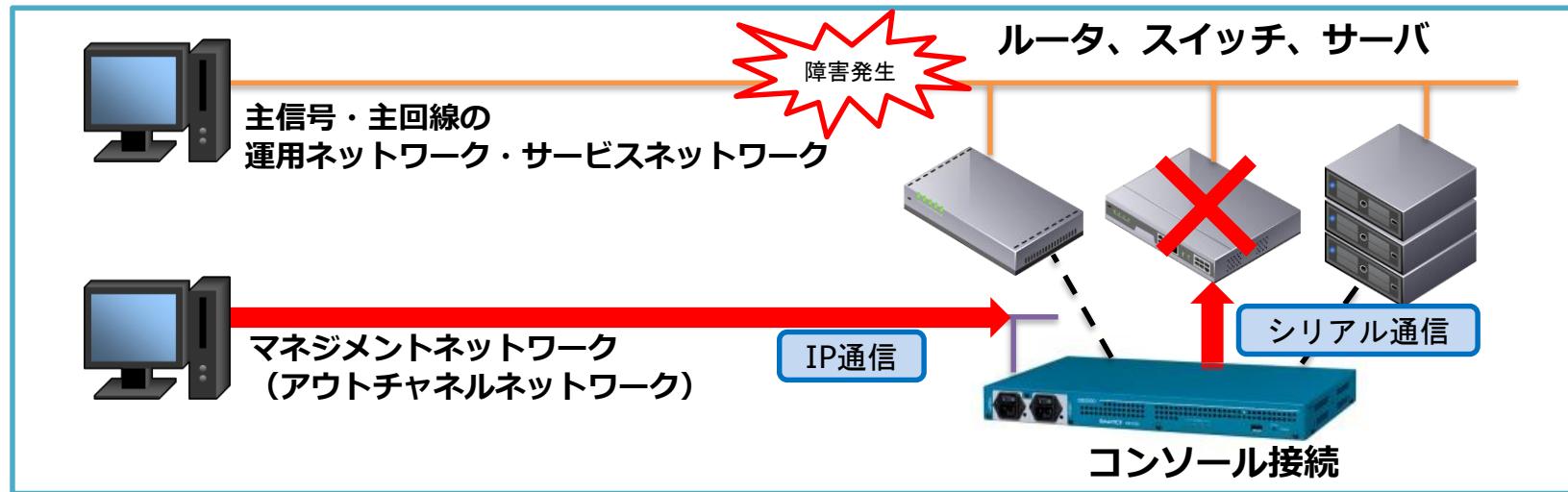


装置裏側

一見、多ポートスイッチのように見えますが、
1台で最大48のコンソールポートを集約可能

- ・通信キャリア様、ISP様など大規模NWを運用しているお客様を中心に、コンソールサーバー SmartCSシリーズは国内で高いシェアを確立
- ・INTEROP shownetのネットワーク構築においても10年以上の実績

コンソールサーバーを利用する場合のNW構成



主回線の運用ネットワークやターゲット機器へのアクセスがNGとなった場合、
コンソールサーバ経由で、監視対象装置にアクセスしてオペレーションを実行



最後のライフライン としてアクセス手段を提供

SmartCSには、下記2通りのアクセス方法があります

<ダイレクトモード>

SmartCSのシリアルポートに割り当てられたTCPポート番号を指定してアクセス

例：

tty1にアクセス → SmartCSの Port8301 にアクセス

<セレクトモード>

一旦SmartCS自身へアクセスし、ポートセレクトメニューからアクセス先を選択

例

SmartCSにSSH(22)でアクセス後、接続したいtty番号を選択

<ダイレクトモード>

- 各ttyに割り当てられているTCPポートを指定するだけで、ダイレクトに接続可能
- 接続するttyにどの機器が繋がっているかを別に管理し、把握しておく必要あり



【ダイレクトモードの接続イメージ】

```
$ telnet 192.168.0.1 8101
Host : "SmartCS-2250"
login from 10.208.36.40
Login: alaxala
Password:
AX2230>
:
:
```

telnetでtty1にダイレクトモードで接続する場合

【ダイレクトモードで使用するTCPポート】

| tty | TCPポート(telnet) | TCPポート(SSH) |
|-----|----------------|-------------|
| 1 | 8101 | 8301 |
| 2 | 8102 | 8302 |
| 47 | 8147 | 8347 |
| 48 | 8148 | 8348 |

<セレクトモード>

- ・ SmartCSの代表ポート(telnet:23/SSH:22)に接続し、リストから選択して接続
- ・ ラベル設定することで、各ttyにどの機器が繋がっているかをリストから把握可能
- ・ 開いているターミナルのまま、別のttyへ操作を切り替えることが可能(切替文字 Ctrl+XX入力)

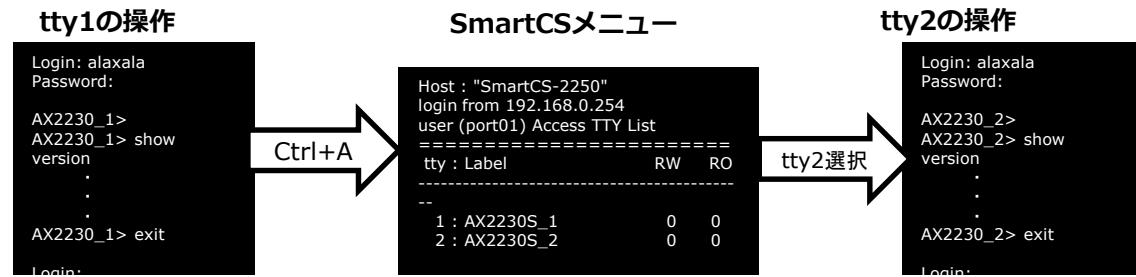


【セレクトモードの接続イメージ】

```
$ telnet 192.168.0.1
Login: port01
Password:
Host : "SmartCS-2250"
login from 192.168.0.254
user (port01) Access TTY List
=====
tty : Label          RW   RO
-----
1 : AX2230S_1        0    0
2 : AX2230S_2        0    0
.
.
.
```

telnetでセレクトモードで接続する場合

【操作対象ttyの切り替え】



コンソールにアクセスする、という用途以外にも、運用管理を支援する便利な機能があります。

<ログ保存/転送機能>

- 装置内部にログを保存するだけでなく、外部サーバへも出力が可能です



<シリアルポートへのアクセス制限>

- ユーザ毎にアクセス可能なシリアルポートを設定できます。



<ポートミラーリング>

- 監視対象機器への操作内容を複数のユーザで確認できます。



コンソールで入出力されるログは以下のような種類があります。

①装置が自発的に出力するログ

- コンソールにしか出力されないログ
- シヤットダウン / 再起動発生時 のログ
- 障害発生直前のエラーログ

②オペレーションログ

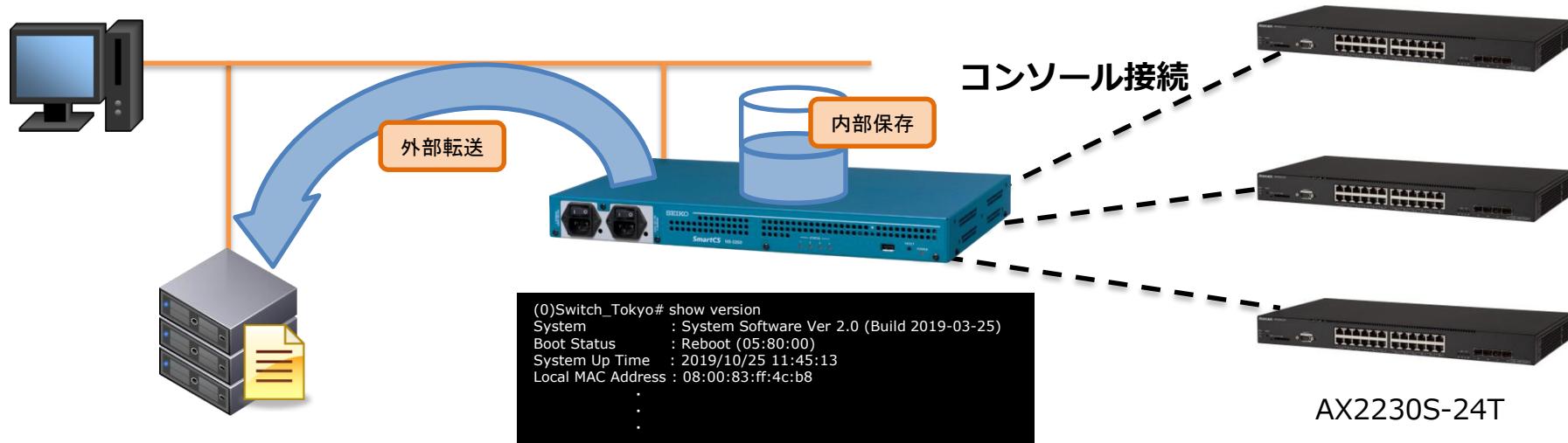
- コンソールサーバを経由して操作したオペレーションログ

■ ログ保存機能

- ・ コンソールに入出力されるログを装置内部に保存します。
 - SmartCS本体に、シリアルポートごとに3Mまで保存可能(最大8Mまで拡張可能)
 - 設定不要で自動的にオペレーションログを装置内部に保存します。
→ ログの保存忘れや誤って削除する事を防ぐことができます。

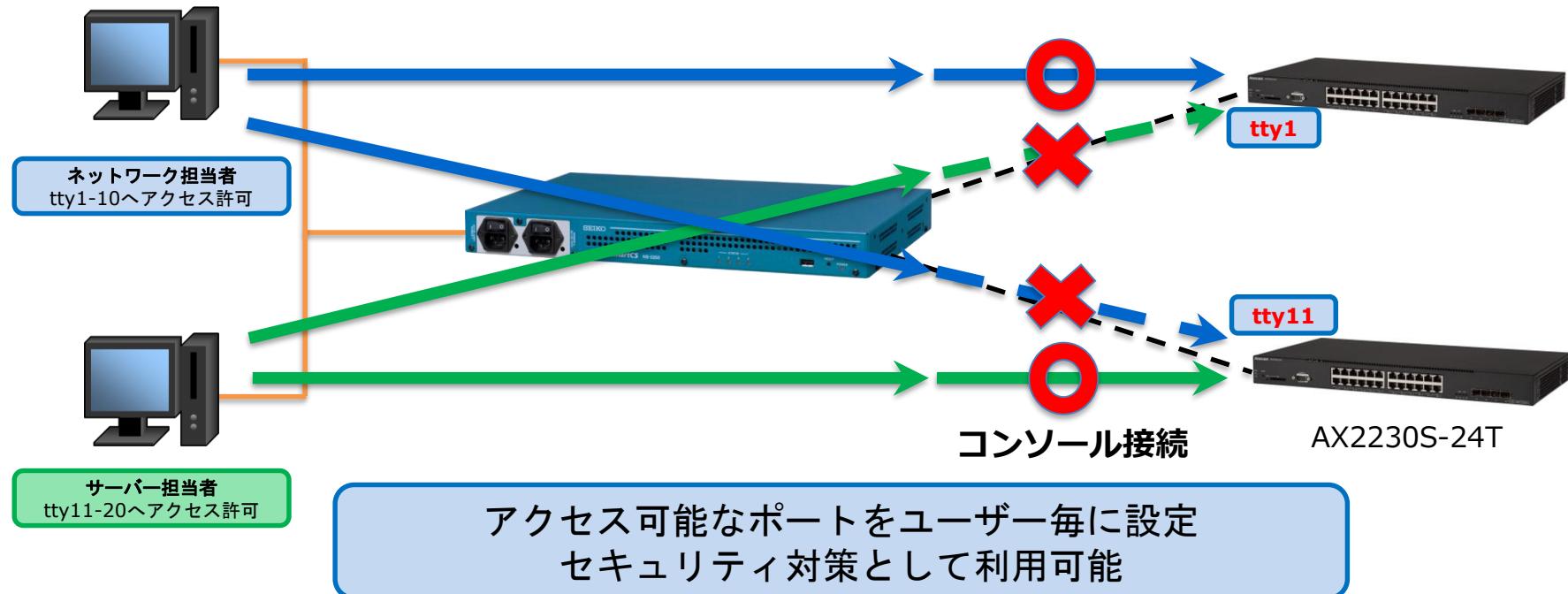
■ ログ転送機能

- ・ 装置内部に保存する以外にも、外部サーバへの転送が可能です
- ・ FTP / Mail
 - 送信時間 / ログの保存領域の閾値に応じた送信タイミングを指定可能
- ・ Syslog / NFS
 - ログが出力されたタイミングでログを送信



