Ansible

ネットワーク自動化チュートリアル

JANOG

これからのネットワーク自動化

Shingo Kitayama & Akira Yokochi

Introduction

Shingo Kitayama

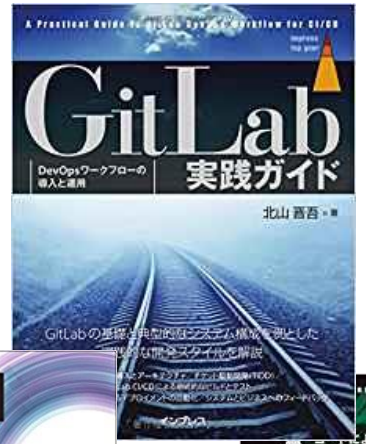
Shingo Kitayama

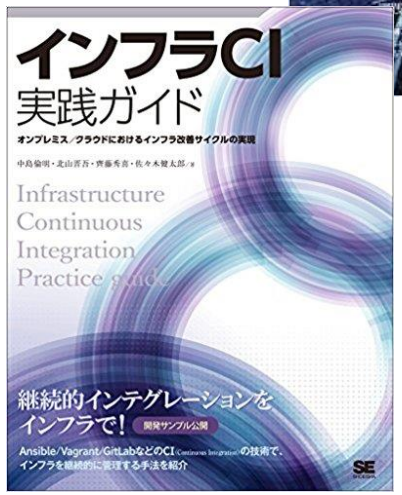
Company: Red Hat K.K.

Role: Solution Architect

Product: OpenShift

Interest: KPP





shkitayama

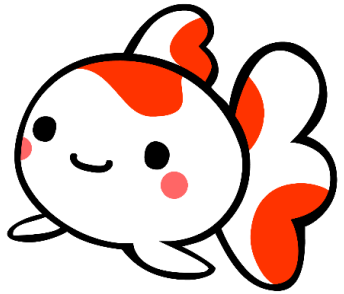
spchildren

JANOG

JANOG
JApan Network Operators' Group

自己紹介

名前	横地 晃
所属	株式会社 エーピーコミュニケーションズ



@akira6592

ブログ（てくなべ）

https://tekunabe.hatenablog.jp

過去発表資料 (JANOG 41 / 41.5 等)

https://www.slideshare.net/akira6592/

Agenda
本日のアジェンダ

前半
Automation with Ansible

1 | これからの自動化

2 | Ansibleによる自動化の世界

3 | Ansibleの仕組み

4 | Playbookを理解する

後半
Ansible Network World

5 | ネットワーク対応の基本

6 | 情報取得サンプル

7 | 設定変更サンプル

8 | ネットワーク対応のまとめ

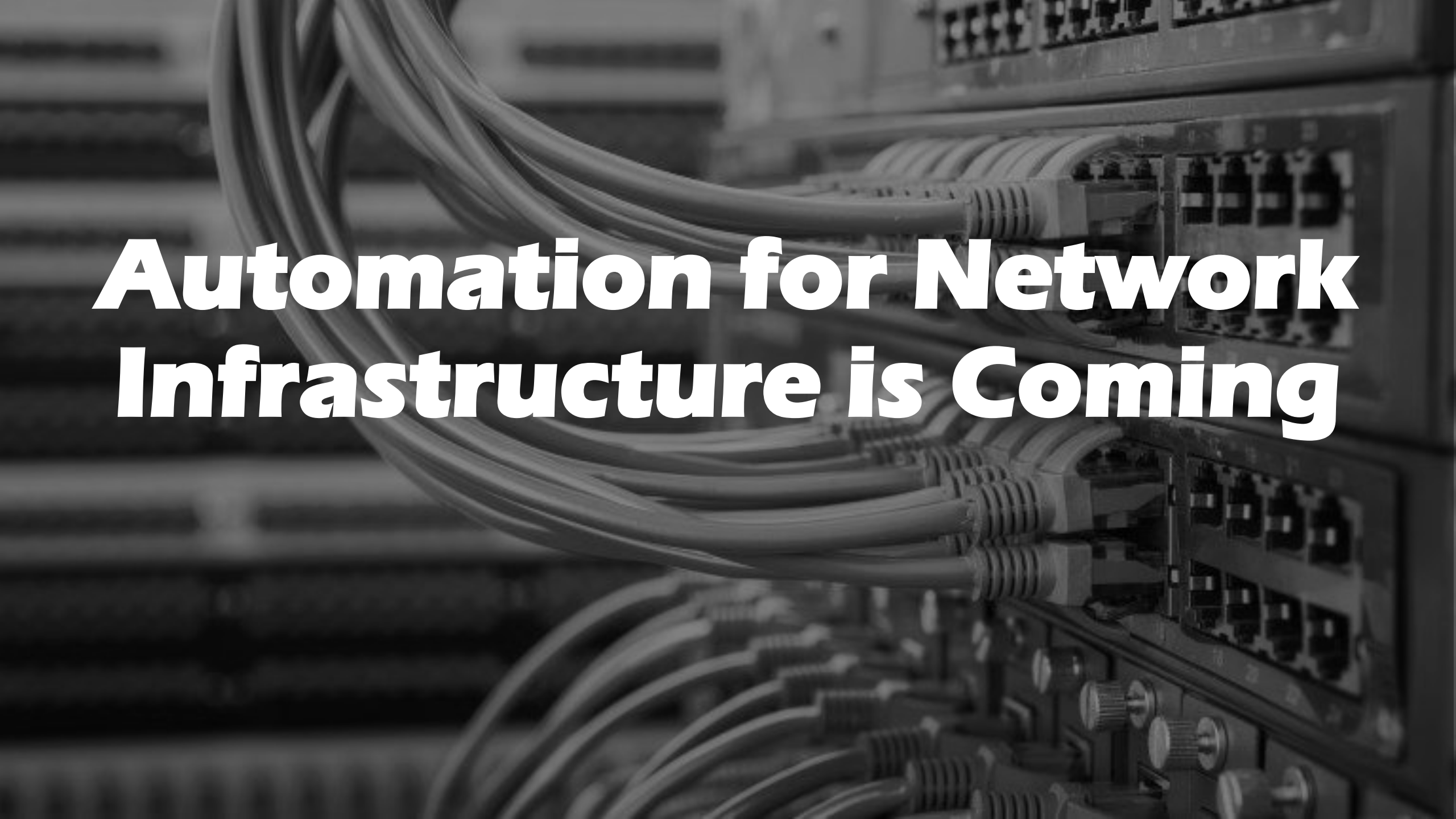
本日お伝えしたいこと

Janogに参加したら、Ansibleを使えるようになったというお土産

Ansible is Simple

Ansibleはネットワーク機器にも使え、
シンプルで導入コストが低く、
様々なプラットフォームに対応している。

1. これからの自動化
Value of Next Automation



Automation for Network Infrastructure is Coming

ネットワーク自動化の状況
ほとんどのユーザーがCLI管理

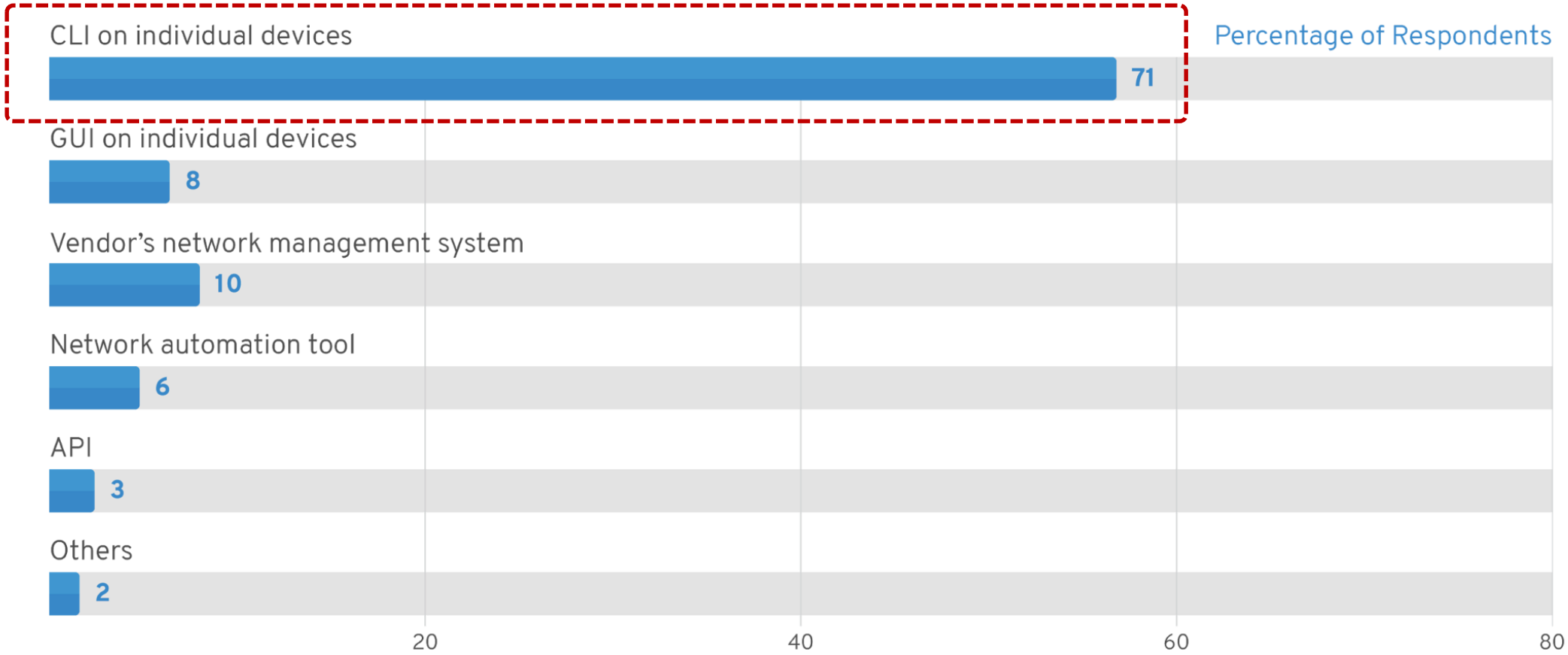


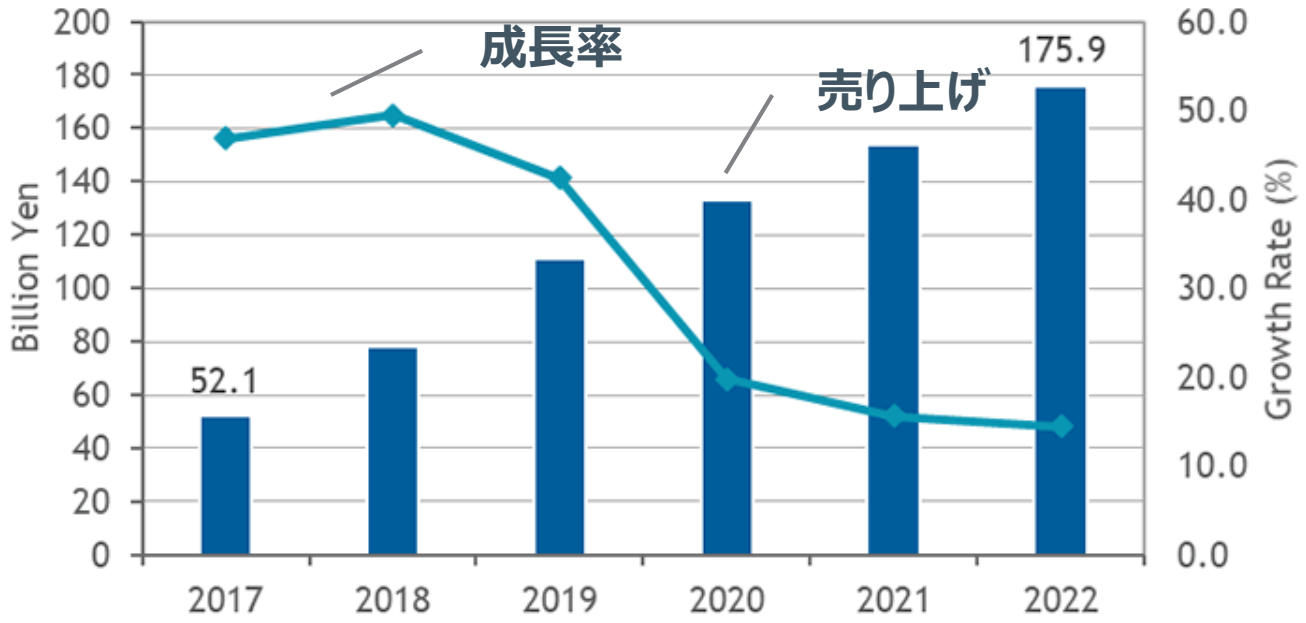
Figure 1
Primary Method for Making Network Changes

参照: Gartner, *Look Beyond Network Vendors for Network Innovation.* January 2018. Gartner ID: G00349636. (n=64)

