

Ansibleによる、ネットワークOSに依存しないコマンド作成・設定変更

株式会社エーピーコミュニケーションズ 川名 賢

自己紹介

- 所属
 - 株式会社エーピーコミュニケーションズ
- 名前
 - 川名 賢
- 過去に経験した業務
 - ネットワークの検証
 - ネットワークの自動化 (Ansible/Python)
- 興味ある技術
 - Ansible Automation Platform
 - コンテナ全般
- その他
 - 息子 (0才4ヶ月)
 - 泣き声が聞こえる場合もありますが、何卒ご容赦ください



今回お伝えしたいこと

- Ansibleを実際に導入して感じた、ネットワーク自動化のメリット
- Ansible導入後にすべきこと

目次

1. Ansible実装前

1.1 業務説明

1.2 作業の問題点

1.3 As-Is / To-Be

2. Ansible実装時

2.1 実装の方針

2.2 実装時の問題と対策

3. Ansible実装後

3.1 得られた効果

3.2 自動化によって生じた課題と対策

- まとめ
- おわりに
- 補足資料

諸注意

- Software Design 2022年12月号に掲載した記事の内容紹介です
- **Ansible実装に向けたエピソードがメインテーマです**
 - AnsibleそのものやPlaybookの詳細に関する解説は実施しません

1. Ansible実装前

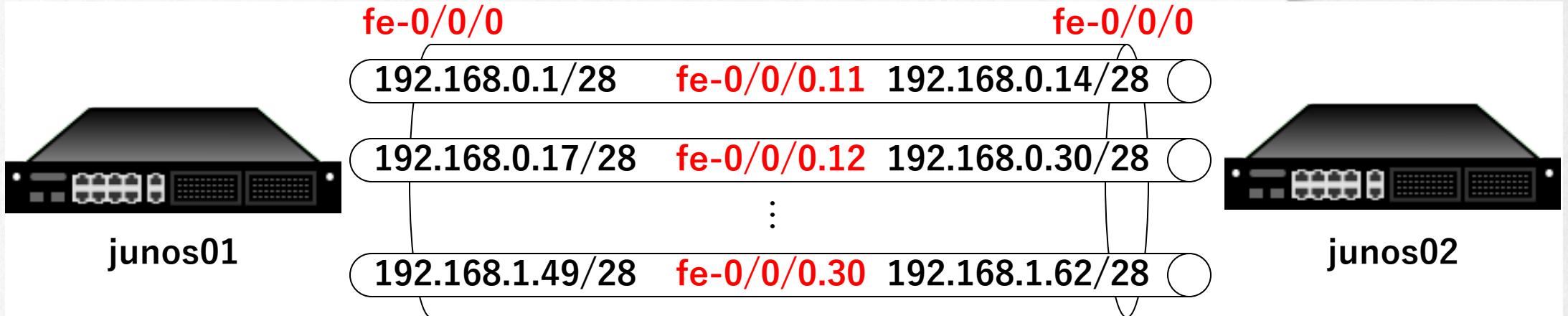
1.1 業務説明

- 主な業務
 - 異なるネットワークOS（Junos/IOS）からなるネットワークの検証
- 特に改善したい作業
 - サブインターフェースを使用したVLANルーティングの設定

1.1 業務説明

サブインターフェースとは

- 1つの物理インターフェースを複数の論理的なインターフェースに分割したもの



1.1 業務説明

- 改善したい理由: 対象機器もコマンドも多く、手作業が大変
 - 機器は30台以上あり、ネットワークOS (Junos/IOS) が異なる
 - 機器ごとにサブインターフェースは100以上あり、コマンドの行数が1万行を超えることもある

1.2 作業の問題点

1. IPアドレス計算にExcelを使っていた
2. ネットワークOSごとにコマンドが異なる

1.2 作業の問題点

1. IPアドレス計算にExcelを使っていた

- Excelでの作業イメージ

	A	B	C	D	E	F	G	H
1								
2	unit	オクテット				プレフィックス		コマンド
3		第1	第2	第3	第4			
4	11	192	168	0	1	28	set interfaces fe-0/0/0 unit 11 family inet address	192.168.0.1/28
5	12	192	168	0	17	28	set interfaces fe-0/0/0 unit 12 family inet address	192.168.0.17/28
6	13	192	168	0	33	28	set interfaces fe-0/0/0 unit 13 family inet address	192.168.0.33/28
7	14	192	168	0	49	28	set interfaces fe-0/0/0 unit 14 family inet address	192.168.0.49/28

