

どう使う？データセンターネットワーク最前線

Yahoo! JAPAN 実用例

ヤフー株式会社 サイトオペレーション本部 ネットワーク開発 高橋 翔



自己紹介

高橋 翔

2017/04: ヤフー入社

サイトオペレーション本部

データセンタネットワーク所属 (2019~兼務)

- ・ L2/L3 スイッチ 構築/運用

2019/04: 同本部

ネットワーク開発所属

- ・ IP Clos の設計/構築/運用



どう使う？データセンターネットワーキング最前線

1. ヤフーのデータセンターネットワークあらまし

2. Clos Network 構築

- Zero Touch Provisioning
- Ansible

3. ネットワーク監視体制

- Prometheus

ヤフーのデータセンターネットワークあらまし

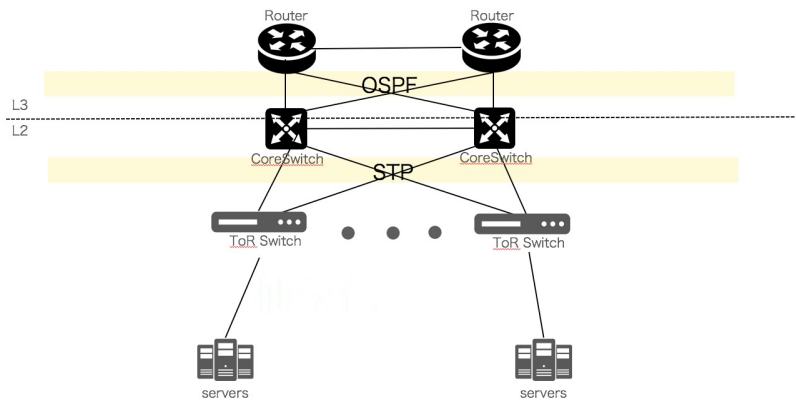
ヤフーの国内データセンター



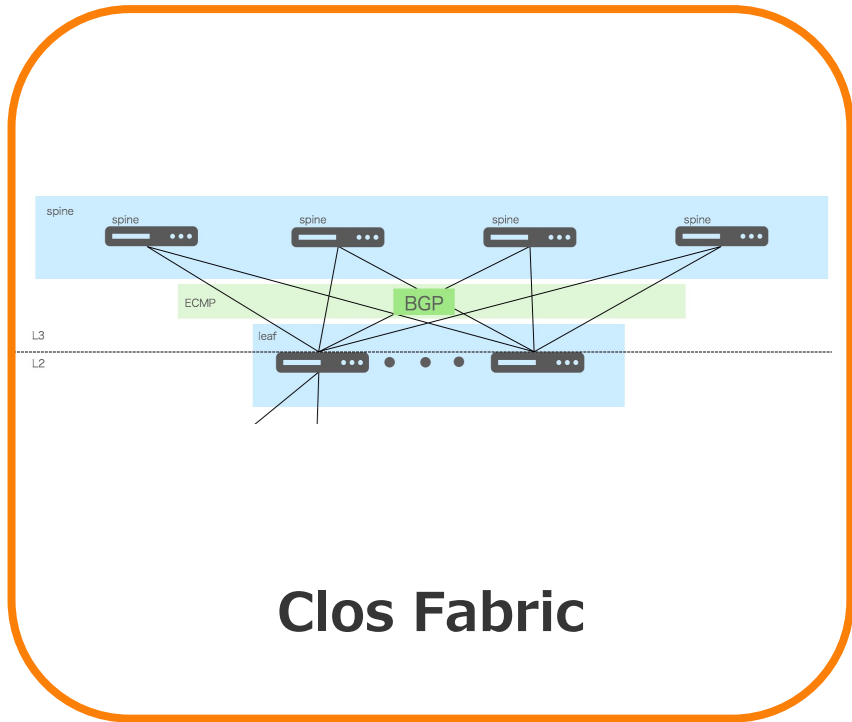
自社管理の大規模データセンター

ヤフーのデータセンタネットワークあらまし

稼働している2種類のネットワークアーキテクチャ



**Traditional
Architecture**



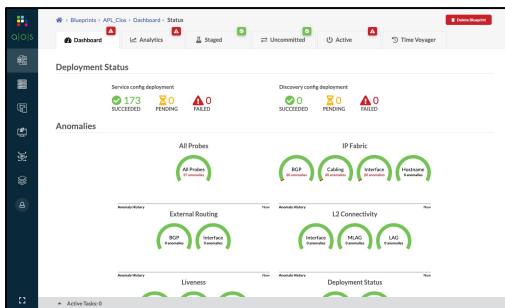
Clos Fabric

ヤフーのデータセンターネットワークあらまし ネットワーク構築に使っているツール



Juniper Apstra™

<https://www.juniper.net/jp/ja/products/network-automation/apstra/apstra-system.html>



Web UI/大規模 NW構築

マルチベンダー対応

サポート体制が充実

ライセンス費用

ヤフーのデータセンターネットワークあらまし

ネットワーク構築に使っているツール



ANSIBLE

今日のトピックはこちら

Open Source

スモールスタート

Linux like な config

Self-support

Clos Network 構築

Zero Touch Provisioning

Ansible

Clos Network に使用している主なハードウェアと Network OS

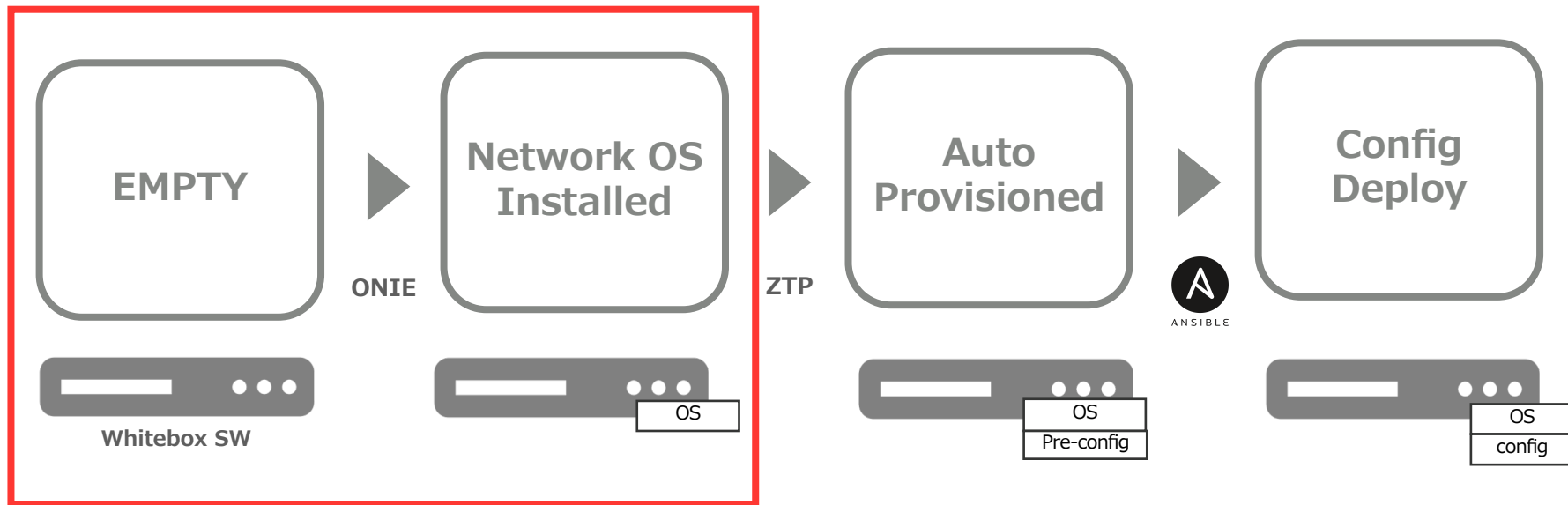
- 以前は
 - Broadcom chip 搭載の Whitebox switch + Cumulus Linux

Mellanox(今は NVIDIA傘下) → Cumulus Networks 買収 (2020)
Broadcom chipのサポート終了を発表

<https://docs.nvidia.com/networking-ethernet-software/cumulus-linux-44/Whats-New/#unsupported-platforms>

- 最近
 - Broadcom chip 搭載の Whitebox switch + Cumulus Linux
 - Mellanox chip 搭載の Whitebox switch + Cumulus Linux
 - 混ぜて使えるように工夫している

Network OS Setup Flow



ONIE

ONIE (Open Networking Install Environment) <https://www.opencompute.org/projects/onie>

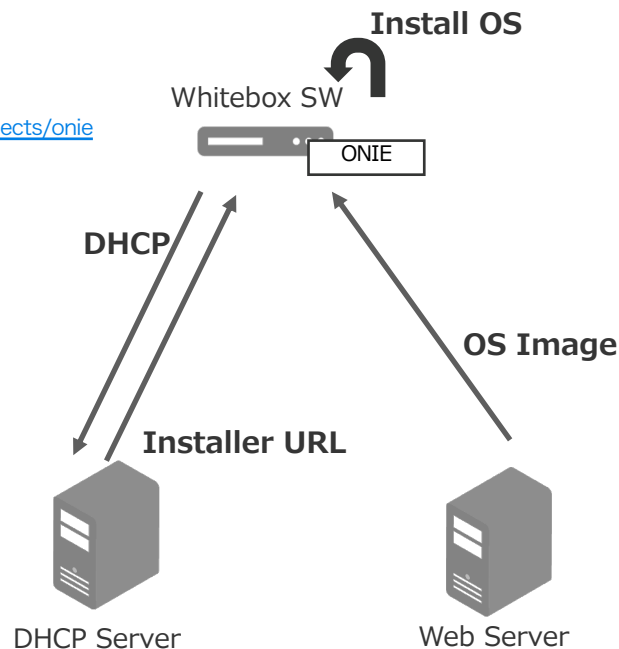
Network OS を Whitebox SW にインストールツール

- DHCP/USB drive 等から Installer を探し出す

Installer を見つけると自動で OS をインストールする

ヤフーでは

- DHCP/Webサーバを組み合わせて OS Install
- 複数のベンダスイッチ対応のため Installer URL を使い分ける工夫



DHCP Option 60 – Vendor class identifier

DHCPクライアントが、自分が何者なのかDHCPサーバに伝えるオプション

onie_vendor:[ARCH]-[VENDOR_MACHINE]-r[REVISION]