SDN市場の拡大

SDNおよびNFVに関する国内市場予測

国内SDN市場は、成長が軌道に乗り2017年には521億円にまで市場規模が拡大。
2022年には1759億円にまで成長するとの予測。

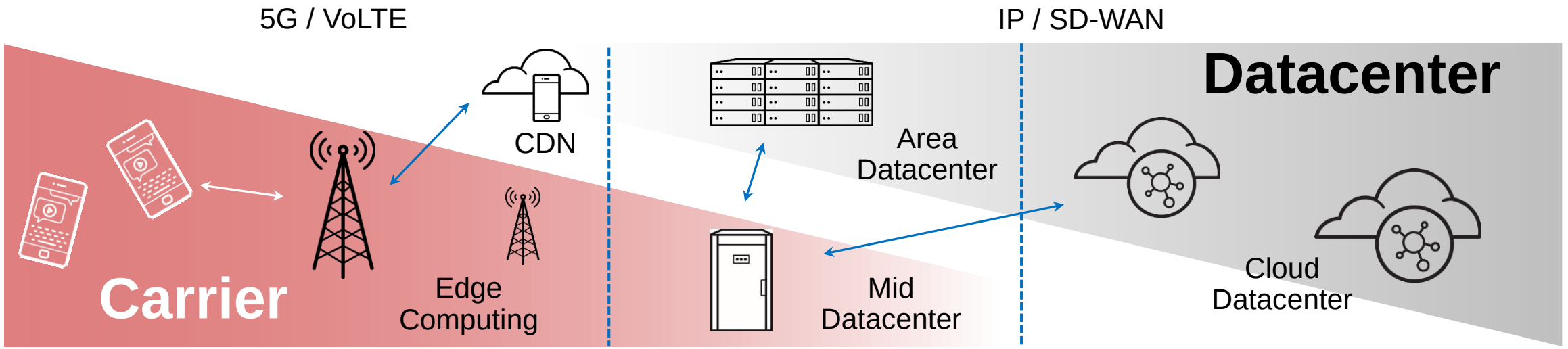


JANOG
JApan Network Operators' Group

参照: https://www.idcjapan.co.jp/Press/Current/20180410Apr.html

高まるネットワーク自動化への期待

キャリア、DCともに自動化に注力



Carrier SDNの注目分野

2017年～2022年のCAGR **32.7%**

「NFV環境の基盤としての導入」
「5Gネットワークにおけるネットワークスライシング」
「**通信事業者のオペレーションの自動化**」

Datacenter SDNの注目分野

2017年～2022年のCAGR **24.0%**

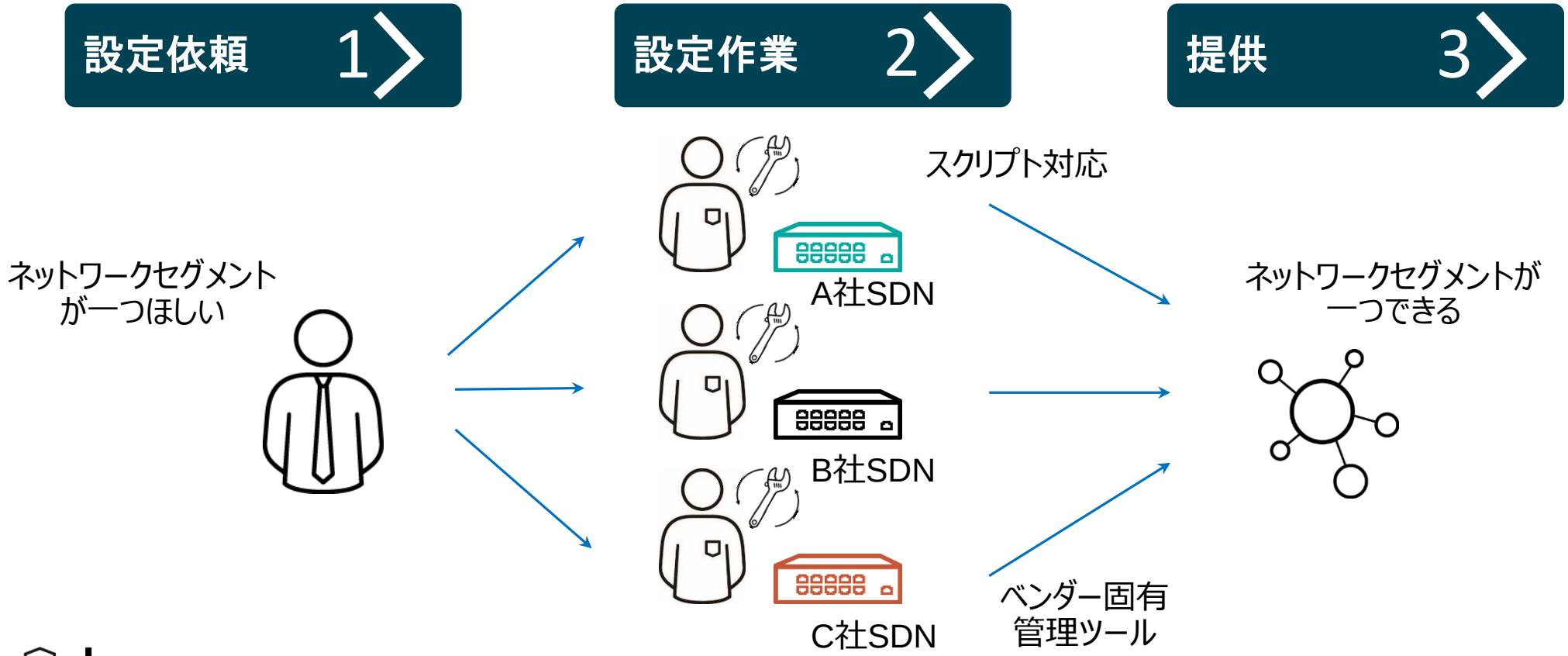
「**データセンターネットワークの自動化**」
「マイクロセグメンテーションの適用」

※ CAGR=Compound Annual Growth Rate(年間平均成長率)

参照: https://www.idcjapan.co.jp/Press/Current/20180410Apr.html

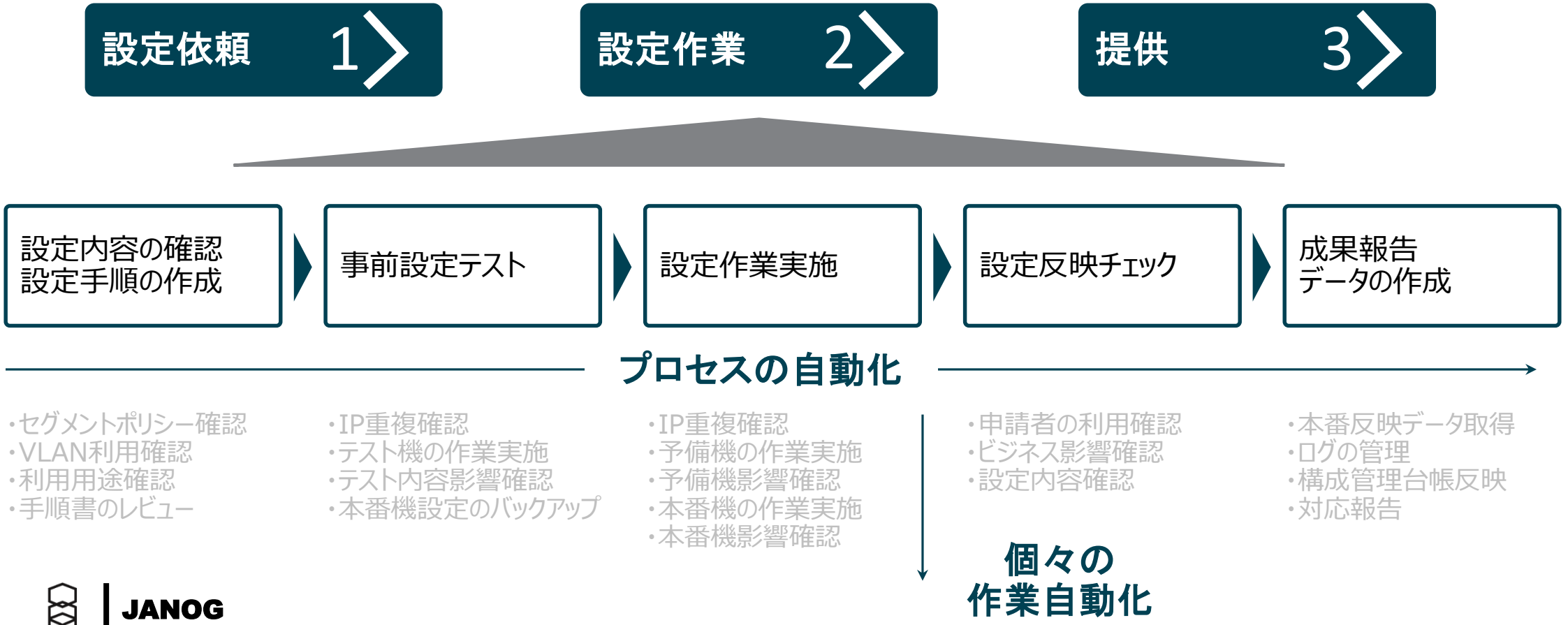
多様化された自動化の課題
製品によって異なるオペレーション

さまざまな領域でSDN技術の適用が進む反面、適用領域ごとに異なるSDNコントローラーや管理コンソールを必要とする「SDNのサイロ化」に対する懸念。



作業の自動化とプロセスの自動化

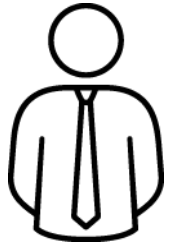
自動化という言葉の範囲



サイロ化する自動化の回避

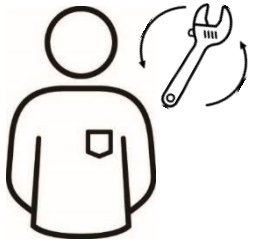
自動化は意識改革

> *Ansible's simple automation framework means that previously isolated network administrators can finally speak the same language of automation as the rest of the IT organization.*



マネジメントの意識改革

工数削減できるという期待で、
自動化することが目的となってしまう。



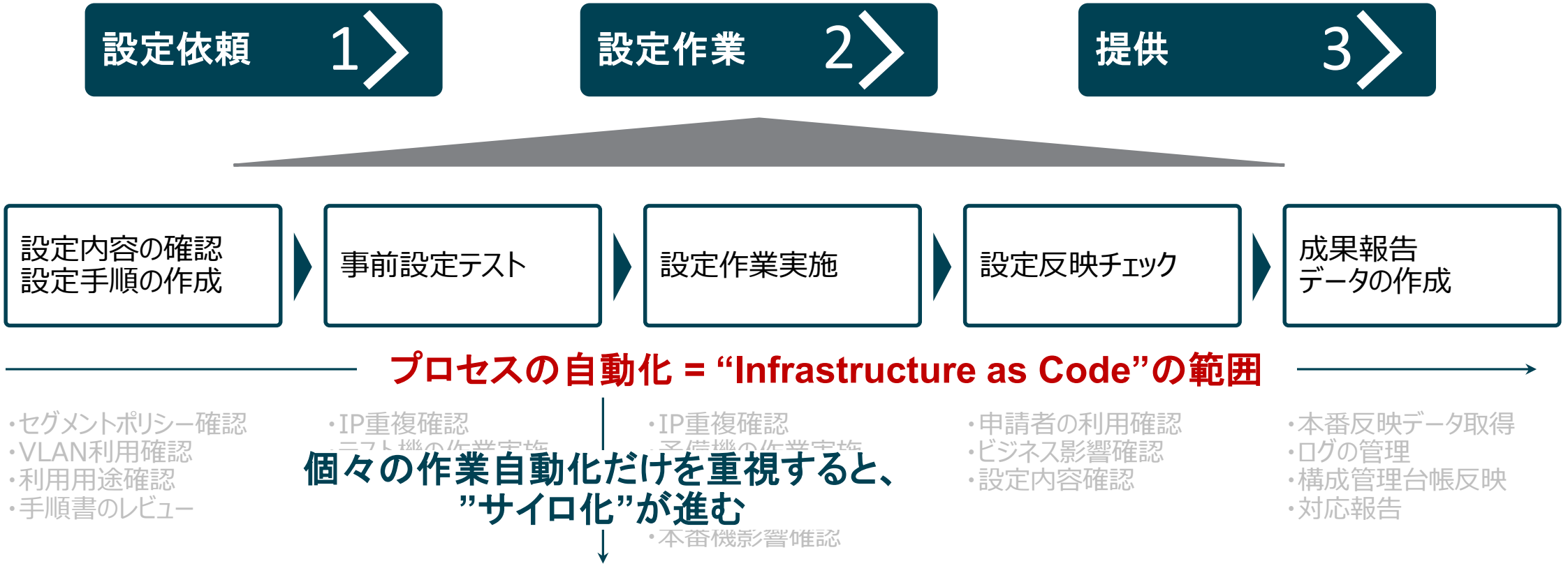
エンジニアの意識改革

属人的に自動化を行うことで、
内容が複雑化されると同時に
寿命が短く、無駄な工数になってしまう。

自動化は手段の一つであり、
自動化することで品質と
短納期を目的とする。

再利用できる自動化を目指し、
最終的な効果を測定できるようにする。

Infrastructure as Code
ネットワークの自動化



Ansibleによる自動化の世界

Ansibleの特徴

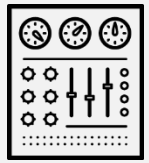
ネットワークにAnsibleを利用する価値
小さく始めて、大きく育てることができる

本日の話の範囲

Build
設定の自動化

- すぐに始められる自動化
- マルチベンダー製品対応
- 既存のネットワーク機器でも、新しいネットワーク機器でも対応可

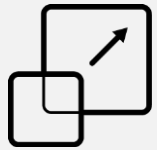
個々の作業自動化



Manage
プロセスの管理

- 構成ドリフトをトラックできる
- 任意のネットワーク機器も同じ方法で管理できる
- 変更したことを検証できる

プロセスの自動化



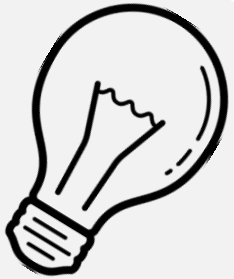
Scale
動的な運用

- 正常状態を定期的に確認する
- 特定のチーム間でロールベースのアクセス制御を行う
- RESTful APIを通してサードパーティ製品の統合をする

運用の自動化

Ansibleの特徴

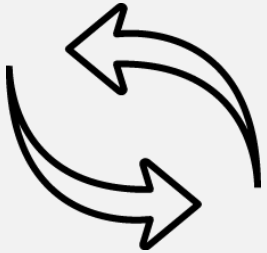
自動化を気軽に始められる要素



Simple

- 人が読みやすい設定
- 特別なスキルが不要
- 順序通り実行される

＞ すぐに始められる



Powerful

- 機器情報や接続情報の取得
- 設定管理
- 自動化のワークフロー

＞ 製品に依存しない管理

Agentless

- エージェント不要なアーキテクチャ
- OpenSSHやparamikoの利用

＞ 効率的かつセキュア

Ansibleの特徴 - Simple -

YAMLファイルによる記述

Simple

- yaml ain't markup languageの略
- 一般的な拡張子は「.yml」
- 構造化データの表現方法
- 仕様を処理する実装が別途必要

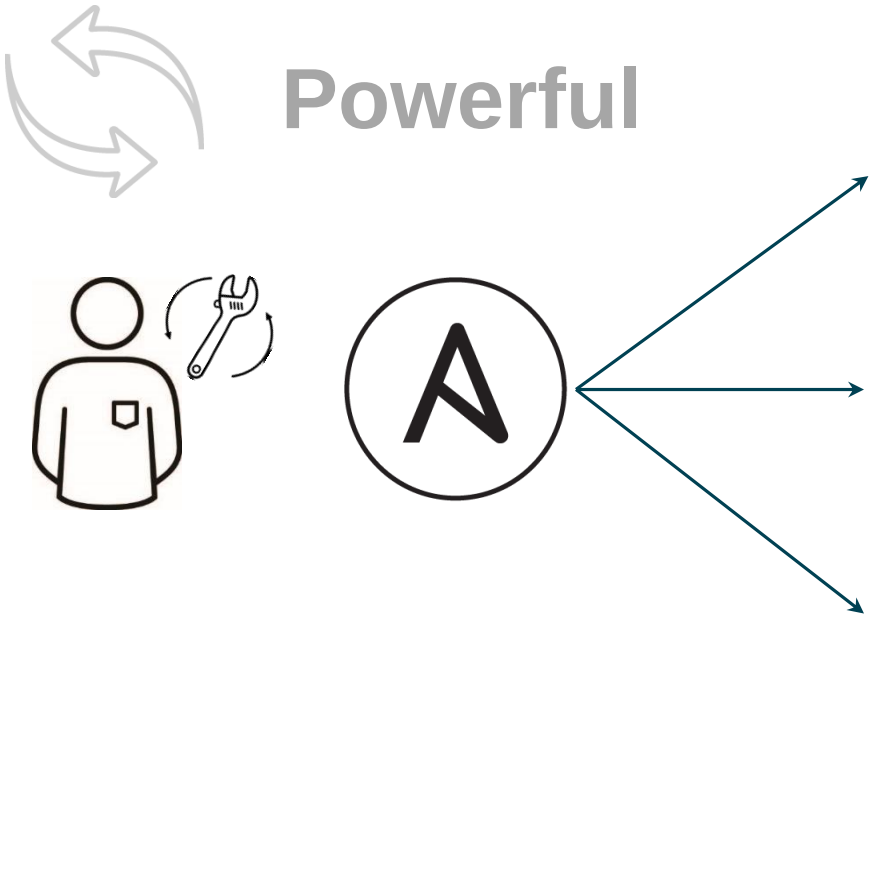
コーディングではなく、手順書の延長としての書式を重視

ios_command.yml

```
- name: run multiple commands
  hosts: ios01
  tasks:
    - name: show version and show interfaces
      ios_command:
        commands:
          - show version
          - show interfaces
```

Ansibleの特徴 - Powerful -

各ベンダーおよび全レイヤーをサポート



Datacenter

Configurations Management
- Linux/Windows Initial Setup
- Cisco/F5 Configuration
- Netapp Management