数式でIPアドレスを等間隔に計算

→ Excelは他のメンバーに数式を壊されるリスクがあり、メンテナンス工数が発生

1.2 作業の問題点

1. IPアドレス計算にExcelを使っていた

- Excelでの作業イメージ

	A	B	C	D	E	F	G	H
1								
2	unit	オクテット				プレフィックス		コマンド
3		第1	第2	第3	第4			
4	11	192	168	0	1	28	set interfaces fe-0/0/0 unit 11 family inet address 192.168.0.1/28	
5	12	192	168	0	17	28	set interfaces fe-0/0/0 unit 12 family inet address 192.168.0.17/28	
6	13	192	168	0	33	28	set interfaces fe-0/0/0 unit 13 family inet address 192.168.0.33/28	
7	14	192	168	0	49	28	set interfaces fe-0/0/0 unit 14 family inet address 192.168.0.49/28	

→ Excelは他のメンバーに数式を壊されるリスクがあり、メンテナンス工数が発生

1.2 作業の問題点

2. ネットワークOSごとにコマンドが異なる

- サブインターフェースを1つ作成する場合のコマンド比較

- Junos

```
set interfaces fe-0/0/0 unit 11 vlan-id 11
set interfaces fe-0/0/0 unit 11 family inet address 192.168.0.1/28
```

- IOS

```
interface GigabitEthernet1.11
  encapsulation dot1Q 11
  ip address 192.168.0.1 255.255.255.240
!
```

→ ネットワークOSごとの手順書や実機コマンドから設定ファイルを作成するので、必要なスキルが高い

1.3 As-Is / To-Be

As-Is（現在の姿）

作業者に求めるスキルの高い、手動の作業

- 作業者はネットワークOSごとのコマンドを理解して設定ファイルを作成する
- 作業台数・作成するコマンドが多く、人海戦術による対処が発生する
- 作業用ファイルがエクセルであり、メンテナンス工数が発生している

To-Be（どうあるべきか）

作業者のスキルに依存しない、自動化された作業

- 作業者はネットワークOSごとのコマンドを意識せずに設定ファイルを作成できる
- 作業台数・コマンドの量に依存せず、より少ない人員で作業ができる
- 作業用ファイルにエクセルを使用せず、シンプルなフォーマットのファイルを使用している

2. Ansible実装時

2.1 実装の方針

1. 問題点への対策
2. 対策①：テンプレートエンジンによるIPアドレスの自動計算
3. 対策②：PlaybookによるネットワークOSの自動識別
4. 実行結果
5. デモ

2.1 実装の方針

1. 問題点への対策

- IPアドレス計算にExcelを使っていた
 - 対策①：テンプレートエンジンによるIPアドレスの自動計算
- ネットワークOSごとにコマンドが異なる
 - 対策②：PlaybookによるネットワークOSの自動識別

