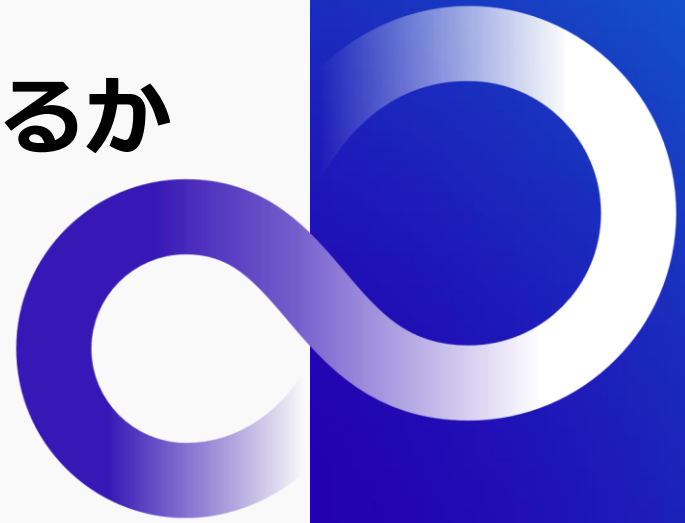
IaaSの裏側で WebAPIを備えていない NW機器にどう設定を入れるか

富士通
システムプラットフォームビジネスグループ

サービスインフラ事業本部
クラウド開発統括部
クラウドネットワーク開発部
山口 将平



自己紹介

FUJITSU

- FJcloud-Vのネットワークサービスを主管するチームのエンジニア
  - ITインフラ系のエンジニアから見たらプログラマ
  - プログラマから見たらITインフラ系のエンジニア

- SNS: @yaaamaaaguuu
  - アイコンはWebプログラマをしていたころの名残
- Like: Emacs, NixOS

本LTで話をするスコープ

FJcloud-V (f.k.a ニフクラ) で提供してるネットワークサービス

- 特にNFV関連の話
- NWのコンポーネントの作成/設定変更/削除が頻繁に起こる環境の話

(余談: 2025/01/27 で FJcloud-V は15周年を迎えます 🎉)

FJcloud-V
NFVサービスの開発運用体制

2020年くらいまでのチーム体制と担務

FUJITSU

- NFV運用チーム
  - 各種NFV製品の使い方設計, 監視運用etc
- APIチーム
  - お客様が利用するWebAPIを作成するチーム
  - NFVチームの設計を受け取り、NFV機器にconfigを入れるAPI作成, 運用

FJcloud-V 15年の運用を経て

FUJITSU

NFVサービスを表現するAPIをNFV運用チームが作る様になった

NFVコンポーネントにいれる設定を迅速に直せる体制がほしいため

- サービスラインナップ拡充や機器リプレースになど伴い,
  設定不可/設定失敗のパターンを見抜く難易度が高くなった
  - バックエンドとなる機器のリプレースに伴う互換性の問題
    - 以前と同様の値を設定可能だが、正しく動かないパターン
    - 同様の動作を実現するために、設定する値を変換する必要があるパターン
    - etc
  - 各コンポーネントの連携が複雑化
    - 複数のコンポーネントにまたがって連携する設定が、操作タイミングによっては変更失敗するなど