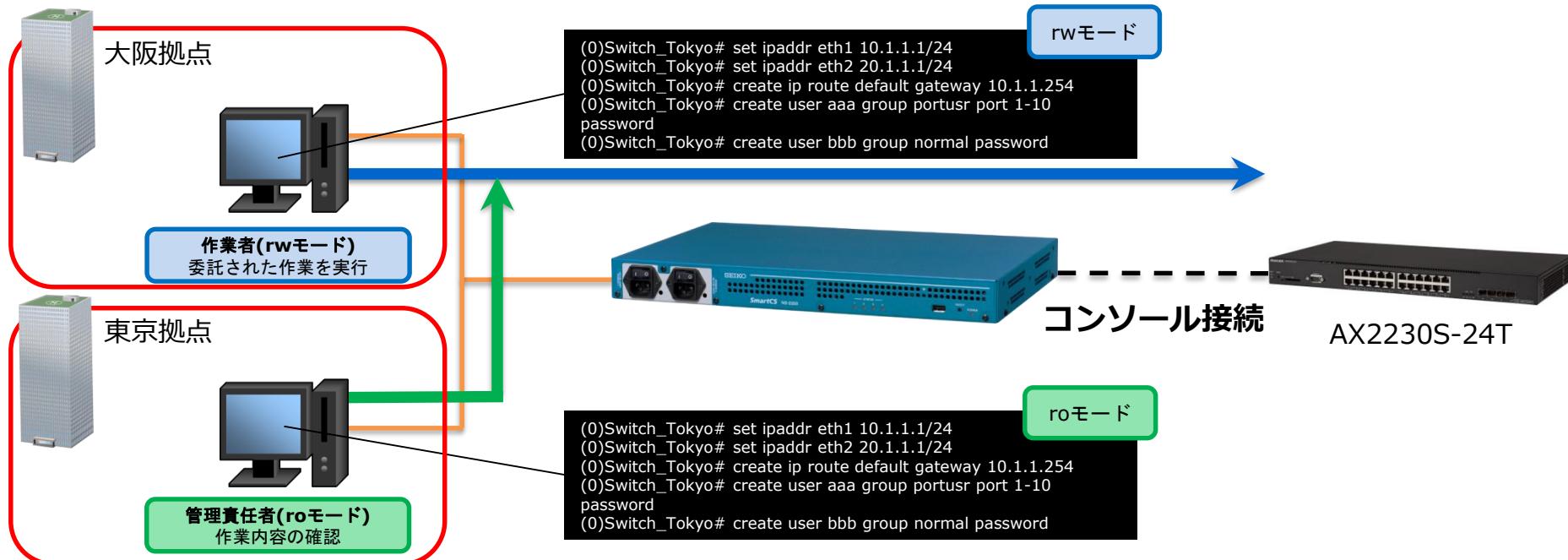
ユーザー毎のアクセス制限

- ミスオペレーションによる操作対象間違い
- 権限の無い機器への不正アクセス などを防止します。



特定のシリアルポートへの操作内容を複数のユーザで確認する事ができます。

- rw権限：送受信可能なモードで、監視しつつ制御も可能
- ro権限：受信のみ可能なモードで、監視のみ可能



演習内容

演習1 ハンズオン環境の確認

- 1.1 演習環境の確認

演習2 SmartCSの基本動作(手動編)

- 2.1 SmartCSを介してALAXALA装置へコンソールアクセスする
- 2.2 SmartCSを介したALAXALA装置へコンソールアクセスをミラーリングする
- 2.3 SmartCSを介したシリアルレセッション情報を確認する

演習3 Ansible×SmartCS×Alaxalaの連携演習(基礎編)

- 3.1 ALAXALA装置にSmartCS経由で初期設定を行う
- 3.2 ALAXALA装置に追加設定を行う
- 3.3 ALAXALA装置の設定情報を取得する
- 3.4 ALAXALA装置の設定情報をSmartCS経由で取得する

演習4 Ansible×SmartCS×Alaxalaの連携演習(応用編)

- 4.1 オペレーションミスからの復旧自動化
- 4.2 通信障害からの復旧自動化
- 4.3 ファームウェアアップデートの自動化
- 4.4 初期化の自動化

演習5 Ansible Tower×SmartCS×Alaxalaの連携演習

- 5.1 Ansible Towerの説明
- 5.2 定期的な死活監視と障害発生時的情報取得を自動化

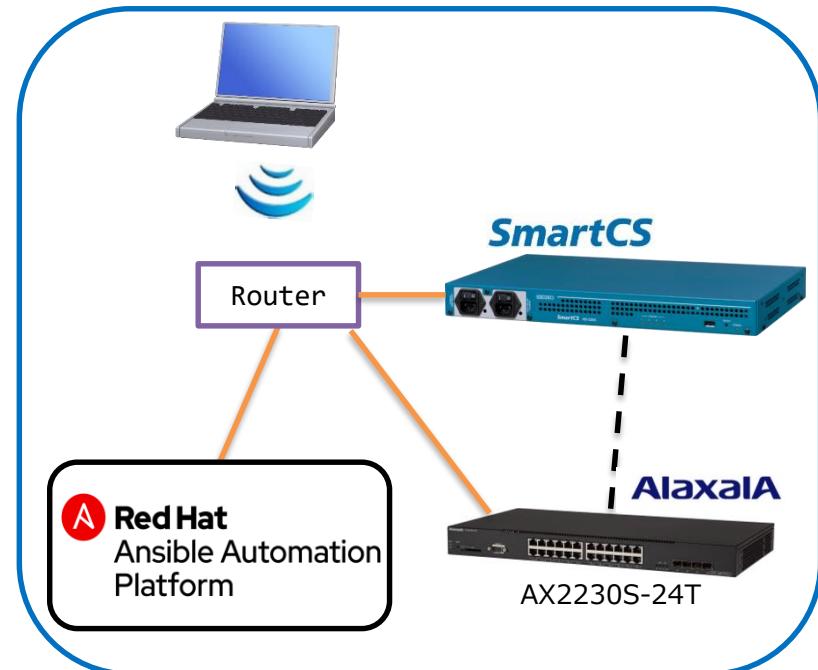
【ハンズオン 演習1】

ハンズオン環境の確認

環境概要

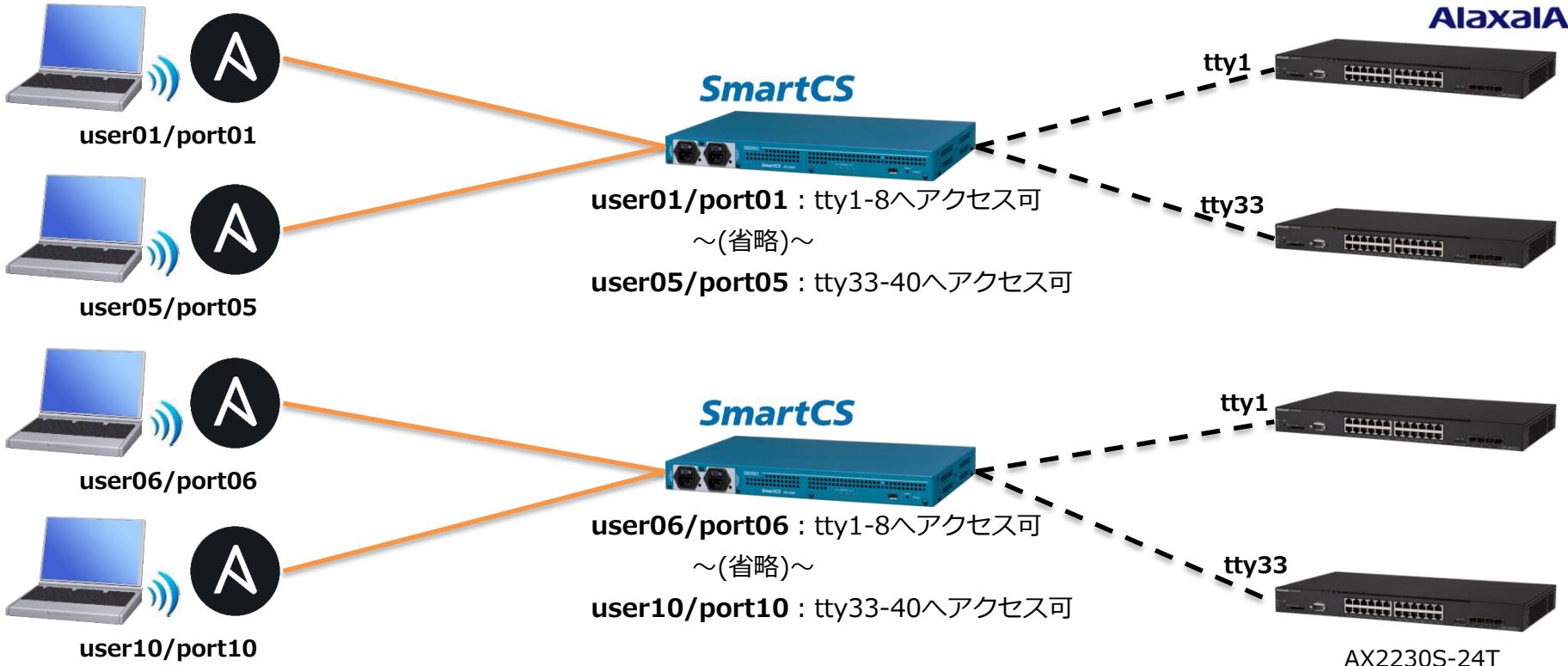
- 参加者1名に対して1つの環境を用意しています。
(SmartCSは5名で1台を使用いただきます。)
- SmartCSのtty1,9,17,25,33に、AX-2230が接続されています。
- 演習2では、SmartCSにSSHでアクセスいただき、
SmartCS経由でのコンソール操作を体感していただきます。
- 演習3以降では、AnsibleノードにSSHでアクセスいただき、
AX2230S-24Tの操作を実施していただきます。

構成



参加者ごとに割り当てられているアドレス/ユーザ/パスワードを利用します。(手順書の演習1を参照)

SmartCSを介してアクセスする場合、各ユーザ(userXX/portXX)には、アクセス可能なttyが設定されています。



■ハンズオン手順書(Github)

<https://github.com/ssol-smartcs/ansible-handson>

■Q&A

bit.ly/smarts_0131

【ハンズオン 演習2】

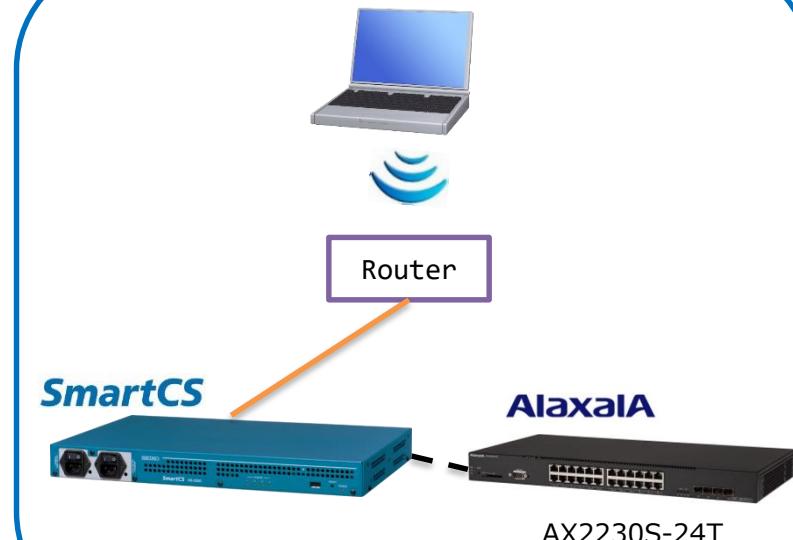
コンソールサーバー SmartCSの基本動作(手動編)

概要

演習2では、SmartCSの基本的な使い方を確認します。

- SmartCSを経由して、AX2230Sのコンソールへアクセス【演習2.1】
- SmartCSを経由したセッションをミラーリング【演習2.2】
- SmartCS上でシリアルセッション情報の確認【演習2.3】

構成

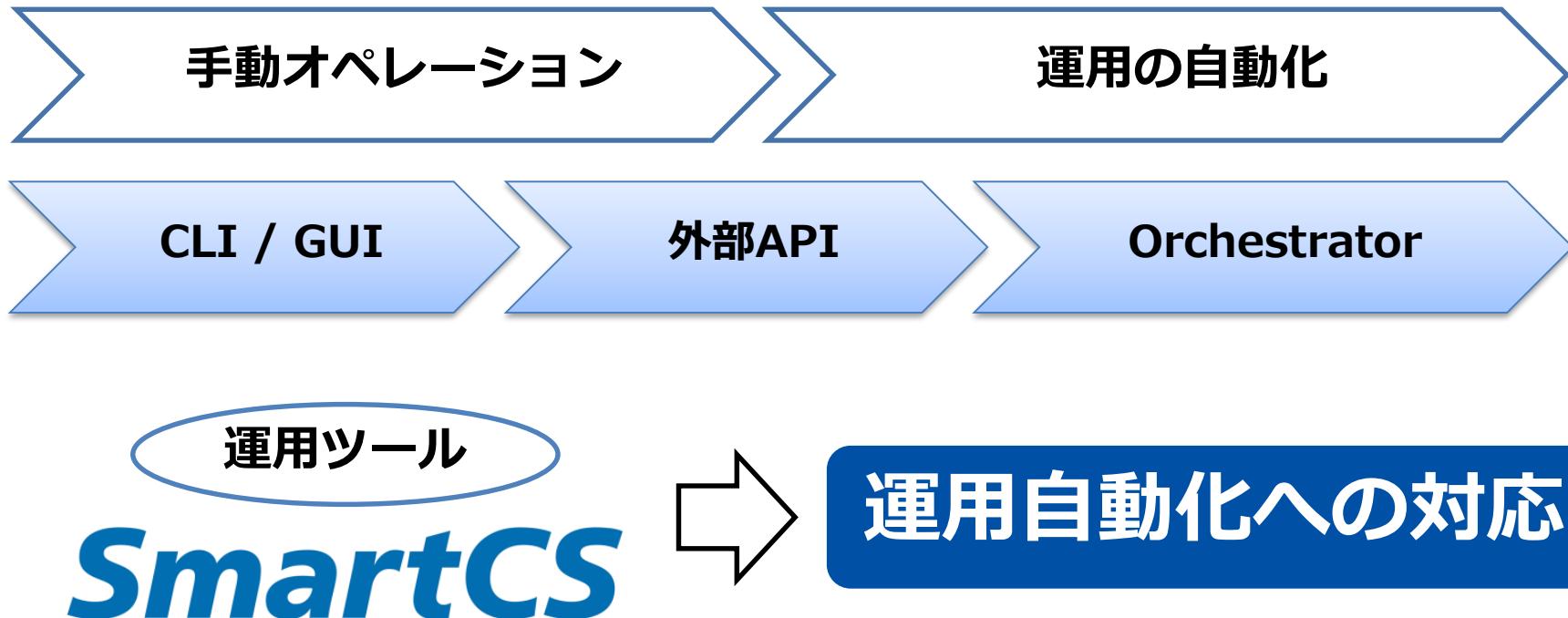


【座学】

Ansible × SmartCS について

背景

ネットワーク運用環境の変化



SmartCS

従来の運用自動化における課題

- リモートからの設定変更により、機器へリモートから接続できなくなる可能性
- リモートからのバージョンアップ作業(失敗)により、通信できなくなる可能性
- データセンタへ駆けつけ、機器のコンソールへPCを直結し復旧しなければならない