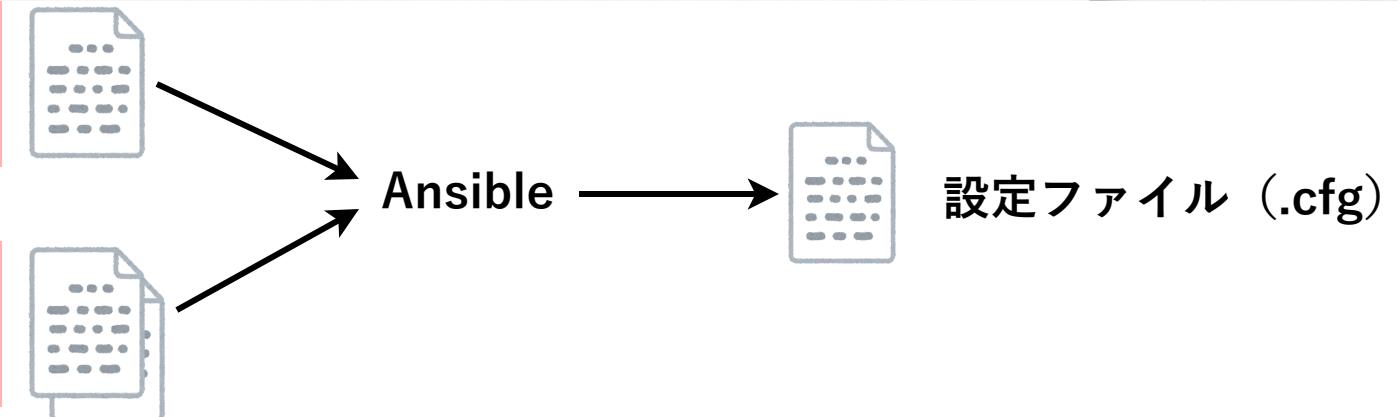
2.1 実装の方針

2. 対策①：テンプレートエンジンによるIPアドレスの自動計算

- IPアドレスの自動計算 & 設定ファイル作成は、Ansibleのtemplateモジュールで可能
- templateモジュールでは、以下のように結果ファイルを生成

YAMLの変数ファイル (.yml)
-> ネットワークOSで共通 の形式

Jinja2テンプレート (.j2)
-> ネットワークOS毎に異なる 形式



2.1 実装の方針

2. 対策①：テンプレートエンジンによるIPアドレスの自動計算

- 変数ファイルの設計

- junos用変数ファイル (junos01.yml)

```
---  
interface: "fe-0/0/0"          # インターフェース名  
interface_count: 20           # サブインターフェース数  
unit_start: 11                # 開始vlan  
address_start: "192.168.0.1"   # 開始アドレス  
mask: 28                      # マスク
```

- ios用変数ファイル (ios01.yml)

```
---  
interface: "GigabitEthernet1"  # インターフェース名  
interface_count: 20           # サブインターフェース数  
unit_start: 11                # 開始vlan  
address_start: "192.168.100.1" # 開始アドレス  
mask: 28                      # マスク
```

→ ネットワークOSで共通の形式なので、ネットワークOSを意識せずに作業可能

2.1 実装の方針

2. 対策①：テンプレートエンジンによるIPアドレスの自動計算

- 変数ファイルの設計

- junos用変数ファイル (junos01.yml)

```
---  
interface: "fe-0/0/0"          # インターフェース名  
interface_count: 20           # サブインターフェース数  
unit_start: 11                # 開始vlan  
address_start: "192.168.0.1"   # 開始アドレス  
mask: 28                      # マスク
```

count	unit	ネットワーク	1	2	...	15	16
1	11	192.168.0.0/28	192.168.0.0	192.168.0.1	...	192.168.0.14	192.168.0.15
2	12	192.168.0.16/28	192.168.0.16	192.168.0.17	...	192.168.0.30	192.168.0.31
...
19	29	192.168.1.32/28	192.168.1.32	192.168.1.33	...	192.168.1.46	192.168.1.47
20	30	192.168.1.48/28	192.168.1.48	192.168.1.49	...	192.168.1.52	192.168.1.53

→ 開始アドレス (address_start) とマスク (mask) を基準に最初のネットワークを算出

2.1 実装の方針

2. 対策①：テンプレートエンジンによるIPアドレスの自動計算

- 変数ファイルの設計
 - junos用変数ファイル (junos01.yml)

```
---
```

```
interface: "fe-0/0/0"          # インターフェース名
interface_count: 20             # サブインターフェース数
unit_start: 11                  # 開始vlan
address_start: "192.168.0.1"    # 開始アドレス
mask: 28                        # マスク
```

count	unit	ネットワーク	1	2	...	15	16
1	11	192.168.0.0/28	192.168.0.0	192.168.0.1	...	192.168.0.14	192.168.0.15
2	12	192.168.0.16/28	192.168.0.16	192.168.0.17	...	192.168.0.30	192.168.0.31
...
19	29	192.168.1.32/28	192.168.1.32	192.168.1.33	...	192.168.1.46	192.168.1.47
20	30	192.168.1.48/28	192.168.1.48	192.168.1.49	...	192.168.1.52	192.168.1.53