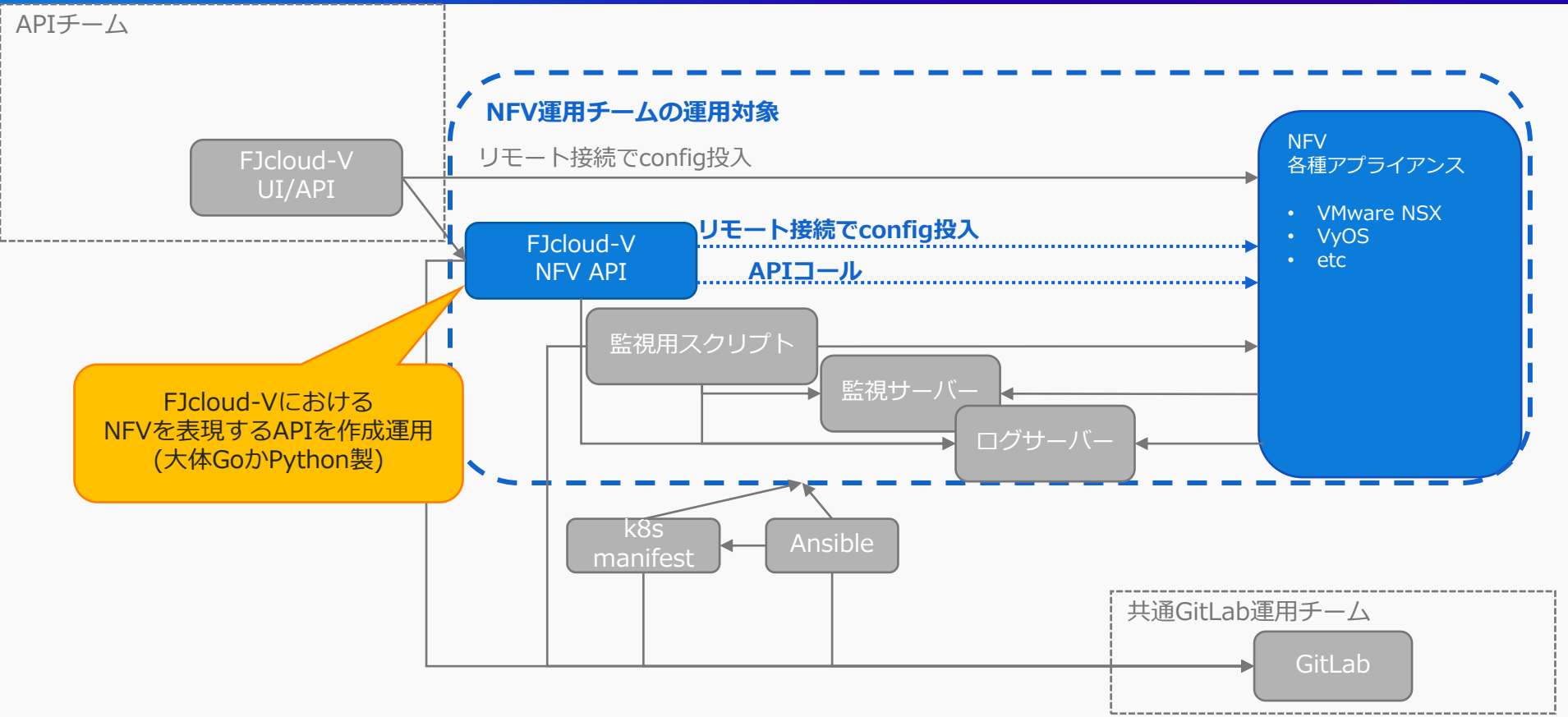
チーム体制と担務の変更

FUJITSU

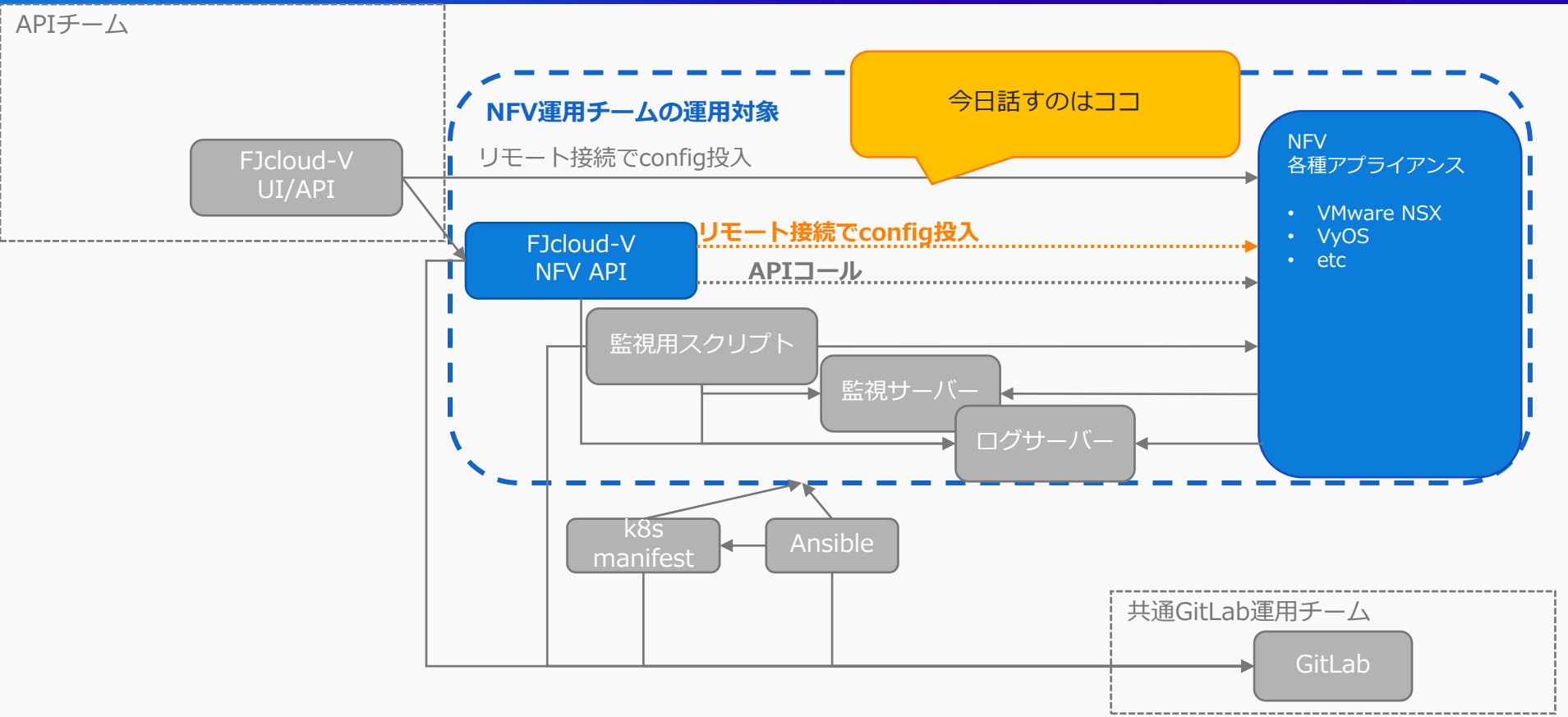
- NFV運用チーム
  - 各種NFV製品の使い方設計, 監視運用etc
  - **FJcloud-VのNFVサービスを表現するAPIを作成中**
    - **FJcloud-VのNFVが実現できることを表現するレイヤを作る**

- APIチーム
  - お客様が利用するWebAPIを作成するチーム
  - NFVチームの設計を受け取り、NFV機器にconfigを入れるAPIを運用
  - **NFV運用チームのAPIをコールする様に徐々に変更中**

