ユーザによるセルフマネージ可能なネットワーク サービス運用システムの事例

NTTコミュニケーションズ株式会社
技術開発部
田島 照久

ONIC 2019@軽井沢大賀ホール
2019/11/01

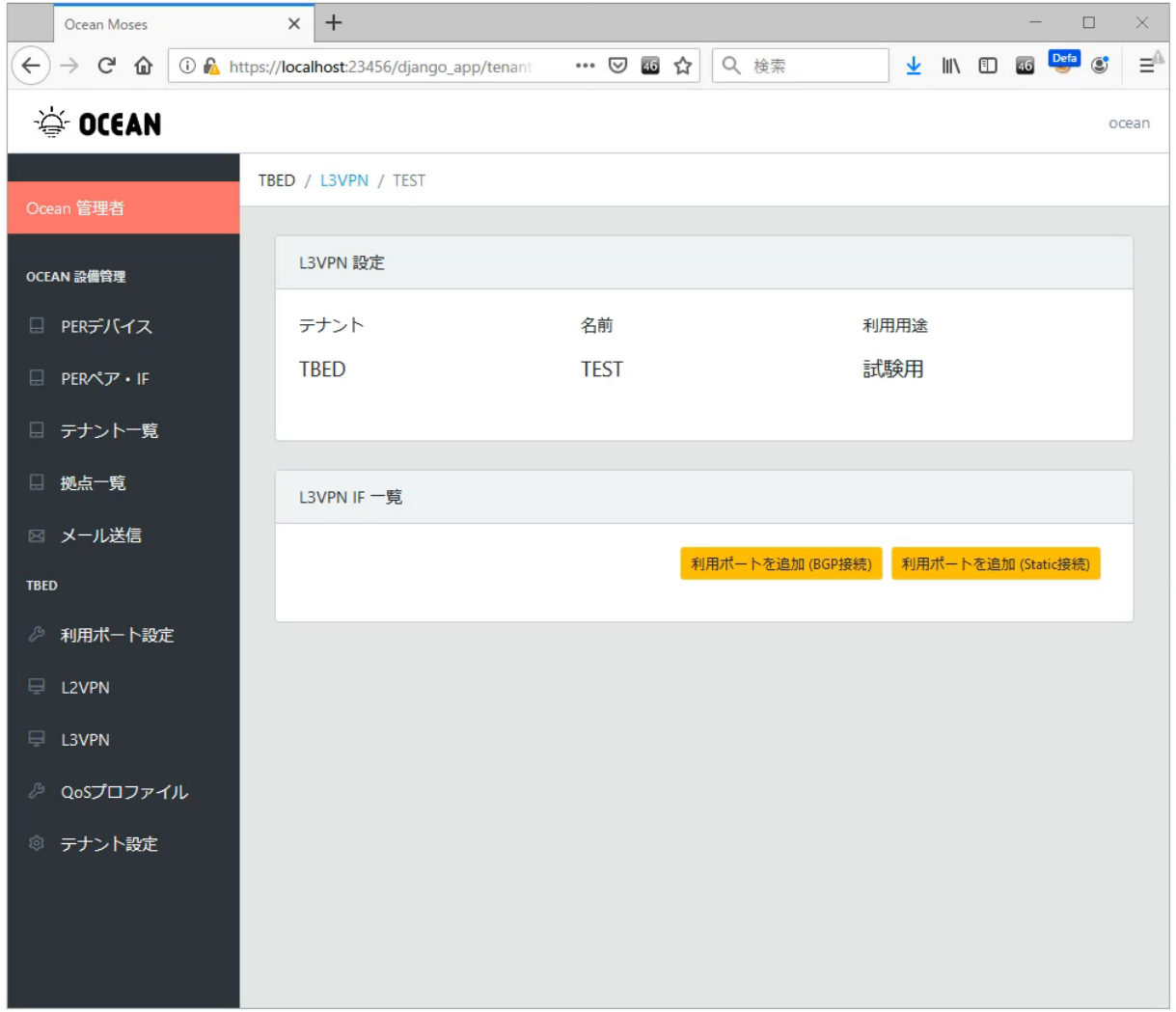
発表概要