SmartCSによる解決



SmartCSがあれば、リモートからコンソールをオペレーション可能！

従来のAnsibleにおける課題

- Ansibleリーチできない状態の機器の操作が難しい（初期設定段階）
- Ansibleモジュールが無い機器の操作にはあまり適していない（ベンダー依存）

SmartCSによる解決



ConsoleからCLI操作可能な機器は、Ansibleによるオペレーション自動化の対象に！

Ansible をさらにパワフルに



従来は ネットワーク機器・サーバ機器 等のターゲットが
IPリーチ (Ansibleリーチ) 可能になっている状態でないと
各モジュールによるオペレーションが実行出来なかつた

Ansible をさらにパワフルに



IPリーチャビリティのないターゲットも運用自動化の対象に
+
Ansibleモジュールのないターゲットも対象に

SmartCS用のAnsibleモジュールをAnsible Engineにインストールすることで
以下のオペレーションをAnsible経由で行うことが可能となります。

- SmartCSのシリアルポートに接続されている監視対象機器にして、
Ansibleから文字列(対象機器のコマンド)を送受信することが可能
→ smartcs_tty_command

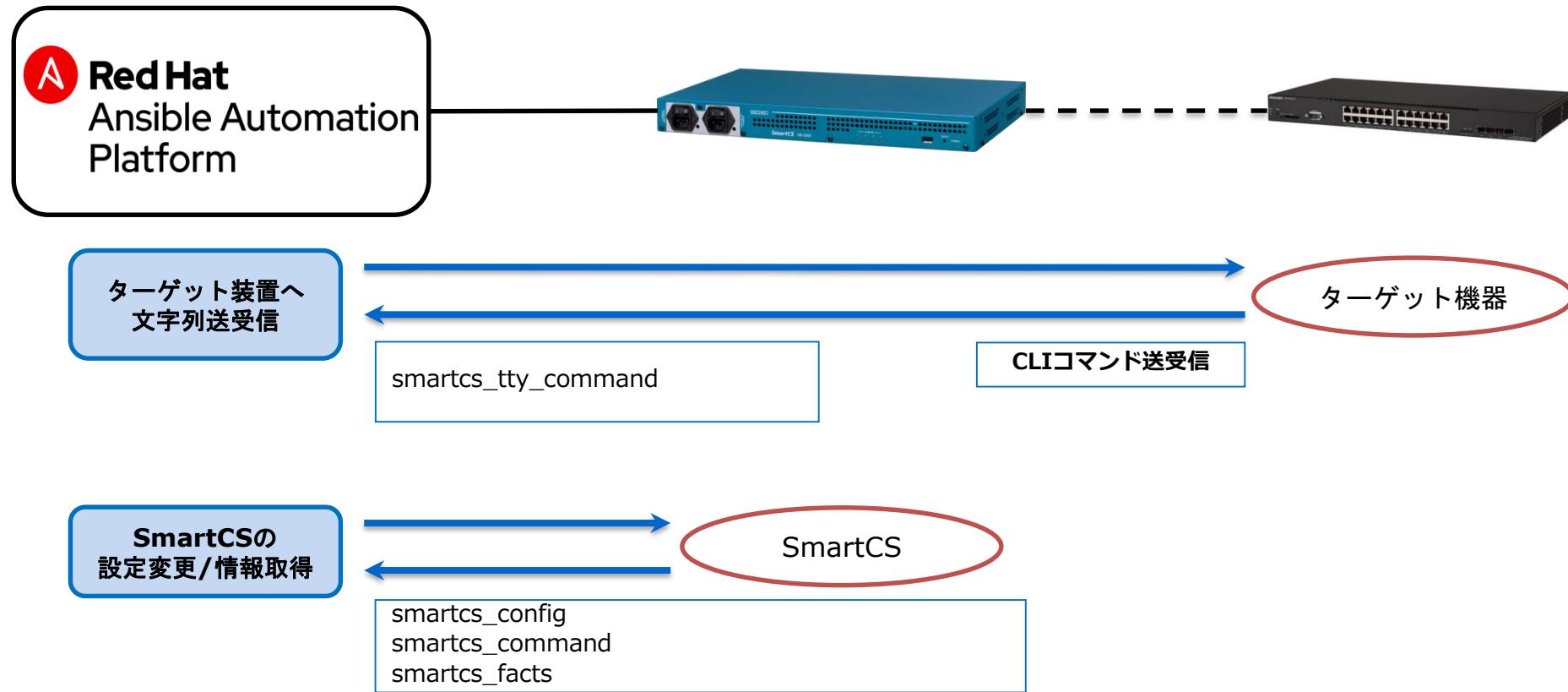
本日のハンズオン
内容

- SmartCS自身の設定変更、および情報取得が可能
→ smartcs_facts, smartcs_command, smartcs_config

リモートからコンソール経由での設定変更、バージョンアップなどの作業を
Ansibleで自動化することができるようになります。

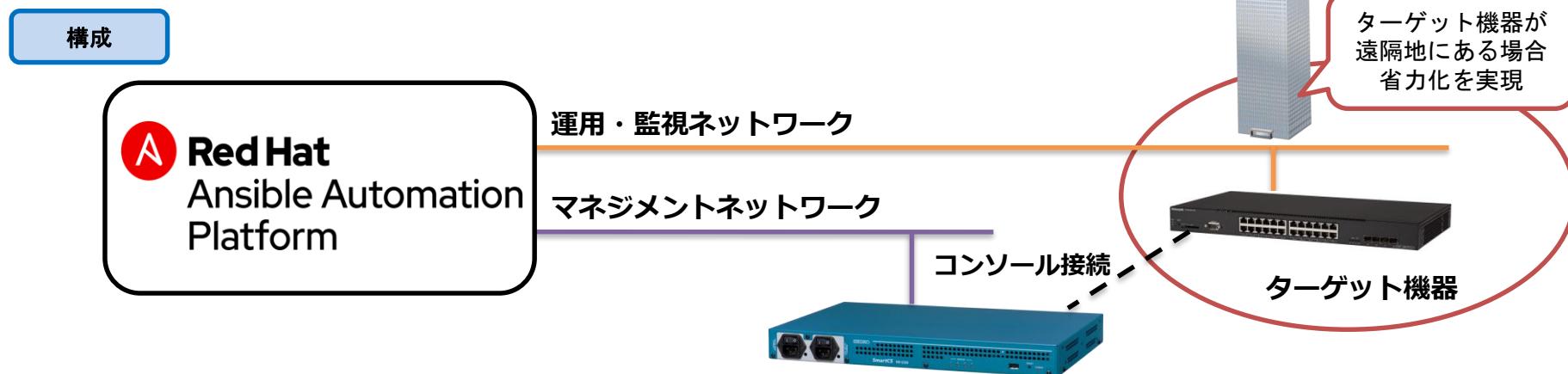
Ansible×SmartCSで実現できること

SEIKO



初期構築（設置時・交換時）

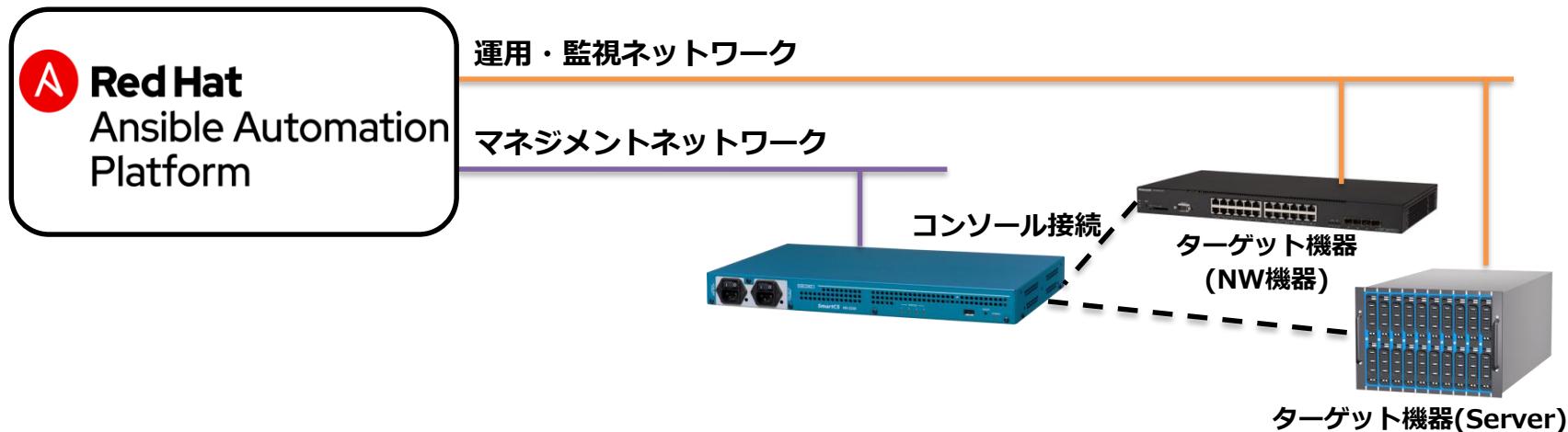
NW機器の設置時や交換時に、コンソール経由でIP設定/ユーザ作成などの初期構築を行います。最低限の初期設定をコンソール経由で実行し、Ansibleリーチ可能な状態になってからはベンダーごとに用意しているモジュールを利用して追加の設定等を行います。



コンソールからのバージョンアップ作業

NW機器やサーバ（Hypervisorのホスト） 機器の設定変更やバージョンアップ作業を、
コンソール経由で安全に行います。

構成



SmartCS用のAnsibleモジュール “smartcs_tty_command” では、
下記の様なパラメータをplaybook内で指定して文字列の送受信を行います。

演習で使用

| パラメータ名 | 設定値 | 概要 |
|------------------------------|-------------------------------------|--|
| tty | 1~48 | 文字列を送信するSmartCSのシリアルポート番号です。1-10の様にリスト形式でも指定可能です。 |
| cmd_timeout | 1~7200 | 文字列を送信してから、recvcharの受信待ちがタイムアウトするまでの時間です。 |
| nl | <u>cr</u> / <u>lf</u> / <u>crlf</u> | 送信文字列として「__NL__」を指定した際に送信する改行コードです。 |
| sendchar (src) | | <p>指定したttyに送信する文字列のリストです。リストの上から順番に送信します。 改行コードや制御文字も送信可能です。</p> <p>【オプション】 __WAIT__:sec 上述のcmd_timeoutを送信文字列毎に指定するオプションです。</p> <p>【オプション】 __NOWAIT__ recvcharで指定した文字列を待たずに、すぐに次の文字列を送信します。</p> <p>【オプション】 __NOWAIT__:sec recvcharで指定した文字列を待たずに、指定した時間経過後に次の文字列を送信します。</p> |
| recvchar (recvchar_regex) | | 文字列を送信後、受信を期待する文字列(プロンプト等)のリストです。 リスト内のいずれかを受信すると、次の文字列を送信します。 期待する文字列は正規表現での記述も可能です。 |

SmartCS用のAnsibleモジュール “smartcs_tty_command” では、
下記の様なパラメータをplaybook内で指定して、送信文字列と受信文字列を区別しやすい
返り値(stdout_lines_custom)とすることができます。

演習で使用

| パラメータ名 | 設定値 | 概要 |
|---------------------------------|----------|--|
| custom_response | boolean値 | stdout、stdout_linesに加えて、sendcharオプションで指定した文字列ごとに、 送信文字列(execute_command)と受信文字列(response)が分かれたフォーマットで 出力するかどうかを指定します。 |
| custom_response_delete_nl | boolean値 | custom_responseの出力内容について、改行のみの行を削除するかどうかを指定します。 |
| custom_response_delete_lastline | boolean値 | custom_responseの出力内容について、responseの最終行を削除するかどうかを指定します。 recvcharオプションで指定した文字列のうち、受信した文字列(主にターゲット装置のプロンプト)が responseに含まれないようにすることができます。 |

参考情報(v1.0)

| パラメータ名 | 設定値 | 概要 |
|--------------------------|--------------------------|--|
| error_detect_on_sendchar | <u>cancel</u> / exec | 文字列を送信後、エラーが発生した場合に、次の文字列を送信するかどうかを指定します。 |
| error_detect_on_module | <u>ok</u> / failed | 文字列を送信後、エラーが発生した場合に、ansibleコマンド(ansible-playbookコマンド)の実行結果をokとするかfailedとするかを指定します。 |
| error_recvchar_regex | | 文字列を送信後、エラーと判定したい受信文字列を正規表現で記述したリストです。 |
| ttycmd_debug | <u>off</u> / on / detail | 文字列送受信処理が終了した後、デバッグ情報を表示します。 |

参考情報(v1.1)

| パラメータ名 | 設定値 | 概要 |
|----------------------------------|------|---|
| initial_prompt | | initial_prompt_check_cmd送信後に受信を期待する文字列です。(「Login:」など) |
| initial_prompt_check_cmd | | 文字列送信の前にコンソールの状態を確認するためのコマンドを指定します。(改行送信など) |
| initial_prompt_check_cmd_timeout | 1~30 | initial_prompt_check_cmd送信後に受信文字列をチェックするまでの時間を指定します。 |
| escape_cmd | | initial_promptを受信できなかった場合に送信するコマンドを指定します。(「exit」など) |
| escape_cmd_timeout | 1~30 | escape_cmd送信後に受信文字列をチェックするまでの時間を指定します。 |
| escape_cmd_retry | 0~8 | escape_cmd送信後にinitial_promptを受信できなかった場合に、initial_prompt_check_cmdの送信リトライ回数を指定します。 |

playbook例とパラメータ概要

SEIKO

```
---  
- name: Login AX-2230  
hosts: smartcs  
gather_facts: no  
  
tasks:  
- name: Login AX-2230  
  smartcs_tty_command:  
    tty : 1  
    nl : cr  
    cmd_timeout : 5  
    recvchar :  
      - "#"  
      - ">"  
      - "(config)#"  
      : 省略  
    sendchar :  
      - operator  
      - enable  
      - configure  
      : 省略
```

```
vars :  
- ansible_command_timeout : 60  
- ansible_connection : network_cli  
- ansible_network_os : smartcs  
- ansible_user: user01  
- ansible_password: secret01
```

■recvchar (recvchar_regex)

- ・コマンド送信後に期待する文字列(プロンプト等)を複数指定します。
- ・指定したいいずれかの文字列を受信したら、sendcharで指定された次の文字列を送信します。