→ サブインターフェース数 (interface_count) に応じて、ネットワークを等間隔に分割・アドレス計算

2.1 実装の方針

2. 対策①：テンプレートエンジンによるIPアドレスの自動計算

- テンプレート

- Junos用テンプレート (junos_interface.j2)

```
{% for i in range(intf_set.interface_count) %}  
{% set unit = intf_set.unit_start + i %}  
{% set address = intf_set.address_start | ipmath(2 ** (32 - intf_set.mask) * i) %}  
{% set address_mask = '{}/{}'.format(address, intf_set.mask) %}  
set interfaces {{ intf_set.interface }} unit {{ unit }} vlan-id {{ unit }}  
set interfaces {{ intf_set.interface }} unit {{ unit }} family inet address {{ address_mask }}  
{% endfor %}
```

- IOS用テンプレート (ios_interface.j2)

```
{% for i in range(intf_set.interface_count) %}  
{% set unit = intf_set.unit_start + i %}  
{% set address = intf_set.address_start | ipmath(2 ** (32 - intf_set.mask) * i) %}  
{% set address_mask = '{}/{}'.format(address, intf_set.mask) %}  
interface {{ intf_set.interface }}.{{ unit }}  
  encapsulation dot1Q {{ unit }}  
  ip address {{ address }} {{ address_mask | ipaddr('netmask') }}  
!  
{% endfor %}
```

➡ テンプレートはネットワークOS毎に作成し、ネットワークOSの違いを吸収させる

2.1 実装の方針

2. 対策①：テンプレートエンジンによるIPアドレスの自動計算

- テンプレートにおける、ipmathフィルター

- 用途

- 単純なIPアドレスの計算

- 使用例

- IPアドレスの加算

```
{{ '192.168.1.1' | ipmath(1) }} -> 192.168.1.2  
{{ '192.168.1.255' | ipmath(1) }} -> 192.168.2.0
```

- IPアドレスの減算

```
{{ '192.168.1.2' | ipmath(-1) }} -> 192.168.1.1  
{{ '192.168.2.0' | ipmath(-1) }} -> 192.168.1.255
```

→ IPアドレスの加算・減算是第3オクテットへの繰り上げ・繰り下げも可能

2.1 実装の方針

3. 対策②：PlaybookによるネットワークOSの自動識別

```
---
- hosts: junos, ios
  gather_facts: false
  tasks:
    # (1) 変数ファイル読み込み
    - name: "read files from settings"
      set_fact:
        intf_set: >-
          {{ lookup("file", "./settings/{{ inventory_hostname }}.yml") |
            from_yaml }}

    # (2) 変数intf_setの内容確認
    - name: "debug intf_set"
      debug:
        var: intf_set

    # (3) 設定ファイル作成
    - name: "create configuration file to configs"
      template:
        src: "./templates/{{ ansible_network_os }}_interface.j2"
        dest: "./configs/{{ inventory_hostname }}_interface.cfg"
      delegate_to: localhost
```

2.1 実装の方針

3. 対策②：PlaybookによるネットワークOSの自動識別

```
# (4-1) 設定変更 (junos)
- name: "setting from configuration file to junos"
  junos_config:
    src: "./configs/{{ inventory_hostname }}_interface.cfg"
    comment: "setting from ansible"
  when:
    - ansible_network_os == "junos"

# (4-2) 設定変更 (ios)
- name: "setting from configuration file to ios"
  ios_config:
    src: "./configs/{{ inventory_hostname }}_interface.cfg"
    save_when: changed
  when:
    - ansible_network_os == "ios"
```

→ 変数ansible_network_osで対象のネットワークOS（Junos/IOS）を判別する

2.1 実装の方針

4. 実行結果

- Junos用設定ファイル (junos01_interface.cfg)

```
set interfaces fe-0/0/0 unit 11 vlan-id 11
set interfaces fe-0/0/0 unit 11 family inet address 192.168.0.1/28
set interfaces fe-0/0/0 unit 12 vlan-id 12
set interfaces fe-0/0/0 unit 12 family inet address 192.168.0.17/28
(..略..)
set interfaces fe-0/0/0 unit 30 vlan-id 30
set interfaces fe-0/0/0 unit 30 family inet address 192.168.1.49/28
```