Cloud

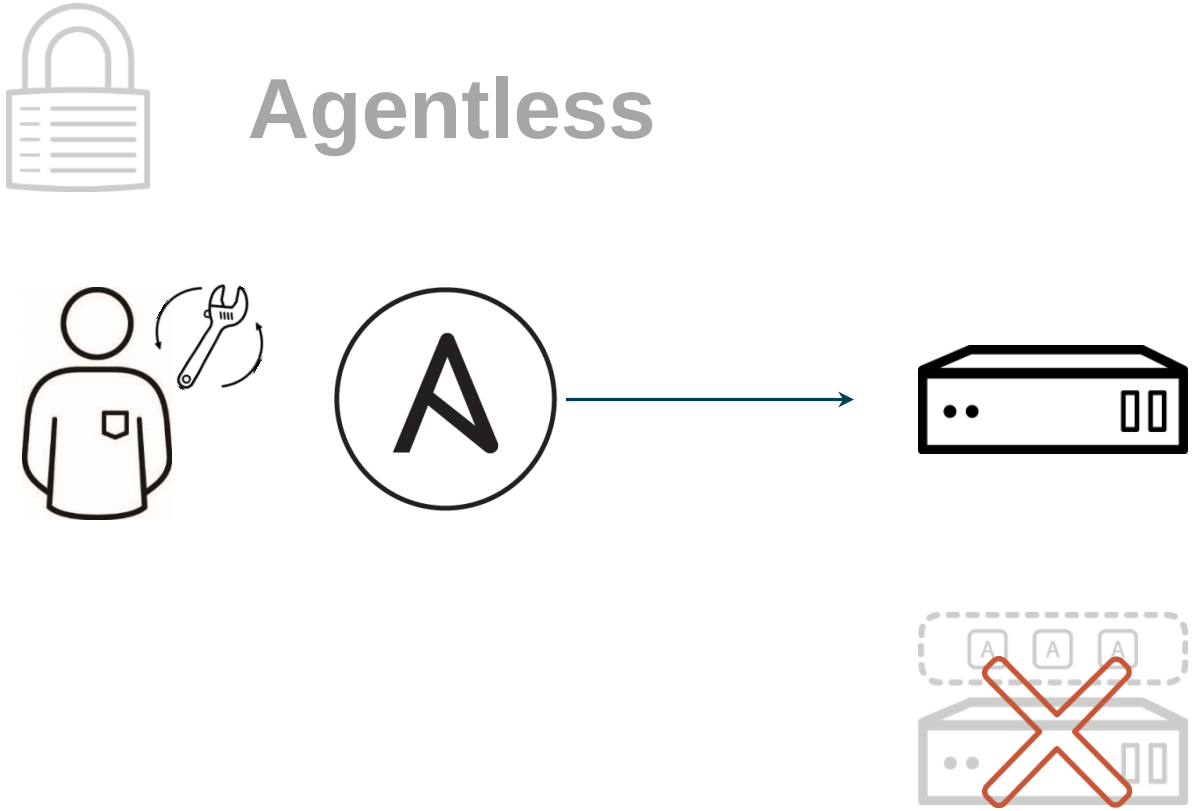
Bootstrapping
- AWS/Azure/GCP Resource Control
- VMware Configuration
- Container Running

Application

Orchestrations
- MySQL/Hadoop Cluster Setup
- 3Tier Web Application
- Cloud Native App Management

Ansibleの特徴 - Agentless -

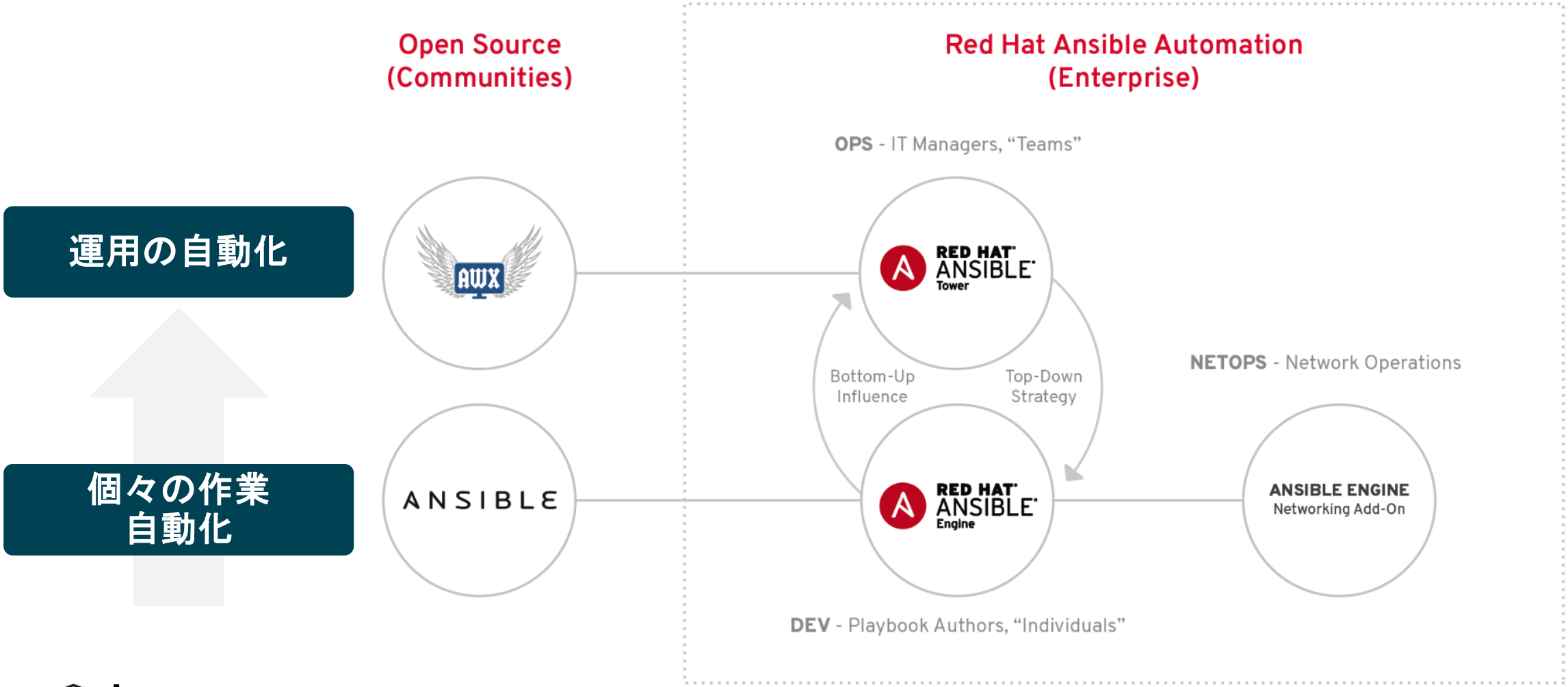
Agentless Connection



- Agentの導入/管理する工数が不要
- 都度SSHによって接続するためセキュア
- 既存の機器にもすぐに導入できる

AnsibleのCommunityとEnterprise

Ansibleの製品一覧

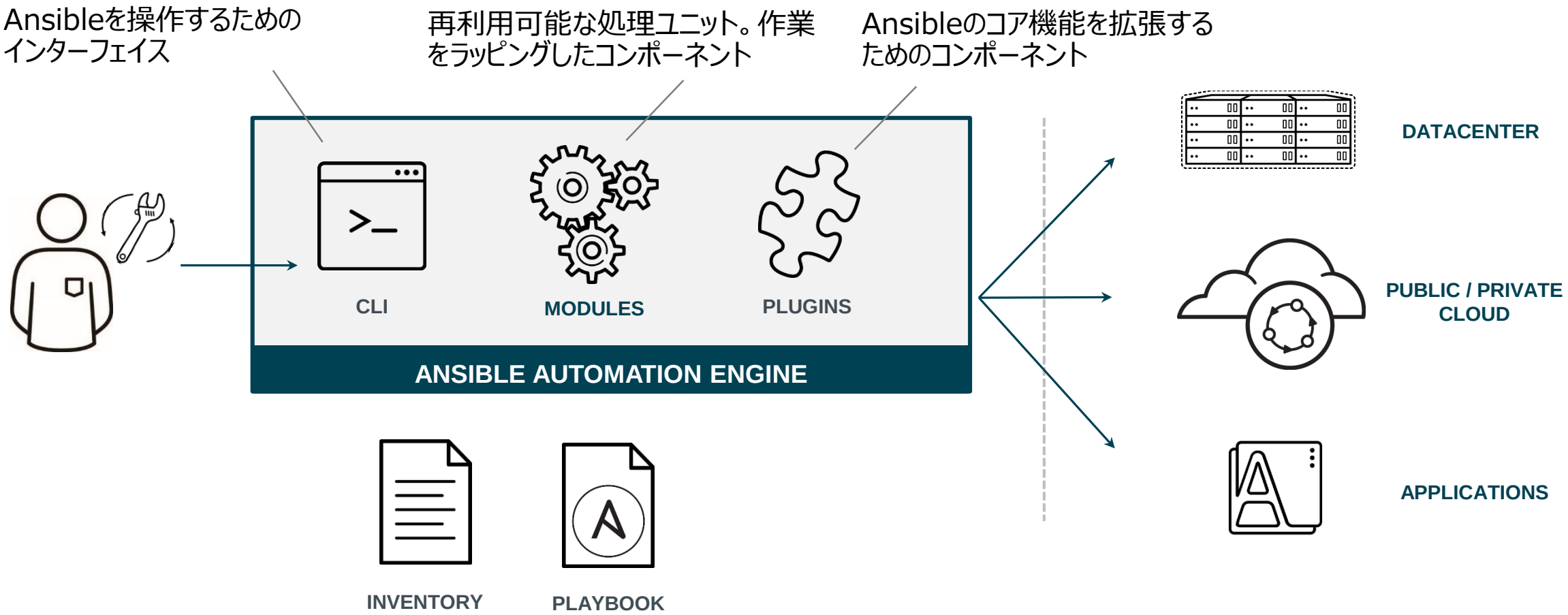


Ansibleの仕組み

Ansibleのコアコンポーネント

Ansibleのコアコンポーネント

Ansible Automation Engine



Ansible Command Line Interface

Ansible Ad-Hoc Command & Playbook Command

基本はコマンドラインからAnsibleを経由して操作を行う。
ターゲットのネットワーク機器を管理するために覚えなければいけないコマンドは、以下の2種類。

1. Ansible Ad-Hoc Command

```
[Usage] ansible <Target Host> -i <Inventory Files> -m <Module>
$ ansible cisco -i inventory -m ping

[Usage] ansible <Target Host> -i <Inventory Files> -m <Module> -a <Argument>
$ ansible cisco -i inventory –m ios_command -a “commands=show version”
```

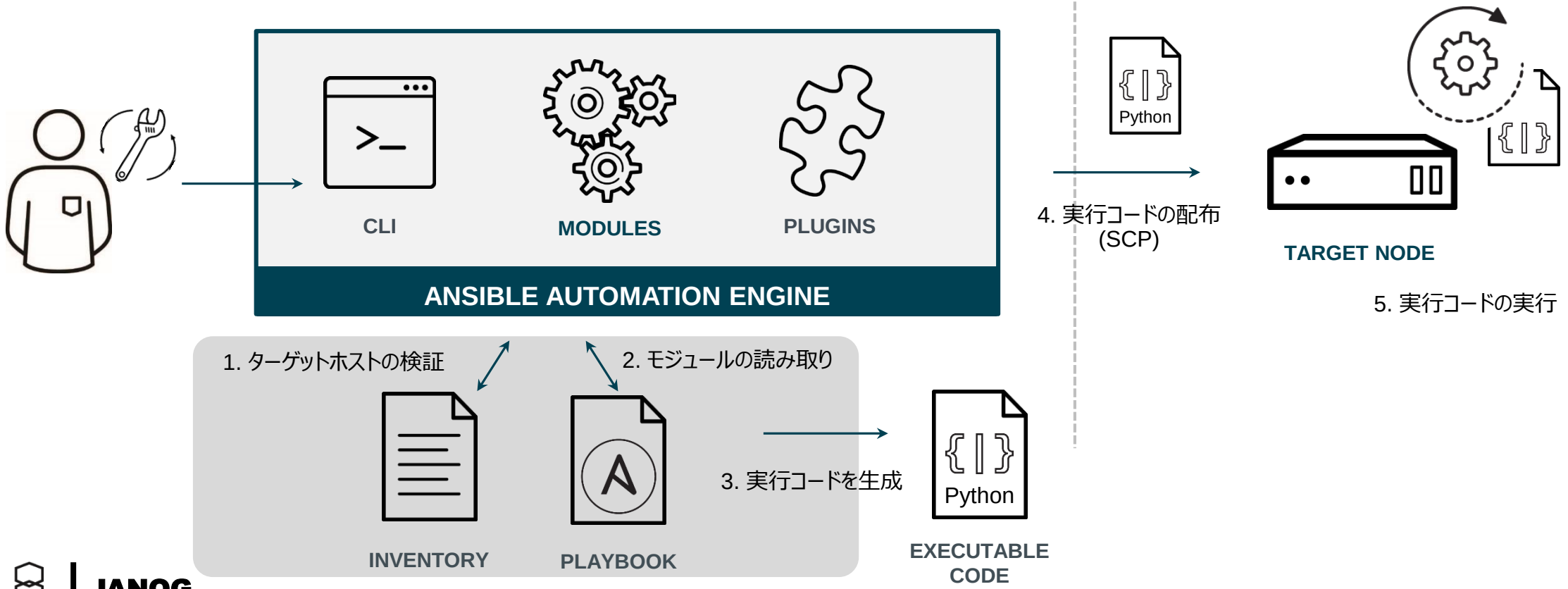
2. Ansible Playbook Command

```
[Usage] ansible-playbook -i <Inventory Files> <Playbook File>
$ ansible-playbook -i inventory site.yml
```