■sendchar

- ・指定した tty に送信する文字列を指定します。
- ・リストの上から順番に送信します。

■vars

・ansible_command_timeout

→ コンソール経由でコマンドを実行する為、通常のモジュールよりも処理時間がかかります。その為、タイムアウト値を延長する必要があります。
(default:10s)

・ansible_connection

→ [network_cli](#) を指定します。

・ansible_network_os

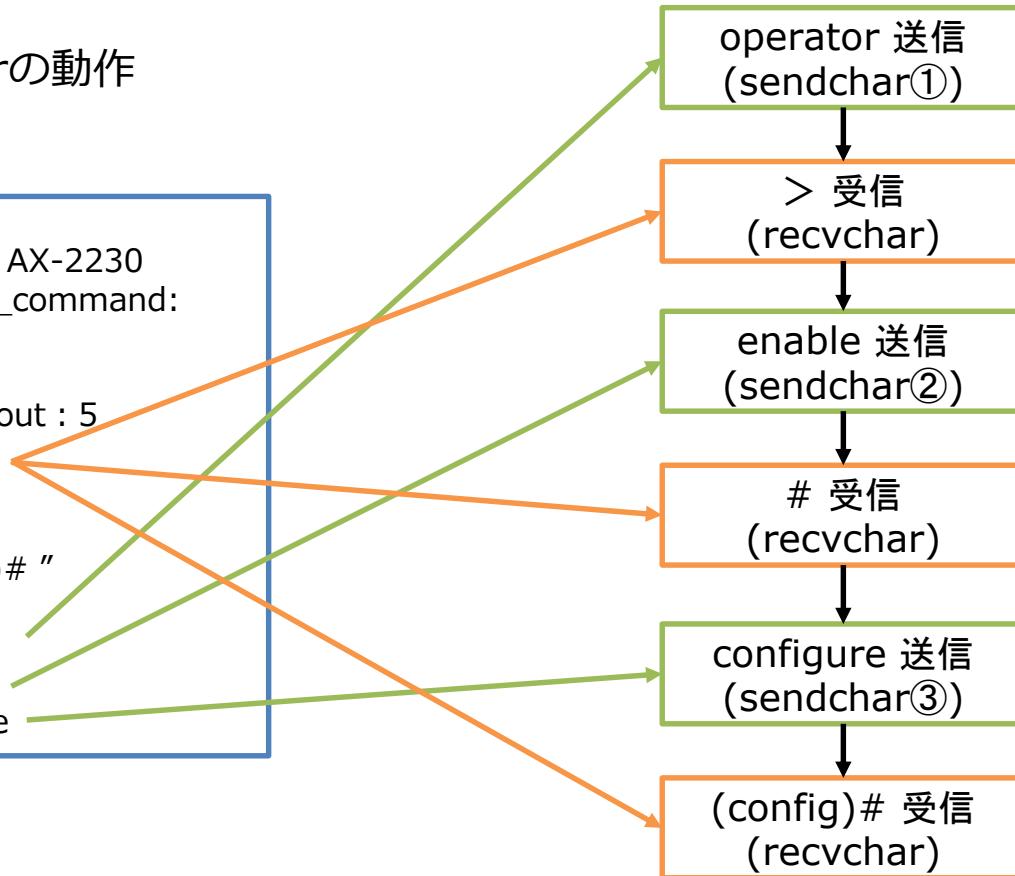
→ [smartcs](#) を指定します。

・ansible_user , ansible_password

→ SmartCSにログインする為の[拡張ユーザ\(extusr\)](#)のログイン情報を指定します。

sendcharとrecvcharの動作

```
tasks:  
- name: Login AX-2230  
  smartcs_tty_command:  
    tty : 1  
    nl : cr  
    cmd_timeout : 5  
    recvchar :  
      - "#"  
      - ">"  
      - "(config)#"  
    sendchar :  
      - operator  
      - enable  
      - configure
```



| 名前 | 説明 | 契機 | タイプ |
|--------------|-----------------------------|----------------|-----|
| stdout | コマンドの実行結果 | | リスト |
| stdout_lines | コマンド実行結果を 送信文字列毎に分割したリスト | コマンドの実行に成功した場合 | リスト |

stdout出力例

```
"stdout": [
    "show version\n\nDate 2020/01/24 13:39:29 UTC\n\nModel:
AX2230S-24T\n\nS/W: OS-LT4 Ver. 2.9 (Build:04)\n\nH/W: AX-2230-
24T-B [CA022B24T000S0000C7S013:0]\n\nAX2230>",
],
```

The diagram shows the output of a 'show version' command. An orange arrow labeled '送信したsendchar' points from the first line 'show version' to the green box labeled '受信したrecvchar'. A green arrow labeled '受信したrecvchar' points from the green box to the second line 'Date 2020/01/24 13:39:29 UTC'.

stdout_lines出力例

```
"stdout_lines": [
    "show version",
    "",
    "",
    "",
    "Date 2020/01/24 13:39:29 UTC",
    "",
    "Model: AX2230S-24T",
    "",
    "S/W: OS-LT4 Ver. 2.9 (Build:04)",
    "",
    "H/W: AX-2230-24T-B [CA022B24T000S0000C7S013:0]",
    "",
    "AX2230>",
],
```

The diagram shows the output of a 'show version' command. An orange arrow labeled '送信したsendchar' points from the first line 'show version' to the green box labeled '受信したrecvchar'. A green arrow labeled '受信したrecvchar' points from the green box to the second line 'Date 2020/01/24 13:39:29 UTC'. A large green bracket on the right side of the code block is labeled 'コマンド(sendchar) 実行結果'.

| 名前 | 説明 | 契機 | タイプ [°] |
|---------------------|---|---------------------------------------|------------------|
| stdout_lines_custom | コンソールの送受信文字列について、送信文字列(execute_command)、受信文字列(response)を区別した形式のリスト。 | custom_response設定が有効、かつコマンドの実行に成功した場合 | リスト |

オプション設定値

- custom_response : **on**
⇒stdout_lines_customでの出力有効
- custom_response_delete_nl : **on**
⇒コマンド実行結果の行間を削除
- custom_response_delete_lastline : off
⇒最終行(プロンプト等)は削除しない

出力例

```
"stdout_lines_custom": [
  {
    "execute_command": "show version",
    "response": [
      "Date 2020/01/24 13:55:13 UTC",
      "Model: AX2230S-24T",
      "S/W: OS-LT4 Ver. 2.9 (Build:04)",
      "H/W: AX-2230-24T-B [CA022B24T000S0000C7S013:0]",
      "AX2230>"
    ]
  }
]
```

【ハンズオン 演習3】

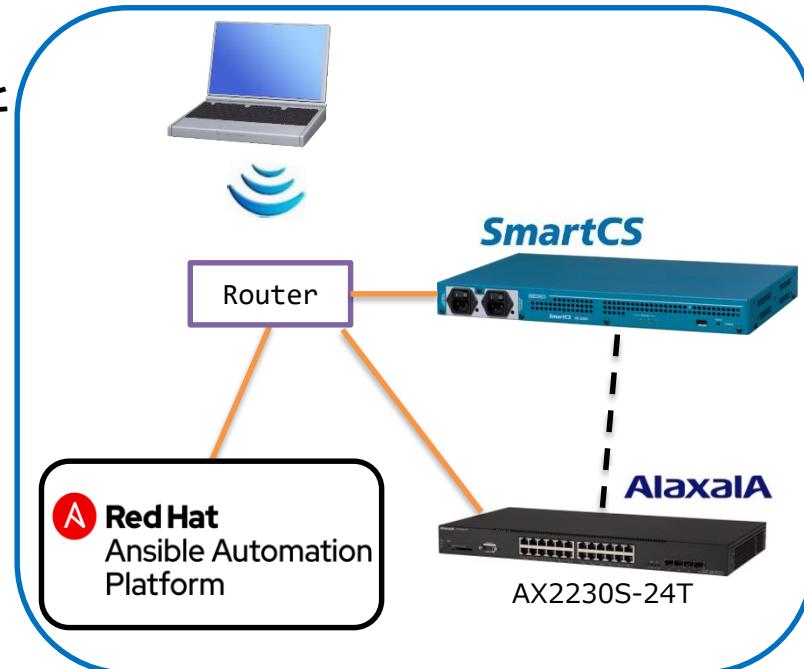
Ansible×SmartCS×Alaxalaの連携演習(基礎編)

概要

演習3では、

- ・SmartCSを経由して設定投入/情報取得を行い、AnsibleとSmartCSの連携方法について理解を深めていただきます。
 - ・Ansibleを使ったネットワーク機器へのオペレーションについて理解を深めていただきます。
- SmartCSモジュールでAX2230Sへの初期設定投入(SmartCS経由)【演習3.1】
- ALAXALA NW機器のご紹介
- AXモジュールでAX2230Sへの追加設定投入【演習3.2】
- AXモジュールでAX2230Sから設定情報取得【演習3.3】
- AXモジュールでAX2230Sから設定情報取得(SmartCS経由)【演習3.4】