- IOS用設定ファイル (ios01_interface.cfg)

```
interface GigabitEthernet1.11
  encapsulation dot1Q 11
  ip address 192.168.100.1 255.255.255.240
!
interface GigabitEthernet1.12
  encapsulation dot1Q 12
  ip address 192.168.100.17 255.255.255.240
!
(..略..)
interface GigabitEthernet1.30
  encapsulation dot1Q 30
  ip address 192.168.101.49 255.255.255.240
!
```

→ ネットワークOSを意識せずに、大量の設定が作成可能

2.1 実装の方針

5. デモ

- junos用のサブインターフェース設定ファイル作成

項目	パターン1	パターン2
インターフェース名 (interface)	fe-0/0/0	fe-0/0/0
サブインターフェース数 (interface_count)	20	20
開始vlan (unit_start)	11	11
開始アドレス (address_start)	192.168.0.1	10.0.0.1
マスク (mask)	28	24

2.2 実装時の問題と対策

1. 開発ツールを統一していなかった
2. コーディングルールがなかった
3. 共有サーバ上のPlaybookを直接編集・実行する運用にしていた

2.2 実装時の問題と対策

1. 開発ツールを統一していなかった

- 実際に発生していた事象
 - 開発ツールがviエディタやサクラエディタで統一性がなく、ノウハウが蓄積されない
 - インデントのずれや余計なスペースがあるなどの些細なミスが発生し、レビューに時間がかかる

➡ 開発ツールをVisual Studio Codeで統一し、インストールする拡張機能をチーム内で共有

2.2 実装時の問題と対策

2. コーディングルールがなかった

- 実際に発生していた事象
 - 開発者間で書き方が統一されていないので、Playbookのレビューが困難
 - Playbookの書き方に迷い、調べる手間が発生
 - 変数名
 - タスク名
 - 変数の確認（assert）方法

→ コーディングルールを作成し、開発者間で共有

2.2 実装時の問題と対策

3. 共有サーバ上のPlaybookを直接編集・実行する運用にしていた

- 実際に発生していた事象
 - 誰かが誤ってPlaybookを編集・削除してしまい、Playbookが使えなくなる
 - 以前のPlaybookに戻すことができないので、定期的にバックアップする手間がかかる

バックアップのイメージ

- backup_directory
 - backup_202301
 - backup_202302

→ Gitを導入し、Playbookの編集者や変更履歴を管理できるようにした

3. Ansible実装後

3.1 得られた効果

1. 作業品質の向上
2. 未経験の作業者でも比較的容易に作業が可能になる

3.1 得られた効果

1. 作業品質の向上

- 主な理由
 - コマンド作成・設定変更が自動で実施されるので、入力ミスが削減された
 - 作業時間のブレが少なくなり、作業時間の見積もりがしやすくなった
 - 変更作業が早く終わるので、状態確認などの事後作業に時間を割くことができるようになった

→ コマンド作成・設定変更の品質向上が、他の作業の品質向上にも繋がっていく

3.1 得られた効果

2. 未経験の作業者でも比較的容易に作業が可能になる

- 主な理由

- ネットワークOSを意識せずに作業可能になった
- 作業用のファイルがYAML形式なため、作業者の学習コストが低くなった

```
---
```

```
interface: "fe-0/0/0"      # インターフェース名
interface_count: 20         # サブインターフェース数
unit_start: 11               # 開始vlan
address_start: "192.168.0.1" # 開始アドレス
mask: 28                   # マスク
```

→ 作業用のファイルがYAML形式でシンプルなので、作業者だけでなくレビュー者の負担軽減に繋がる

3.2 自動化によって生じた課題と対策

1. 作業者にとって、自動化された作業の内容がイメージしづらい
2. 実行エラーが起きるたびに開発者に連絡がきてしまうようになった

3.2 自動化によって生じた課題と対策

1. Ansible未経験者にとって、自動化された作業の内容がイメージしづらい

- 主な理由
 - 自動化によって実作業がきれいに隠蔽されてしまった (=ブラックボックス化)
 - 自動化実装後に参入したメンバーは、そもそも手動の作業をしたことがない