ONIE共通

```
ONIE:/ # onie-sysinfo
x86_64-dellemc_z9264f_c3538-r0
```

自分のスイッチが何者か
ONIE console から調べることができる



Whitebox SW

DHCP Discover



Installer URL
for Vendor A



DHCP Server

```

  > Option: (61) Client identifier
      Length: 7
      Hardware type: Ethernet (0x01)
      Client MAC address: 18:5a:58:
  > Option: (57) Maximum DHCP Message Size
  > Option: (55) Parameter Request List
  > Option: (60) Vendor class identifier
      Length: 41
      Vendor class identifier: onie_vendor:x86_64-dellemc_z9200_c3538-r0
```

DHCP Discover
ペイロードの一部

<https://opencomputeproject.github.io/onie/design-spec/discovery.html>

Vendor-class-identifier の設定例

dhcpd.conf

```
class "mellanox-switch" {
    match if (substring(option vendor-class-identifier, 0, 23) = "onie_vendor:x86_64-mlnx");
    option default-url = "http://192.0.2.1/images/onie-installer-x86_64-mlnx.bin";
}

class "dell-switch" {
    match if (substring(option vendor-class-identifier, 0, 26) = "onie_vendor:x86_64-dellemc");
    option default-url = "http://192.0.2.1/images/onie-installer-x86_64-bcm.bin";
}
```

OS Image URL

kea-dhcp4.conf

```
{
  "Dhcp4": {
    "client-classes": [
      {
        "name": "dell-switch",
        "test": "substring(option[60].hex,0,26) == 'onie_vendor:x86_64-dellemc'",
        "option-data": [
          {
            "name": "default-url",
            "data": "http://192.0.2.1/images/onie-installer-x86_64-bcm.bin"
          }
        ]
      }
    ]
  }
}
```

インストーラファイル名は
ONIE の命名規則に従う

OS image は onie-installer**** のフォーマットでホストする

Default File Name Search Order

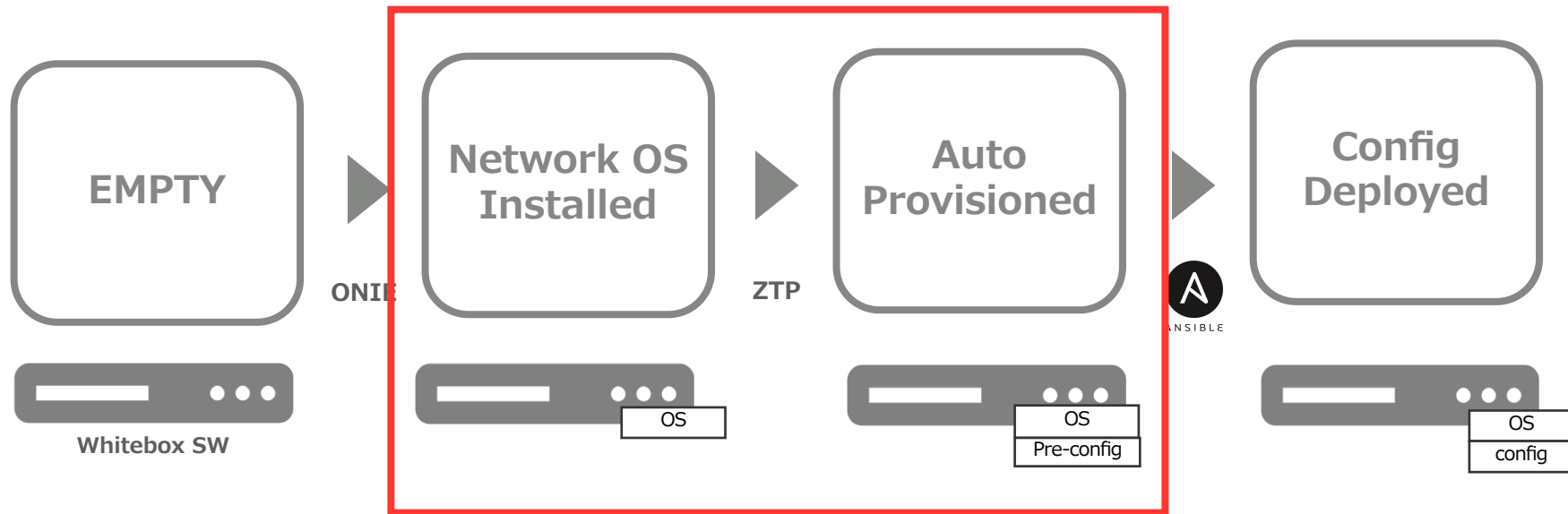
In a number of the following methods, ONIE searches for default file names in a specific order. All the methods use the same default file names and search order, which are described in this section.

The default installer file names are searched for in the following order:

1. `onie-installer-<arch>-<vendor>_<machine>-r<machine_revision>`
2. `onie-installer-<arch>-<vendor>_<machine>-r<machine_revision>.bin *`
3. `onie-installer-<arch>-<vendor>_<machine>`
4. `onie-installer-<arch>-<vendor>_<machine>.bin *`
5. `onie-installer-<vendor>_<machine>`
6. `onie-installer-<vendor>_<machine>.bin *`
7. `onie-installer-<cpu_arch>-<switch_silicon_vendor>`
8. `onie-installer-<cpu_arch>-<switch_silicon_vendor>.bin *` `onie-installer-x86_64-mlnx.bin`
9. `onie-installer-<arch>`
10. `onie-installer-<arch>.bin *`
11. `onie-installer`
12. `onie-installer.bin *`

<https://opencomputeproject.github.io/onie/design-spec/discovery.html>

Network OS Setup Flow




Cumulus Zero Touch Provisioning

Cumulus Linux 初回起動時、ZTP プロセスが 1度だけ立ち上がる

DHCP/USB Drive/Local Drive のスクリプトを読み込んで任意の設定を行う機能

- Perl
- Python
- Ruby
- **Shell**



専用の DHCP option を使って
Scriptの場所 を DHCPサーバからスイッチに伝える

Auto Provisioning では必要最低限 (SSH できる状態まで) 設定する

<https://docs.nvidia.com/networking-ethernet-software/cumulus-linux-44/Installation-Management/Zero-Touch-Provisioning-ZTP/>

DHCP Option 239

ZTP script URL を Cumulus に渡すために、Cumulus Linux が指定する DHCP Option番号

DHCPサーバからの応答に Option 239 がある場合、指定された URL から ZTP script をダウンロードして実行

dhcpd.conf

```
option cumulus-provision-url code 239 = text;  
option cumulus-provision-url = "http://192.0.2.1/config/cumulus/boot";
```

Option 239 を定義

kea-dhcp4.conf

```
{  
  "Dhcp4": {  
    "option-def": [  
      {  
        "name": "cumulus-provision-url",  
        "code": 239,  
        "type": "string"  
      }  
    ],  
    "option-data": [  
      {  
        "name": "cumulus-provision-url",  
        "data": "http://192.0.2.1/config/cumulus/boot"  
      }  
    ]  
  }  
}
```

Option 239 で渡す URL をセット

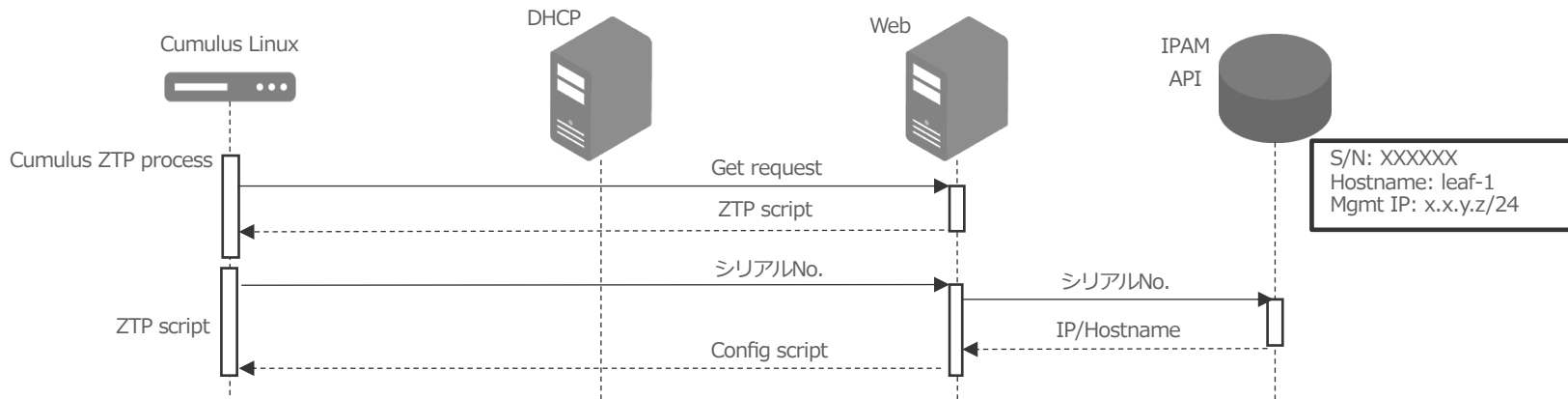
ZTP Script

Mgmt IP を Static に埋め込みたい

社内IPAMと連携して IP/Hostname 収集

内部的に script を呼び出す工夫

```
config/cumulus/boot
1 #!/bin/bash -e
2
3 function error() {
4     echo -e "ERROR: something occurred $BASH_COMMAND at line $BASH_LINENO." >&2
5     exit 1
6 }
7
8 # Log all output from this script
9 exec >> /var/log/autoprovision 2>&1
10 date "+%FT%T ztp starting script $0"
11
12 trap error ERR
13
14 # Pick up serial
15 ID=$(sudo decode-syseeprom | grep Serial | awk '{print $5}')
16
17 curl -sL "http://192.0.2.1/config/cumulus/init?serial=$ID" | bash
18
19 # CUMULUS-AUTOPROVISIONING
20 exit 0
```