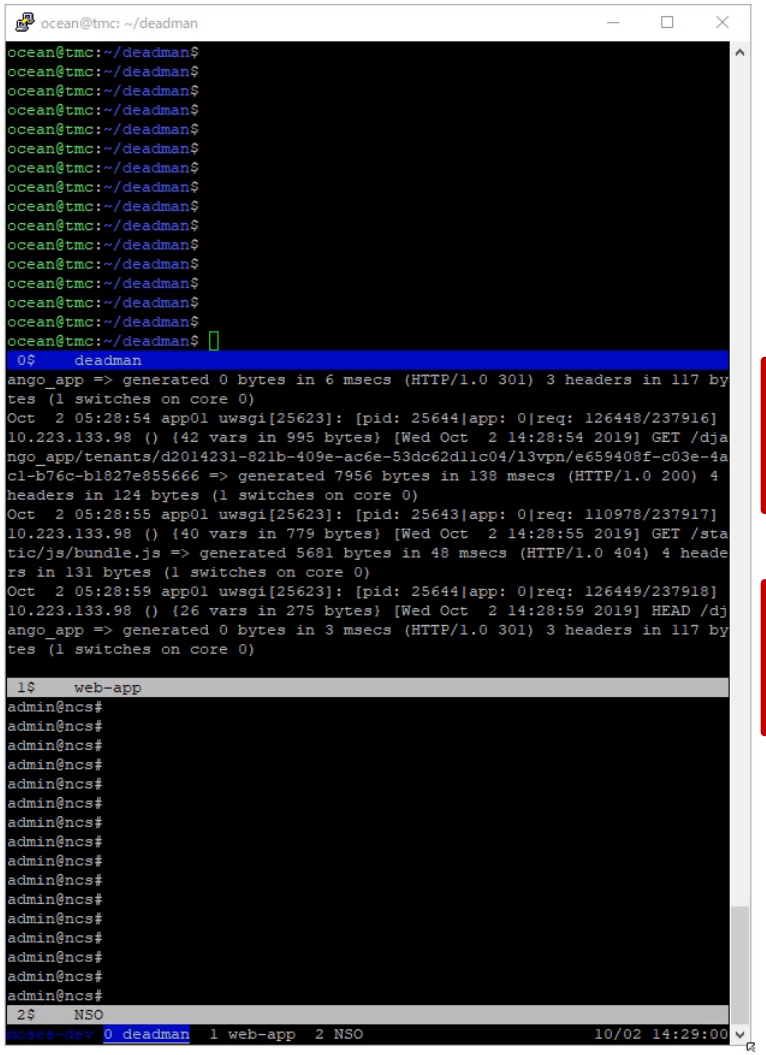
- スライス提供型のネットワークサービスを提供
- オンデマンド提供などの利点を活かしつつ実装を抽象化できる運用システムの開発が重要
  - 技術を理解しているNWエンジニアだけでなく
  - ユーザーのセルフポータル利用や他システムからの連携