➡ テスト環境で手動の作業を体験してもらい、自動化後の作業との違いを体験してもらう

3.2 自動化によって生じた課題と対策

1. Ansible未経験者にとって、自動化された作業の内容がイメージしづらい

- 手動の作業を体験してもらうによる追加効果
 - Ansibleが使えなくなった場合の手動の対応がスムーズになる
 - 自動化による効果を再認識してもらうきっかけになる
- 手動と自動の違いを理解してもらうことが、自動化のイメージアップにつながる

3.2 自動化によって生じた課題と対策

2. 実行エラーが起きるたびに開発者に連絡がきてしまうようになった

- 主な理由
 - Ansible未経験者にとって、エラーログから原因を特定するのが難しい
- エラー時の対処方法をドキュメント化し、問い合わせフローを策定する

まとめ

- Ansibleを実際に導入して感じた、ネットワーク自動化のメリット
 - 作業品質の向上
 - 未経験の作業者でも比較的容易に作業が可能になる
- Ansible導入後にすべきこと
 - 手動の作業を体験してもらい、自動化後の作業との違いを体験してもらう
 - エラー時の対処方法をドキュメント化し、問い合わせフローを策定する

さいごに

- 自動化は導入が目的ではありません
 - 導入後の運用方法も考えて、継続的な運用を実現する必要がある
- 自動化で仕事が減るのではなく、価値の高い作業の時間が増やせると考える

ご清聴ありがとうございました

補足資料

完成したファイル一式

📁: フォルダ, 📄: ファイル

- 📁 inventory … 対象機器への接続情報
 - 📄 hosts.yml
- 📁 templates … ネットワークOSごとのテンプレート (j2)
 - 📄 junos_interface.j2
 - 📄 ios_interface.j2
- 📁 settings … テンプレートと組み合わせる機器ごとの変数ファイル (yml)
 - 📄 junos01.yml
 - 📄 junos02.yml
 - 📄 ios01.yml
 - 📄 ios02.yml
- 📁 configs … 作成された機器ごとの設定ファイル (Playbook実行後に生成)
 - 📄 junos01_interface.cfg
 - 📄 junos02_interface.cfg
 - 📄 ios01_interface.cfg
 - 📄 ios02_interface.cfg
- 📄 ansible.cfg
- 📄 network_setting_interface.yml

➡ 対象機器ごとの変数ファイルを作成し、Playbookを実行すれば作業が完了する

完成したファイル式 hosts.yml

```
all:
  children:
    junos:
      hosts:
        junos01:
          ansible_host: xxx.xxx.xxx.xxx
      vars:
        ansible_network_os: junos
        ansible_connection: netconf
        ansible_user: user
        ansible_password: password
    ios:
      hosts:
        ios01:
          ansible_host: xxx.xxx.xxx.xxx
      vars:
        ansible_network_os: ios
        ansible_connection: network_cli
        ansible_user: user
        ansible_password: password
        ansible_become: yes
        ansible_become_method: enable
```

必要なPythonパッケージ

- Ansible以外のPythonパッケージ

Pythonパッケージ	用途
ncclient	Junosのネットワーク機器接続用
paramiko	IOSのネットワーク機器接続用
netaddr	IPアドレスの計算用

- インストール手順

```
$ source $HOME/venv/bin/activate
(venv)$ python --version
Python 3.9.6
(venv)$ cat requirements.txt
ansible==2.9.27
ncclient
netaddr
paramiko
(venv)$ pip install -r requirements.txt
```

Visual Studio Codeの拡張モジュール

- **indent-rainbow**
 - インデントを見やすく色付けしてくれる
- **Trailing Spaces**
 - 末尾のスペースをハイライトしてくれる
- **Code Spell Checker**
 - コード内の英語のスペルが正しいかを確認してくれる

IPアドレスに等間隔で加算するための計算方法

- テンプレートにおける該当部分

```
{% set address = intf_set.address_start | ipmath(2 ** (32 - intf_set.mask) * i) %}
```

- 計算式

開始アドレス + $2^{32 - \text{マスク}} \times \text{加算する回数}$

開始アドレス（例: 192.168.0.1）

マスク（例: 28）

加算する回数（0, 1, 2, ...）

- 例

- 開始アドレス=192.168.0.1, マスク=28, 加算する回数=1の場合

- $192.168.0.1 + 2^{32 - 28} * 1$

- $192.168.0.1 + 16$

- $192.168.0.17$