ZTP Script

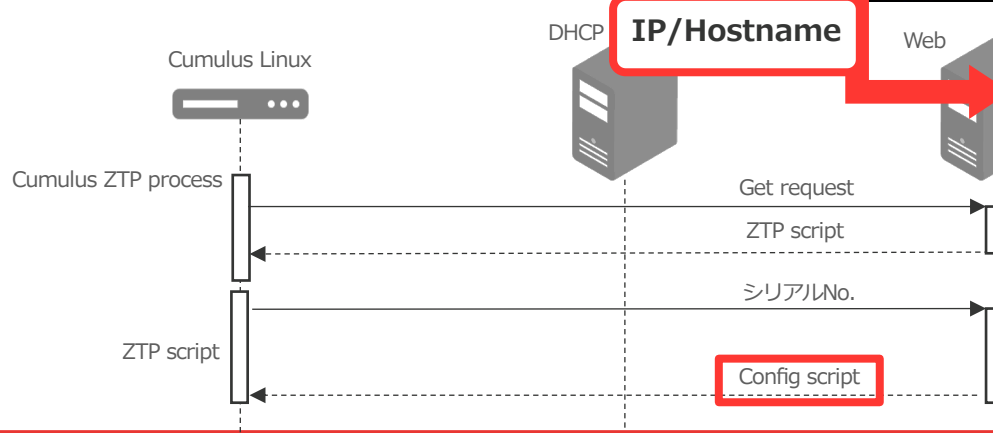
S/N をキーにして設定用 script を呼び出し

初期ログイン用の Credential も設定

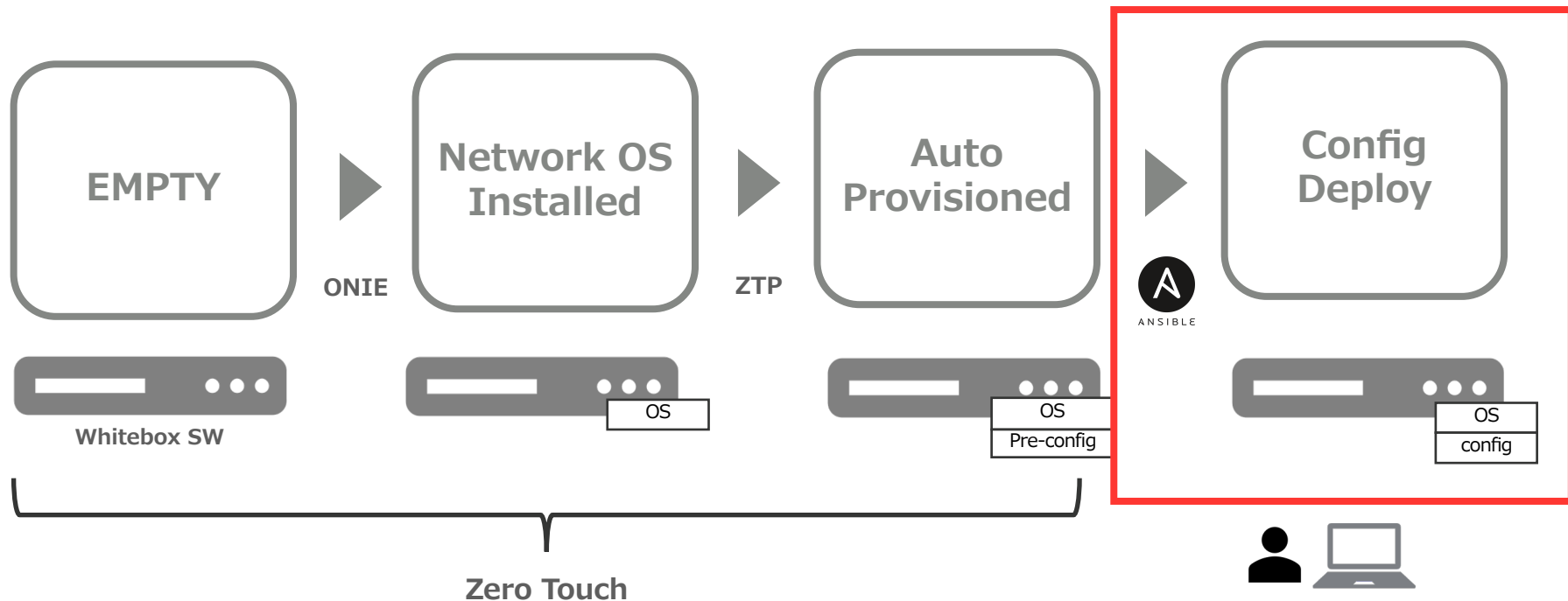
SSH login ready

```
config/cumulus/boot
1 #!/bin/bash -e
2
3 function error() {
4     echo -e "ERROR: something occurred $BASH_COMMAND at line $BASH_LINENO." >&2
5     exit 1
6 }
7
8 # Log all output from this script
9 exec >> /var/log/autoprovision 2>&1
10 date "+%FT%T ztp starting script $0"
11
12 trap error ERR
13
14 # Pick up serial
15 ID=$(sudo decode-syseeprom | grep Serial | awk '{print $5}')
16
17 curl -sL "http://192.0.2.1/config/cumulus/init?serial=$ID" | bash
18
19 # CUMULUS-AUTOPROVISIONING
20 exit 0
```

```
config/cumulus/init
1 #!/bin/bash
2
3 # Wait
4 last_code=1
5 while [ "1" == "$last_code" ]; do
6     net show interface &> /dev/null
7     last_code=$?
8 done
9
10 # Set management IP
11 net add vrf mgmt
12 net add interface eth0 ip address {{mgmtip}}/{{network_cidr}}
13 net add interface eth0 ip gateway {{gateway}}
14
15 net add hostname {{hostname}}
16 net commit
17
18 # Fix login secret
19 echo 'cumulus:$6$XXXXXXXXXXXXXXXXXXXXX' | chpasswd -e
```



ここまで Zero Touch, ここから Ansible



Ansible を使った Network 設定

Cumulus Linux

基本的にファイルのコピー/Jinja2 template にレンダリングした設定ファイルを転送

設定更新は プロセス restart OR reload

注力するポイント

1. 手に入れる変数を少なくする
2. テンプレートの種類を少なくする。できる限り使い回す
3. 何でもできるテンプレートにしない（何でもできるようにがんばらない）
 - If 文が増えるとメンテしにくい