構成



【ALAXALA NW機器紹介】

アラクサラネットワークス社

Alaxala



ソリューション・製品ラインナップ

The Guaranteed Network
いちばん近くで、もっと先へ。

Solutions

Security

AX-Security-Controller

サイバー攻撃自動防御
端末トレース



Visualization

AX-Collector

AX-Sensor

ネットワーク可視化・分析
異常検知



Operation

AX-Network-Manager

AX module for Ansible



運用自動化・効率化

Products

Layer 3 Switches

Box

AX3800Sシリーズ

IPv4/IPv6
1G/10G/40G
電源冗長



AX3600Sシリーズ

IPv4/IPv6
1G/10G/40G/100G
電源冗長



High Density

AXscala 3900Sシリーズ

QSFP28 (100GbE) 32port



Coming Soon

SFP28 (25GbE) 48port +
QSFP28 (100GbE) 8port



Chassis

AX4600Sシリーズ



AX8300Sシリーズ



AX8600Sシリーズ



Router

AX8600R
シリーズ



AX620R
シリーズ



Layer 2 Switches

AXprimoM210シリーズ

1U/100M/1G
ファンレス
/PoE/PoE+



AX2130Sシリーズ

10/100M/1G
ファンレス/認証
/PoE/PoE+



AX260Aシリーズ

10/100M/1G
大容量ホワイトリスト



AX2530Sシリーズ

1G/10G PoE/PoE+
認証/ファンレス
耐環境



DX時代のスマートなネットワーク

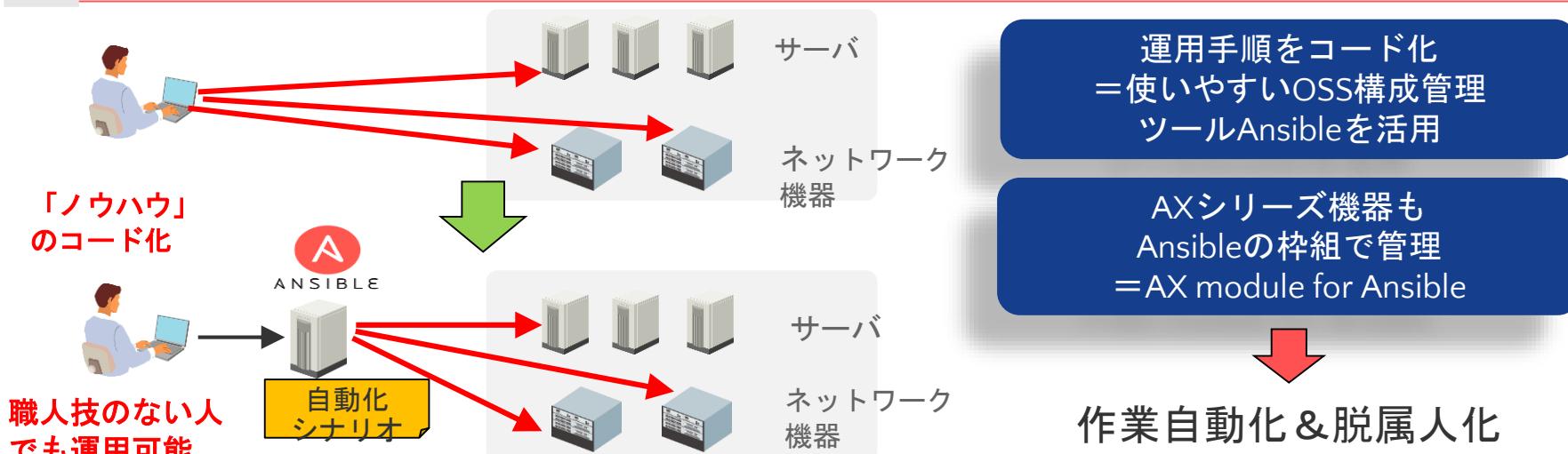


ネットワークの運用自動化

Ansibleを用いてネットワーク運用を自動化することで、
運用コスト削減＆働き方改革に貢献

課題

- ◆ ネットワーク運用管理は作業量が膨大
- ◆ ネットワークの大規模化/仮想化/複雑化に伴い、確認すべき内容も複雑化
→ 作業の長時間化、運用作業の属人化



アラクサラ機器をAnsibleから制御するための
AX modules for Ansibleをご提供中

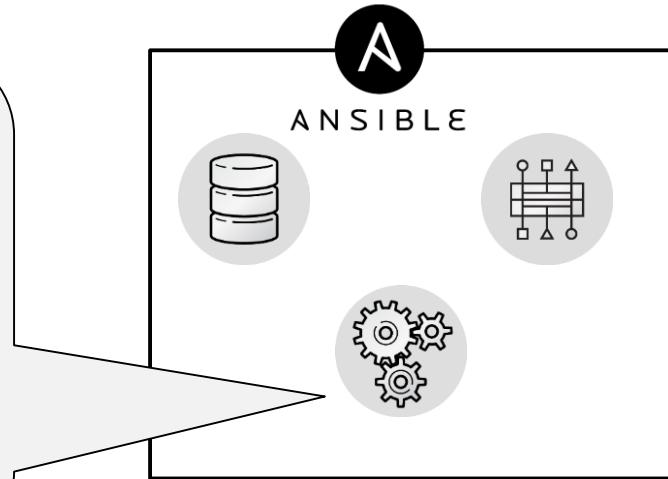


アラクサラ機器で「運用自動化」が！

AX modules for Ansibleの構成

- ◆ ax_facts
機器情報の収集
- ◆ ax_command
運用コマンドを実行
- ◆ ax_config
コンフィグを投入

AX modules for Ansible



Engine

◆ACL設定

セキュリティ情勢の移り変わりや外部クラウドサービス(例:Office365)のIPアドレス変更に伴う定期的なACLの設定変更

◆VLAN設定

新規事業・サービス・アプリケーション立ち上げでプラットフォーム準備のためや、回線契約に伴うユーザの追加・削除

◆OSアップデート

脆弱性対応したバージョンのファームウェアへアップデート

◆ NTP/Syslog/SNMP Trap等のサーバ登録変更

サーバの老朽化に伴うリプレースやセキュリティアプライアンス・NMSの追加

◆ 障害発生時の情報採取

ポート障害や期待しない動作をしている際に装置の情報を採取

◆ パスワード更新・SNMPコミュニティストリング変更

セキュリティポリシーとして定期的なパスワードや機器へのアクセス情報を更新



挙げたものは一例です。

基本的にはコマンドラインで実現できる事は自動化が可能です。



ドキュメント更新しないと・・・

問い合わせ来てたな・・・

NW更改時期近いから情報収集しないと・・・

アラートがあがってる・・・、だと・・・



新しい技術の学習についてまで
なかなか手を回せないのが現状ではないでしょうか

ネットワーク運用自動化への取り組みの一環として、
Ansibleの学習コストを低減し、
運用自動化のお試し・導入をより手軽にする施策
を展開中！



運用に沿った**実践的なPlaybook**を作成し、
インターネットで公開

実践的playbookをGitHubに公開中！

The Guaranteed Network
いちばん近くで、もっと先へ。

パスワード変更



障害情報採取



サーバ追加設定



GitHub

<https://github.com/alaxalanetworks/>



ACL設定



VLAN設定

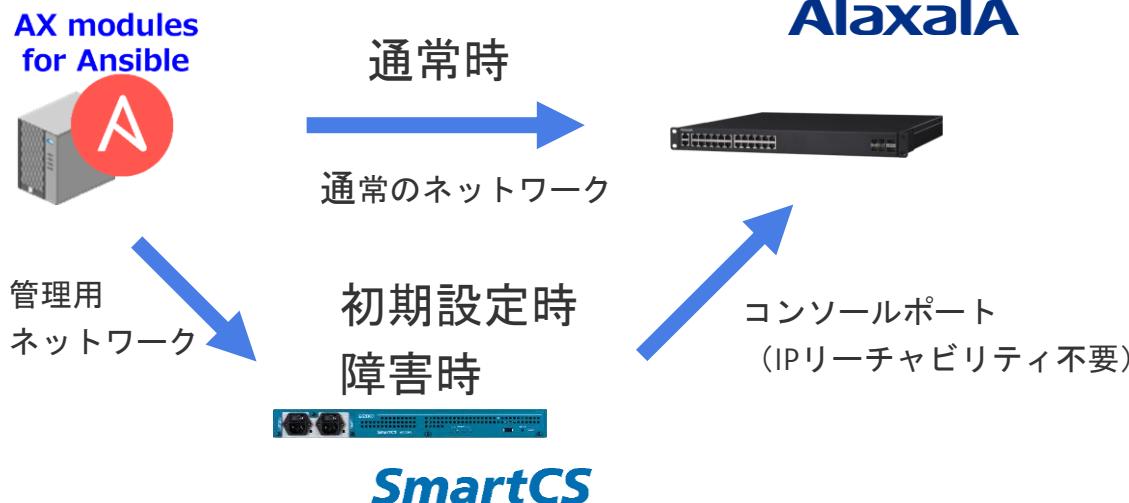


OSアップデー
ト

さらに、適用範囲を拡大| 本日のハンズオンのテーマ

The Guaranteed Network
いちばん近くで、もっと先へ。

SmartCS経由でAX modules for Ansibleを動かすことで、初期設定や障害時の情報収集の自動化を実現



ユースケース 1

初期状態の機器に管理用IPやSSHの設定
(多台数時に効果あり)

ユースケース 2

通常のネットワークで障害が発生した場合でも、機器の自動制御を継続
(クリティカルなネットワークに効果あり)

ボックス型L2スイッチ
AX2200Sシリーズ



高速性とハイコストパフォーマンスを両立した、モバイル環境のフロアスイッチに最適なギガビットレイヤ2スイッチ。

主な適用位置

フロアスイッチ

■ SmartCS経由で、Ansibleの他ベンダーモジュールを利用可能

※演習3.4以降で本機能を利用した演習を実施いたします。

smartcs_tty_commandのみを利用

- ・ベンダー製のAnsibleモジュールがないターゲットにアクセスする場合
- ・smartcs_tty_commandモジュールを使って全ての制御を完結させたい場合
→ 1つのPlaybookで全ての処理を行いたい場合

【課題】 Playbookの作成が難しい

→ 実施したい操作のコンソール経由の入出力情報(特にrecvchar)が必要、署等性担保×

smartcs_tty_command と 他ベンダーモジュールを連携して利用

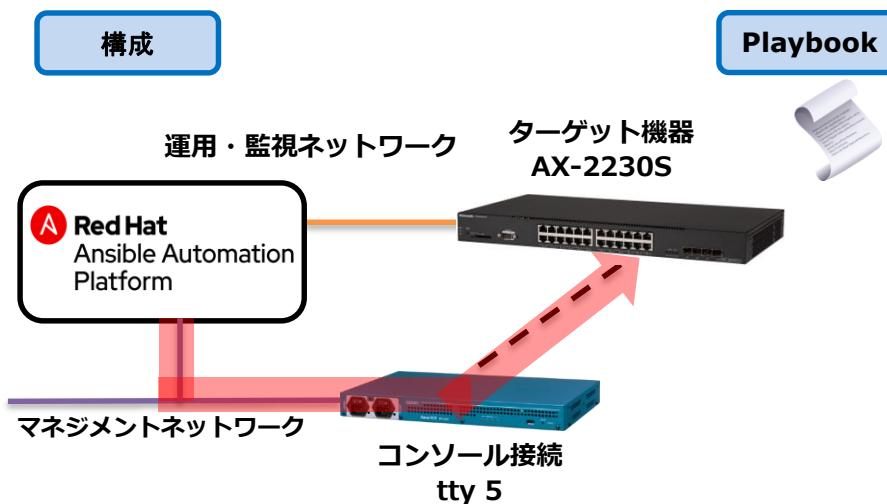
- ・ベンダー製モジュールを利用してターゲット機器の制御を行いたい場合

【メリット】 Playbookの作成が比較的容易

→ ベンダー製モジュールを利用したPlaybookがそのまま流用出来る、署等性担保○

他ベンダーモジュールの利用

- SmartCS経由で、Ansibleの他ベンダーモジュールを利用する場合の接続構成とPlaybookイメージ



Playbook

```

---
- hosts: smartcs
  gather_facts: no
  tasks:
    - name: "Task ax_command"
      ax_command:
        commands:
          - show version
          - show system
          - show ip interface
          - show vlan

  vars:
    - ansible_connection: network_cli
    - ansible_user: port01
    - ansible_password: secret01
    - ansible_ssh_port: 9305
    - ansible_network_os: ax
    - ansible_become: yes
    - ansible_become_method: enable
    - ansible_become_password: secret2230
    - ansible_command_timeout: 60
  
```

接続先はSmartCS

他社ベンダのモジュールを指定

コネクションプラグインは network_cli を指定

ログインID パスワード には SmartCS の ポートユーザを指定

SSHの接続ポートを SmartCS の sshxpt用のポートに変更

OS種別は 使用する他社ベンダを指定

■他ベンダーモジュールの実行（Playbook構成例）



Playbook ①

Module: smartcs_tty_command

ユーザ：拡張ユーザ

ポート：SSHポート（22）



Playbook ②

Module: 他社ベンダーモジュール

ユーザ：ポートユーザ

ポート：sshxpt ポート（93xx）



Playbook ③

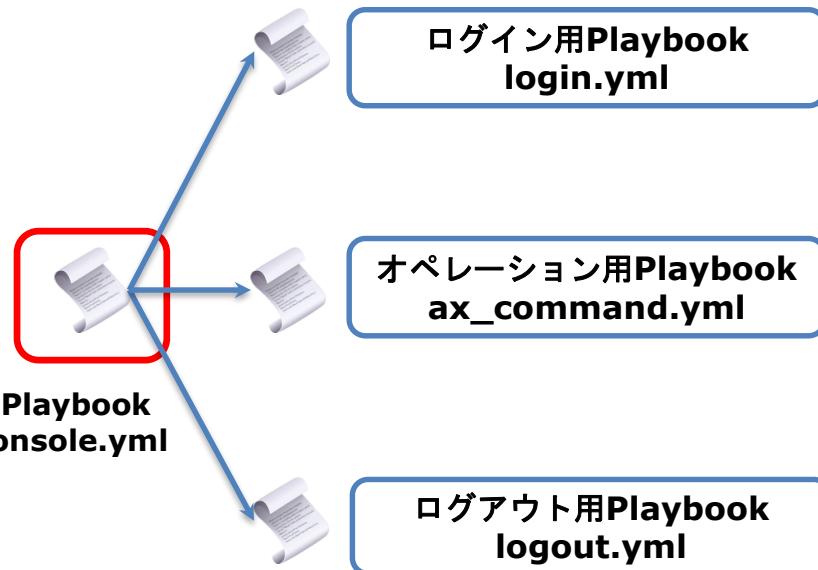
Module: smartcs_tty_command

ユーザ：拡張ユーザ

ポート：SSHポート（22）



■他ベンダーモジュールの実行（Playbook構成例）



制御用Playbook例 **ax_console.yml**
※実際に実行するPlaybook

```
---
```

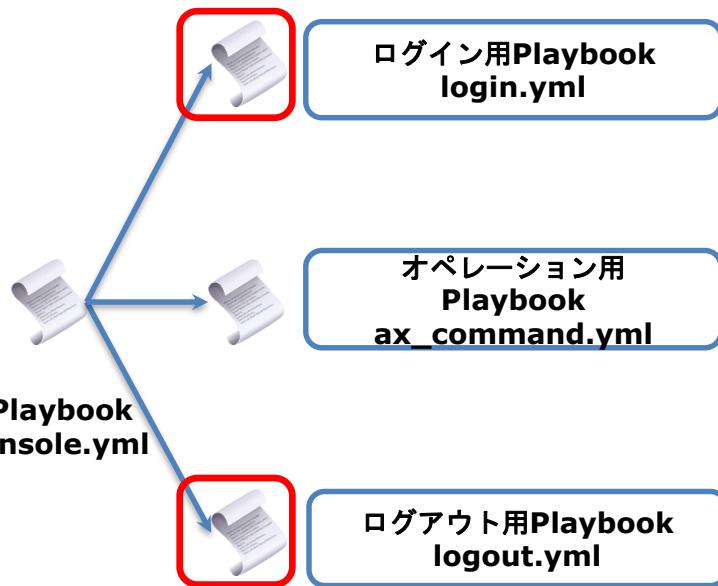
```
- name: "LOGIN with smartcs_tty_command"
  import_playbook: login.yml

- name: "Exec Task with ios_command"
  import_playbook: ax_command.yml

- name: "LOGOUT with smartcs_tty_command"
  import_playbook: logout.yml
```

他ベンダーモジュールの利用

■他ベンダーモジュールの実行 (Playbook構成例)



```

---
- hosts: smartcs
  tasks:
    - name: "Login by Console"
      smartcs_tty_command:
        tty: 5
        recvchar_regex:
          - '[Uu]sername: '
          - '[Pp]assword: '
          - '^(^|¥r|¥n|!)[a-zA-Z0-9_.-]*(>|#)'
        sendchar :
          - alaxala ←ax装置へのログインID
          - secret2230 ←ax装置へのログインパスワード ...
  
```

login.yml

NW機器に応じて
コンソール経由の
ログインプロンプトを指定

```

---
- hosts: smartcs
  tasks:
    - name: "Logout by Console"
      smartcs_tty_command:
        tty: 5
        recvchar :
          - "Press RETURN to get started."
        recvchar_regex:
          - '^(^|¥r|¥n|!)[a-zA-Z0-9_.-]*(>|#)'
        sendchar :
          - exit
  
```

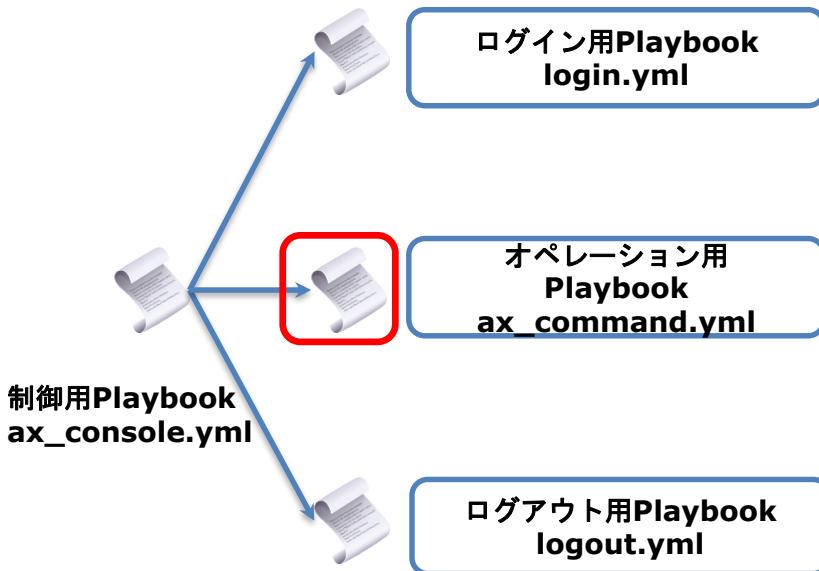
logout.yml

NW機器の汎用的な
プロンプト例

装置のコンソールに
ログインする際の
ID,パスワードを指定

exit 送信回数は
NW機器に応じて
複数回指定

■他ベンダーモジュールの実行 (Playbook構成例)



```

---                                         ax_command.yml
- hosts: smartcs
  gather_facts: no
  tasks:
    - name: "Task ax_command"
      ax_command:
        commands:
          - show version
          - show system
          - show ip interface
          - show vlan

vars:
  - ansible_connection: network_cli
  - ansible_user: port01
  - ansible_password: secret01
  - ansible_ssh_port: 9305
  - ansible_network_os: ax
  - ansible_become: yes
  - ansible_become_method: enable
  - ansible_become_password: secret2230
  - ansible_command_timeout: 60