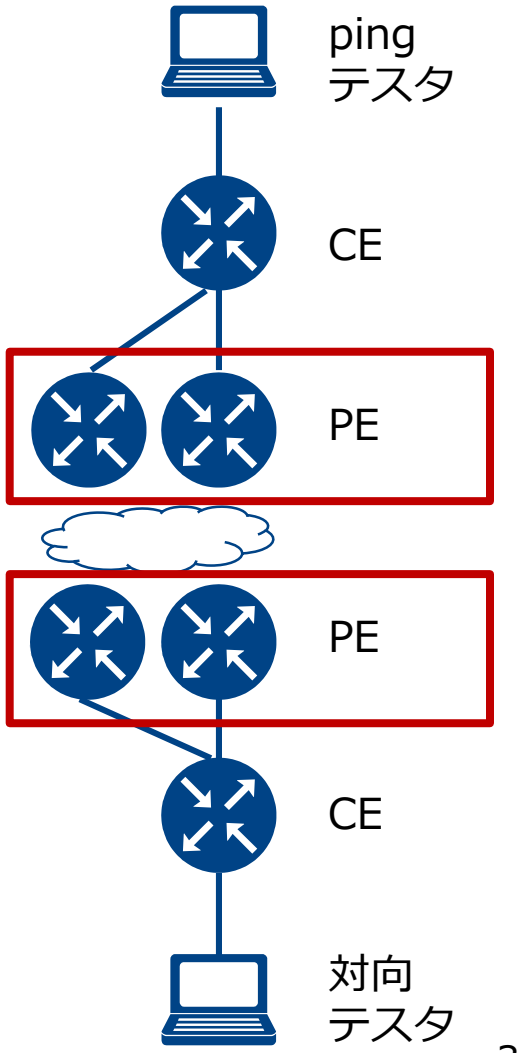
- 我々が開発したWeb UIを例にシステム開発の事例を紹介

話題にするシステムの「表側」

■ ユーザがGUIでオンデマンドで通信を開通・閉鎖させるデモ



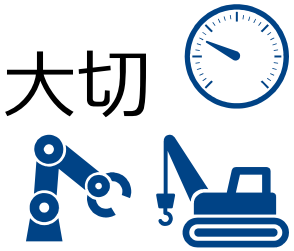




使いやすいネットワークサービス

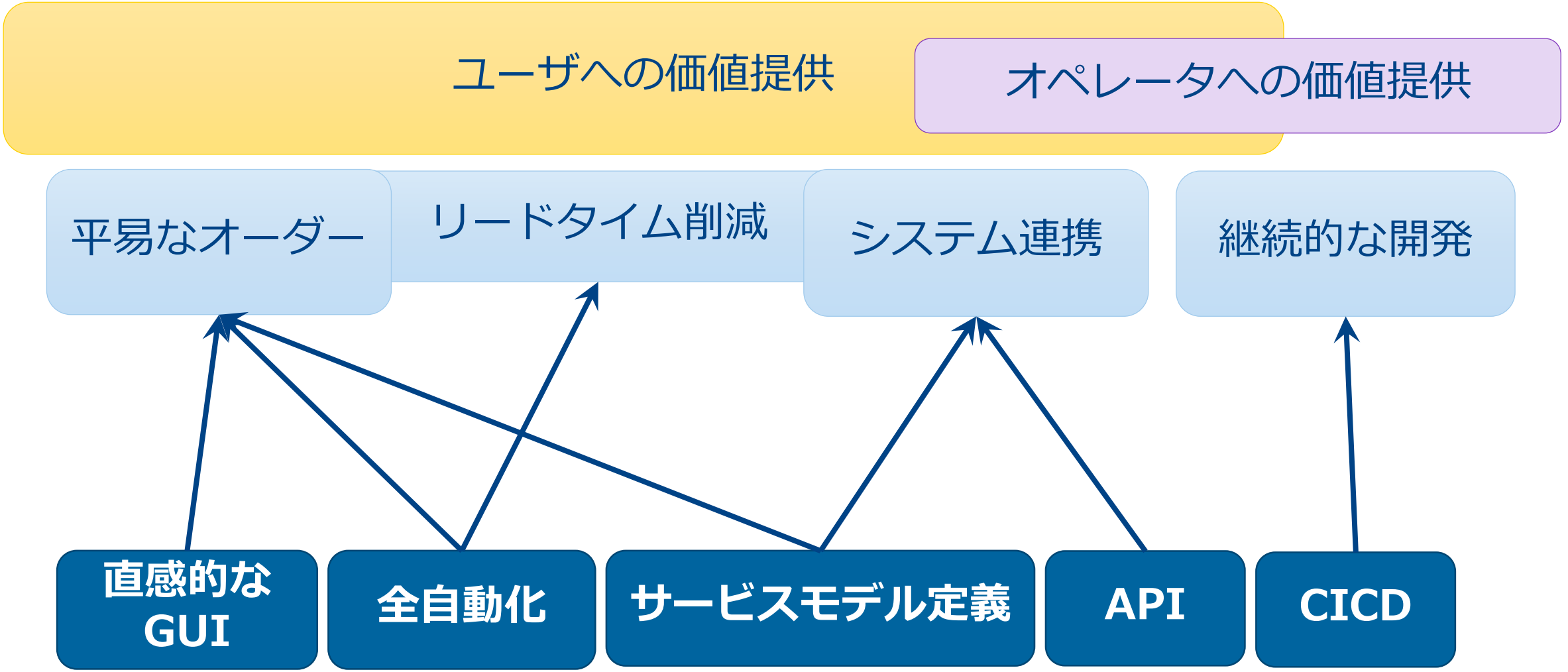
- こんなネットワークサービスは嫌だ
  - リードタイムが長い
  - 技術ベースのオーダーは難解
  - サービスを追加しようにも改修しづらい