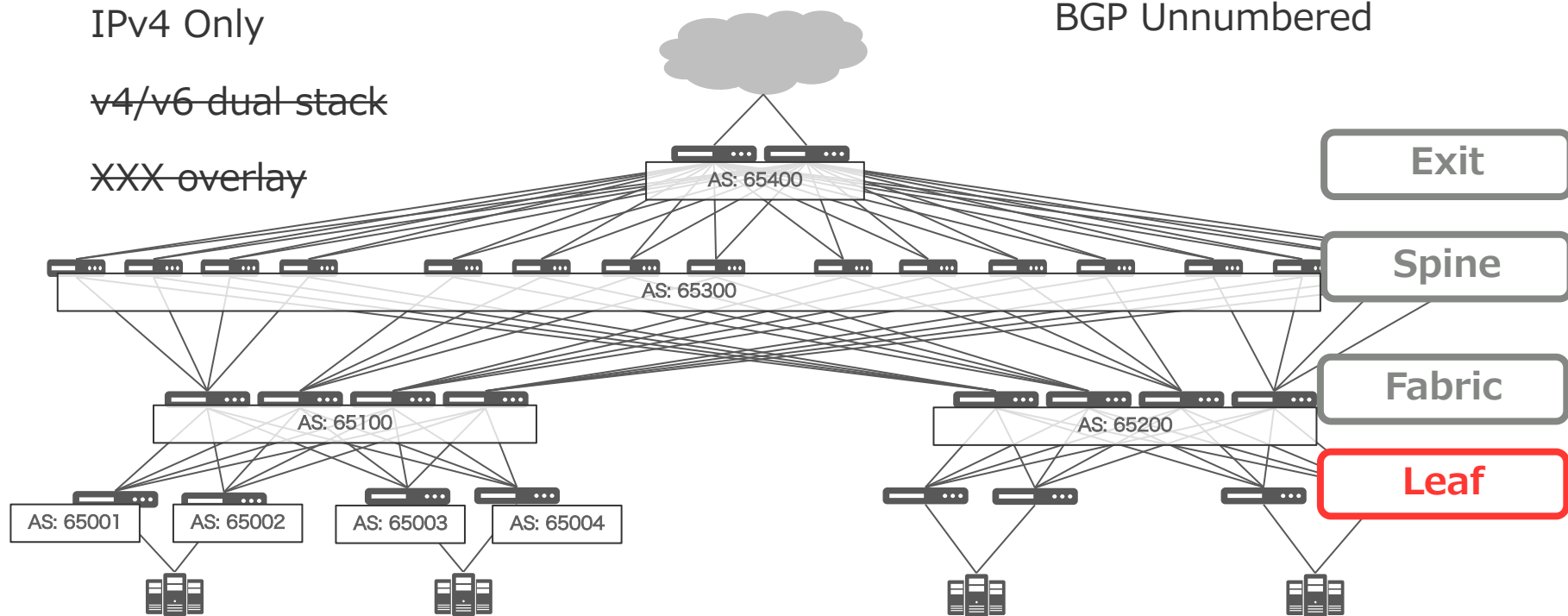
ヤフーの Clos Network トポロジ

IPv4 Only

~~v4/v6 dual stack~~

~~XXX overlay~~

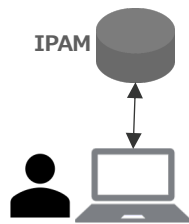
BGP Unnumbered



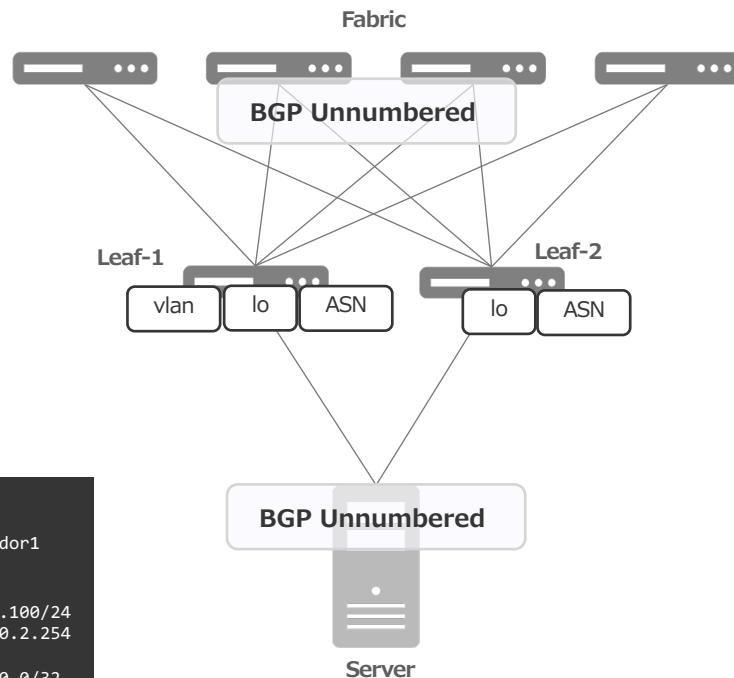
Leaf スイッチの設定に必要な情報

1. IPv4 Loopback → **IPAM**
2. ASN → **Loopback IP** からスクリプトで生成
 - 4-Byte Private-AS 4200000000-4294967294
 - Shared IP, だいたい 4200000個
3. Vlan interface IPv4 (Leaf-1 のみ) → **IPAM**
 - サーバの初期設定にどうしても L2 が必要
4. Management IP/Gateway → **IPAM**

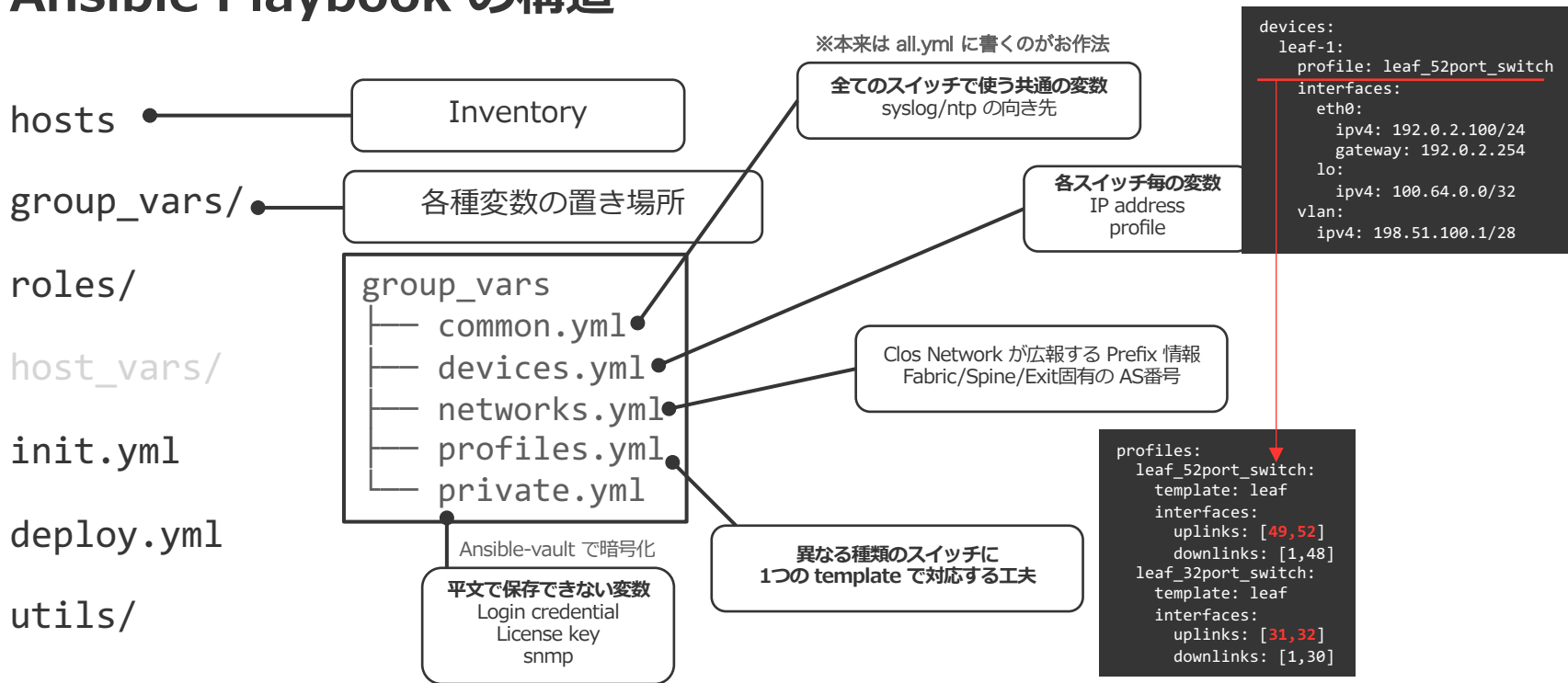
```
leaf-1:  
  profile: leaf_vendor1  
  interfaces:  
    eth0:  
      ipv4: {{ ip }}/{{ subnet }}  
      gateway: {{ gateway }}  
    lo:  
      ipv4: {{ loopback }}/32  
  vlan:  
    ipv4: {{ vlan_ip }}/{{ vlan_subnet }}
```



```
devices:  
  leaf-1:  
    profile: leaf_vendor1  
    interfaces:  
      eth0:  
        ipv4: 192.0.2.100/24  
        gateway: 192.0.2.254  
      lo:  
        ipv4: 100.64.0.0/32  
    vlan:  
      ipv4: 198.51.100.1/28
```



Ansible Playbook の構造



Ansible Playbook の構造

hosts

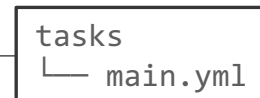
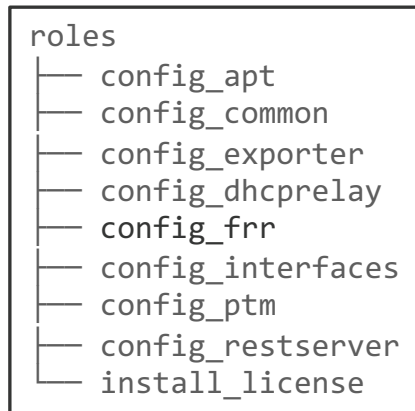
group_vars/

roles/ ● ——— 大量の task を分類するための roles

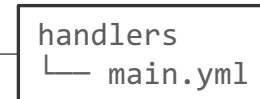
init.yml

deploy.yml

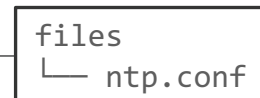
utils/



Role の中で実行する task



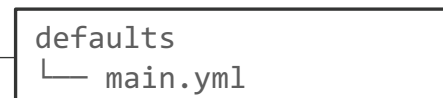
特定の task を trigger にして実行される



静的ファイルの置き場所



Jinja2 template の置き場所



変数のデフォルト値を定義
各 Role の task 実行時に参照される

Ansible Playbook の構造

hosts

group_vars/

roles/

init.yml

deploy.yml

utils/

```
- name: Initial setup
  hosts: cumulus
  roles:
    - install_license
    - config_common
    - config_appt
    - config_ptm
    - config_exporter
    - config_restserver
    - config_interfaces
    - config_dhcprelay
    - config_frr
  vars_files:
    - "{{ inventory_dir }}/group_vars/private.yml"
    - "{{ inventory_dir }}/group_vars/devices.yml"
    - "{{ inventory_dir }}/group_vars/common.yml"
    - "{{ inventory_dir }}/group_vars/profiles.yml"
    - "{{ inventory_dir }}/group_vars/networks.yml"
  vars:
    - init_playbook: this_is_just_flag
```

初期設定用 playbook

全て設定

```
roles
├── config_appt
├── config_common
├── config_exporter
├── config_dhcprelay
├── config_frr
├── config_interfaces
├── config_ptm
├── config_restserver
└── install_license
```

この Playbook を実行するときだけ定義される変数

Ansible Playbook の構造

hosts

group_vars/

roles/

init.yml

deploy.yml

utils/

```
- name: Deploy
  hosts: cumulus
  roles:
    - config_interfaces
    - config_dhcprelay
    - config_frr
  vars_files:
    - "{{ inventory_dir }}/group_vars/private.yml"
    - "{{ inventory_dir }}/group_vars/devices.yml"
    - "{{ inventory_dir }}/group_vars/common.yml"
    - "{{ inventory_dir }}/group_vars/profiles.yml"
    - "{{ inventory_dir }}/group_vars/networks.yml"
```