```

接続先はSmartCS

他社ベンダのモジュールを指定

コネクションプラグインはnetwork_cliを指定

ログインID
パスワードにはSmartCSのポートユーザを指定

SSHの接続ポートをSmartCSのsshpt用のポートに変更

OS種別は使用する他社ベンダを指定

■他ベンダーモジュール実行時のポイント

・利用できるモジュール

- SSHで装置にログインしてCLIを実行する処理をコンソール経由で行う内部処理となる為、コネクションプラグインとして**network_cli**をサポートしているものに限ります。

例

```
vars:  
  - ansible_connection: network_cli
```

- SSH接続時とコンソールアクセス時のプロンプト定義が同じでないと動作しない (**terminal** プラグインの定義)

・タイムアウト値の設定

- 他社ベンダのモジュールは通常SSH接続して動作するが、本連携ではコンソール経由で動作する事になる
その為、処理速度が遅いのでタイムアウト時間の延長が必要。（コマンド実行時間など）

例

```
vars:  
  - ansible_command_timeout: 60
```

【ハンズオン 演習4】

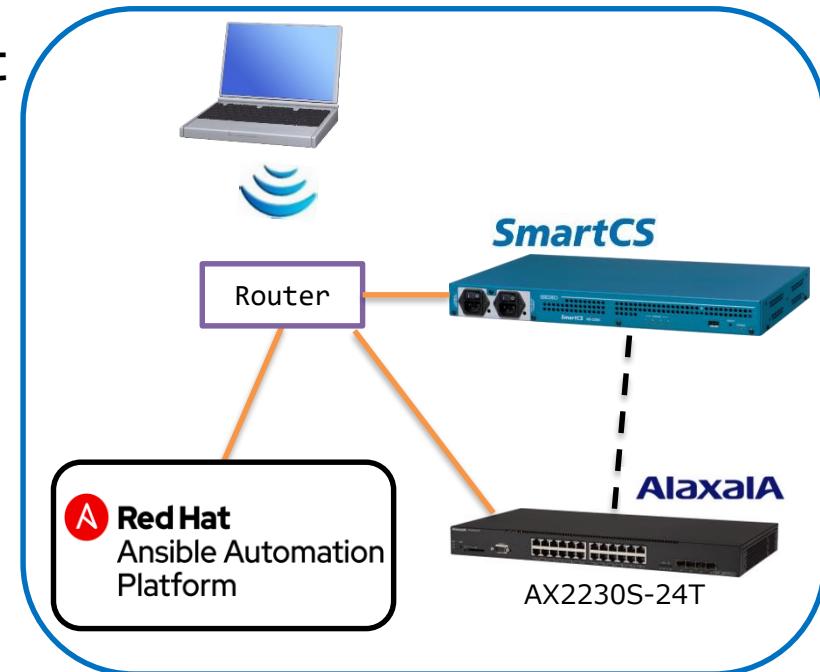
Ansible×SmartCS×Alaxalaの連携演習(応用編)

概要

演習4では、コンソールアクセスが必要となるユースケースに沿って、SmartCSとAnsibleの連携方法の理解を深めていただきます。

- 設定ミスによる障害からの復旧自動化【演習4.1】
- 通信障害からの復旧自動化【演習4.2】
- ファームウェアアップデートの自動化【演習4.3】
- 設定初期化の自動化(演習5終了後、最後に実行します)【演習4.4】

構成



【Ansible Towerの説明】

Red Hat社



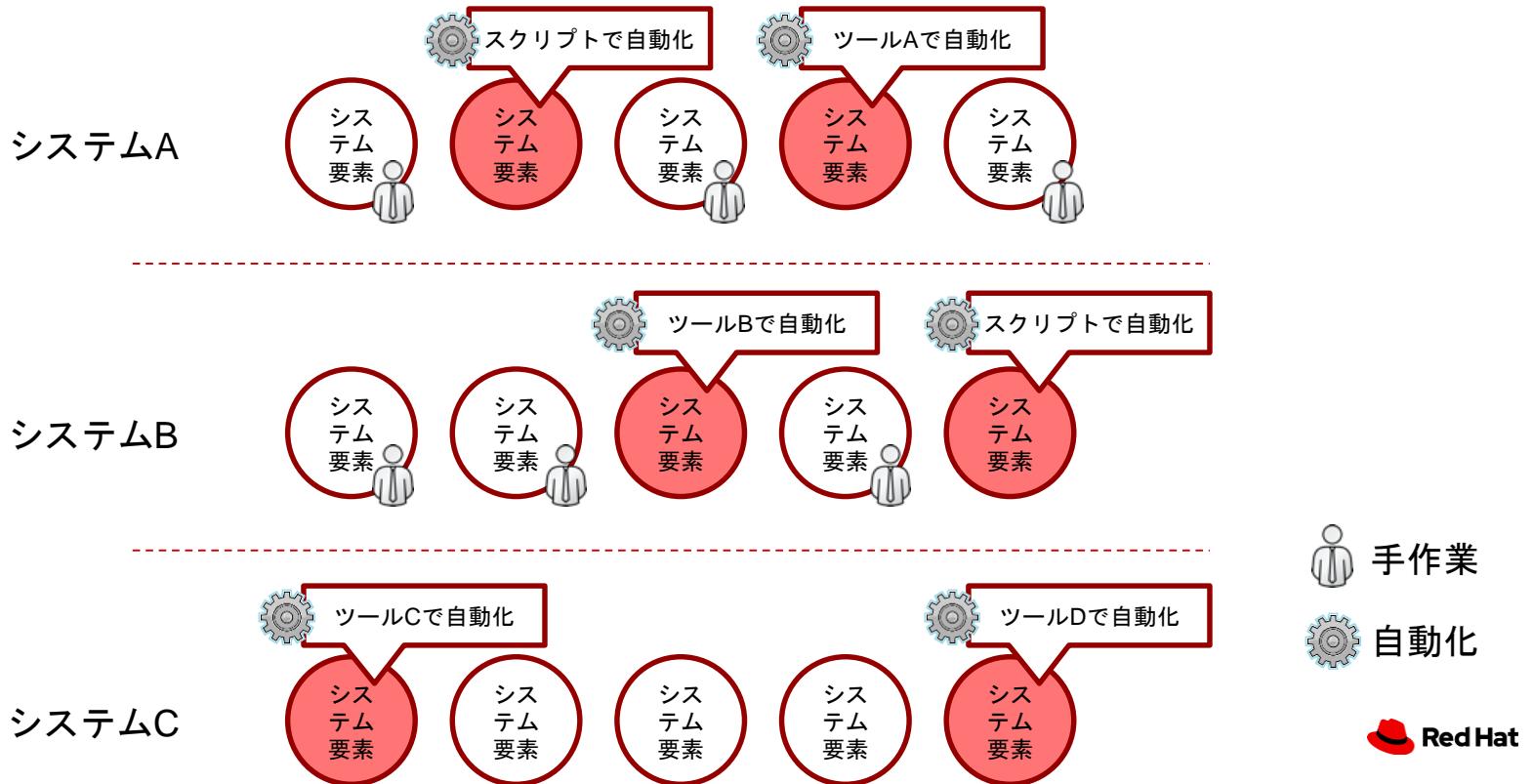
Red Hat
Ansible Automation
Platform

Ansible Engine と Ansible Tower

Ansible (Engine)はお一人様専用

- チームで使う場合のAnsible (Engine)のイケテナイ所
 - 誰でも見られるパスワードの管理、大丈夫？
 - 実行ログが残らない、
 - 誰が何をしたか、
 - 今の状態はどうなっているかを追えない
 - Playbookの編集履歴が追えない。
 - 誰でも編集できてしまう。
 - どうなっているか確認しないと使えない

