JANOG49 発表資料

ネットワーク運用自動化(NetOps)の実運用 苦労/解決/妥協した話

2022年1月26日 15:15-15:45

NRIセキュアテクノロジーズ株式会社
マネージドセキュリティサービス事業本部
MSS基盤マネジメント部

三輪 佳紀
Oyunbileg Chingun

自己紹介

三輪 佳紀 (みわ よしき)

Japan
1982年生まれ(39歳)
グループマネージャー

■ **業務**
・ MSS（マネージドセキュリティ）サービスを提供する会社で業務を支えるインフラからアプリケーションまで広く浅くやっています
(AS運用、サーバ基盤、アプリ開発etc...）

Chingun S.Oyunbileg（オユンビレグ チングン）

Mongolia → Japan
1984年生まれ (37歳)

■ **略歴**
・以前某通信キャリアの情シス/技術調査開発の部隊でいろんなPoC開発・検証開発をやっていた
・NRIセキュアではMSS共通プラットフォームの保守運用、サーバ/ネットワーク運用自動化が主務

■**趣味**
バスケ、水泳（聖地の辰巳プールがスケートリンクに改修されることになり落胆している）

はじめに

わたしたちの運用業務

2,500 運用機器の台数

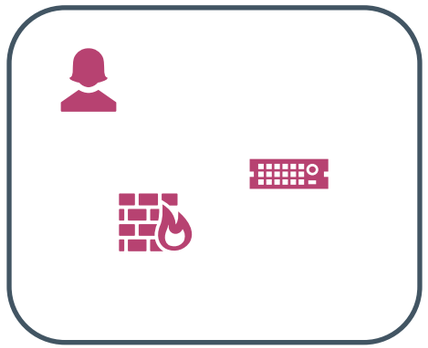
5,000 1年の変更作業の数

わたしたちの運用業務

金融系A社
Web環境

A社担当

サービス系B社
OA環境



B社担当

インフラ系C社
リモートアクセス環境

C社担当

金融系D社
EDR環境

D社担当

・・・

わたしたちの変更作業

【作業手順】 作業概要: 【 XXX01 】 Firewall設定変更
作業日時: yyyy/mm/dd 0:00 ～ 0:00

作業体制:

NRI Secure	xxx	87-****
作業担当	www	87-****
サポート	-	82-9567

項番	日時	作業項目	ダイヤ21F : www 作業内容	ダイヤ20F : xxx 作業内容	確認 時刻記入
	yyy/mm/d		■ Firewall設定変更 (xxxsrx01)		
1	0:00	監視切替え	87-**** ← ←	■ ユーザーへの作業開始連絡（サポート） ■ 作業開始を連絡（サポート）	
2		ログイン経路	■ Telnetログイン fncway >> xxxsrx01 (1) xxxsrx01 : telnet xxxsrx01		
3		xxxsrx01 事前確認	■ 下記コマンドにより機器の状態を確認 (1) show system uptime → システム時刻が合っていることを確認 (2) show ntp associations → NTPが引けていることを確認 (3) show version → ホスト名、OSバージョンを確認 (4) show interfaces terse → Interface情報が想定通りであることを確認 (5) show interface extensive | no-more (6) show log messages | last 100 → 通常出ないアラームが無いことを確認 (7) show chassis routing-engine → CPU使用率が上昇していないことを確認 (8) show chassis fan → fanのstatusが"OK"であることを確認（コマンド自体無い時は確認不要）		
4		xxxsrx01 事前確認	■ config を取得し、オリジナルconfig と比較 (1) show configuration | display set | no-more xxxsrx01-1.txt で保存 (2) xxxsrx01_org.txt とdiff → 差分が無いことを確認		
5	0:10	xxxsrx01 設定変更～設定保存	■ paste用 config を使用し、設定変更を実施 (1) configure コンフィグレーションモードへ移動 (2) xxxsrx01_yyyymmdd_paste.txt		