一部だけ設定

```
roles
├── config_apt
├── config_common
├── config_exporter
├── config_dhcprelay
├── config_frr
├── config_interfaces
├── config_ptm
├── config_restserver
└── install_license
```

サービスインに使う Playbook

トラフィックを落とさずに動く Playbook
- FRR は reload で設定を更新する
(BGP Peer が切断される restart はしない)

Ansible Playbook の構造

hosts

group_vars/

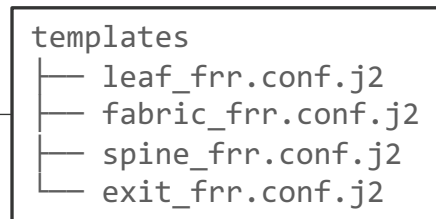
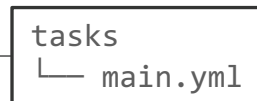
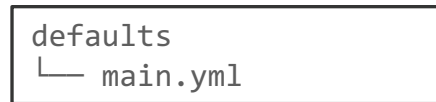
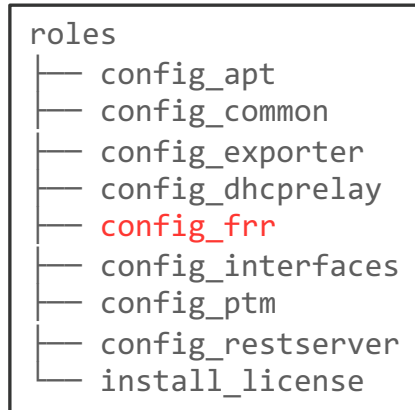
roles/ ●

大量の task を分類するための roles

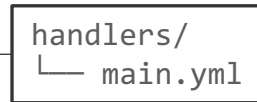
init.yml

deploy.yml

utils/



役割毎に template を用意



FRRouting の設定 - task

IPからAS番号を生成する CLIコマンドを開発

```
Ex:  
$ asncalc 100.64.0.0 → 4200000000  
$ asncalc 100.64.0.1 → 4200000001
```

tasks/main.yml

```
- name: generate AS number (leaf)  
  shell: asncalc "{{ devices[inventory_hostname].interfaces.lo.ipv4.split('/')[0] }}"  
  delegate_to: localhost  
  register: asncalc  
  when: profiles[device_profile].template in ["leaf"]  
  tags:  
    - local  
  
- set_fact:  
  asnum: "{{ asncalc.stdout }}"  
  when: profiles[device_profile].template in ["leaf"]  
  tags:  
    - local  
  
- name: Configure frr daemons  
  become: yes  
  copy:  
    src: "daemons.j2"  
    dest: /etc/frr/daemons  
    owner: frr  
    group: frr  
    mode: u=rw,g=r,o=  
  notify:  
    - Enable and start frr.service
```

```
profiles:  
  leaf_52port_switch:  
    template: leaf  
  interfaces:  
    uplinks: [49,52]  
    downlinks: [1,48]  
  leaf_32port_switch:  
    template: leaf  
  interfaces:  
    uplinks: [31,32]  
    downlinks: [1,30]
```

defaults/main.yml

config_frr role の中で使える変数を定義

```
device_profile: "{{ devices[inventory_hostname].profile }}"
```

```
- name: Configure /etc/frr/frr.conf  
  become: yes  
  template:  
    src: "{{ profiles[device_profile].template }}_frr.conf.j2"  
    dest: /etc/frr/frr.conf  
    owner: frr  
    group: frr  
    mode: u=rw,g=r,o=  
  notify:  
    - Enable and start frr.service  
    - Reload frr.service
```

```
- name: Restart frr at init  
  become: yes  
  systemd:  
    name: frr  
    state: restarted  
  when: init_playbook is defined
```

初期設定用 Playbook を動かすときだけ
process restart

FRRouting の設定 - template

templates/leaf_frr.conf.j2

```
{% set switch = devices[inventory_hostname] %}
{% set device_profile = devices[inventory_hostname].profile %}
{% set downlinks = profiles[device_profile].interfaces.downlinks %}
{% set uplinks = profiles[device_profile].interfaces.uplinks %}
frr defaults datacenter
hostname {{ inventory_hostname }}
username cumulus nopassword
!
service integrated-vtysh-config
!
log syslog informational
!
router bgp {{ asnum }}
{% if init_playbook is defined %}
  bgp graceful-shutdown
{% endif %}
  bgp bestpath as-path multipath-relax
  bgp bestpath compare-routerid
  bgp always-compare-med
  update-delay 300 90
  bgp router-id {{ switch.interfaces.lo.ipv4.split("/") [0] }}
```

異なる機種で使い回せるように
ポート構成はハードコードしない

初期設定 (init.yml 実行時) は
トラフィックが流れてほしくない

サービスイン (deploy.yml 実行時) は
bgp graceful-shutdown
が外れてトラフィックが流れ始める

```
neighbor DOWNLINK peer-group
{% for downlink in range(downlinks[0], downlinks[1] + 1) %}
{%- set bgpinfo = switch.bgp.peer["swp"+downlink|string] %}
  neighbor swp{{ downlink }}.100 interface peer-group DOWNLINK
  neighbor swp{{ downlink }}.200 interface peer-group DOWNLINK
{% if bgpinfo.state | default() == "down" %}
  neighbor swp{{ downlink }}.100 shutdown
  neighbor swp{{ downlink }}.200 shutdown
{% endif %}
{% endfor %}
{% endif %}
{% for uplink in range(uplinks[0], uplinks[1] + 1) %}
{%- set bgpinfo = switch.bgp.peer["swp"+uplink|string] %}
  neighbor swp{{ uplink }} interface peer-group UPLINK
{% if bgpinfo.state | default() == "down" %}
  neighbor swp{{ uplink }} shutdown
{% endif %}
{% endfor %}
!
~~~ snip ~~~
```

その他の定形作業に使う Ansible Playbook

hosts

group_vars/

roles/

init.yml

deploy.yml

utils/

ネットワークオペレーション用 Playbook

utils/graceful-shutdown.yml

```
tasks:  
  - name: Maintenance  
    shell: /usr/bin/vtysh -c "config t" -c "router bgp" -c "bgp graceful-shutdown" -c "end" -c "wr"
```

シェルから操作する定形作業を playbook として作成

例

- FRR に “bgp graceful-shutdown” を入れてトラフィック迂回する playbook
- BGP peer の状態をチェックする playbook
- OS upgrade する playbook
- Reboot する playbook

OS upgrade メンテナンスの一例

```
ansible-playbook -i hosts -l leaf-1 utils/graceful-shutdown.yml
```

bgp graceful-shutdown でトラフィックを迂回

```
ansible-playbook -i hosts -l leaf-1 utils/reboot.yml
```

再起動して OS をクリーンインストール

```
ansible-playbook -i hosts -l leaf-1 utils/check-bgp-state.yml
```

BGP Peer が想定通りに Up しているかチェック
いわゆる「確認系作業」も playbook の assert で実行

utils/check-bgp-state.yml

```
tasks:
  - name: Collect bgp neighbors to fabric
    shell: net show bgp summary | grep fabric- | grep -v -e Idle -e Active | awk '{print $1}' | wc -l
    register: peers_fabric
    changed_when: false

  - name: Confirm that BGP peers to all fabric are established
    assert:
      that:
        - peers_fabric.stdout | int == 4
    fail_msg: "{{peers_fabric.stdout}} peers to fabric"
```