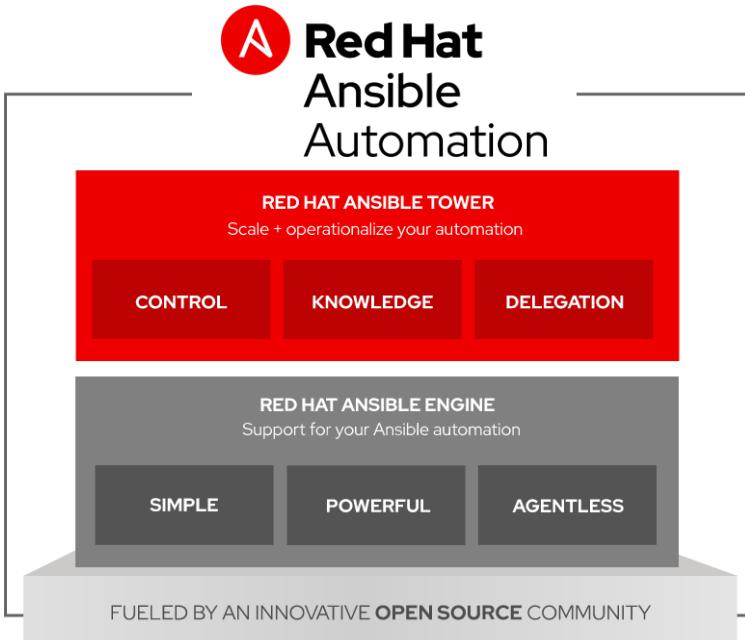
一般的な日本企業で起きている状態 = 属人化&サイロ化



サイロ化が招く負のスパイラル



Red Hatが提供する Ansible



Red Hat Ansible Automation

Red Hatが提供するAnsible
以下の2つの要素で構成

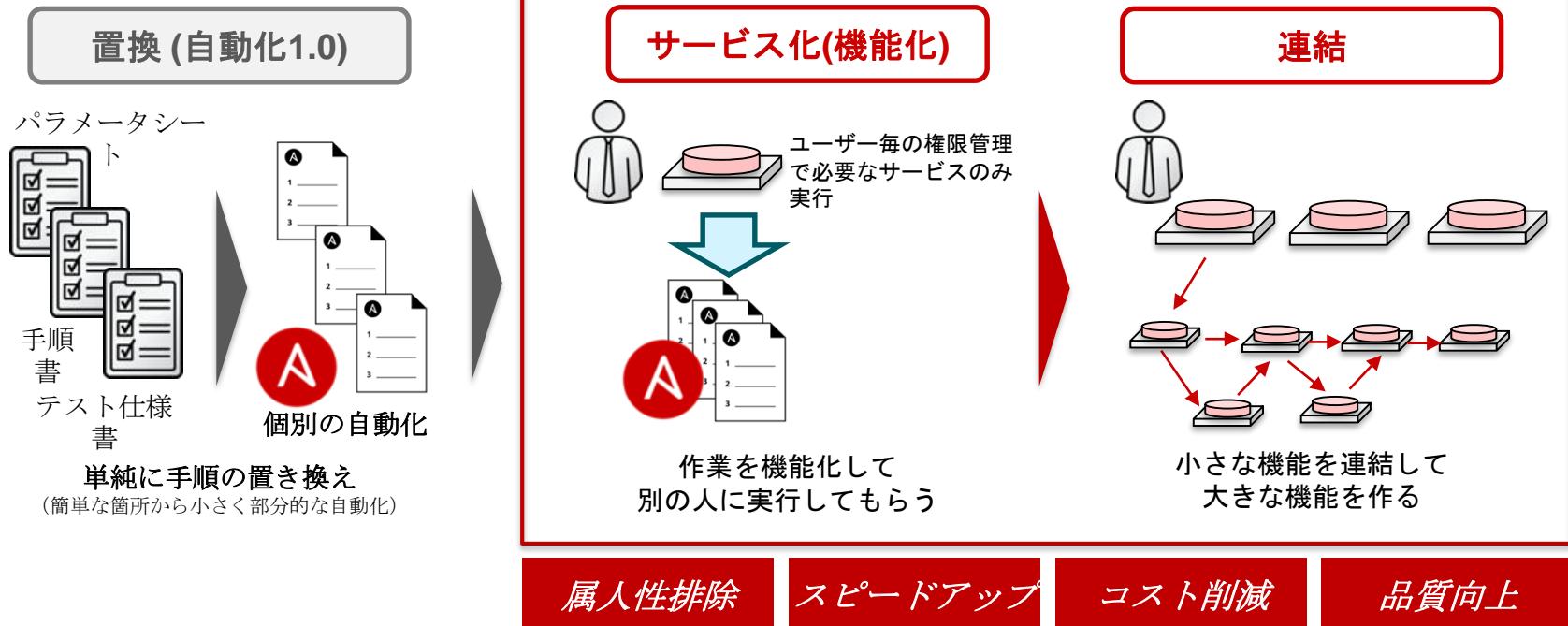
Ansible Tower

自動化プラットフォーム
(GUI / REST API / 権限管理 / 履歴管理 /
拡張性・冗長性などを提供

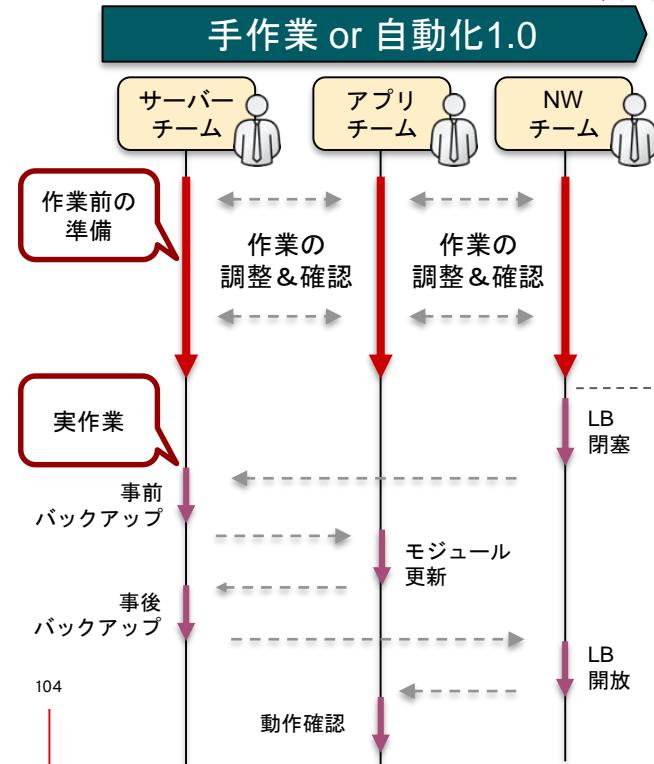
Ansible Engine

自動化の処理実行エンジン
オープンソースのAnsible Projectに相当

Ansible による自動化2.0の実現（作業のサービス化）

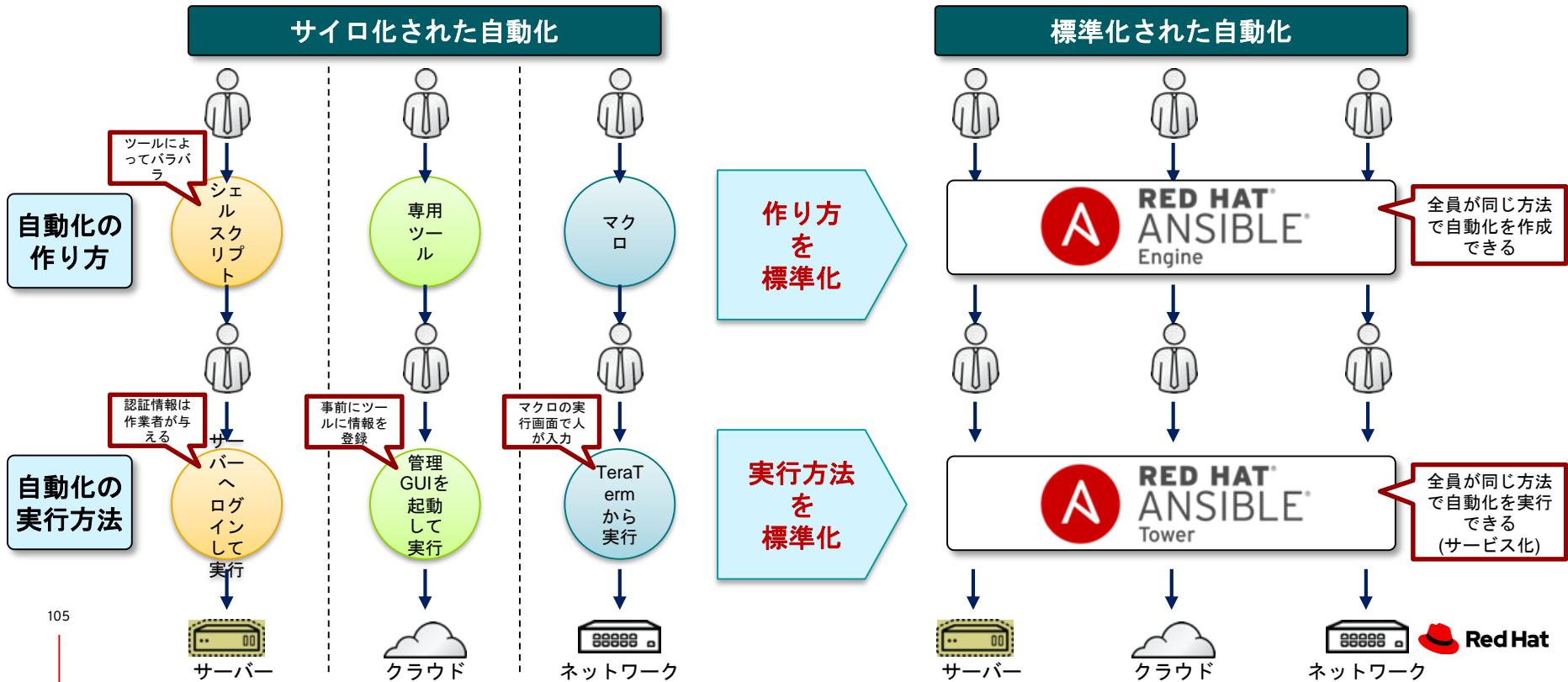


自動化からサービス化へ 組織間オーバヘッド削減のために



WEBアプリケーションのリリース作業例

Ansibleは、自動化手法の”標準化”を実現



真に経営へ貢献するITインフラのエンジニアリングを実現

- Ansible Automationは、従来は効果が発揮されづらかった自動化の効果を最大化するソフトウェア
- Ansible Automationは、自動化の標準化を可能にするソフトウェア

Ansibleの自動化

今できている事、今できていない事の

コスト削減
スピードアップ
品質向上

経営者は
企業価値の向上

現場のエンジニアは
自分が楽に安全に

Ansible Tower の要素

リソース

- **認証情報 (Credential) :**
様々な認証で使用されるIDやパスワード情報など。Towerでは、認証情報は暗号化され保存される
- **プロジェクト (Project) :**
Playbookを管理する論理コレクション。Towerでは、ソースコード管理システムにPlaybookを配置して管理される
- **インベントリ (Inventory) :**
管理対象ノードのリスト。Ansibleのインベントリと同じ
- **ジョブ (Job) :**
Ansibleにおいて、インベントリに対して実施された（されている）処理
- **ジョブテンプレート (Job Template) :**
Ansibleジョブを実行するためのパラメータセット。インベントリ、（プロジェクト&）Playbook、認証情報が含まれる
- **ワークフロージョブ(Workflow Job) :**
複数のジョブテンプレート等の順序実行を制御するためのセット

Ansible Tower の要素

その他

- 組織 (Organization) :

Ansible Towerの最上位の管理単位。前述の各種リソースなどの要素は組織に属している。

- パーミッション (Permission) :

権限のセット、前述のそれぞれのリソース要素一つ一つに対して、ユーザー毎に編集、閲覧、実行などの権限を設定・管理できる

- スケジュール (Schedule) :

ジョブ実行、プロジェクトの更新、インベントリの更新などの処理を特定の時刻、間隔で実施させるための機能

- 通知 (Notification) :

ジョブ実行、プロジェクトの更新、インベントリの更新などの処理が行われた際の通知設定。メール、Slackなど10種の通知方法に対応

【ハンズオン 演習5】

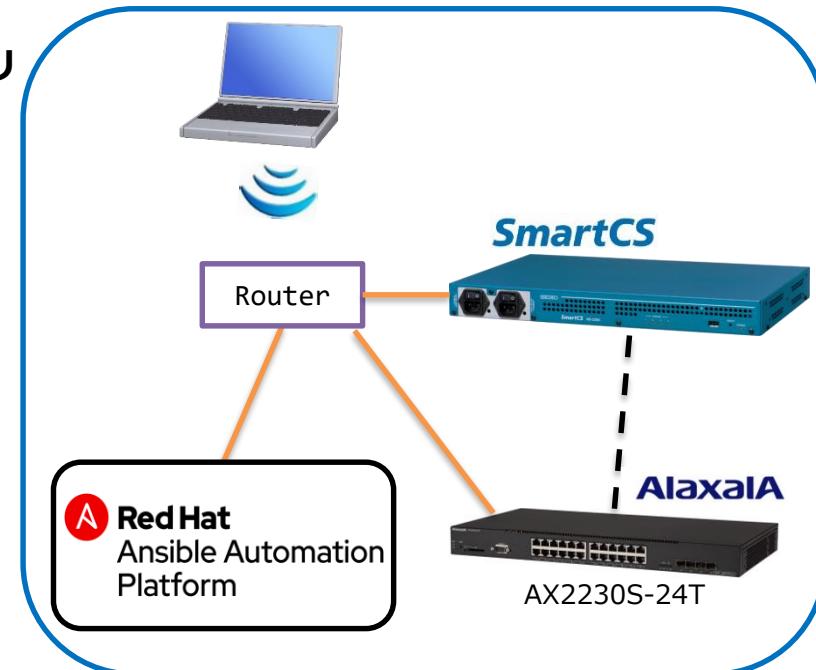
Ansible Tower×SmartCS×Alaxalaの連携演習

概要

演習5では、Ansible Towerを利用し、作業のスケジューリングとワークフロー化について理解を深めていただきます。

- 監視～障害時の情報取得のワークフロー化【演習5.2】
- 監視～障害時の情報取得のスケジューリング実行【演習5.2】

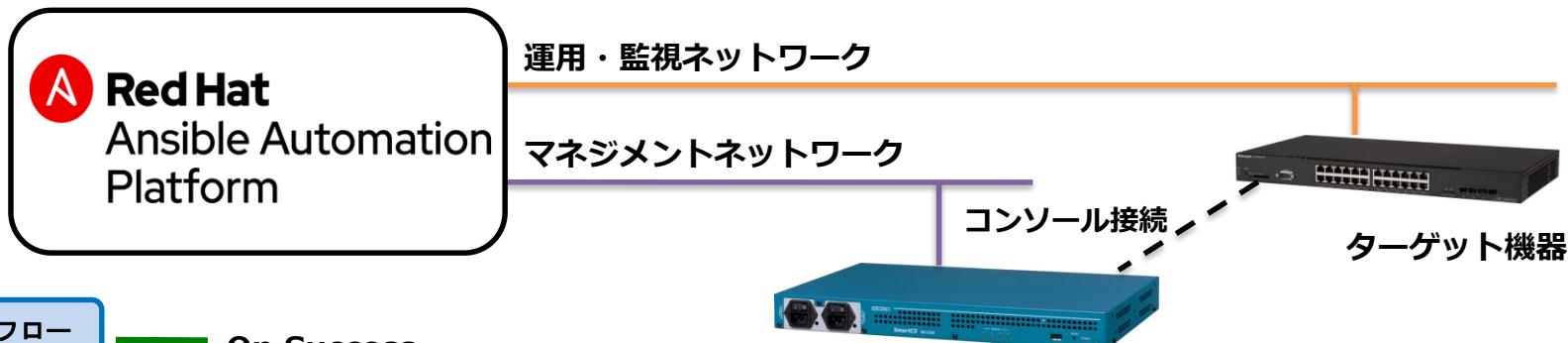
構成



監視と情報取得のワークフローをスケジューリング実行

- ネットワーク経由でターゲット機器を定期的に監視 (**スケジューリング機能**)
- 監視で障害検知した際に、コンソール経由で一次情報取得を自動実行しログ保存 (**ワークフロー機能**)

構成



ワークフロー

- On Success
- On Fail
- Always



Ansible Tower×SmartCSの連携

■ワークフロー機能

複数のPlaybookを続けて実行するように組み合わせることが可能。

次に続くPlaybookの実行条件として、前段のPlaybook実行結果(失敗時/成功時/常時)を指定可能なため、**死活監視失敗時にコンソール経由で情報取得を実行する**という様な条件分岐するオペレーションが容易に実現可能となります。

■スケジューリング機能

Playbook、およびワークフローをスケジューリング実行可能。

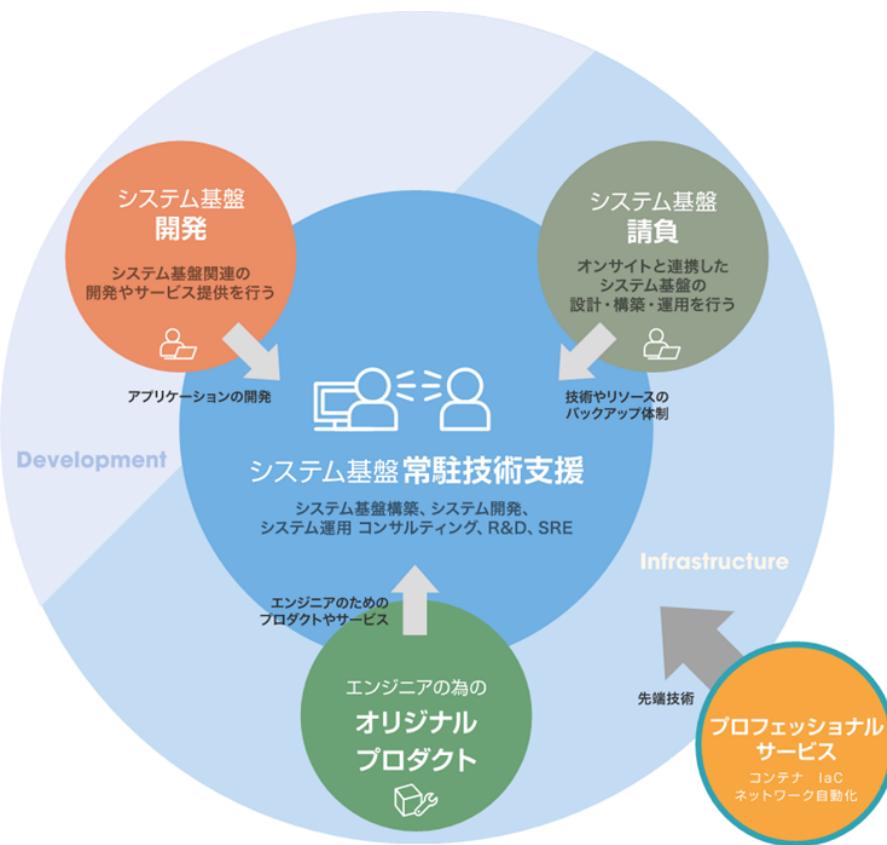
監視～情報取得のワークフローや設定情報の定期バックアップなど、
定期的に実行したいオペレーションが容易に実現可能となります。

【本日のまとめ】
エーピーコミュニケーションズ社

本日はSmartCS x ALAXALA x Ansible ハンズオン
へのご参加まことにありがとうございました。

各社様のご理解を賜り
弊社の取り組みについてご紹介させていただきます。





会社概要

会社名：株式会社エーピーコミュニケーションズ

URL : <http://www.ap-com.co.jp/>

住所：本社 / 東京都千代田区鍛冶町2丁目9番12号 神田徳力ビル3階

関連会社：ミランティス・ジャパン株式会社、株式会社APアシスト

代表：内田武志

設立：1995年11月16日

資本金：9,250万円

従業員数：381名（2019年7月現在）

最新技術に精通し、「お客様が満足すればするほど、我々もベネフィットを得ることができる構図」を創出することが出来る「NeoStar（ネオエスアイヤー）」として、日本のSI業界を活気に溌ちた面白い業界にすることを目指す、マルチエンジニア集団です。