運用自動化への取り組み

目指したかった姿

金融系A社
Web環境

サービス系B社
OA環境

インフラ系C社
リモートアクセス環境

金融系D社
EDR環境

画一的なInterfaceで変更作業を効率化/自動化したい

NETOPS SRX

A社担当

B社担当

C社担当

D社担当

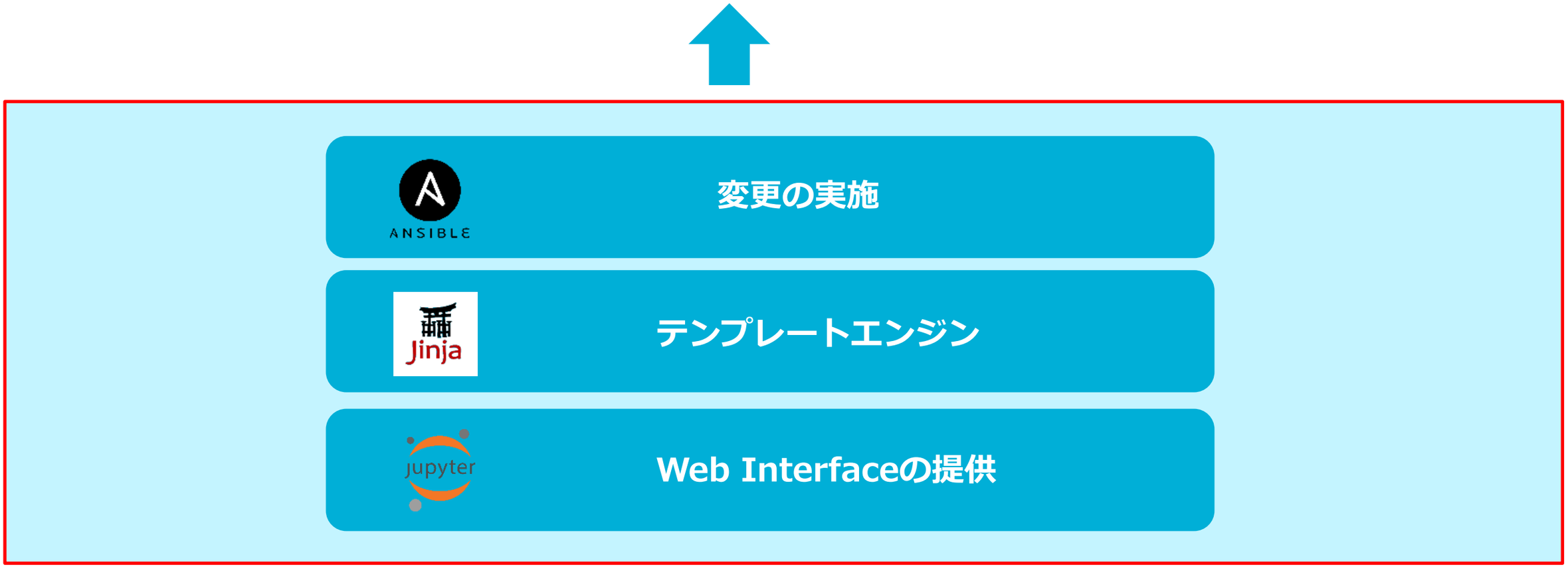
いろいろ苦労しました

作るぞ！

いろいろ苦労しました（設計編）

試行錯誤がありました

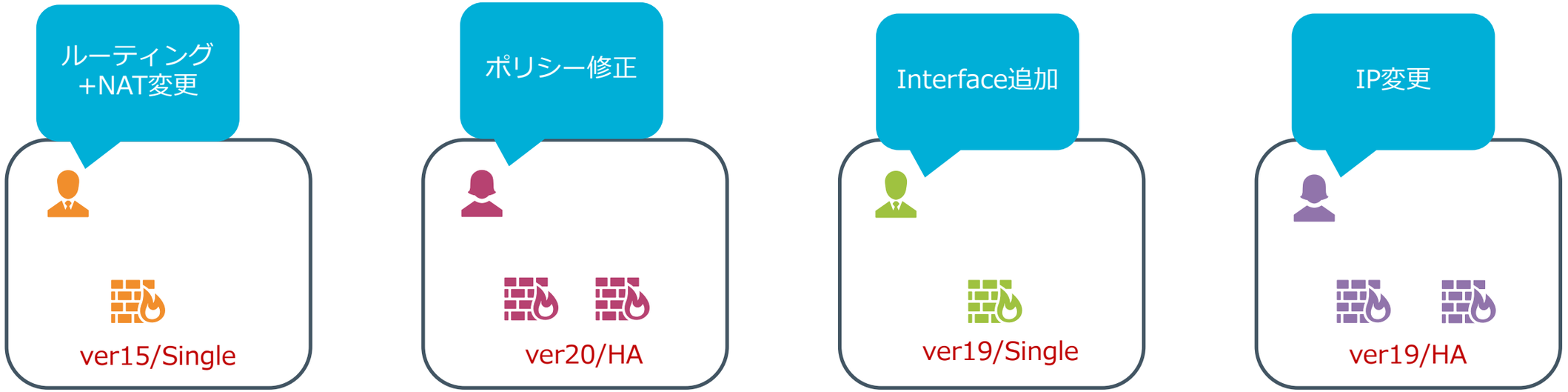
初期構成案：**雛形手順をテンプレ化する方式**(Jupyter + Jinja2 + Ansible)