```
ansible-playbook -i hosts -l leaf-1 utils/os-upgrade.yml
```

OS upgrade を実行

```
ansible-playbook -i hosts -l leaf-1 init.yml
```

初期設定を実行

```
ansible-playbook -i hosts -l leaf-1 deploy.yml
```

graceful-shutdown を解除してトラフィックを流す

全ての作業を playbook 化
オペミス回避

(最後に) すべての手順を Playbook化した次にやりたくなること

Playbook の実行を自動化したい

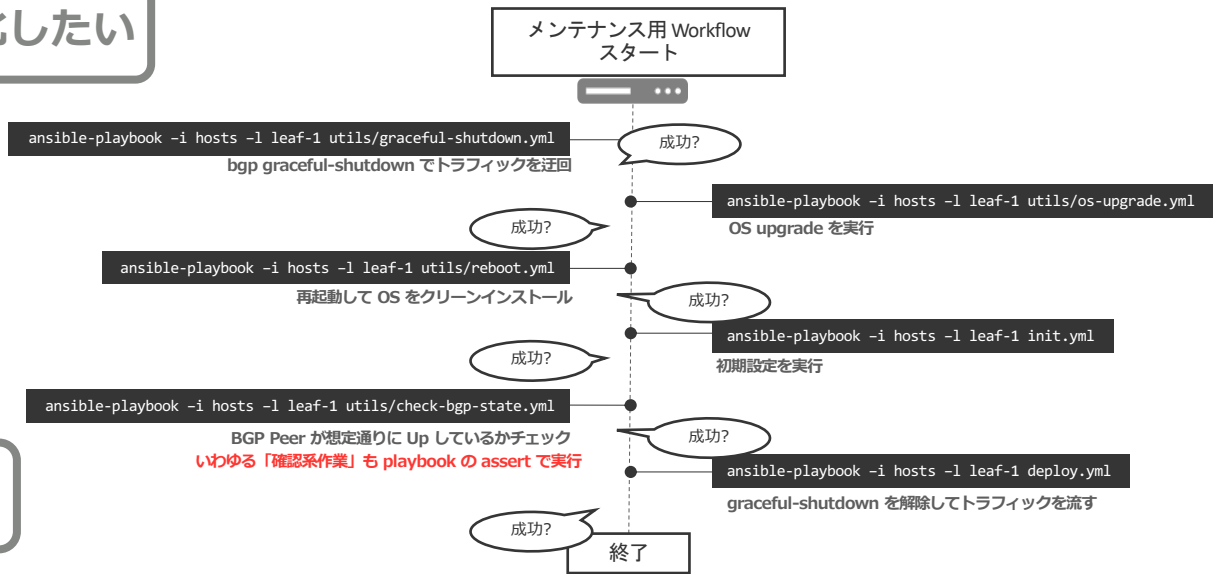


<https://github.com/ansible/awx>

Ansible の Task Engine

実行履歴の管理

Production の運用に導入できないか検討中



ネットワーク監視体制

監視体制の紹介

Prometheus

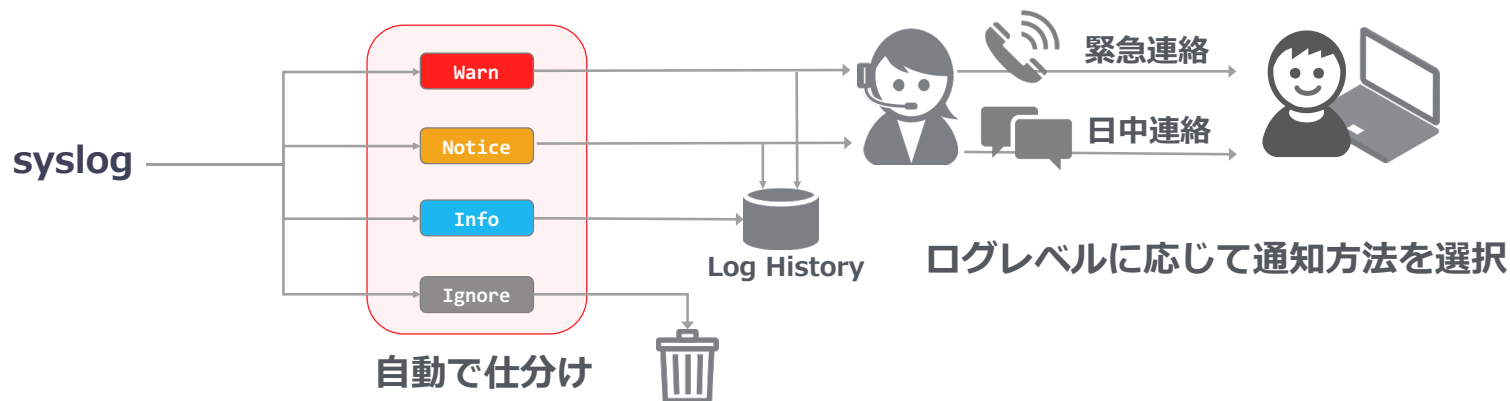
Syslog を使った監視

Flap 系ログの取り扱いに困る
センサ系とか

ログのパーズは大変

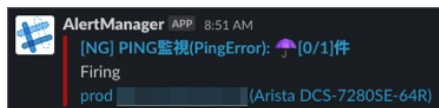
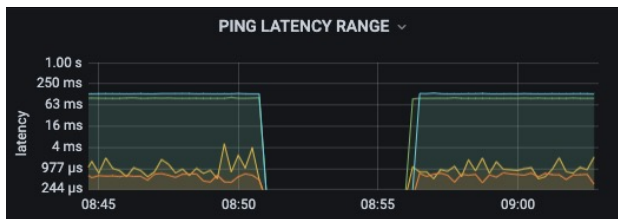
事前に syslog をレベル別仕分け

Warn	watchfrr[18602]: [EC 268435457] bgpd state -> down : read returned EOF
Notice	bgpd[10014]: %ADJCHANGE: neighbor swp55(neighbor_hostname) in vrf default Down Interface down
Info	systemd[1]: Reloading FRRouting.
Ignore	switchd[2573]: sync.c:3802 IPv4 Route Summary (2553556) : 1 Added, 0 Deleted, 0 Updated, 0 Skipped in 23468 usecs



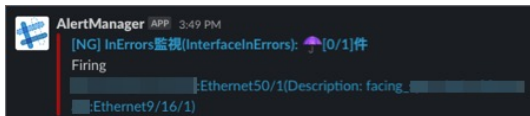
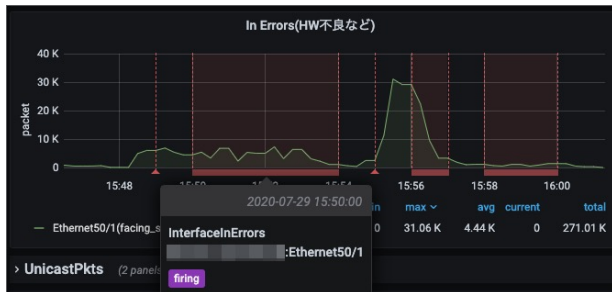
Prometheus によるモニタリング

PING



緊急度が高いアラートは通知+電話(夜間)

Interface Error



集めた Metrics 情報を分析

Prometheusとは

P 15

- Pull型(HTTP)のメトリクス監視ツール
 - ◆ Inspired by Google's Borgmon
- Alert管理機能を標準装備
 - ◆ Alertを発生させることが出来るし、管理ができる