- サービスデリバリを短くするのは確かに大切
  - 拡張性や開発を続けられるのも大事



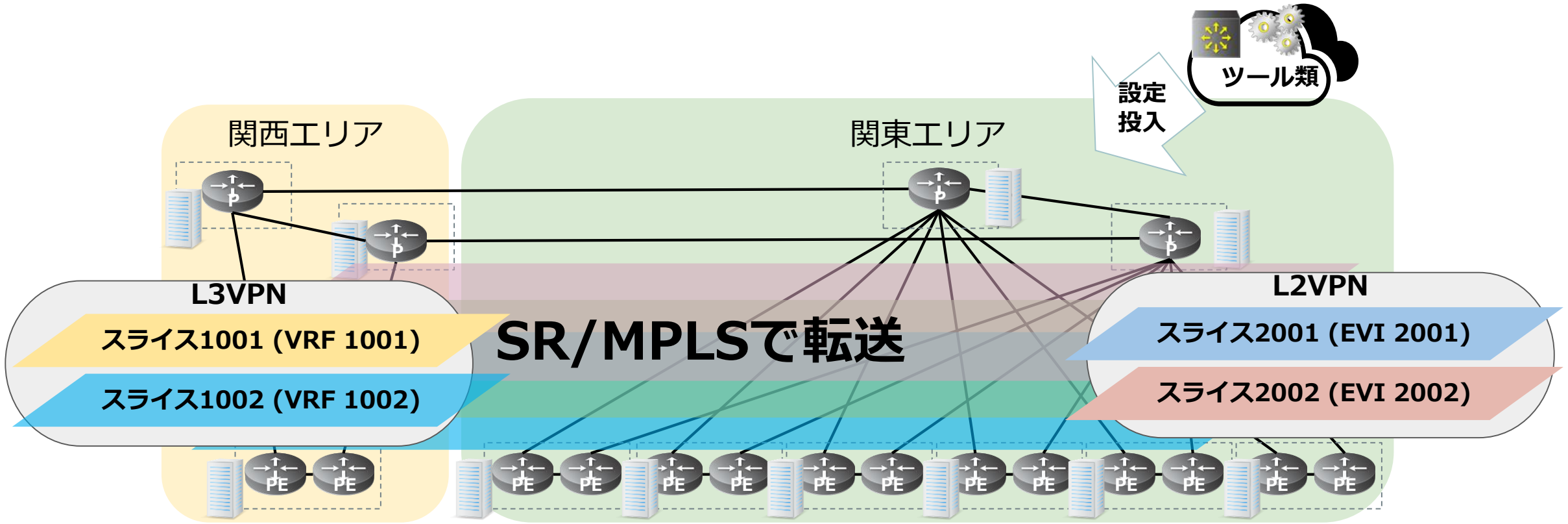
- ユーザーフレンドリーなサービスは、オペレータにもフレンドリー

実現へのアプローチ

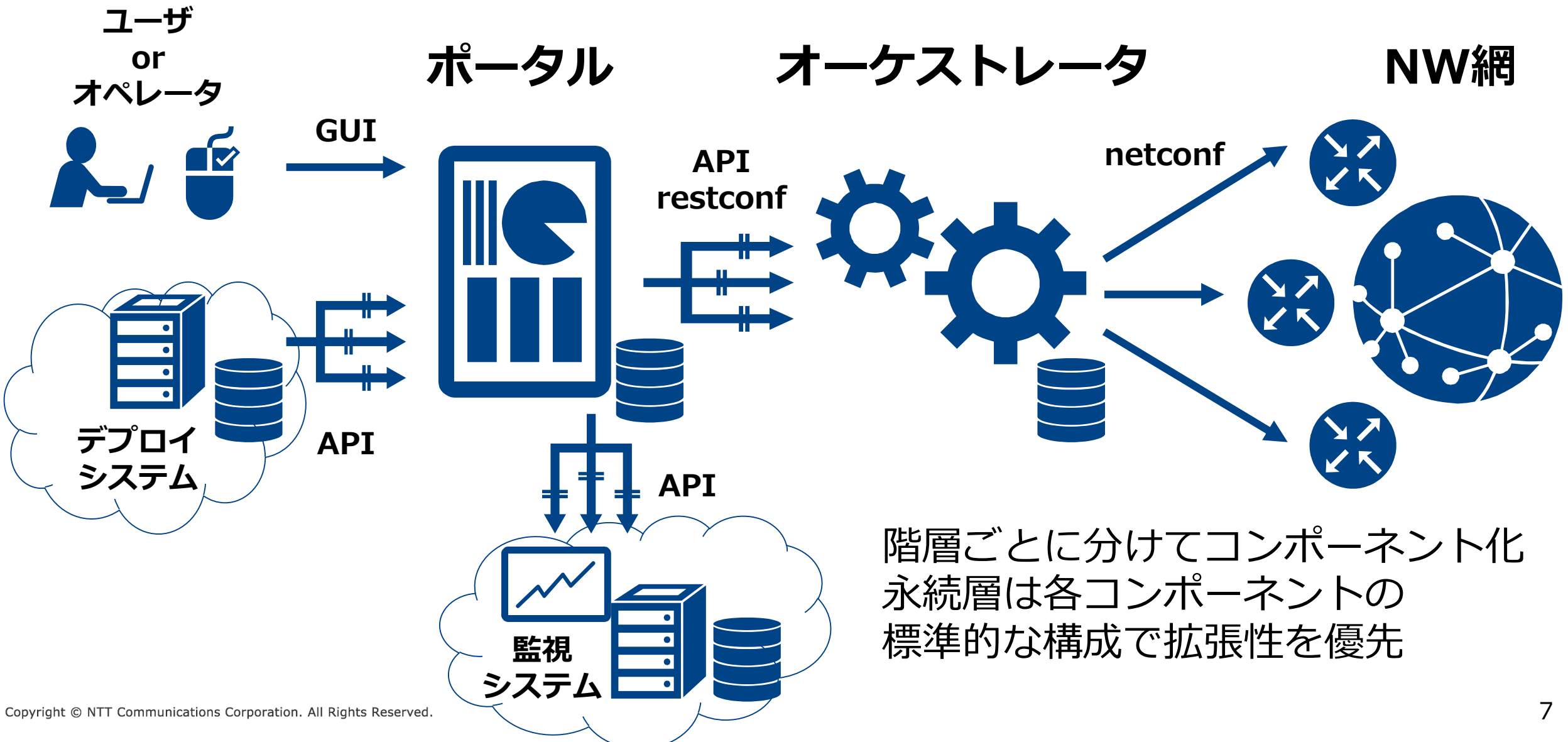


制御対象のNW

木村 安宏, SR-MPLSの導入事例と今後の展望について, MPLS JAPAN 2019, Oct. 2019.
一部加筆



システム全体像



階層ごとに分けてコンポーネント化
永続層は各コンポーネントの
標準的な構成で拡張性を優先

アプローチ

1. サービスモデル定義
    - サービス仕様をデータ構造に落とし込む
2. 全自動化
3. GUI作成
4. API連携
5. システムのCI/CD

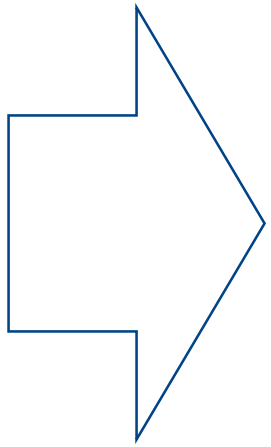
サービスのモデリング

- モデル化できないサービスは機械化できない
  - サービス仕様である程度正規化

- サービス仕様 → 抽象化
  - 正規化はこだわらない
    - ✓ 過度な正規化は理解の妨げ
    - ✓ 一方、正規化しないフィールドは1トランザクションで変更が必要
  - モデルからconfigを生成
    - ✓ 正しく抽象化されたモデルでは設定箇所以外に影響を及ぼさない
    - ✓ configからモデルを作ると失敗しがち