うまくいきませんでした

環境が色々あるが故に、**変更作業のパターンが多すぎる**

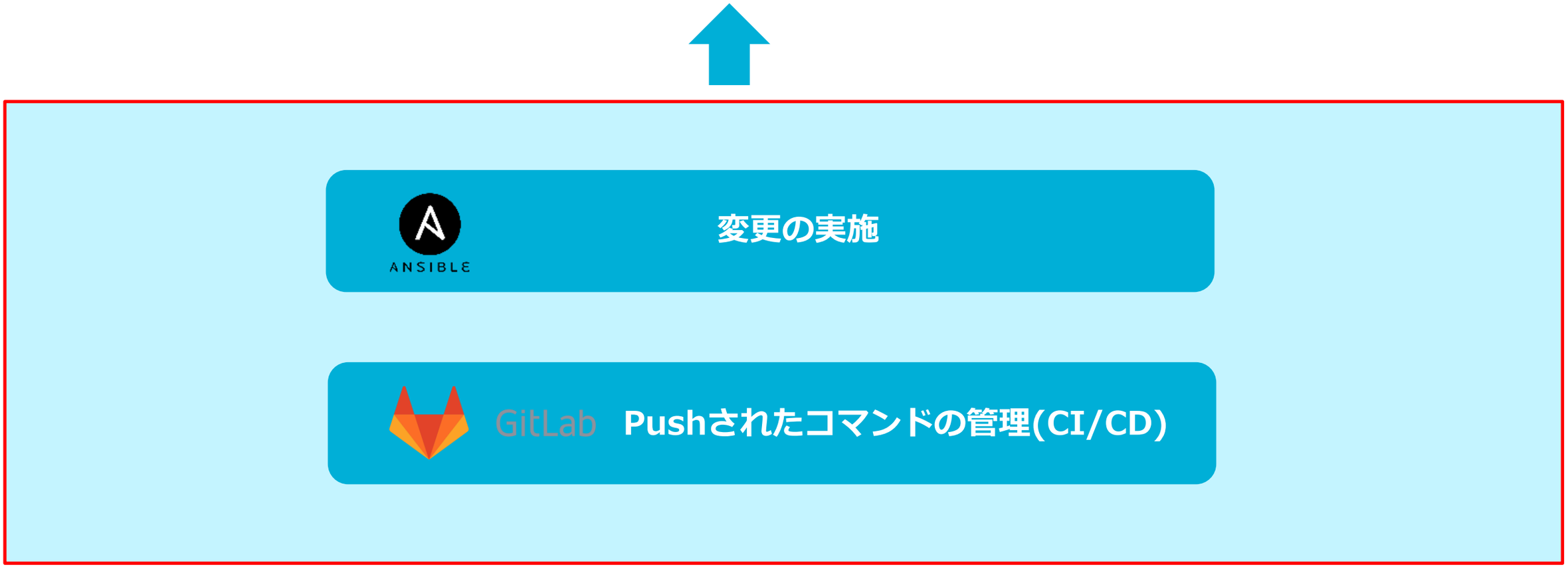
環境が色々あるが故に、**機器の構成やバージョンなども様々**



全ての構成に応じた変更パターンをテンプレート化することは非現実的

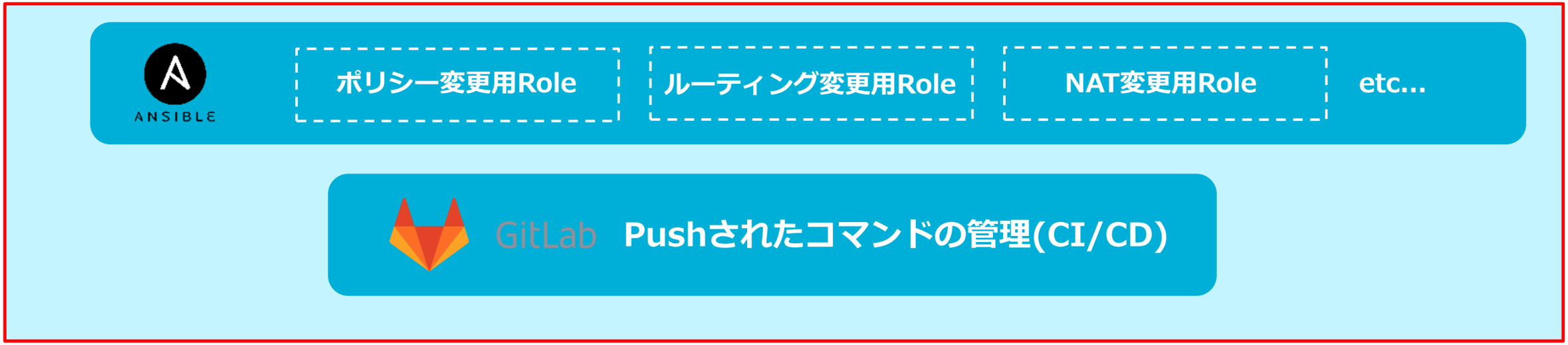
試行錯誤がありました その2

初期構成案：**設定変更コマンドを入力させる方式**(Gitlab CI/CD + Ansible)



やっぱりうまくいきませんでした

- AnsibleのRoleを変更作業のパターンに応じて実装しなくてはいけない
- 利用側もある程度Ansibleに習熟している必要がある



? ? ? ?