- 多彩なService Discoveryに対応
 - ◆ OpenStack, Kubernetes, StaticFile ...
 - ◆ 監視対象を自動的に見つけてくれる
- 公式で様々なメトリクス取得方法を提供
 - ◆ `snmp_exporter`, `blackbox_exporter`, `node_exporter` ...



<https://www.janog.gr.jp/meeting/janog40/program/soft>
JANOG40: フロントエンドエンジニアがY!のNW部隊で取り組んだこと

Prometheus の構成要素

- **Prometheus**
 - Exporter に対して Scraping を掛けて Metrics を収集する
 - Exporter から集めた情報を貯める
 - Alertmanager に対してアラートを通知する
- **SNMP Exporter / Blackbox Exporter / XXXX Exporter**
 - NW機器から情報を取得して Metrics を生成する実体
 - Prometheus からScrape されると、実際に Metrics を生成する
- **Alertmanager**
 - 通知されたアラートのハンドリング

Prometheus を使う理由

MRTG や Syslog では実現の難しい監視/可視化を Prometheus を使うことで実現したい

綺麗な UI で可視化
Grafana + PromQL

Intent ベースのアラート

Prometheus サーバの冗長性

スケーラビリティ
長期間の Metrics 可視化

Prometheus

Clos 向けネットワーク機器の監視状況

SNMP/Blackbox Exporter に加えて

Cumulus のセンサ情報/HWリソース/FRRの各種エントリ数

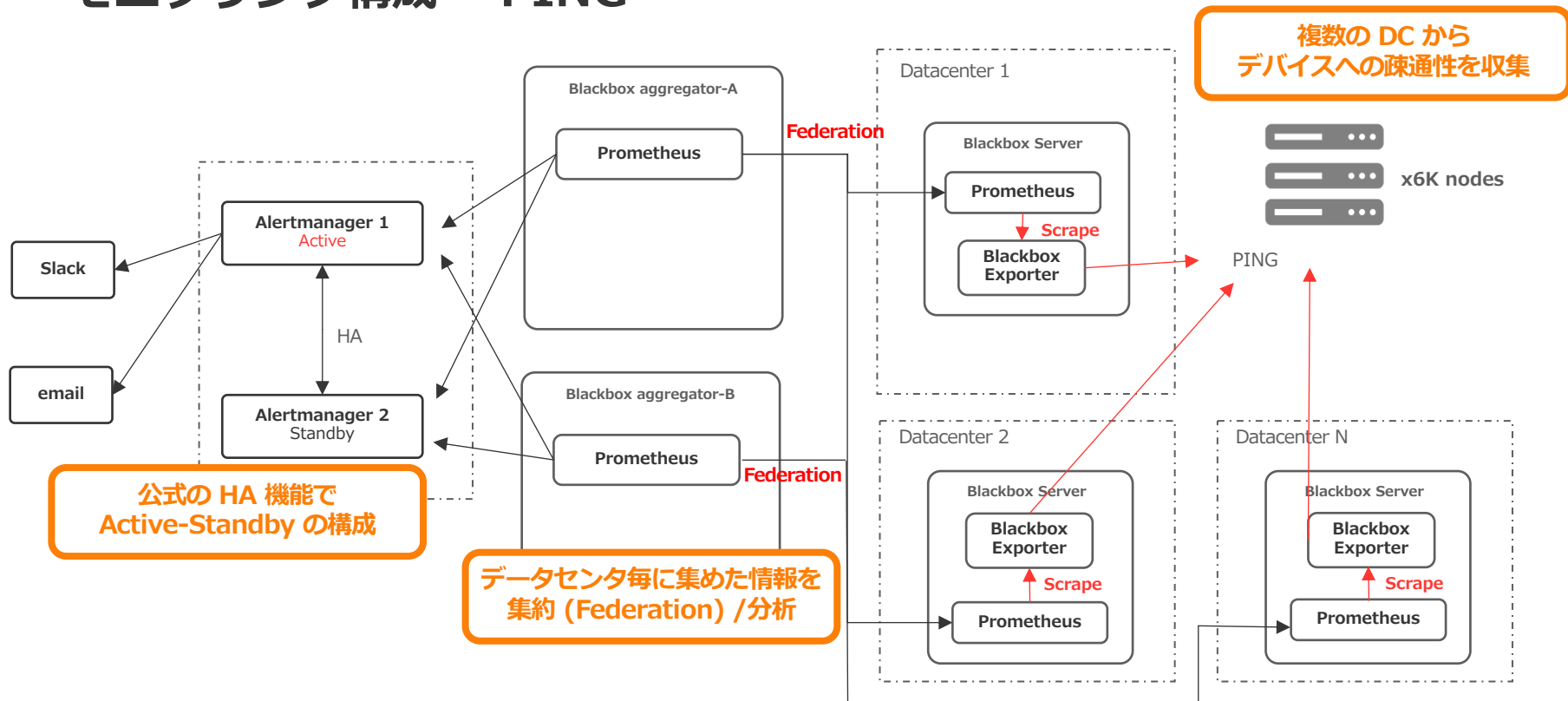
PING

SNMP

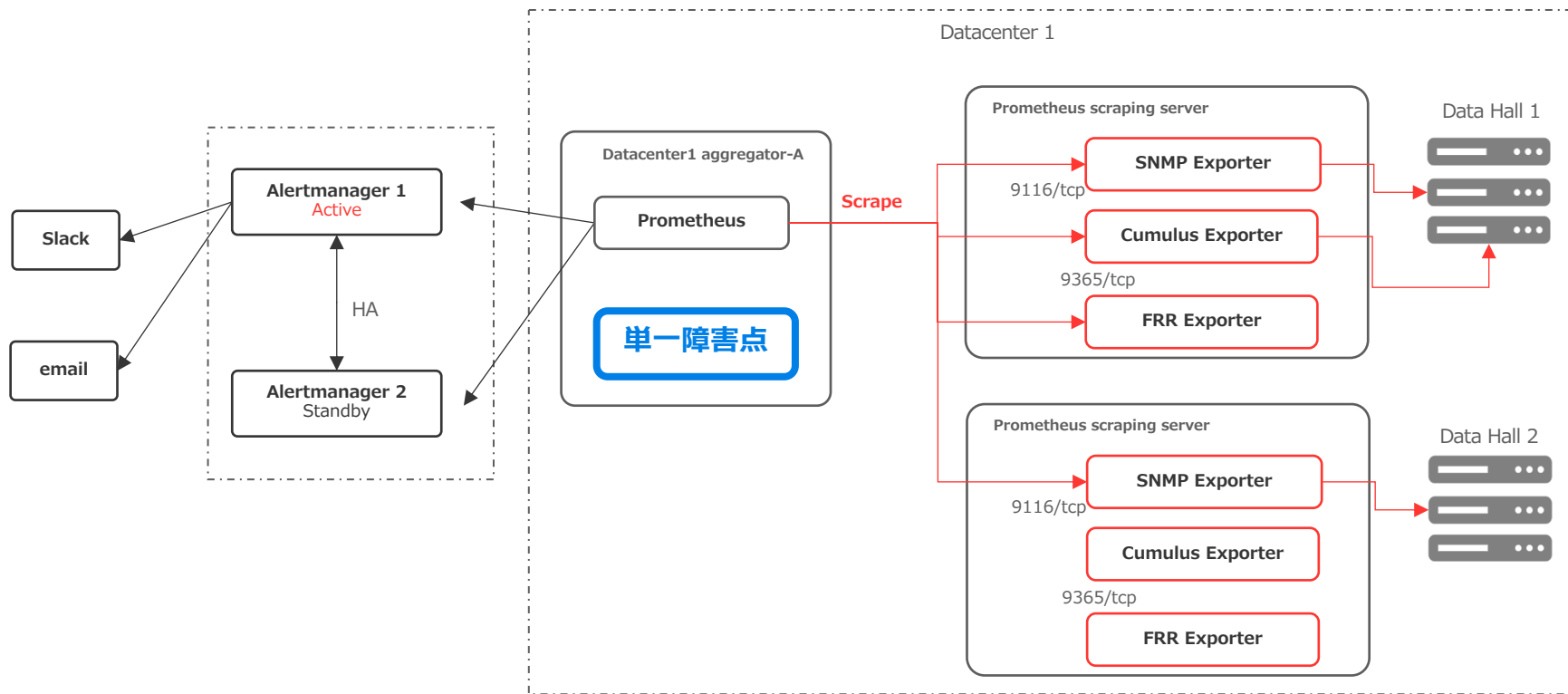
Cumulus
HWセンサ/
FRR utilization

https://github.com/tytany/cumulus_exporter
https://github.com/tytany/frr_exporter

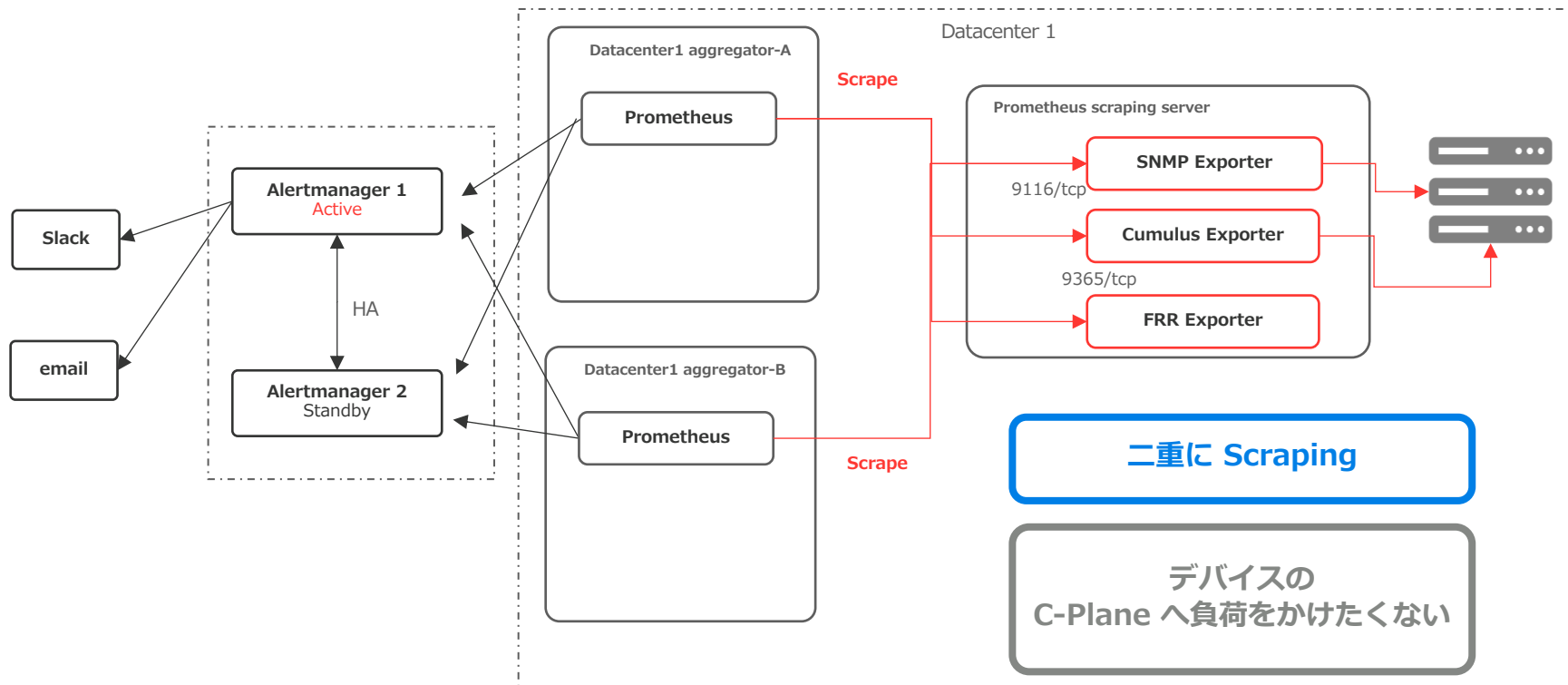
モニタリング構成 - PING



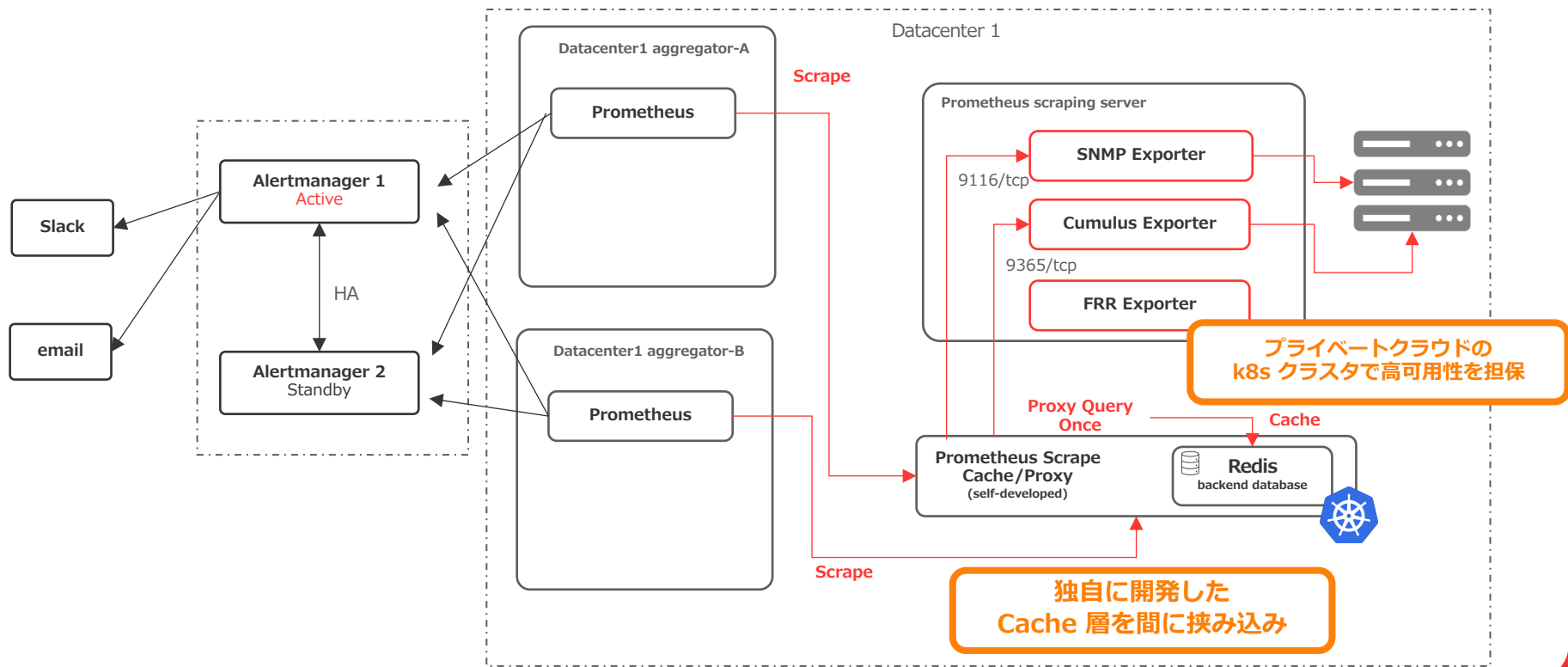
モニタリング構成 – SNMP/3rd party exporter



モニタリング構成 – 冗長性 vs Scrape頻度



モニタリング構成 - キャッシュ

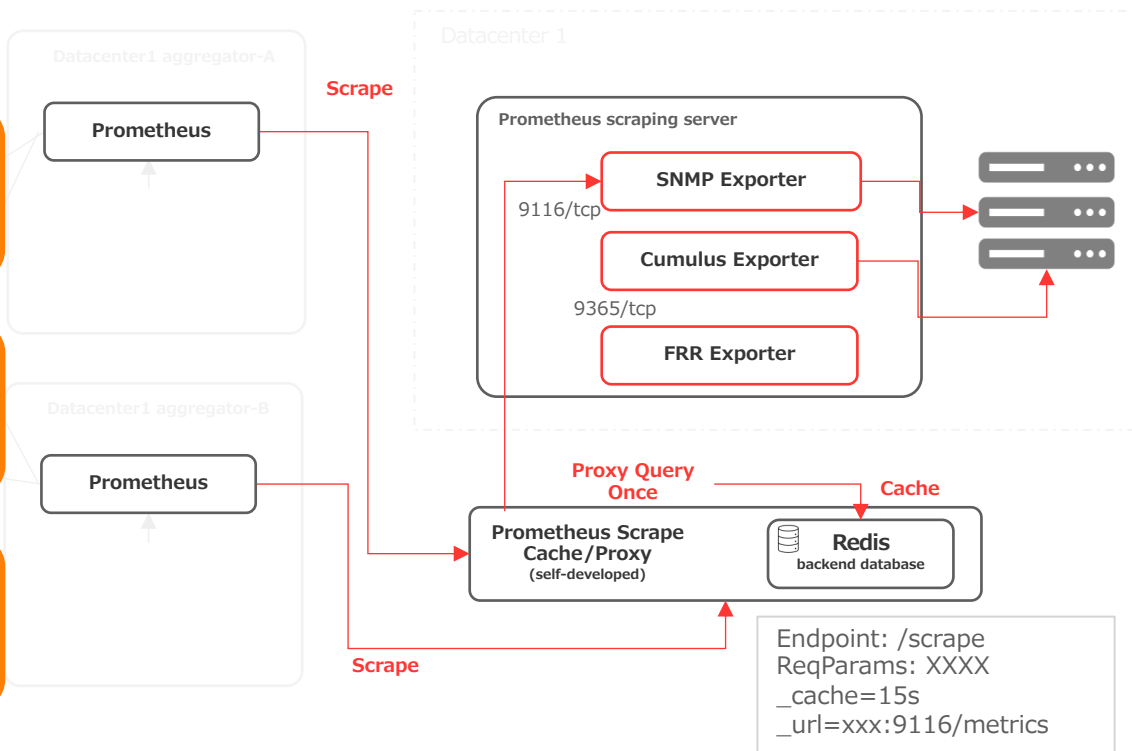


Prometheus Scrape Cacher