組織的な自動化を加速するには

本日ご参加いただいている皆様は

ワークショップを通じ

自動化の必要性、技術的な理解を深めていただけたと思います。

またセイコーソリューションズ様 SmartCSとAnsible連携を学習し連携の可能性についても知見を広げていただけたことでしょう。

しかし実際の現場への自動化適用には様々な「壁」があります。

自動化推進にあたってよくある「壁」



- Ansibleを学習したがどこから手を付けて良いかわからない。
- 一部のワークフローを自動化したところで取り組みが頓挫してしまった。
- 内製でAnsible化を進めているが半年経ってもPoCのまま。
- サーバーの自動化は着手できたがネットワークは手付かず。
- Ansibleでの自動化を進めているが、見様見真似で進めてきたのでベストプラクティスかどうか不安である。
- 自動化推進の議論をしていたら、話が大きくなりすぎて、いつの間にか自動化の話から脱線していた。

ネットワーク自動化推進の背景には、
既存のネットワークにおける優先度の高い問題
対応するエンジニアの多忙
リスクとコスト
といった問題が多く
内製で思ったように進まないケースが多い

弊社の自動化サービスによって解決可能です！

自動化推進にあたって壁を感じたら是非ご一考ください。

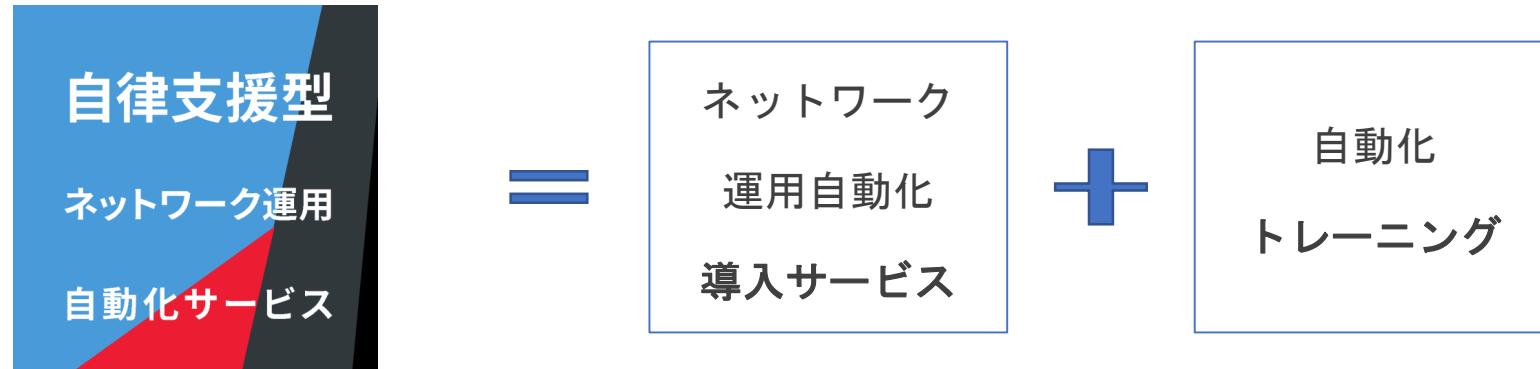


エンジニアが自ら育てる自動化へ

お客様ごとの最適な業務プロセスコンサルティングから自動化の導入（実際の運用）

最終的にはお客様が自動化を自律して運用するためのスキル習得トレーニングまでを、パッケージ化して提供いたします。

自律支援型ネットワーク運用自動化導入サービスとは





ネットワーク運用自動化 導入サービス

組織で自動化を推進するために

Ansible Towerの導入をサポートします。

Ansible Towerの導入は、次の3ステップで進めます。

APC自律支援型ネットワーク運用自動化サービス

1. ネットワーク運用自動化導入サービス

組織で自動化を推進するためにAnsible Towerの導入をサポートします。

Ansible Towerの導入は、以下の3ステップで進めます。

Phase① 業務プロセスのコンサルテーション

一連の業務プロセスを分解・選択し、自動化を前提としたプロセスへ変換します。自動化されにくい業務は見直しを行い、自動化に適したプロセスを目指します。

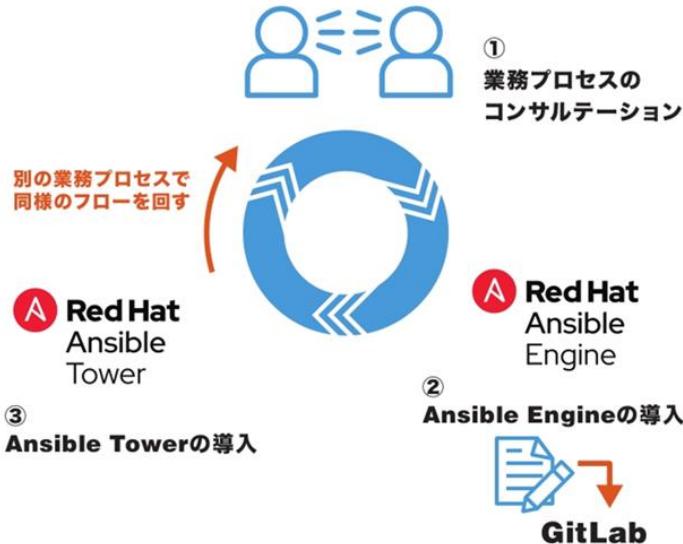
Phase② Ansible Engineの導入

変換したプロセス内の手順をPlaybook化すると共に、お客様が自動化導入後にご自身で作成されるPlaybookの品質が維持される仕組みを構築します。

あわせて、GitLabサーバを導入することで、ConfigおよびPlaybookをバージョン管理し、「いつ」「誰が」「何を」変更したかを記録します。

Phase③ Ansible Towerの導入

ワークフロー機能を用いて、Phase2で作成したPlaybookを取り込み、一連の業務プロセスの自動化を実現します。セルフサービスポータルのよう利用可能なGUIや権限管理、実行履歴管理を実装し、エンタープライズに最適な自動化ソリューションを提供します。



Phase1で複数ある業務プロセスから優先度が高いプロセスを選択して、Phase2、Phase3を回します。Phase3が完了したら再度Phase1に戻り、別の業務プロセスを選択して同様のフローを回す流れとなっています。

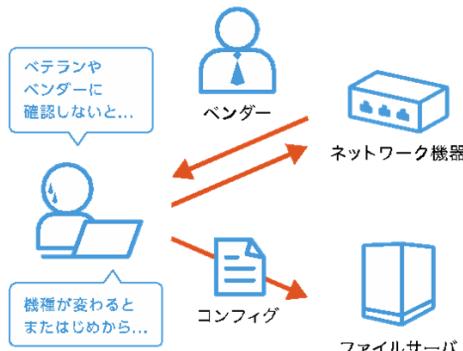
エンジニアが自ら育てる自動化へ

人手による品質維持からシステム化された品質維持に移行。

お客様にて自律的に自動化を推進できるように

仕組み化された自動化を提供します。

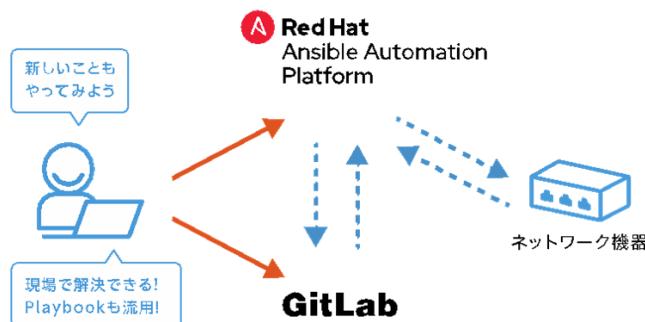
Before



- 運用フロー
- ①手順書作成
 - ②コマンド実行
 - ③コンフィグ取得
 - ④ファイルサーバにアップロード

× 手順書、マクロ作成が大変
× 実作業に時間がかかる
× 機器ごとの変更履歴が煩雑

After



- 運用フロー
- ①Playbookの作成
 - ②ワークフローの実行
 - ③コマンドの実行 [自動]
 - ④コンフィグの取得 [自動]
 - ⑤GitLabへのコミット [自動]

◎ Playbookによる手順の標準化 [属性入排他]
◎ ネットワーク機器への一括設定 [スピードアップ]
◎ コンフィグ・Playbookのバージョン管理 [品質向上]

弊社サービスのメリット

メリット 1

循環型の自動化変換プロセスを前提としているため、
スマートスタートでき、
効果が実感しやすいサービス設計になっている。

メリット 2

導入段階から全体最適化を図るべく、
形骸化しづらく、
メンテナンスしても**品質劣化しづらい**仕組みとなっている。

メリット 3

ネットワーク自動化に関するトレーニングを前提としている。
組織の文化とスキルセットを自動化へ適応することで
自律と**継続**を図る。



Red Hat
Ansible Automation
Platform

Red Hat Ansible Automation Platformによる

自動化トレーニング

Red Hat Ansible Automation Platformを一貫して学習可能

(Ansible EngineおよびAnsible Tower)

案件実績およびコミュニティ貢献、書籍執筆などを
手掛けるエンジニアによるハンズオンサポート

ネットワーク領域における実践的なトレーニング

自動化トレーニング

自動化の効果を最大化するためには組織を横断し継続的に改善サイクルを回していくことが重要です。

Ansible Automationによる実践的なIaC(Infrastructure as Code)自動化技術を学習することで、組織の文化とスキルセットの両面から自律的な自動化を推進できるようトレーニングを提供いたします。

【トレーニングの概要】

IaC自動化技術領域で共通言語となっているAnsibleに関し、基本的な知識習得から実践的なハンズオンを通して理解を深めていきます。また、Ansible Engineによるコードレベルの自動化だけでなく、Ansible Towerによる、自動化のセルフサービスポータル化の実現までを一貫して学習します。

トレーニングはサーバ編とネットワーク編の2パターンをご用意しております。

自動化トレーニング

【開催場所】

弊社カンファレンスルーム(定員20名)または、ご訪問での開催
(最小催行人数5名以上)

【トレーニング価格】

サーバ編 7.5万円(税抜) / 1名 ネットワーク編 7.5万円(税抜) / 1名



自動化トレーニング サーバ編

座学によるAnsibleの基礎の学習、OS～ミドルウェアのデプロイに関するハンズオンを実践し、Ansibleによるサーバ構築の自動化と構成管理を習得します。

■日数

1.5日間 (1日目 7.5時間 / 2日目 4.5時間)

■応募要件

Linuxコマンドラインの基本操作

(テキストエディタを含む) ができること。

Ansibleの事前知識は不要です。

■カリキュラム

1日目

- Ansible とは
- Ansible の特徴
- Ansible を利用する際の注意点
- Ansible の基本動作
- Ansible の内部コンポーネント
- インストールの準備
- インストールの実態
- 事前準備
- コマンドの実行
- ホスト変数とグループ変数
- YAML の基本
- 変数
- 特殊なディレクトリ
- タスクのグループ化
- プレイブックの活用
- WordPress のデプロイメント(ロールの活用)
- パフォーマンスチューニング
- トラブルシューティング
- 暗号化

2日目

- Ansible Tower とは
- Ansible Tower の要件
- Ansible Tower のインストール
- Ansible Tower のログイン
- チーム作成
- ユーザー作成
- イベントリ作成
- クレデンシャル作成
- プロジェクト作成
- ジョブの作成および実行
- ワークフローの作成および実行
- APIについて
- 通知について
- バックアップ/リストア
- トラブルシューティング

自動化トレーニング ネットワーク編

座学によるAnsibleの基礎の学習、GitLabと連携した運用の自動化に関するハンズオンを通じ、Ansibleによるネットワーク運用の自動化を習得します。

■日数

2日間（1日目 7.5時間 / 2日目 7.5時間）

■応募要件

Linuxコマンドラインの基本操作

（テキストエディタを含む）ができること。

Ansibleの事前知識は不要です。

■カリキュラム

1日目

- Ansible とは
- Ansible の特徴
- Ansible を利用する際の注意点
- Ansible の基本動作
- Ansible の内部コンポーネント
- インストール
- 事前準備
- コマンドの実行
- ホスト変数とグループ変数
- YAML の基本
- 変数
- 特殊なディレクトリ
- タスクのグループ化
- プレイブックの活用
- ネットワーク機器設定のプレイブック化
- パフォーマンスチューニング
- トラブルシューティング
- 暗号化

2日目

- Git とは
- Git コマンドの基本
- GitLab とは
- GitLab の要件
- GitLab のインストール
- GitLab のログイン
- プロジェクトの作成
- ユーザ、グループの作成
- GitLab リポジトリの作成
- GitLab リポジトリの活用
- GitLab 便利機能の説明
- バックアップ/リストア(GitLab)
- トラブルシューティング(GitLab)
- Ansible Tower とは
- Ansible Tower の要件
- Ansible Tower のインストール
- Ansible Tower のログイン
- チーム作成
- ユーザ作成
- インベントリ作成
- クレデンシャル作成
- プロジェクト作成
- ジョブの作成および実行
- ワークフローの作成および実行
- APIについて
- 通知について
- バックアップ/リストア(Ansible Tower)
- トラブルシューティング(Ansible Tower)

組織的な自動化を加速するには

弊社の「自律支援型ネットワーク自動化サービス」をご検討ください。
スタートダッシュのお手伝いをし、
自動化の取り組みを加速し継続するご支援が可能です。



お問合せ先
株式会社エーピーコミュニケーションズ 自動化グループ

automation@ap-com.co.jp

セミナー開催にあたりご協力いただきました企業の皆さん

SEIKO

セイコーソリューションズ株式会社

AlaxalIA



Red Hat

Ansible Acceleration Partner

レッドハット株式会社



SB C&S



AP Communications

ありがとうございました

ご清聴ありがとうございました