Ansibleに明るくないメンバーが多いため教育コストが高くなってしまう

設計で意識しないといけなかったこと

事業の特性

- **カバーしなくてはいけない業務**は何か？
- **想定される変更作業のパターン**はどのようなものがあるのか？

要員の特性

- **要員のスキル**はどの程度か？

他社の正解事例が自社の正解とは限らない

最終的にこうなりました

- **事業の特性**　変更作業のパターンが多く、定型化が難しい
- **要員の特性**　ネットワーク機器を触っているためconfigは作れる

→ configを機器にloadする方式で作業を自動化

A社担当　B社担当　C社担当　D社担当

システム構成案の比較

運用者観点

様々な変更作業をカバーしているか？

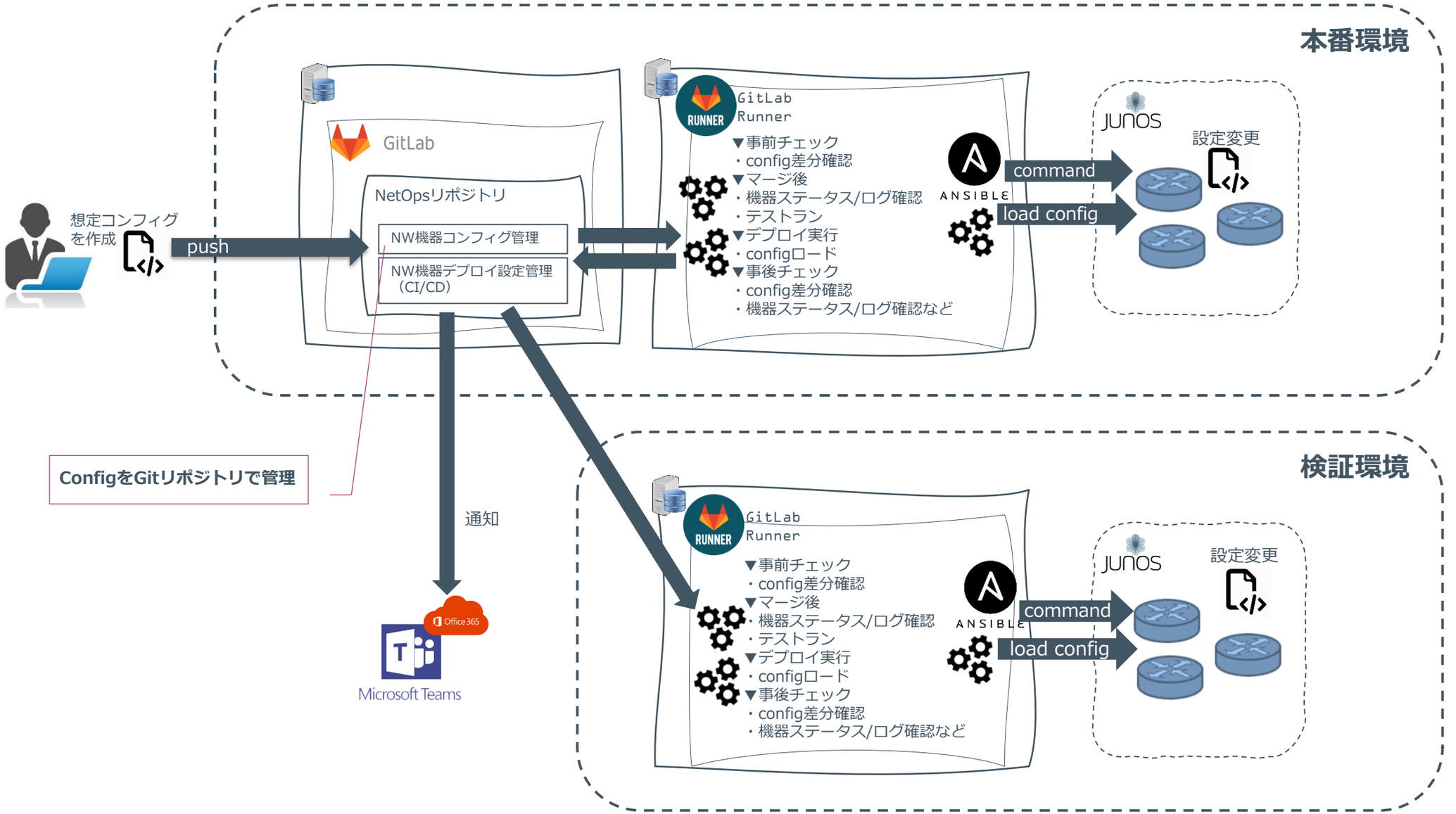
運用者が手間なく利用出来るようになっているか？

開発者観点

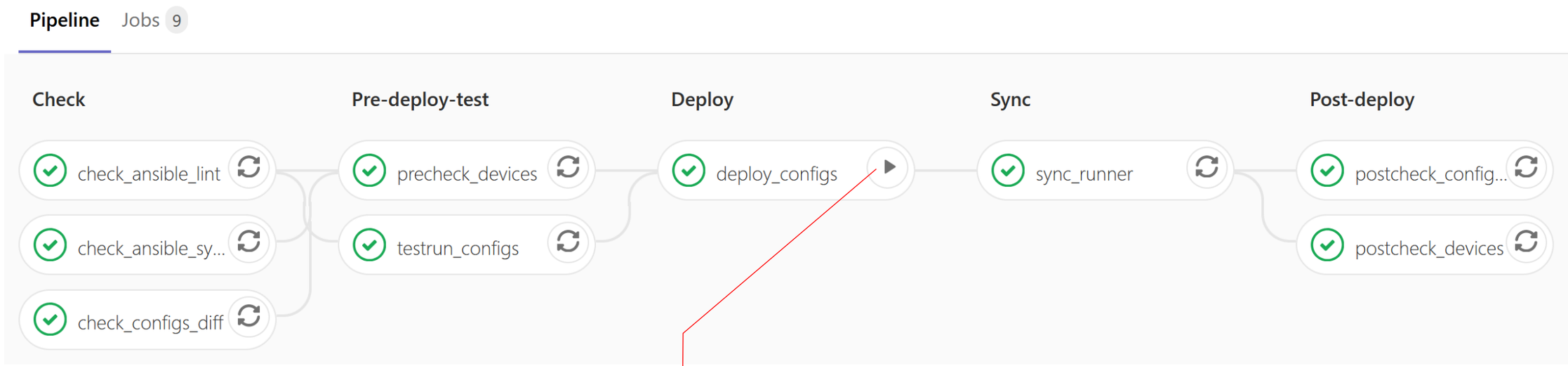
システムが複雑になっていないか？

	カバーできる変更作業のパターン	導入(学習)コスト	システムのメンテ性	
① Jupyter + Jinja + Ansible	△ 作業パターンに応じてテンプレートを準備する必要がある。	○ 直感的なIFなので、学習コストは低い	△ 対象の作業が増えるとテンプレートを準備する必要がある	よくある作業をテンプレート化して実装。**作業が定型化されていれば高い効果が見込める**方式
② Gitlab + Ansible(コマンド投入方式)	△ 作業パターンに応じてAnsible Roleを実装する必要がある。	× Ansibleへの理解が必要	× Ansibleの実装が複雑になりがち	Ansibleを駆使して変更作業の仕組みを実装。**開発側/利用者側ともにAnsibleの理解が必要。**
③ Gitlab + Ansible(config投入方式)	○ 基本的に単一のフォーマットで変更作業に対応	△ Gitの運用について少々理解は必要	○ そこまで複雑な構成にはならない	変更後の想定フルconfigをGitにPUSHしてリロードさせる方式。**作業の定型化が難しい場合においては、運用者/開発者の両面でバランスの良い方式。**

Netops構成(概要)



Gitlab CI/CDのPipeline処理



事前チェック1

・各種構文チェック
・Git ⇔ 実機のconfig比較(作業前)