Connectionの仕組み

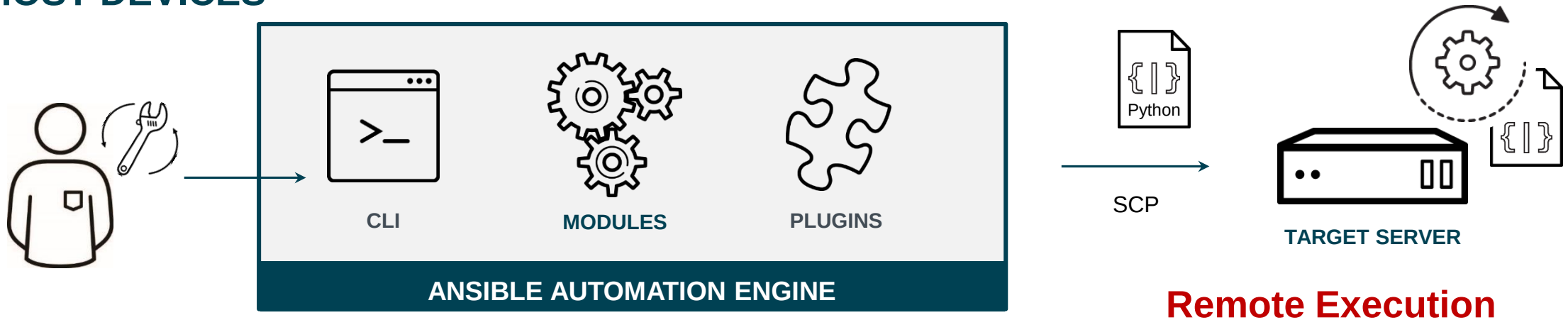
実行コードによるオペレーションの実行

1. ターゲットホストの検証
2. モジュールの読み取り
3. 実行コードを生成
4. 実行コードの配布
5. 実行コードの実行



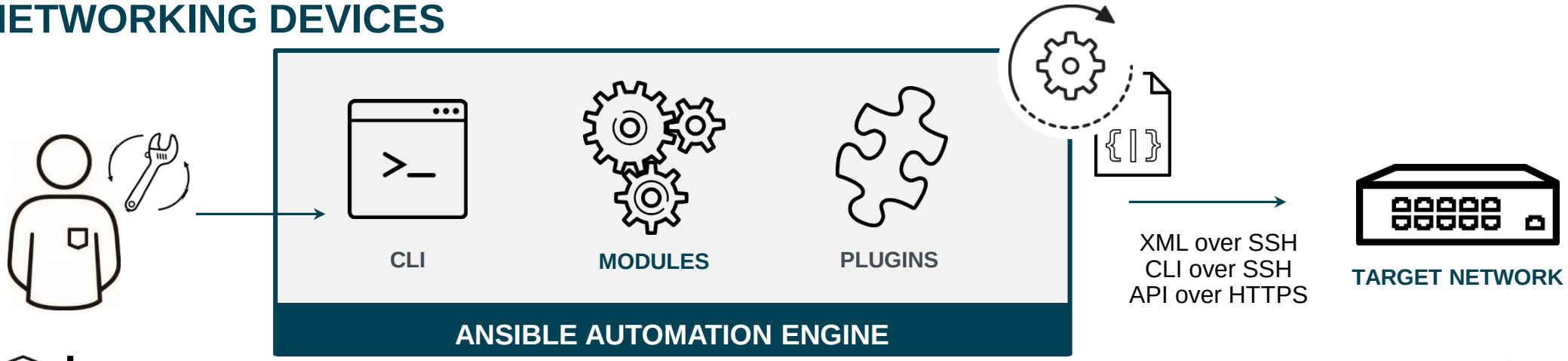
Network Connection Plugin

HOST DEVICES



Remote Execution

NETWORKING DEVICES

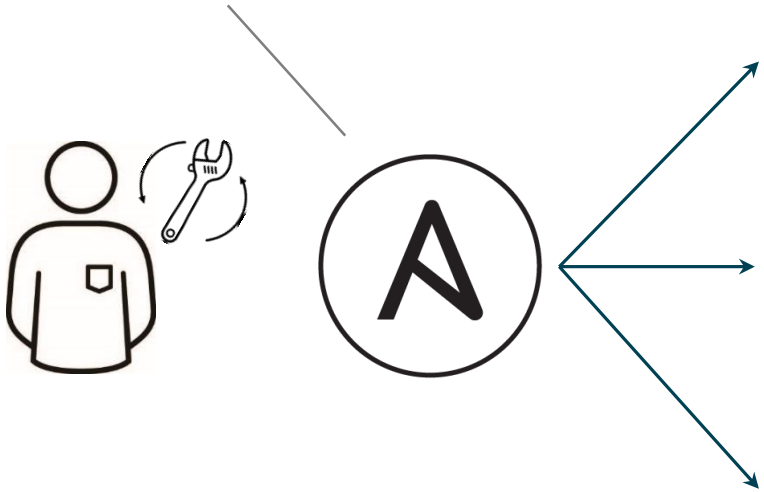


Local Execution

冪等性の担保

再利用可能な処理ユニットとは

新しいVLAN(ID:120)を作成して、
GigabitEth0/4,0/5に割り当てる。



Ansible Modules

再利用可能な処理ユニット

各種モジュールを利用して個別のタスクを実行する

```
- name: Assign new VLAN
  hosts: cisco
  vars:
    NEW_VLAN_ID: 120
  tasks:
    - name: Create VLAN
      ios_vlan:
        vlan_id: "{{ NEW_VLAN_ID }}"
        name: janog-vlan
        state: present
    - name: Add interface to VLAN
      ios_vlan:
        vlan_id: "{{ NEW_VLAN_ID }}"
        interfaces:
          - GigabitEthernet0/4
          - GigabitEthernet0/5
```

宣言型プログラミング
どういう状態にするのか

← Module(iso_vlan)

新しいVLAN(ID:120)を作成

GigabitEth0/4,0/5に割り当てる

ios_vlan - Manage VLANs on IOS network devices

New in version 2.5.

- Synopsis
- Parameters
- Notes
- Examples
- Return Values
- Status
  - Support
  - Author

Synopsis

- This module provides declarative management of VLANs on Cisco IOS network devices.

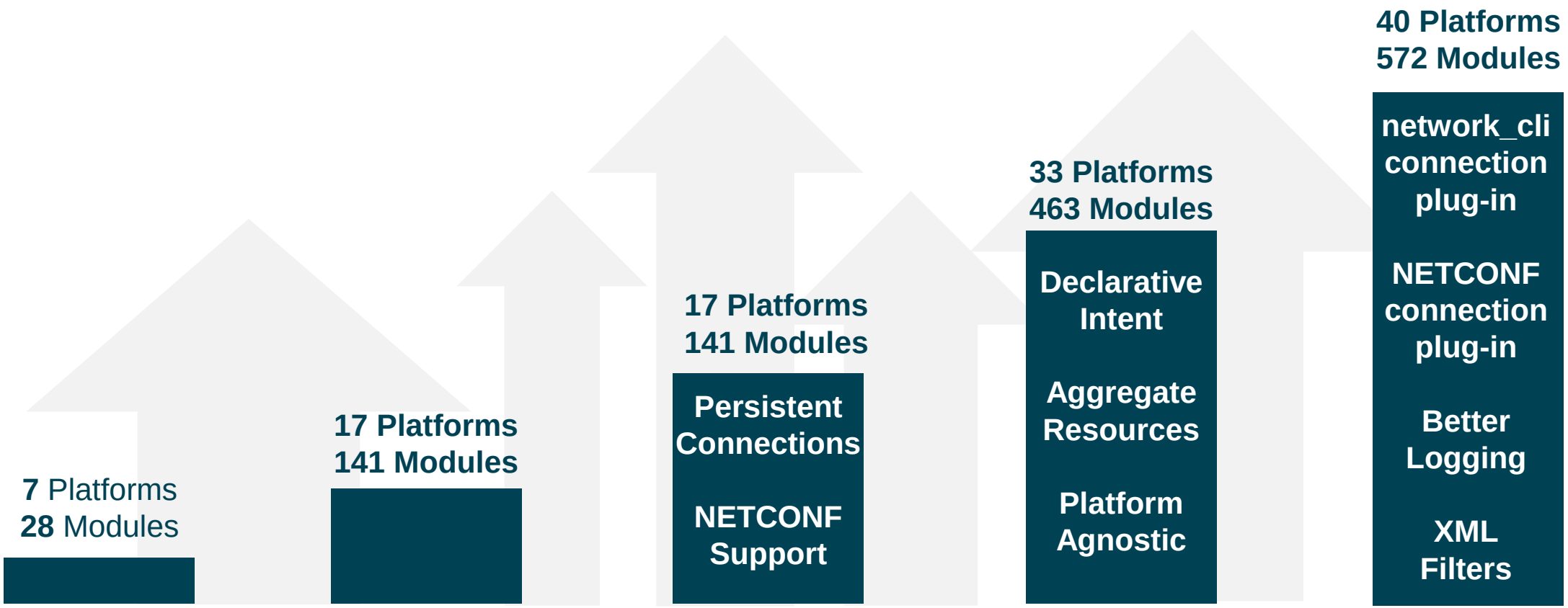
Parameters

Parameter	Choices/Defaults	Comment
aggregate		List of VLANs definitions.
associated_interfaces (added in 2.5)		This is a intent option and checks the operational state of the value in the `associated_interfaces` does not match with the result in failure.

https://docs.ansible.com/ansible/latest/modules/ios_vlan_module.html

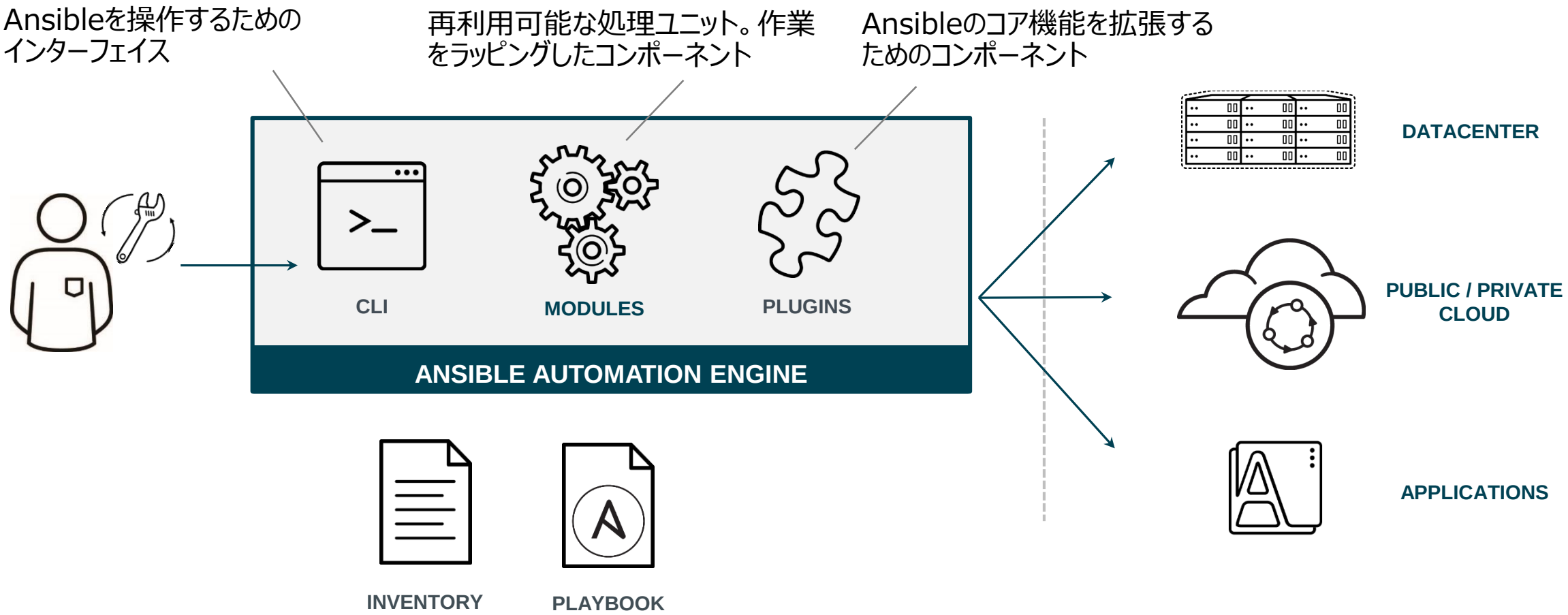
ネットワークモジュールの拡大

Ansible Network Progress



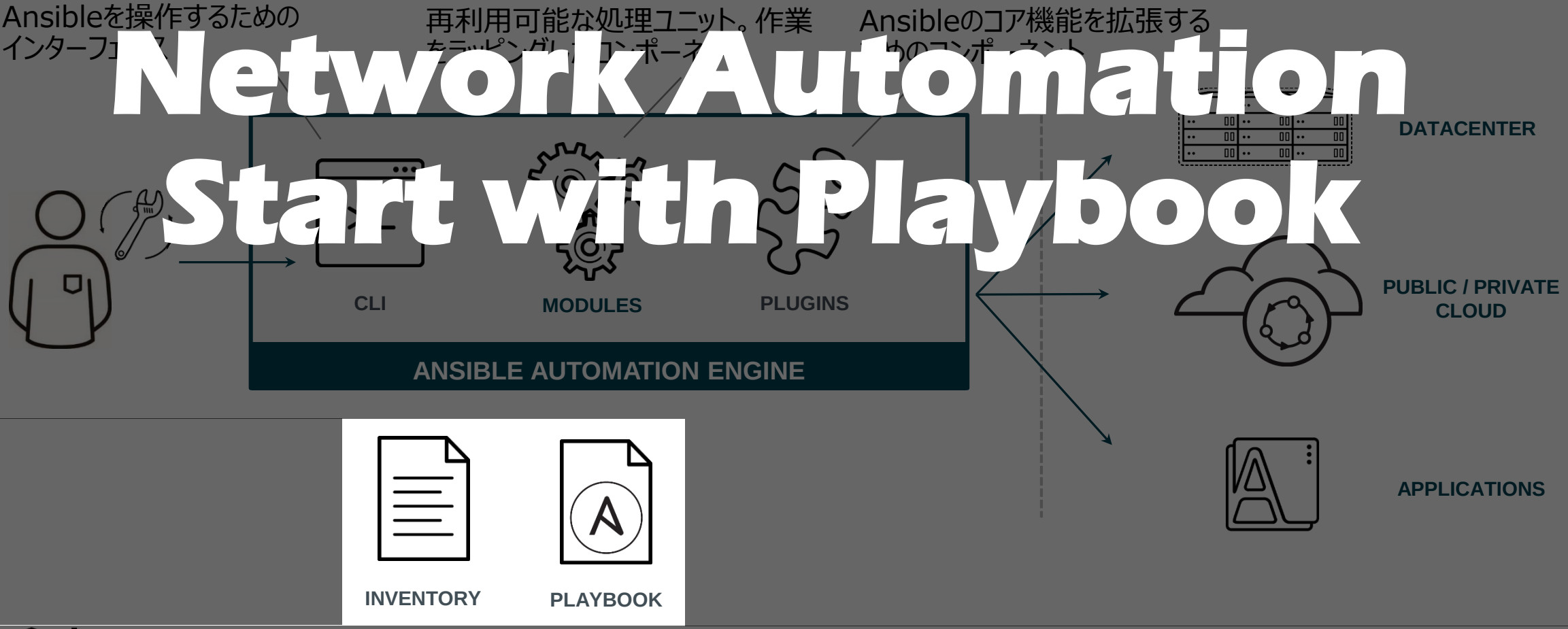
Ansibleのコアコンポーネント

Ansible Automation Engine



Ansibleのコアコンポーネント

Ansible Automation Engine



Network Automation Start with Playbook

Playbookを理解する

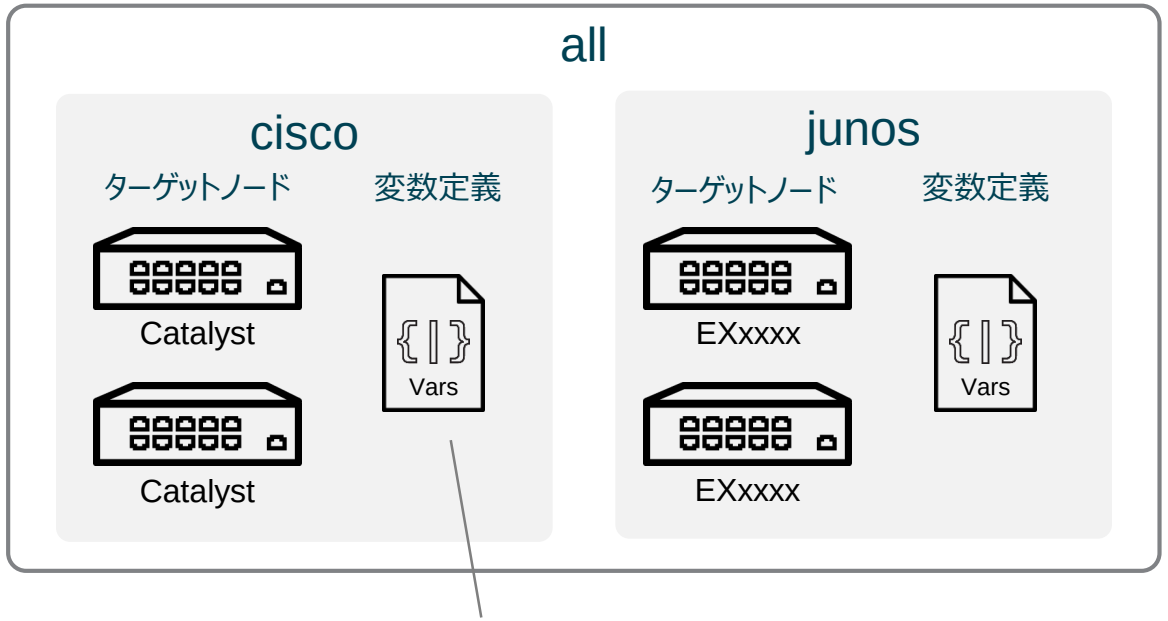
AnsibleのPlaybookが読めるまで

Inventory

Static Inventory

Inventoryとは、処理対象サーバの接続情報を記載したファイルです。

```
# ciscoグループを定義
[cisco]
csr01 ansible_host=192.168.1.1
csr02 ansible_host=192.168.1.2

# junosグループを定義
[junos]
192.168.100.1 # EXxxxx 1台目
192.168.100.2 # EXxxxx 2台目

# ciscoグループ共通の変数定義
[cisco:vars]
# Connection Plugin
ansible_connection=network_cli
# 対象OS
ansible_network_os=ios
…
```