FJcloud-V NFV運用の概略図



FJcloud-V NFV運用の概略図



FJcloud-VのNFVを表現するAPIにおけるNFVのconfigの入れかた

今回の話の対象

FUJITSU

WebAPIを利用せずに設定している機器

- VyOS
  - JUNOS likeな設定方法/設定体系を持つLinuxのアプライアンス

対象外) WebAPIがある機器

- WebAPIを順序よく利用するのは別の課題があるが、別の機会があれば話します

VyOSの設定方法

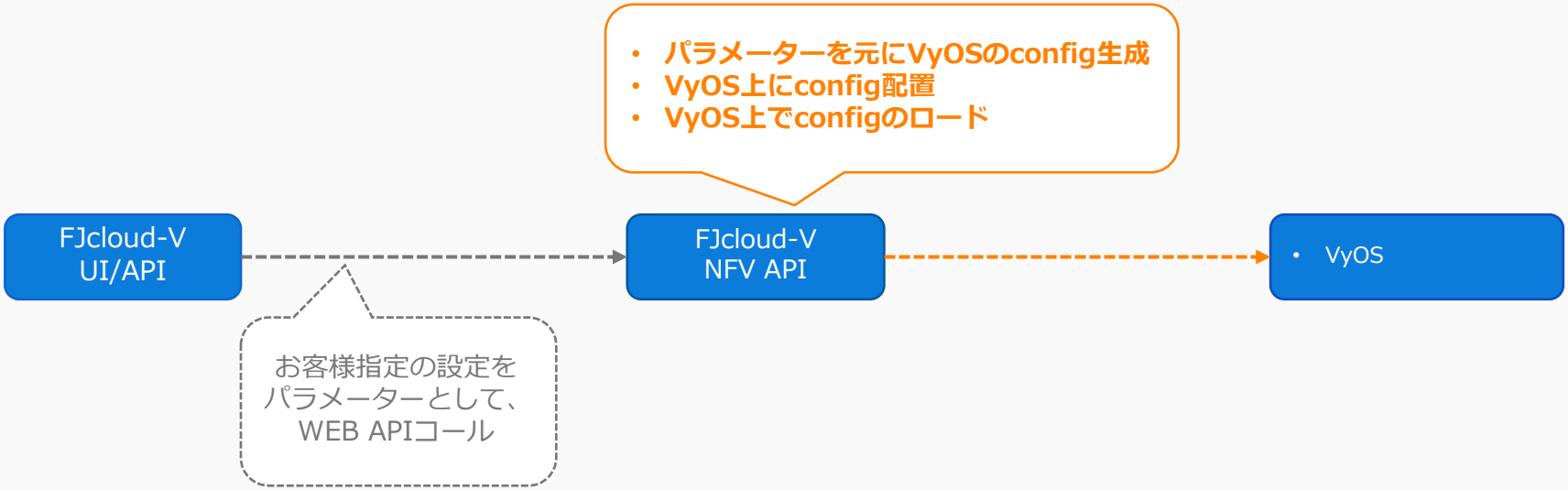
FUJITSU

- APIによる設定
  - 新しいverのVyOSはAPIを備えている
  - 歴史的経緯でAPIは利用していない
- CLIからインタラクティブな操作
  - リモートから接続し、正しい順序でコマンドを複数回発行し自動化するのは、辛いことを過去の経験で学習済みなので避けたい
- CLIを利用して組み立て済みのconfigファイルの読み込み
  - VyOS内に最終目標とするconfigのファイルを配置し、ロードコマンドを実行することで設定変更
  - 使えそう！

VyOSのconfig例

```
interfaces {
    ethernet eth0 {
        address dhcp
        duplex auto
        hw-id 00:00:53:00:53:01
        smp_affinity auto
        speed auto
    }
}
service {
    ssh {
        disable-host-validation
        disable-password-authentication
    }
}
system {
    host-name vyos
    name-server 127.0.0.1
    syslog {
        host 192.0.2.1 {
            facility all {
                level info
            }
            facility protocols {
                level debug
            }
        }
    }
    time-zone Asia/Tokyo
}
```