事前チェック2

（マージ後）

・実機の状態/ログ確認
・テストラン

設定変更の実行

Manual実行

・load overrideでGit上のconfigで上書き

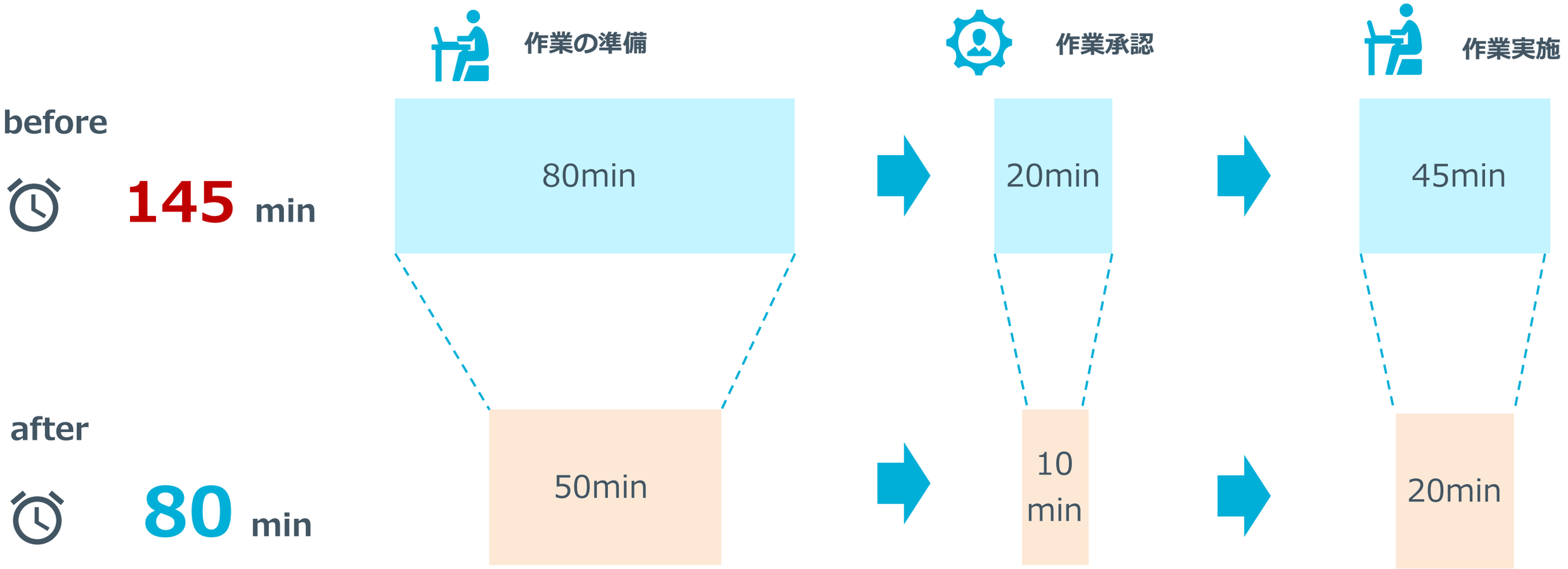
事後処理1

・Gitlab-Runner最新化/リポジトリ同期

事後処理2

・Git ⇔ 実機のconfig比較(作業後)
・設定変更後の実機の状態/ログ確認

導入による効果



1回あたりの作業時間としては**約45%削減**

いざ運用開始、でも・・・

出来た！

いろいろ苦労しました（運用編）

なんとかなってないこともたくさんあります

実機が直接操作されてしまうことが度々ある

- **障害対応**
- **セキュリティインシデント対応**
- **複雑な設定変更**

こういったケースは**ある程度発生するものと割り切る**

予防的観点ではなく、**発見的観点**で差分が出ている状況を検出できるようにする

なんとかなってないことの続き

構成やモデル、OSのバージョンによってコマンドの結果は変わる

投入したconfigもそのまま実機に反映されるとは限らない

Juniper SRXの例

```
## Last commit: 2021-11-11 00:28:22 JST by root
version 20.4R3.8;
system {
    host-name netinfra-fw03;
    root-authentication {
        encrypted-password "(snip)"; ## SECRET-DATA
    }
    login {
        user admin {
            uid 101;
```

注釈/コメント文が付与される

```
# show chassis fan     →   モデルによってFanの有無が変わる
# show chassis cluster status  →  シングル /HA で出力が変わる
```

構成でコマンドの実行結果が変わる

```
application [ junos-ping ];   →   application junos-ping;
```

投入configが実機側で良しなに変更される

こういったケースはある程度発生するものと割り切る（2度目）