共通の接続情報などはInventoryに設定しておく

グループ化されたターゲットノードは同時に実行されます

Ansible Ad-Hoc Command

```
$ ls ./inventory/hosts

[Usage] ansible <Target Host> -i <Inventory Files> -m <Module>
$ ansible cisco -i inventory -m ios_command -a "commands='show version'"
csr01 | SUCCESS => {
    "changed": false,
    "stdout": [
        "Cisco IOS XE Software, Version 16.04.01¥nCisco IOS Software [Everest],
CSR1000V Software (X86_64_LINUX_IOSD-UNIVERSALK9-M), Version 16.4.1, RELEASE SOFTWARE
(fc3)¥nTechnical Support: http://www.cisco.com/techsupport¥nCopyright (c) 1986-2016 by
Cisco Systems, Inc.¥nCompiled Wed 30-Nov-16 22:13 by mcpre¥n¥n¥nCisco IOS-XE software,
Copyright (c) 2005-2016 by cisco Systems, Inc.¥nAll rights reserved.  Certain
components of Cisco IOS-XE software are¥nlicensed
```

YAMLの規則

Playbook is YAML

```yaml
- name: Assign new VLAN
  hosts: cisco
  vars:
    NEW_VLAN_ID: 120
  tasks:
    - name: Create VLAN
      ios_vlan:
        vlan_id: "{{ NEW_VLAN_ID }}"
        name: janog-vlan
        state: present
    - name: Add interface to VLAN
      ios_vlan:
        vlan_id: "{{ NEW_VLAN_ID }}"
        interfaces:
          - GigabitEthernet0/4
          - GigabitEthernet0/5
```

- Title: YAMLの掟

Rules:
- Rule1: 二種類の文法を主に使用する
  type:
  - シーケンス
    (List形式でデータ構造を表す)
  - マッピング
    (Key Value形式でデータ構造を表す)
- Rule2: コメントアウトは [#]
- Rule3: 入れ子構造はインデントで表す
  attention:
  - インデントにタブ記号は使えない
  - インデントは空白2文字
- Rule4: データ型の宣言は不要

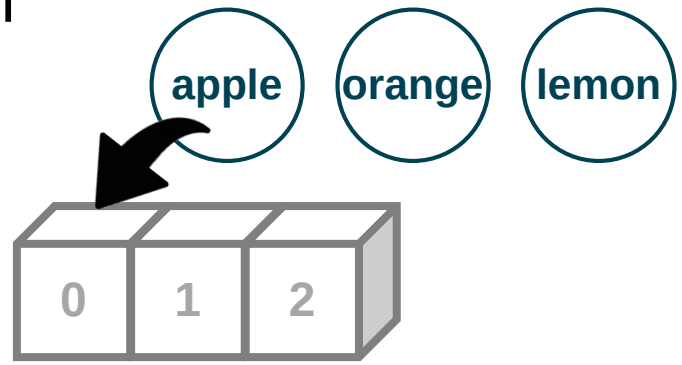
YAMLの文法
シーケンスとマッピング

シーケンス(List)

書式）- <value>

ハイフン+スペース

例）
- apple
- orange
- lemon

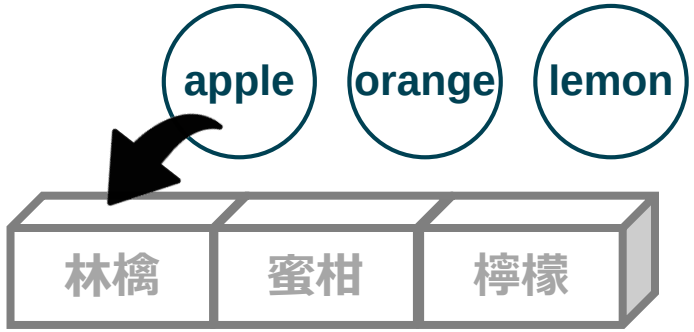


マッピング(Dictionary)

書式）<key>: <value>

コロン+スペース

例）
林檎: apple:
蜜柑: orange
檸檬: lemon



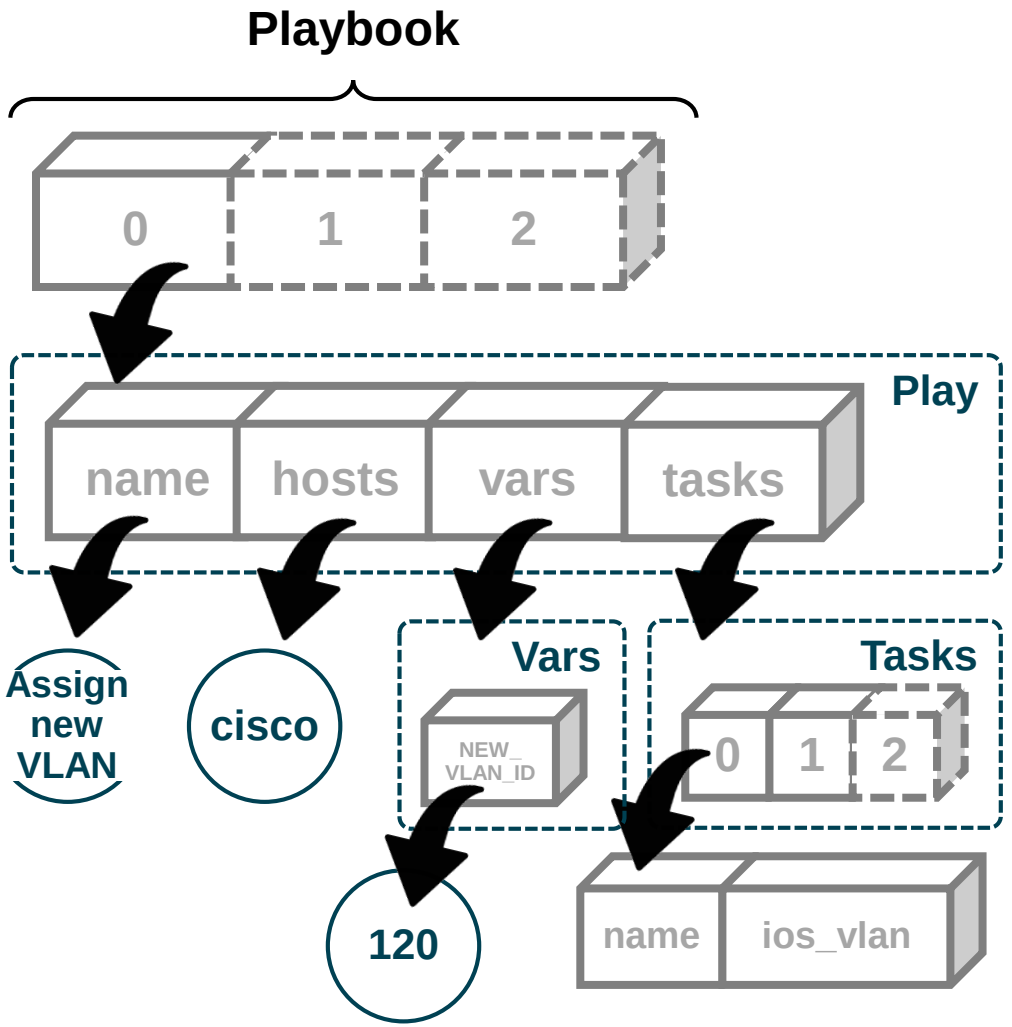
YAMLの文法

YAMLの入れ子構造を理解する

```
- name: Assign new VLAN
  hosts: cisco
  vars:
    NEW_VLAN_ID: 120
  tasks:
    - name: Create VLAN
      ios_vlan:
        vlan_id: "{{ NEW_VLAN_ID }}"
        name: janog-vlan
        state: present
    - name: Add interface to VLAN
      ios_vlan:
        vlan_id: "{{ NEW_VLAN_ID }}"
        interfaces:
          - GigabitEthernet0/4
          - GigabitEthernet0/5
```

(1) シーケンスの中にマッピングをネスト
(2) マッピングの中にシーケンスをネスト

(1)
(1)
(2)



Playbookの基本
セクションと関係性

```
- name: Assign new VLAN
  hosts: cisco                          Target Section
  vars:                                 Vars Section
    NEW_VLAN_ID: 120
  tasks:                                Task Section
    - name: Create VLAN
      ios_vlan:
        vlan_id: "{{ NEW_VLAN_ID }}"
        name: janog-vlan
        state: present
    - name: Add interface to VLAN
      ios_vlan:
        vlan_id: "{{ NEW_VLAN_ID }}"
        interfaces:
          - GigabitEthernet0/4
          - GigabitEthernet0/5
```

Target Section

対象機器やグループを指定するセクション。
また、接続オプションなどを定義する。

Vars Section (※省略可)

独自の変数を指定するセクション。
マッピング形式で変数定義を行い、Keyを指定することで、変数値を参照できる。

Task Section

Playで実施する個別のタスクを指定するセクション。
上から定義した順で実行されていく。

変数について
変数の定義と参照

変数の定義

Playの中や、Taskの中で変数を定義できる。

```
- name: Assign new VLAN
  hosts: cisco
  vars:
    NEW_VLAN_ID: 120
```

変数の参照

変数の参照には、Jinja2テンプレートエンジンを利用して呼び出す。

```
- name: Create VLAN
  ios_vlan:
    vlan_id: “{{ NEW_VLAN_ID }}”   ← 120
    name: janog-vlan
    state: present
```

Jinja2とは

Python用のテキストベースのテンプレートエンジンであり、HTMLやXMLなどの生成に利用される。

基本文法は以下の2通り。

{{…}} で囲むことにより、変数の参照が可能
{%…%}で囲むことにより、制御構文

その他、変数参照時のFilterなども用意されている。
- {{ var|first }} リストの始めを取得
- {{ var|min }} リストの最小値を取得
- {{ var|max }} リストの最大値を取得
- {{ var|random }} ランダム値を取得
- {{ var|hash(‘sha1’) }} sha1値を取得
- {{ var|basename }} フォルダパス取得

などなど
https://docs.ansible.com/ansible/2.5/user_guide/playbooks_filters.html

主な変数定義

いろいろなところで変数が取扱できる

ファクト変数

機器情報を取得した変数

```
- name: Gather IOS Facts
  ios_facts:
    gather_subset:
      - config
```

レジスター変数

タスクを実行した戻り値を取得した変数

```
- name: Gather IOS Facts
  ios_facts:
    gather_subset:
      - config
  register: ios_result

- name: Debug Register Var
  debug:
    var: ios_result
```

タスク変数

タスクの定義の中に設定した変数

```
- name: Create VLAN
  ios_vlan:
    vlan_id: “{{ VLAN_ID }}”
    name: janog-vlan
    state: present
  vars:
    VLAN_ID: 120
```

Ansible Playbook Command

Execute Playbook

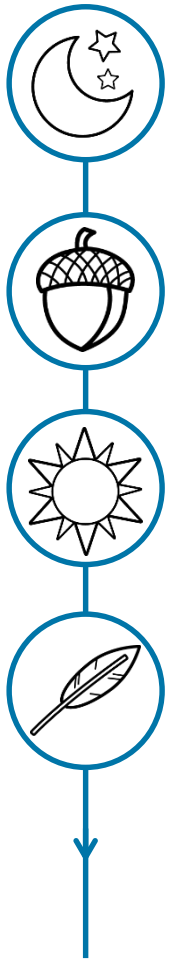
```
$ ls ./inventory/hosts

[Usage] ansible-playbook -i <Inventory Files> <Playbook File>
$ ansible-playbook -i inventory site.yml

TASK [Command Sample for IOS]
*********************************************************************************
ok: [csr01]

TASK [Gathering Facts]
*********************************************************************************
ok: [csr01]

PLAY RECAP
*********************************************************************************
csr01                      : ok=3    changed=0    unreachable=0    failed=0
```

まとめ

自動化とAnsible

まとめ

Automation with Ansible



ネットワークも自動化の時代になってきているが、自動化による弊害も多いので
自動化の範囲とメリットを検討すべき

簡単に始められ、大きな効果が得られるAnsibleは、ネットワーク自動化にも大きな影響がある。

ネットワークモジュールが拡大しており、よりベンダーニュートラル、かつ標準化しやすい製品になっている。

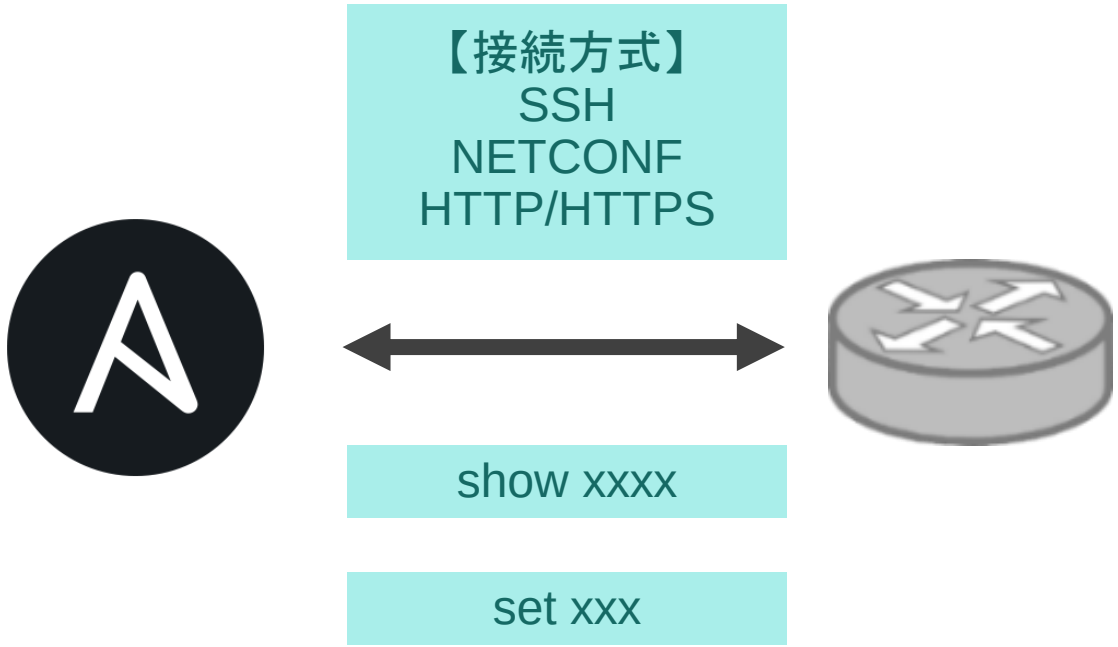
YAMLさえ理解できれば、Playbookの理解もでき、すぐに書けるようになる。

ネットワーク対応の基本1
なにができるか

接続方式とできること

– 接続方式
  – SSH、NETCONF、HTTP/HTTPS（API）などによってネットワーク機器に接続する
    – 利用するモジュールによっては追加ライブラリのインストールが必要（例：NETCONFの場合はncclient）
  – エージェントレスアーキテクチャならでは