① リクエストパラメータ毎に
二重scrapeを防止する
2回目の scrape は cache を返す

② リクエストパラメータ毎に
任意の時間
Metrics情報を Cache できる

③ 1つのエンドポイントを利用して
任意の宛先ポートに Proxy できる



Prometheus を使う理由

MRTG や Syslog では実現の難しい監視を Prometheus を使うことで実現したい

綺麗な UI で可視化
Grafana + PromQL

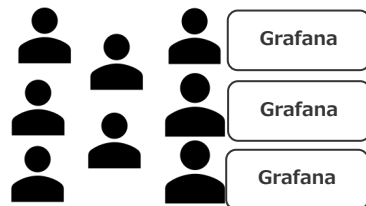
Intent ベースのアラート

Prometheus サーバの冗長性

スケーラビリティ
長期間の Metrics 可視化

Prometheus

可視化の課題



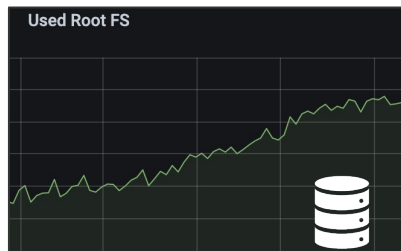
高速に閲覧したい



キャッシュ

<https://github.com/trickstercache/trickster>

Prometheus への Query 結果をキャッシュ



Prometheus の時系列DBが
増え続ける
貯め続けたい



Thanos

<https://thanos.io/>

オブジェクトストレージを利用して
長期間の Metrics を管理するアプリケーション

Prometheus

Thanos のコンポーネント (一部)

Thanos Sidecar

Prometheus へのクエリを発行

オブジェクトストレージに
TSDBをアップロード

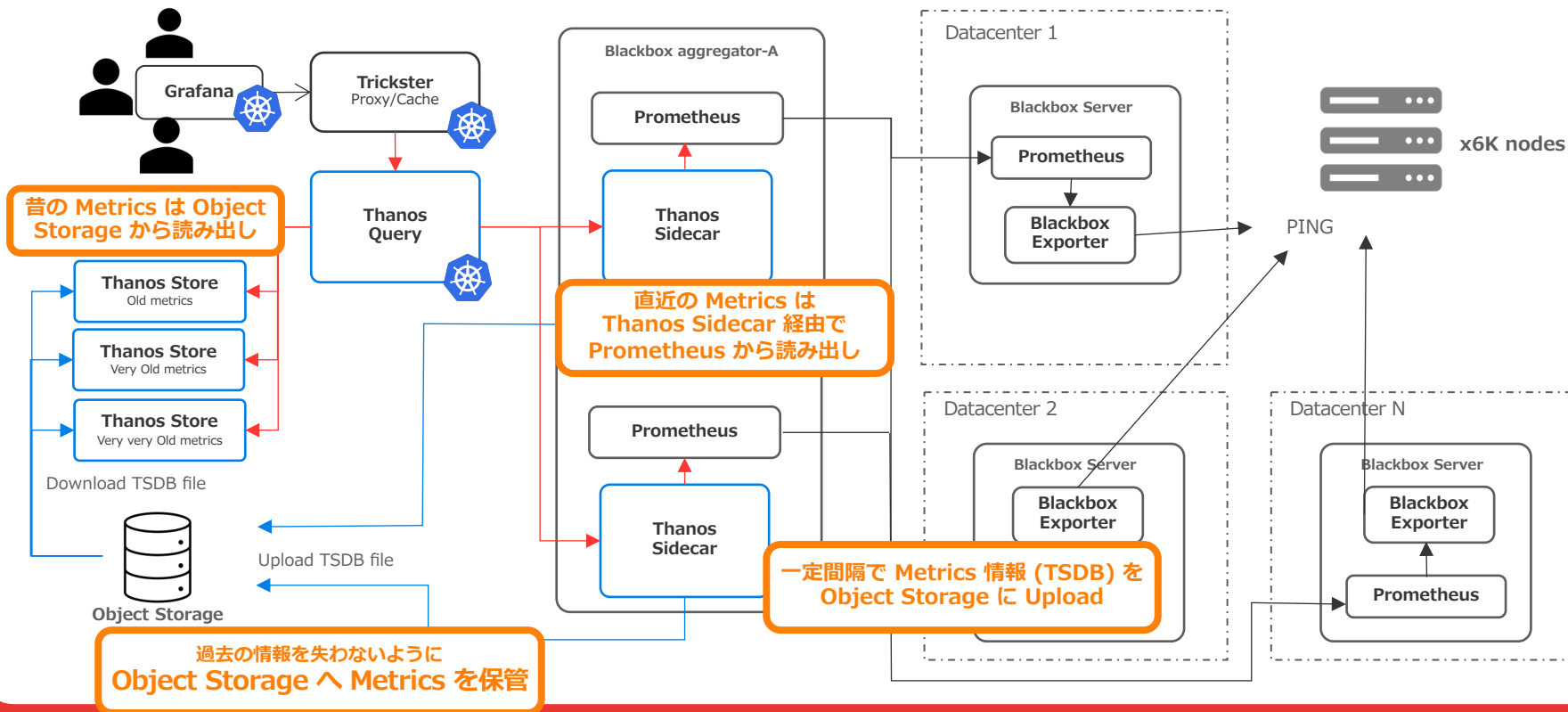
Thanos Store

オブジェクトストレージに
保管されたTSDBを参照する
API

Thanos Query

Thanos Sidecar や
Thanos Store に
PromQL を発行/
結果をマージする

Thanos を併用した可視化構成



本日のまとめ

- Zero Touch Provisioning + Ansible を組み合わせてネットワークを構築
 - 一般的な DHCPサーバ/Webサーバを使って SSH できるまで Zero Touch
- Ansible
 - 複数の機種で config template を使い回せる工夫
 - 構築だけでなく、オペレーションも Ansible Playbook 管理で事故防止
 - 常により良い構成管理をするための改良を重ねている
- Prometheus を使ったネットワーク監視
 - 様々な Exporter を駆使して情報を収集
 - 冗長化された Prometheusサーバで情報を分析、アラートの仕組みを構築
 - 集めた情報は大変価値がある
 - Thanos を利用して Object Storage に過去のMetricsを保管
 - 後から参照できるように

YAHOO!
JAPAN