差分を許容する設計にする（完璧を求めない）

なんとかなってないこと

変更自体は自動化できても作業全体の自動化は難しい

変更作業の全体

要件の発生

- 変更要件を伝える

作業の準備

- configを作る
- configのpush

作業の承認

- configの妥当性を確認する

作業の実施

- 反映前の状態確認
- 反映作業の実施

変更パターンが多様であるが故に確認内容も多様。確認作業も完全には自動化出来ない。
(ポリシーを消したら該当の通信が流れていないことを確認するなど・・・)

定型的な確認作業は自動化。作業に応じてアドリブが必要な部分についてはシステム化のスコープからは外す（完璧を求めない その2）

これからのお話

今後の展望

テンプレ方式もうまく併用したい

作業パターンは多岐に渡るとは言え、**よくある系の作業があることも事実**。
テンプレート化することによるメリットも多く、うまく併用できるようにしたい・・・

- 可変パラメータを入れるだけの**簡単なUI**にすることが出来る
- 設定箇所が明確なので、**想定外の設定不備を作りこみにくい**
- 設定箇所が明確なので、**確認作業も含めて定型化しやすい**

検証でも楽をしたい

変更作業は自動化できても**検証は自動化できていない・・・**
今後対象製品やバージョンが増えると 機器 × バージョン × 自動化基盤 の組み合わせで検証工数が大変なことになってしまうため、なんとか**検証も自動化/効率的な仕組みを模索したい・・・**

さいごに

まとめのようなもの

- オペレーションの自動化を考える上では、**事業の特性や携わる要員を理解することが重要**。

- プロジェクトの立ち上がりにおいては理想を求めがち。
  自動化という言葉に夢見がちだが、**時には妥協も必要**。

他社様の事例やモデルケースなど、**隣の芝は青く見えるが青い芝が決して良いとも限らない・・・**

QA / みなさんの経験等を聞かせて下さい

- 変更作業どうされていますか？
- 困っていること、うまくいっていることはありますか？
- 昨今は有償ツールも色々。使われているところはありますか？
- 検証ってどうやってるの？バージョン違いや機種の違いなど。
  自動化とかでうまくやってます？
- そのほかなんでも・・・

NRI SECURE