– できること（例）
  – 参照（show）コマンドの実行による情報取得
  – 設定コマンドの実行による設定追加・変更・削除



対応プラットフォーム

– Cisco IOS、IOS-XR、NX-OS、Juniper Junos、Arista EOSなど、40以上のプラットフォームに対応
  – 他、サードパーティモジュールとして提供しているベンダーもある
  – モジュールにはコマンドをそのまま指定するタイプや、各オプションをパラメータとして指定するタイプがある

- A10
- ACI
- Aireos
- AOS
- Aruba
- ASA
- Avi
- Bigswitch
- Citrix
- CloudEngine
- CloudVision
- CNOS
- Cumulus
- DellOS10
- DellOS6
- DellOS9
- EdgeOS
- ENOS
- EOS
- EXOS
- F5
- Fortimanager
- FortiOS
- Illumos
- IOS
- IOS-XR
- IronWare
- Junos
- Meraki
- Netact
- Netscaler
- Netvisor
- NSO
- Nuage
- NXOS
- ONYX
- Ordnance
- OVS
- PAN-OS
- Radware
- SLX-OS
- SR OS
- VyOS

ネットワークモジュール一覧 https://docs.ansible.com/ansible/latest/modules/list_of_network_modules.html

Ansible を利用するメリット（Tera Termマクロ との比較）

– Ansible の他の機能と連携しやすい
  – 例:
    – コマンド出力結果を copy モジュールに渡して、ファイルに保存
    – テンプレート機能を利用して、コンフィグを生成して投入
    – 監視サーバーに登録されているホスト情報を接続に利用

– ログインやモード変更の処理を簡略化できる
  – 認証情報を定義しておくだけで自動でログイン、ログアウト
  – 設定系モジュールであれば暗黙的にコンフィグレーションモードへ移行

– コマンド投入エラーを標準で検出できる
  – エラーの検出処理がAnsible モジュールに組み込まれている
  – 自前で「どのようなプロンプトが返ってきたら正常か」という指定は不要

ネットワーク対応の基本2
ネットワークモジュール固有のポイント

ネットワークモジュールで利用するコネクションタイプ

- Ansible 2.5 以降ネットワークモジュール用のコネクションタイプが用意された
  - network_cli: Cisco IOS、Arista EOS、VyOS など向け
  - netconf: Juniper Junos 向け
  - httpapi: Cisco NX-OS、Arista EOS の HTTP(S) API アクセス向け（Ansible 2.6 以降）
- Ansible 2.4まで
  - local: ネットワークモジュール用ではないが、性質上こちらを利用するモジュールもある

Settings by Platform

		ansible_connection: settings available			
Network OS	ansible_network_os:	network_cli	netconf	httpapi	local
Arista EOS*	eos	in v. >=2.5	N/A	in v. >=2.6	in v. >=2.4
Cisco ASA	asa	in v. >=2.5	N/A	N/A	in v. >=2.4
Cisco IOS*	ios	in v. >=2.5	N/A	N/A	in v. >=2.4
Cisco IOS XR*	iosxr	in v. >=2.5	N/A	N/A	in v. >=2.4
Cisco NX-OS*	nxos	in v. >=2.5	N/A	in v. >=2.6	in v. >=2.4
F5 BIG-IP	N/A	N/A	N/A	N/A	in v. >=2.0
F5 BIG-IQ	N/A	N/A	N/A	N/A	in v. >=2.0
Junos OS*	junos	in v. >=2.5	in v. >=2.5	N/A	in v. >=2.4
Nokia SR OS	sros	in v. >=2.5	N/A	N/A	in v. >=2.4
VyOS*	vyos	in v. >=2.5	N/A	N/A	in v. >=2.4

https://docs.ansible.com/ansible/latest/network/user_guide/platform_index.html#settings-by-platform

インベントリファイルで network_cli を指定する例

```
[ios:vars]
ansible_connection=network_cli
```

Playbookで network_cli を指定する例

```
- hosts: ios
  connection: network_cli
```

ネットワークモジュール用の変数

– ansible_network_os 変数
  – プラットフォームに応じて「ios」「junos」「eos」「vyos」等の値を設定する必要あり

Settings by Platform

		`ansible_connection:` settings available			
Network OS	`ansible_network_os:`	network_cli	netconf	httpapi	local
Arista EOS*	`eos`	in v. >=2.5	N/A	in v. >=2.6	in v. >=2.4
Cisco ASA	`asa`	in v. >=2.5	N/A	N/A	in v. >=2.4
Cisco IOS*	`ios`	in v. >=2.5	N/A	N/A	in v. >=2.4
Cisco IOS XR*	`iosxr`	in v. >=2.5	N/A	N/A	in v. >=2.4
Cisco NX-OS*	`nxos`	in v. >=2.5	N/A	in v. >=2.6	in v. >=2.4
F5 BIG-IP	N/A	N/A	N/A	N/A	in v. >=2.0
F5 BIG-IQ	N/A	N/A	N/A	N/A	in v. >=2.0
Junos OS*	`junos`	in v. >=2.5	in v. >=2.5	N/A	in v. >=2.4
Nokia SR OS	`sros`	in v. >=2.5	N/A	N/A	in v. >=2.4
VyOS*	`vyos`	in v. >=2.5	N/A	N/A	in v. >=2.4

https://docs.ansible.com/ansible/latest/network/user_guide/platform_index.html#settings-by-platform

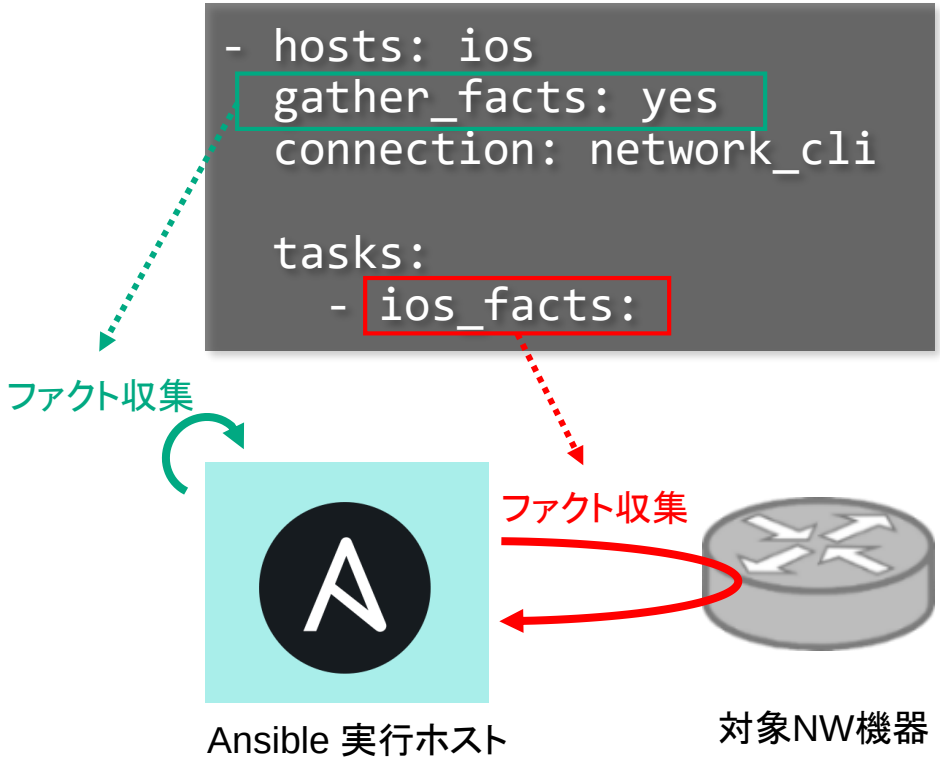
インベントリファイルで指定する例

```
[ios:vars]
ansible_connection=network_cli
ansible_network_os=ios
```

ネットワークモジュール利用時のファクトの収集方法

- 対象ネットワーク機器のファクト（システム情報）を収集するためには「*_facts」モジュールを利用する
  - 「gather_facts: yes」ではAnsible実行ホスト自身の情報となる
    - 不要であれば「gather_facts: no」を指定する
- *_facts モジュールの例
  - ios_facts
  - junos_facts
  - eos_facts
  - vyos_facts
- 収集できるシステム情報例（ios_facts の場合）
  - コンフィグ
  - ホスト名
  - インターフェース情報
  - ファームウェア情報
  - LLDPネイバー情報

```
- hosts: ios
  gather_facts: yes
  connection: network_cli

  tasks:
    - ios_facts:
```



特権モードの指定

- IOS や EOS などで enable コマンドによる特権モードへの移行が必要な場合は変数で指定する
  - ansible_become 変数
    - 特権モードへ移行が必要かどうか（yes/no: デフォルトno）
  - ansible_become_method 変数
    - 特権モードへ移行するコマンド（network_cli では enable のみ指定可）
  - ansible_become_pass 変数
    - 特権モード移行時に必要なパスワード
- 対象ネットワーク機器へのログインユーザーに特権が付与されている場合は指定不要

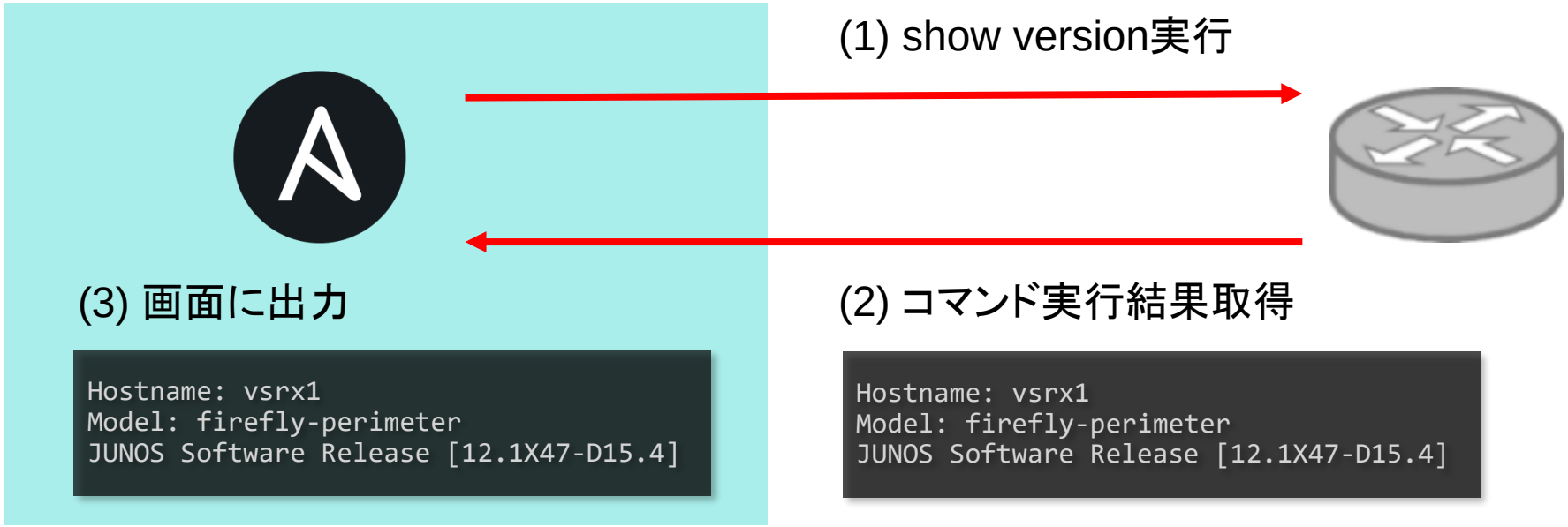
インベントリファイルで指定する例

```
[ios:vars]
ansible_connection=network_cli
ansible_network_os=ios
ansible_become=yes
ansible_become_method=enable
ansible_become_pass=enablepass999
```

情報取得サンプル【1-1】
状態の表示

サンプル1-1: 状態の表示 （流れ）

- show version コマンド実行結果をAnsibleホストの画面に表示クアップ



サンプル1-1: 状態の表示 （インベントリとPlaybook）

- インベントリファイル （inventory）

```
[junos]
172.16.0.1

[junos:vars]
ansible_connection=netconf
ansible_network_os=junos
ansible_user=testuser
ansible_ssh_pass=testpassword99
```

コネクションタイプやOS、
認証情報を指定

このインベントリファイルは、以降のサンプルでも共通

- Playbook （show01.yml）

```
- hosts: junos
  gather_facts: no

  tasks:
    - name: show command test
      junos_command:
        commands:
          - show version
      register: result

    - name: debug output
      debug:
        msg: "{{ result.stdout_lines[0] }}"
```

junosコマンド実行モジュール
junos_command(※1) で
実行したいコマンドを指定

実行結果を変数 result に格納

debugモジュールで
コマンド実行結果を出力

※1 junos_command モジュール詳細
https://docs.ansible.com/ansible/latest/modules/junos_command_module.html

サンプル1-1: 状態の表示 （実行結果）

- 実行結果

ansible-playbook コマンドを実行

```
[vagrant@centos7 janog]$ ansible-playbook -i inventory show01.yml

PLAY [junos] *******************************************************************

TASK [show command test] *******************************************************

ok: [172.16.0.1]

TASK [debug output] ************************************************************

ok: [172.16.0.1] => {
    "msg": [
        "Hostname: vsrx1",
        "Model: firefly-perimeter",
        "JUNOS Software Release [12.1X47-D15.4]"
    ]
}

PLAY RECAP *********************************************************************
172.16.0.1                 : ok=2    changed=0    unreachable=0    failed=0
```

show version 実行結果が表示される

サンプル1-1: 状態の表示 （showコマンド実行結果の構造について補足）

```
    - name: debug output
      debug:
      msg: "{{ result }}"
```

全体を指定

```
    - name: debug output
      debug:
      msg: "{{ result.stdout_lines[0] }}"
```

実行結果を1行1リスト要素で格納した
stdout_lines 内の0番目
（commandsオプションの最初に指定したコマンド）

```
"msg": {
        "changed": false,
        "failed": false,
        "stdout": [
            "Hostname: vsrx1¥nModel: firefly-perimeter¥nJUNOS Software Release [12.1X47-D15.4]"
        ],
        "stdout_lines": [
            [
                "Hostname: vsrx1",
                "Model: firefly-perimeter",
                "JUNOS Software Release [12.1X47-D15.4]"
            ]
        ]
    }
```

情報取得サンプル【1-2】
コンフィグのバックアップ

サンプル1-2: コンフィグのバックアップ （流れ）

- show configuration コマンド実行結果をファイルにバックアップ

サンプル1-2: コンフィグのバックアップ （Playbook）

- Playbook （show02.yml）

```
- hosts: junos
  gather_facts: no

  tasks:
    - name: show command test
      junos_command:
        commands:
          - show configuration
      register: result

    - name: save config to file
      copy:
        content: "{{ result.stdout[0] }}"
        dest: "show_config_{{ inventory_hostname }}.txt"
```

junosコマンド実行モジュール junos_command で実行したいコマンドを指定

copyモジュールでコマンド実行結果をファイルに保存する

ファイル名に名インベントリ名（ここではIPアドレス）を含ませる

サンプル1-2: コンフィグのバックアップ （実行結果）

- 実行結果

ansible-playbook コマンドを実行

```
[vagrant@centos7 janog]$ ansible-playbook -i inventory show02.yml

PLAY [junos] *******************************************************************

TASK [show command test] *******************************************************

ok: [172.16.0.1]

TASK [save config to file] *****************************************************

changed: [172.16.0.1]

PLAY RECAP *********************************************************************
172.16.0.1                 : ok=2    changed=1    unreachable=0    failed=0
```