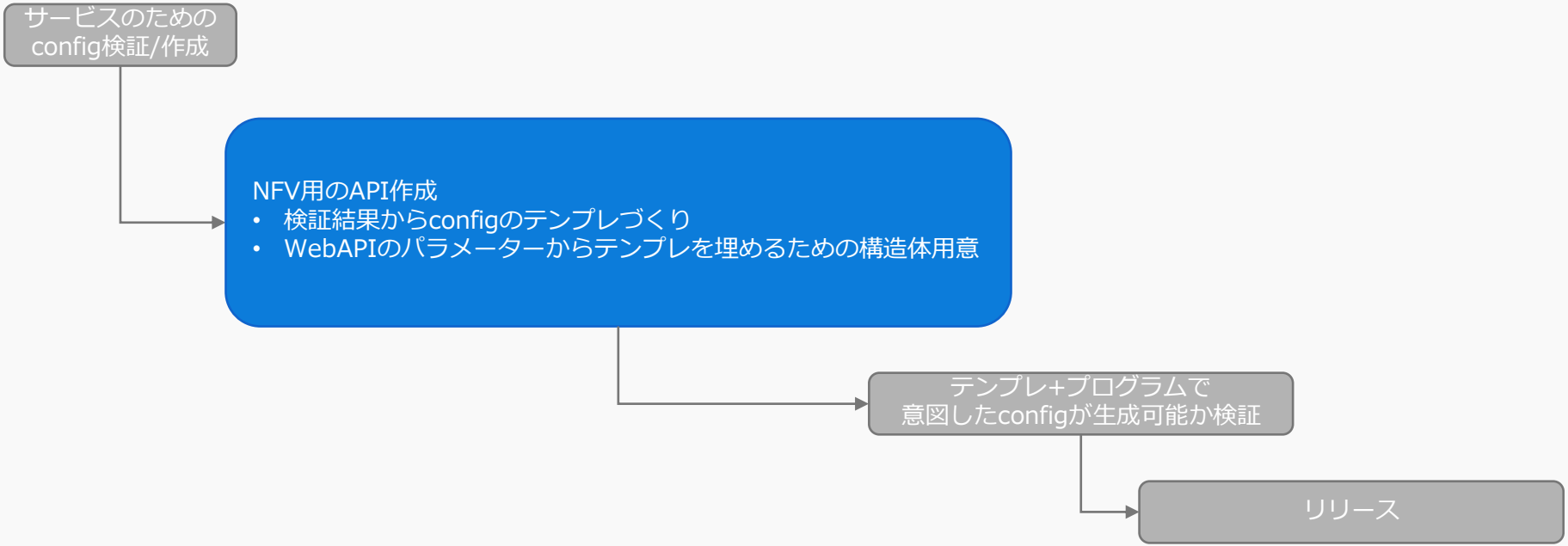
実現したいこと



config投入APIを作るためにまず思いつく案

FUJITSU

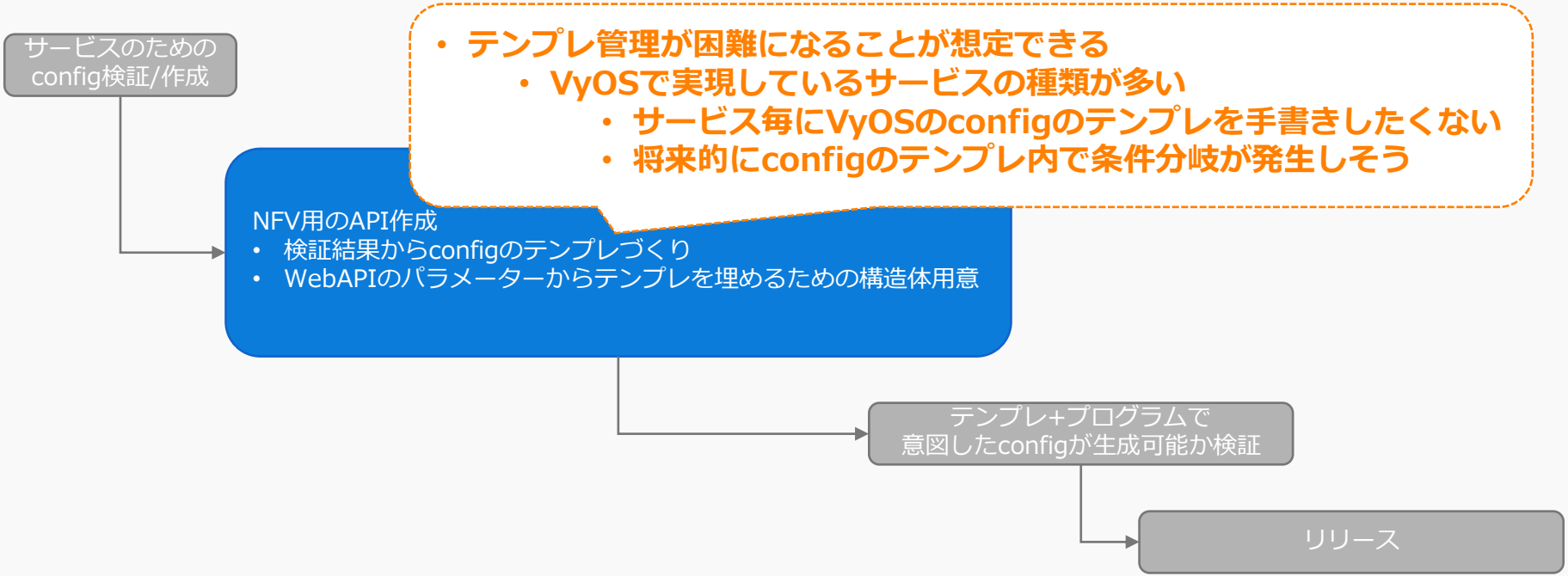
サービスに合わせて、VyOSのconfigのテンプレを作る案



configテンプレで事前に思いつく課題

FUJITSU

サービスに合わせて、VyOSのconfigのテンプレを作る案



課題解決案

- NW機器のconfigは構造的
- API内ではconfigに埋めるデータを構造的に持たせている
- プログラムのデータ構造をJSONやYAMLに変換できるなら、NW機器のconfigにも変換できるのでは？
- プログラムのデータ構造からVyOSのconfigを生成できれば、configのテンプレ運用から離れられるのでは？

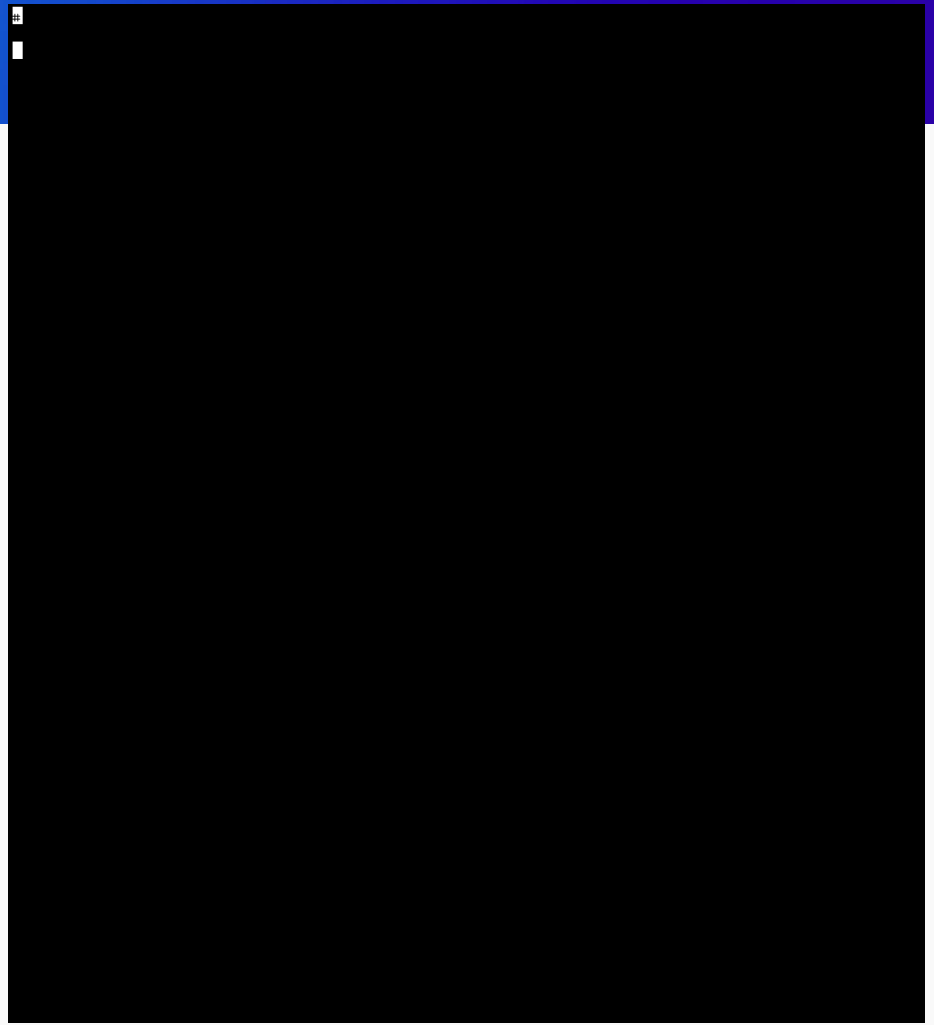
```
interfaces {
    ethernet eth0 {
        address dhcp
        duplex auto
        hw-id 00:00:53:00:53:01
        smp_affinity auto
        speed auto
    }
}
service {
    ssh {
        disable-host-validation
        disable-password-authentication
    }
}
system {
    host-name vyos
    name-server 127.0.0.1
    syslog {
        host 192.0.2.1 {
            facility all {
                level info
            }
            facility protocols {
                level debug
            }
        }
    }
    time-zone Asia/Tokyo
}
```