コンフィグがファイルとして
新たに保存されたログ

サンプル1-2: コンフィグのバックアップ （ファイル内容の確認）

- 実行結果

インベントリ名（ここではIPアドレス）を含むファイル名で保存されている

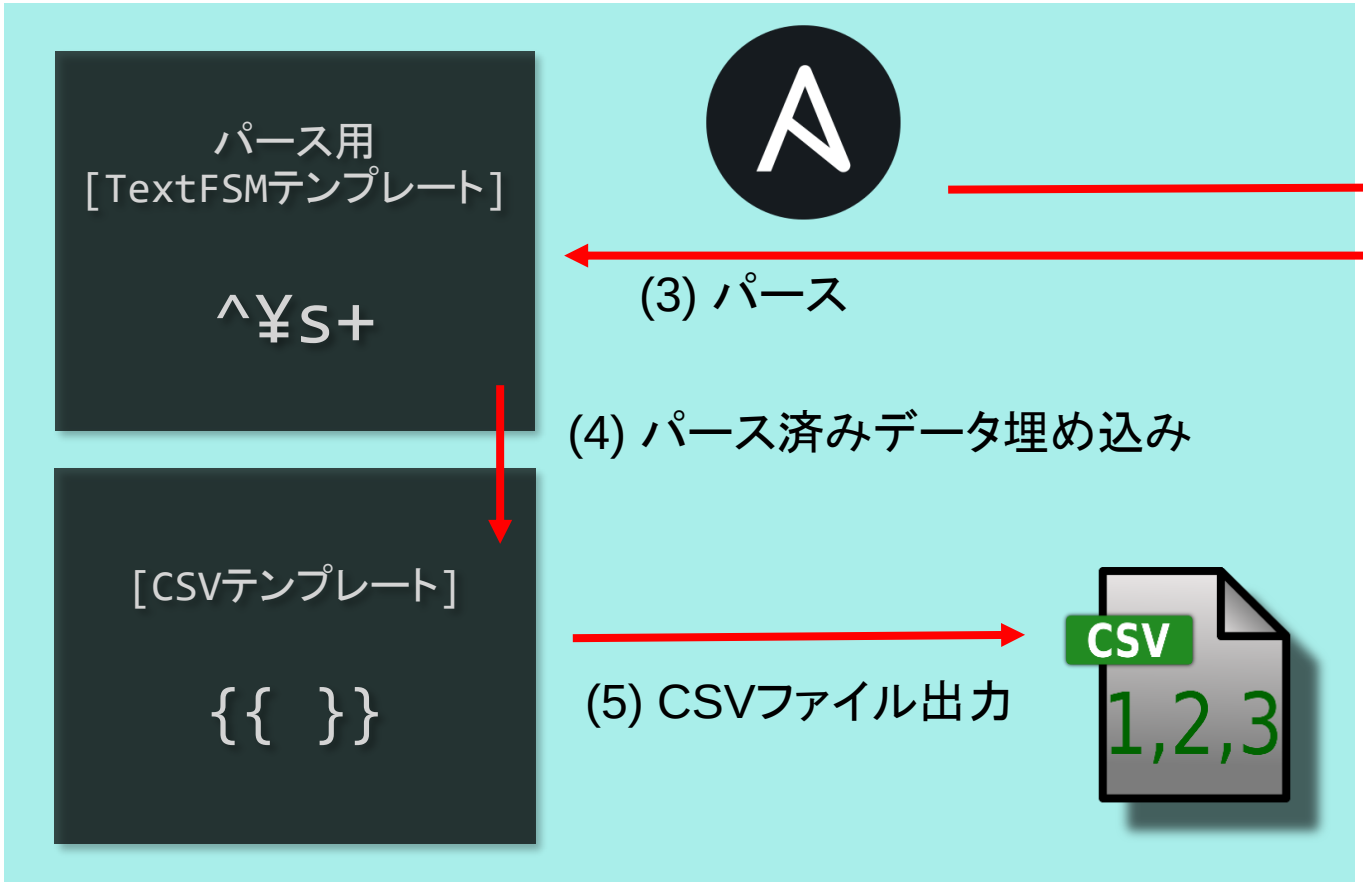
```
[vagrant@centos7 janog]$ cat show_config_172.16.0.1.txt
## Last changed: 2018-06-28 05:45:50 UTC
version 12.1X47-D15.4;
system {
    host-name vsrx1;
    root-authentication {
        encrypted-password "$1$nq.N1UsY$Jx...(略)...";
        ssh-rsa "ssh-rsa AAAAB3NzaC1yc2...(略)....";
    }
...(略)....
 ge-0/0/1 {
        unit 0 {
            family inet {
                address 172.16.0.1/24;
            }
        }
    }
...(略)....
}
```

show configuration 実行結果

情報取得サンプル【1-3】
showコマンド結果のCSV出力

サンプル1-3: showコマンド結果のCSV出力 （流れ）

- show interfaces コマンド実行結果をパースして、CSVファイルに出力




```
Physical interface: ge-0/0/0, Enabled, Physical link is Up
  Interface index: 134, SNMP ifIndex: 507
  Link-level type: Ethernet, MTU: 1514, Link-mode: Full-duplex, Speed: 1000mbps,…
  Flow control: Enabled, Auto-negotiation: Enabled, Remote fault: Online
  Device flags   : Present Running
  Interface flags: SNMP-Traps Internal: 0x4000
  Link flags     : None
  CoS queues     : 8 supported, 8 maximum usable queues
  Current address: 08:00:27:ae:f4:51, Hardware address: 08:00:27:ae:f4:51
  Last flapped   : 2018-06-28 05:45:29 UTC (04:54:28 ago)
  Input rate     : 3432 bps (7 pps)
  Output rate    : 4256 bps (4 pps)
  Active alarms  : None
  Active defects : None
  Interface transmit statistics: Disabled

  Logical interface ge-0/0/0.0 (Index 69) (SNMP ifIndex 509)
    Flags: SNMP-Traps 0x4000 Encapsulation: ENET2
    Input packets : 522
    Output packets: 580
    Security: Zone: Null
    Protocol inet, MTU: 1500
…
```

サンプル1-3: showコマンド結果のCSV出力 （Playbook）

- Playbook （show03.yml）

```
- hosts: junos
  gather_facts: no

  tasks:
    - name: show command test
      junos_command:
        commands:
          - show interfaces
      register: result

    - name: output csv file
      template:
        src: "./template_interface_junos.txt"
        dest: "./result_interface.csv"
      vars:
        parsed_result: "{{ result.stdout[0] | parse_cli_textfsm('juniper_junos_show_interfaces.template') }}"
```

junosコマンド実行モジュール junos_command で
実行したいコマンドを指定

templateモジュールで、CSVテンプレートファイル名と、
出力先ファイル名を指定

コマンド実行結果をTextFSMテンプレートに渡して、
パースされた結果を変数 parsed_result に格納

※ parse_cli_textfsm フィルターの利用にはあらかじめTextFSMのインストール（pip install TextFSM）が必要

サンプル1-3: showコマンド結果のCSV出力 （CSVテンプレート）

- CSVテンプレート （template_interface_junos.txt）

show interfaces 実行結果
がパースされた内容が入る

```
"INTERFACE","ADMIN_STATE","LINK_STATUS","HARDWARE_TYPE"
{% for r in parsed_result %}
"{{ r['INTERFACE'] }}","{{ r['ADMIN_STATE'] }}","{{ r['LINK_STATUS'] }}","{{ r['HARDWARE_TYPE'] }}"
{% endfor %}
```

CSVのヘッダー

for 文による繰り返し

r[‘XXXX’] に各項目の値が埋め込まれる

※ テンプレートエンジン jinja2 を利用

サンプル1-3: showコマンド結果のCSV出力 （TextFSMテンプレート）

- TextFSMテンプレート （ juniper_junos_show_interfaces.template ）

```
Value Required INTERFACE (¥S+)
Value LINK_STATUS (¥w+)
Value ADMIN_STATE (¥S+)
Value HARDWARE_TYPE (¥S+)
Value MTU (¥d+|Unlimited)

Start
  ^¥s+Logical¥s+interface¥s+${INTERFACE}
  ^Physical¥s+interface:¥s+${INTERFACE},¥s+${ADMIN_STATE},¥s+Physical¥s+link¥s+is¥s+${LINK_STATUS}
  ^.*ype:¥s+${HARDWARE_TYPE},.*MTU:¥s+${MTU}.* -> Record
  ^.*MTU:¥s+${MTU}.* -> Record
  ^.*flags -> Record

EOF
```

TextFSMというパーサー用のテンプレートファイル
ここでは Junos の「show interfaces」用のものを利用

https://raw.githubusercontent.com/networktocode/ntc-templates/master/templates/juniper_junos_show_interfaces.template

サンプル1-3: showコマンド結果のCSV出力 （実行結果）

- 実行結果

ansible-playbook コマンドを実行

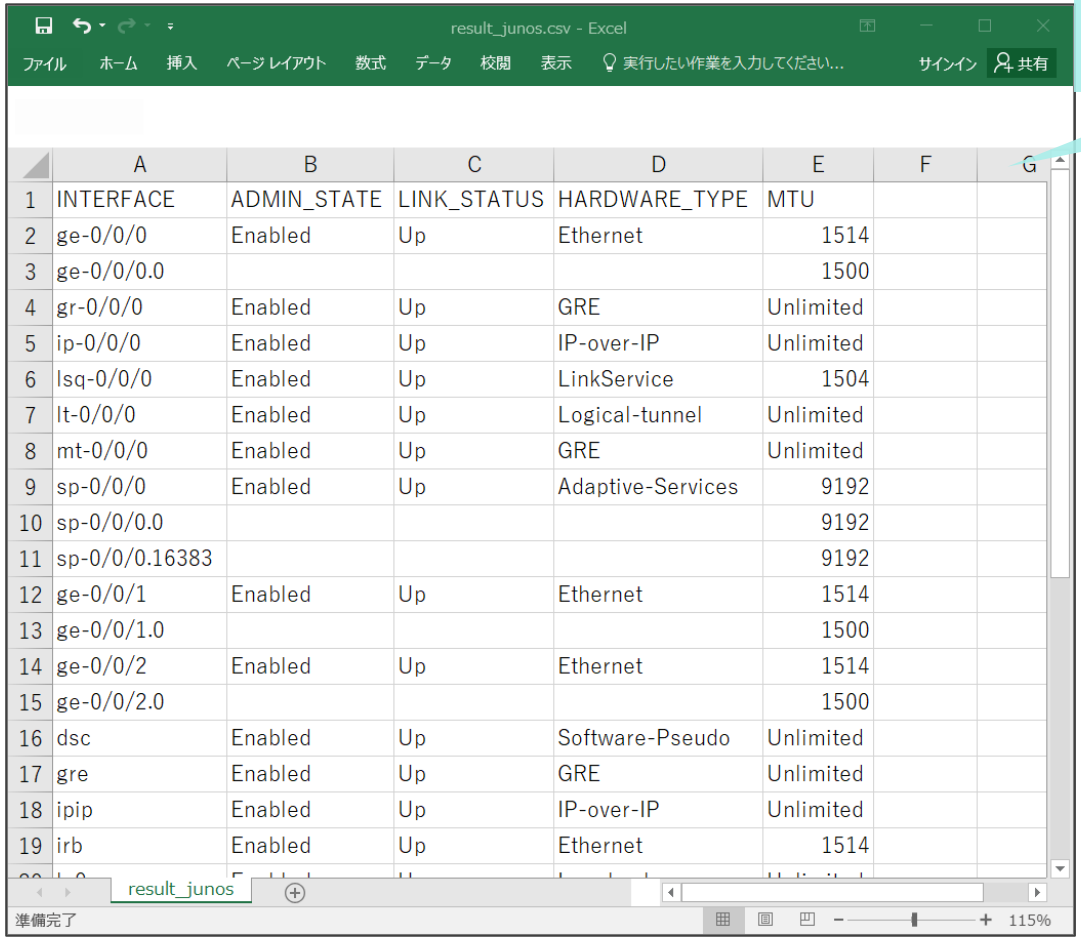
```
[vagrant@centos7 janog]$ ansible-playbook -i inventory show03.yml

PLAY [junos] *******************************************************************

TASK [show command test] *******************************************************

ok: [172.16.0.1]

TASK [output csv file] *********************************************************

changed: [172.16.0.1]

PLAY RECAP *********************************************************************
172.16.0.1                 : ok=2    changed=1    unreachable=0    failed=0
```

CSVファイルが生成された

サンプル1-3: showコマンド結果のCSV出力 （CSVファイル内容の確認）

- 出力CSVファイル （result_interface.csv）

CSVテンプレートの書式に
パースされた情報が埋め込まれて
CSVファイルが生成された



	A	B	C	D	E	F	G
1	INTERFACE	ADMIN_STATE	LINK_STATUS	HARDWARE_TYPE	MTU		
2	ge-0/0/0	Enabled	Up	Ethernet	1514		
3	ge-0/0/0.0				1500		
4	gr-0/0/0	Enabled	Up	GRE	Unlimited		
5	ip-0/0/0	Enabled	Up	IP-over-IP	Unlimited		
6	lsq-0/0/0	Enabled	Up	LinkService	1504		
7	lt-0/0/0	Enabled	Up	Logical-tunnel	Unlimited		
8	mt-0/0/0	Enabled	Up	GRE	Unlimited		
9	sp-0/0/0	Enabled	Up	Adaptive-Services	9192		
10	sp-0/0/0.0				9192		
11	sp-0/0/0.16383				9192		
12	ge-0/0/1	Enabled	Up	Ethernet	1514		
13	ge-0/0/1.0				1500		
14	ge-0/0/2	Enabled	Up	Ethernet	1514		
15	ge-0/0/2.0				1500		
16	dsc	Enabled	Up	Software-Pseudo	Unlimited		
17	gre	Enabled	Up	GRE	Unlimited		
18	ipip	Enabled	Up	IP-over-IP	Unlimited		
19	irb	Enabled	Up	Ethernet	1514		

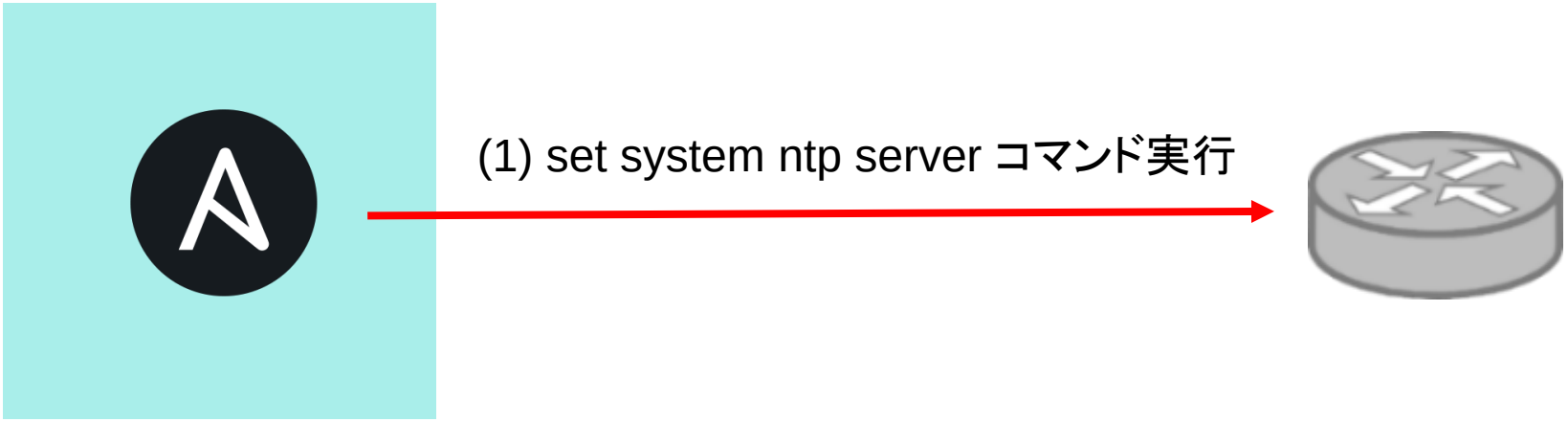
result_junos
準備完了 115%

```
"INTERFACE","ADMIN_STATE","LINK_STATUS","HARDWARE_TYPE","MTU"
"ge-0/0/0","Enabled","Up","Ethernet","1514"
"ge-0/0/0.0","","","","1500"
"gr-0/0/0","Enabled","Up","GRE","Unlimited"
"ip-0/0/0","Enabled","Up","IP-over-IP","Unlimited"
"lsq-0/0/0","Enabled","Up","LinkService","1504"
"lt-0/0/0","Enabled","Up","Logical-tunnel","Unlimited"
"mt-0/0/0","Enabled","Up","GRE","Unlimited"
"sp-0/0/0","Enabled","Up","Adaptive-Services","9192"
"sp-0/0/0.0","","","","9192"
"sp-0/0/0.16383","","","","9192"
"ge-0/0/1","Enabled","Up","Ethernet","1514"
"ge-0/0/1.0","","","","1500"
"ge-0/0/2","Enabled","Up","Ethernet","1514"
"ge-0/0/2.0","","","","1500"
"dsc","Enabled","Up","Software-Pseudo","Unlimited"
"gre","Enabled","Up","GRE","Unlimited"
"ipip","Enabled","Up","IP-over-IP","Unlimited"
"irb","Enabled","Up","Ethernet","1514"
"lo0","Enabled","Up","Loopback","Unlimited"
"lo0.16384","","","","Unlimited"
"lo0.16385","","","","Unlimited"
"lsi","Enabled","Up","LSI","1496"
"mtun","Enabled","Up","GRE","Unlimited"
"pimd","Enabled","Up","PIM-Decapsulator","Unlimited"
"pime","Enabled","Up","PIM-Encapsulator","Unlimited"
"pp0","Enabled","Up","PPPoE","1532"
"ppd0","Enabled","Up","PIM-Decapsulator","Unlimited"
"ppe0","Enabled","Up","PIM-Encapsulator","Unlimited"
"st0","Enabled","Up","Secure-Tunnel","9192"
"tap","Enabled","Up","Interface-Specific","Unlimited"
"vlan","Enabled","Down","VLAN","1518"
```