実際にやってみた

VyOSとGo言語を相互変換するものはなかったので作った

- 作ったもの
  - VyOSのconfigをもとに、Go言語の構造体を生成
  - Go言語の構造体をもとに、VyOSのconfigを生成
- 参考にしたもの
  - **低レイヤを知りたい人のためのCコンパイラ作成入門**
  - **Go言語のencoding/json**

デモ



FUJITSU

デモ解説1

- vyos2goというプログラムに, 検証後のVyOSのconfigを入力
  - 入力したconfigに対応したGoの構造体が生成される
  - デモでは vyos.go に保存
- app.go から vyos.go 内の構造体を利用
  - 値を埋めてシリアライズすると、VyOSの設定が生成される
    - ※デモでは省略しているが、バリデーション済みの値を利用することを想定している
- WebAPIでのユースケース
  - 生成したconfigをリモート接続して配置
  - リモート接続してconfigをロードするコマンドの発行

デモ解説2

FUJITSU

VyOSのconf

```
interfaces {
    ethernet eth0 {
        address dhcp
        duplex auto
        hw-id 00:00:53:00:53:01
        smp_affinity auto
        speed auto
    }
}
service {
    ssh {
        disable-host-validation
        disable-password-authentication
    }
}
system {
    host-name vyos
    name-server 127.0.0.1
    syslog {
        host 192.0.2.1 {
            facility all {
                level info
            }
            facility protocols {
                level debug
            }
        }
    }
    time-zone Asia/Tokyo
}
```

VyOSのconfから生成したGoの構造体

```go
// Code generated by vyos2go DO NOT EDIT.

package main

type HostFacility struct {
        FacilityLevel string `vyos:"level"`
}

type SyslogHost struct {
        HostFacility map[string]HostFacility `vyos:"facility"`
}

type SystemSyslog struct {
        SyslogHost map[string]SyslogHost `vyos:"host"`
}

type RootSystem struct {
        SystemHostName   string       `vyos:"host-name"`
        SystemNameServer string       `vyos:"name-server"`
        SystemSyslog     SystemSyslog `vyos:"syslog"`
        SystemTimeZone   string       `vyos:"time-zone"`
}

type ServiceSsh struct {
        SshDisableHostValidation         bool `vyos:"disable-host-validation"`
        SshDisablePasswordAuthentication bool `vyos:"disable-password-authentication"`
}

type RootService struct {
        ServiceSsh ServiceSsh `vyos:"ssh"`
}

type InterfacesEthernet struct {
        EthernetAddress     string `vyos:"address"`
        EthernetDuplex      string `vyos:"duplex"`
        EthernetHwId        string `vyos:"hw-id"`
        EthernetSmpAffinity string `vyos:"smp_affinity"`
        EthernetSpeed       string `vyos:"speed"`
}

type RootInterfaces struct {
        InterfacesEthernet map[string]InterfacesEthernet `vyos:"ethernet"`
}

type Root struct {
        RootInterfaces RootInterfaces `vyos:"interfaces"`
        RootService    RootService    `vyos:"service"`
        RootSystem     RootSystem     `vyos:"system"`
}
```

デモ解説3

VyOSのconfから生成したgoを利用したアプリケーション

```go
package main

import (
	"fmt"
	"vyos2go-demo/govyos"
)

func main() {
	root := Root{
		RootInterfaces: RootInterfaces{
			InterfacesEthernet: map[string]InterfacesEthernet{
				"eth0": InterfacesEthernet{
					EthernetAddress: "dhcp",
					EthernetDuplex: "auto",
					EthernetHwId: "00:00:53:00:53:01",
					EthernetSmpAffinity: "auto",
					EthernetSpeed: "auto",
				},
			},
		},
		RootService: RootService{
			ServiceSsh: ServiceSsh{
				SshDisableHostValidation: true,
				SshDisablePasswordAuthentication: true,
			},
		},
		RootSystem: RootSystem{
			SystemHostName: "example",
			SystemNameServer: "127.0.0.1",
			SystemSyslog: SystemSyslog{
				SyslogHost: map[string]SyslogHost{
					"192.0.2.2": SyslogHost{
						HostFacility: map[string]HostFacility{
							"all": HostFacility{
								FacilityLevel: "info",
							},
							"protocols": HostFacility{
								FacilityLevel: "debug",
							},
						},
					},
				},
			},
			SystemTimeZone: "Asia/Tokyo",
		},
	}
	if result, err := govyos.Marshal(root); err != nil {
		panic(err)
	} else {
		fmt.Println(string(result))
	}
}
```

VyOSのconf

```
interfaces {
    ethernet eth0 {
        address dhcp
        duplex auto
        hw-id 00:00:53:00:53:01
        smp_affinity auto
        speed auto
    }
}
service {
    ssh {
        disable-host-validation
        disable-password-authentication
    }
}
system {
    host-name example
    name-server 127.0.0.1
    syslog {
        host 192.0.2.2 {
            facility all {
                level info
            }
            facility protocols {
                level debug
            }
        }
    }
    time-zone Asia/Tokyo
}
```

やってみた結果

FUJITSU

- 大体3日くらいで作れた
- 狙い通りテンプレの手書きは避けられている
- configにシリアライズしてロードさせるだけなのでアプリの実装も楽

- 新規サービスでは割とうまく行っている
  - 2023年にメジャーバージョンアップしたサービスではこの仕組みを使っている
- 既存サービスに対する適用はこれから

展望と取り組みからの考察

FUJITSU

- 今回紹介したプログラムは需要があればOSSとして公開するかも
- 今回はVyOSのconfigを対象としたが、他のNW機器も同様のことができるはず
  - 実際、NW機器のconfigをPythonの辞書に変換するようなOSSはありそう
- NW機器のconfigとプログラミング言語の構造を相互変換可能ならば、JSONやYAMLなどにも変換できるはず
  - 各種機器のconfigをJSONやYAMLに落とし込める仕組みを作ると運用を楽にできるかもしれない
  - configをかき集めてJSONに変換して、jqで集計するシェルスクリプトをつくるなど

Thank you