設定変更サンプル【2-1】
簡単な設定変更

サンプル2-1: 簡単な設定変更 （流れ）

- 参照NTPサーバー設定コマンドを投入



サンプル2-1: 簡単な設定変更 （Playbook）

- Playbook （set01.yml）

```
- hosts: junos
  gather_facts: no

  tasks:
    - name: config test
      junos_config:
        lines:
          - set system ntp server 10.0.0.123
```

Junos設定コマンド実行モジュール
junos_config(※1) で設定したいコマンドを指定

※1 junos_config モジュール詳細
https://docs.ansible.com/ansible/latest/modules/junos_config_module.html

サンプル2-1: 簡単な設定変更 （実行結果）

- 実行結果

ansible-playbook コマンドを実行

設定変更された

```
 [vagrant@centos7 janog]$ ansible-playbook -i inventory set01.yml

PLAY [junos] ***************************************************************

TASK [config test] *********************************************************
changed: [172.16.0.1]

PLAY RECAP
*****************************************************************************
172.16.0.1                 : ok=1    changed=1    unreachable=0    failed=0
```

サンプル2-1: 簡単な設定変更 （作業後コンフィグの確認）

- 実行結果（ネットワーク機器側）

```
root@vsrx1> show configuration system ntp | display set
set system ntp server 10.0.0.123

root@vsrx1>
```

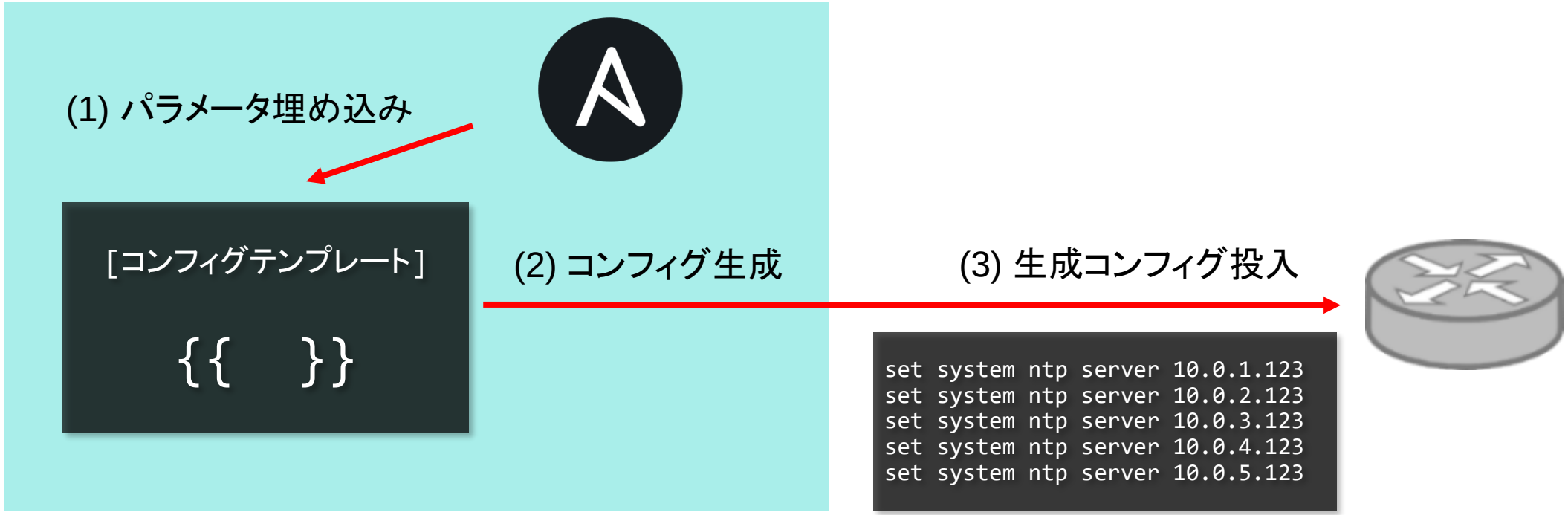
設定が反映された

設定変更サンプル【2-2】

テンプレートを利用した設定変更

サンプル2-2: テンプレートを利用した設定変更 （流れ）

- 参照NTPサーバー設定コマンドを投入



サンプル2-2: テンプレートを利用した設定変更 （Playbook）

- Playbook （set02.yml）

```
- hosts: junos
  gather_facts: no

  tasks:
    - name: config test
      junos_config:
        src: template_ntp_junos.txt

  vars:
    ntp_servers:
      - 10.0.1.123
      - 10.0.2.123
      - 10.0.3.123
      - 10.0.4.123
      - 10.0.5.123
```

Junos設定コマンド実行モジュール junos_config で
流し込みたいコンフィグファイルを指定

NTPサーバーのIPアドレスのリストを
変数「ntp_servers」として定義

サンプル2-2: テンプレートを利用した設定変更 （コンフィグテンプレート）

- コンフィグテンプレート （template_ntp_j.txt ）

```
{% for n in ntp_servers %}
set system ntp server {{ n }}
{% endfor %}
```

Playbook内で定義された変数
ntp_servers の内容が入る

for 文による繰り返し

※ テンプレートエンジン jinja2 を利用

- 生成されるコンフィグ

```
set system ntp server 10.0.1.123
set system ntp server 10.0.2.123
set system ntp server 10.0.3.123
set system ntp server 10.0.4.123
set system ntp server 10.0.5.123
```

サンプル2-2: テンプレートを利用した設定変更 （実行結果）

- 実行結果

ansible-playbook コマンドを実行

設定変更された

```
 [vagrant@centos7 janog]$ ansible-playbook -i inventory set02.yml

PLAY [junos] *******************************************************************

TASK [config test] *************************************************************
changed: [172.16.0.1]

PLAY RECAP
*********************************************************************************
172.16.0.1                 : ok=1    changed=1    unreachable=0    failed=0
```

サンプル2-2: テンプレートを利用した設定変更（作業後コンフィグの確認）

- 実行結果（ネットワーク機器側）

```
root@vsrx1> show configuration system ntp | display set
set system ntp server 10.0.1.123
set system ntp server 10.0.2.123
set system ntp server 10.0.3.123
set system ntp server 10.0.4.123
set system ntp server 10.0.5.123

root@vsrx1>
```

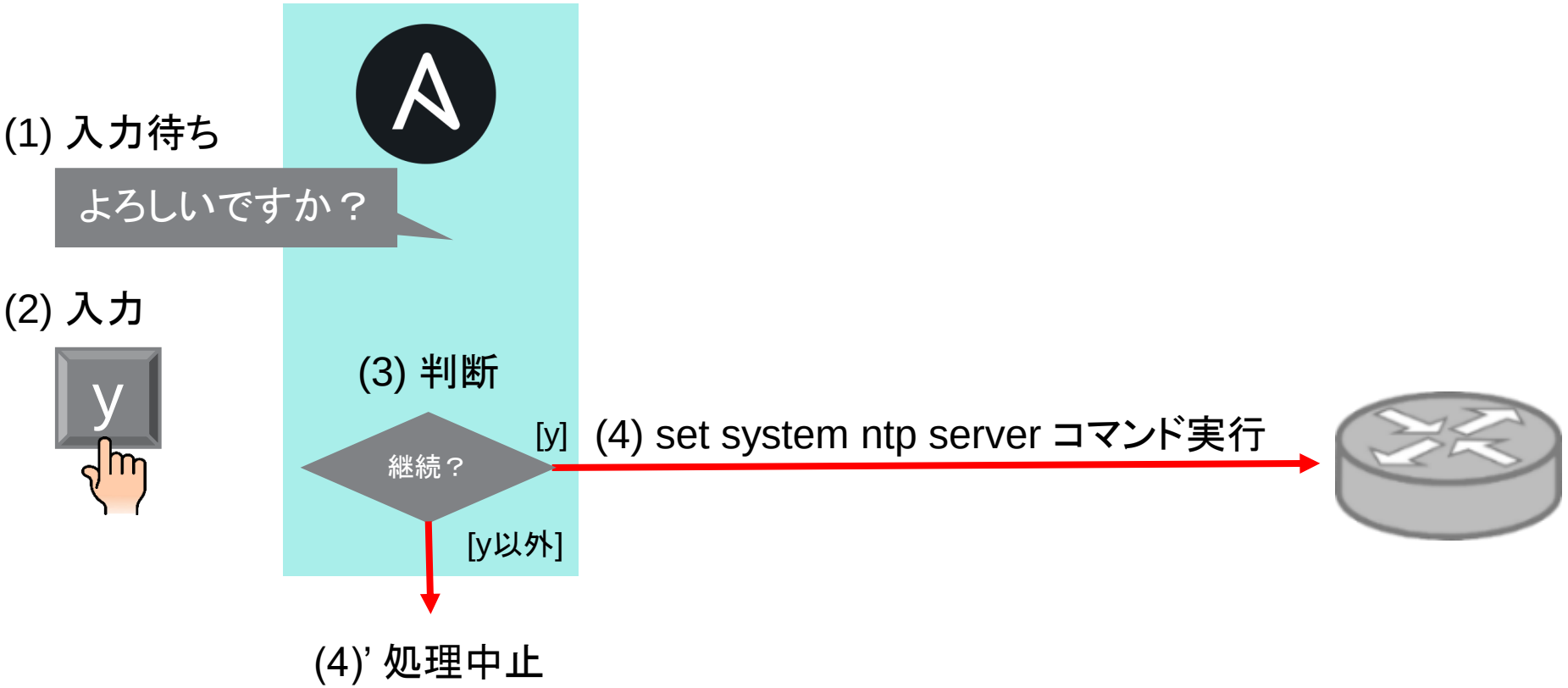
設定が反映された

設定変更サンプル【2-3】

処理継続確認付きの設定変更

サンプル2-3: 処理継続確認付きの設定変更 （流れ）

- 処理継続してよいかの入力を待ち、OKであれば設定コマンドを投入



サンプル2-3: 処理継続確認付きの設定変更 （Playbook）

- Playbook （set03.yml）

```
- hosts: junos
  gather_facts: no

  tasks:
    - name: confirm
      pause:
        prompt: "continue? [y/N]"
      register: input

    - name: abort
      fail:
      when: input.user_input != "y"

    - name: config test
      junos_config:
        lines:
          - set system ntp server 10.0.0.123
```

入力待ち状態で待機するpauseモジュールで表示する文字列

入力された文字が変数 input に入る

入力された文字が「y」でなければ処理を中断

Junos設定コマンド実行モジュール junos_config で設定したいコマンドを指定

サンプル2-3: 処理継続確認付きの設定変更 （実行結果1：処理継続）

- 実行結果（処理継続）

ansible-playbook コマンドを実行

```
[vagrant@centos7 janog]$ ansible-playbook -i inventory set03.yml

PLAY [junos] *******************************************************************

TASK [confirm] *****************************************************************

[confirm]
continue? [y/N]: y
ok: [172.16.0.1]

TASK [abort] *******************************************************************
skipping: [172.16.0.1]

TASK [config test] *************************************************************
changed: [172.16.0.1]

PLAY RECAP *********************************************************************
172.16.0.1                 : ok=2    changed=1    unreachable=0    failed=0
```

y を入力

y が入力されたため中止処理はスキップ

設定変更された

サンプル2-3: 処理継続確認付きの設定変更 （実行結果2:処理中止）

- 実行結果（処理中止）

ansible-playbook コマンドを実行

```
[vagrant@centos7 janog]$ ansible-playbook -i inventory set03.yml

PLAY [junos] *******************************************************************

TASK [confirm] *****************************************************************

[confirm]
continue? [y/N]: N
ok: [172.16.0.1]

TASK [abort] *******************************************************************
fatal: [172.16.0.1]: FAILED! => {"changed": false, "msg": "Failed as requested from task"}

PLAY RECAP *********************************************************************
172.16.0.1                 : ok=1    changed=0    unreachable=0    failed=1
```

N を入力

y 以外 が入力されたため中止処理

ネットワーク対応のまとめ

ネットワーク対応のまとめ

– Ansible は 40以上のネットワークプラットフォームに対応し、参照や設定などができる
– ネットワークモジュール固有のポイントがいくつかある
  – コネクションタイプ、ansible_network_os 変数、ファクト収集方法、特権モードなど
– 他の Ansible の機能と連携して様々な処理ができる
  – サーバー、クラウド、監視ツール、通知、ファイル管理など、他にも様々な用途のモジュールがある

参考資料

- 公式ドキュメント
  - トップ
    - https://docs.ansible.com/
  - Ansible for Network Automation
    - https://docs.ansible.com/ansible/latest/network/index.html
  - ネットワークモジュール一覧
    - https://docs.ansible.com/ansible/latest/modules/list_of_network_modules.html
- 書籍
  - Ansible実践ガイド 第2版
    - https://book.impress.co.jp/books/1117101100
  - Ansible徹底入門
    - https://www.shoeisha.co.jp/book/detail/9784798149943
- その他
  - ネットワークエンジニア的Ansibleの始め方
    - https://www.slideshare.net/akira6592/networkengineermeetsansible-85889620
  - もっと気軽に始めるAnsible（JANOG41.5）
    - https://www.slideshare.net/akira6592/ansibleadhocnetworkautomation

イベント情報

– Ansibleユーザー会
  – 【7/12】Ansible Night in Osaka 2018.07　@大阪
    – https://ansible-users.connpass.com/event/90117/
  – 【7/17】Ansibleもくもく会（第4回） @東京
    – https://ansible-users.connpass.com/event/92701/
  – 【8/3】Ansible Night in Fukuoka 2018.08　@福岡
    – https://ansible-users.connpass.com/event/93620/

– JANOG42 Meeting in Mie
  – 【7/11 16:30 ～】ネットワーク運用自動化BoF 対話編 @多目的ホール

このあと、ここで

Thank you