- yangで記述

サービス仕様からサービスモデルへ

- L3VPNサービス仕様
  - static or BGP
  - subIFの有無
  - 経路上限
  - QoS (tos)
- L2VPNサービス仕様
  - LACP接続必須
  - QoS (cos)

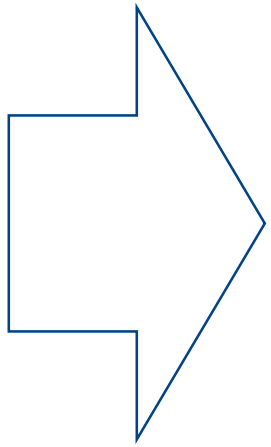


- IF
  - テナントID
  - 速度
  - L2用orL3用
    - ✓ L2の場合 LAG-IFのID

- L3VPN
  - スライスID
  - 接続拠点とポート
    - ✓ 接続=BGP
      - ✓ AS番号
      - ✓ P2Pの/30が2つ
    - ✓ 接続=static
      - ✓ VRRPの/29
      - ✓ VIP
      - ✓ 静的経路
    - ✓ 経路数上限
    - ✓ QoS

サービスモデルからconfigへ

- L3VPN
  - スライスID
  - 接続拠点とポート
    - ✓ 接続=BGP
      - ✓ AS番号
      - ✓ P2Pの/30が2つ
    - ✓ 接続=static
      - ✓ VRRPの/29
      - ✓ VIP
      - ✓ 静的経路
    - ✓ 経路数上限
    - ✓ QoS
- ・・・



```
vrf 1000
 description Test-vrf
 address-family ipv4 unicast
  import route-target
   64639:1000
  !
  export route-target
   64639:1000
```

```
router bgp 64639
 vrf 1000
  rd 10.0.0.1:1000
  address-family ipv4 unicast
   redistribute connected
   redistribute static
  !
  neighbor 172.17.1.253
   remote-as 65000
   timers 30 90
   address-family ipv4 unicast
    route-policy OCEAN-IN-v4 in
    maximum-prefix 51 80
    route-policy OCEAN-OUT-v4 out
    as-override
    next-hop-self
    soft-reconfiguration inbound always
    site-of-origin 10.0.0.1:1000
```

```
interface TenGigE0/0/0/12
 description Cat9300#2_L3VPN
 vrf 1000
 ipv4 mtu 1500
 ipv4 address 172.17.1.254 255.255.255.252
```

レコードの論理削除・物理削除問題

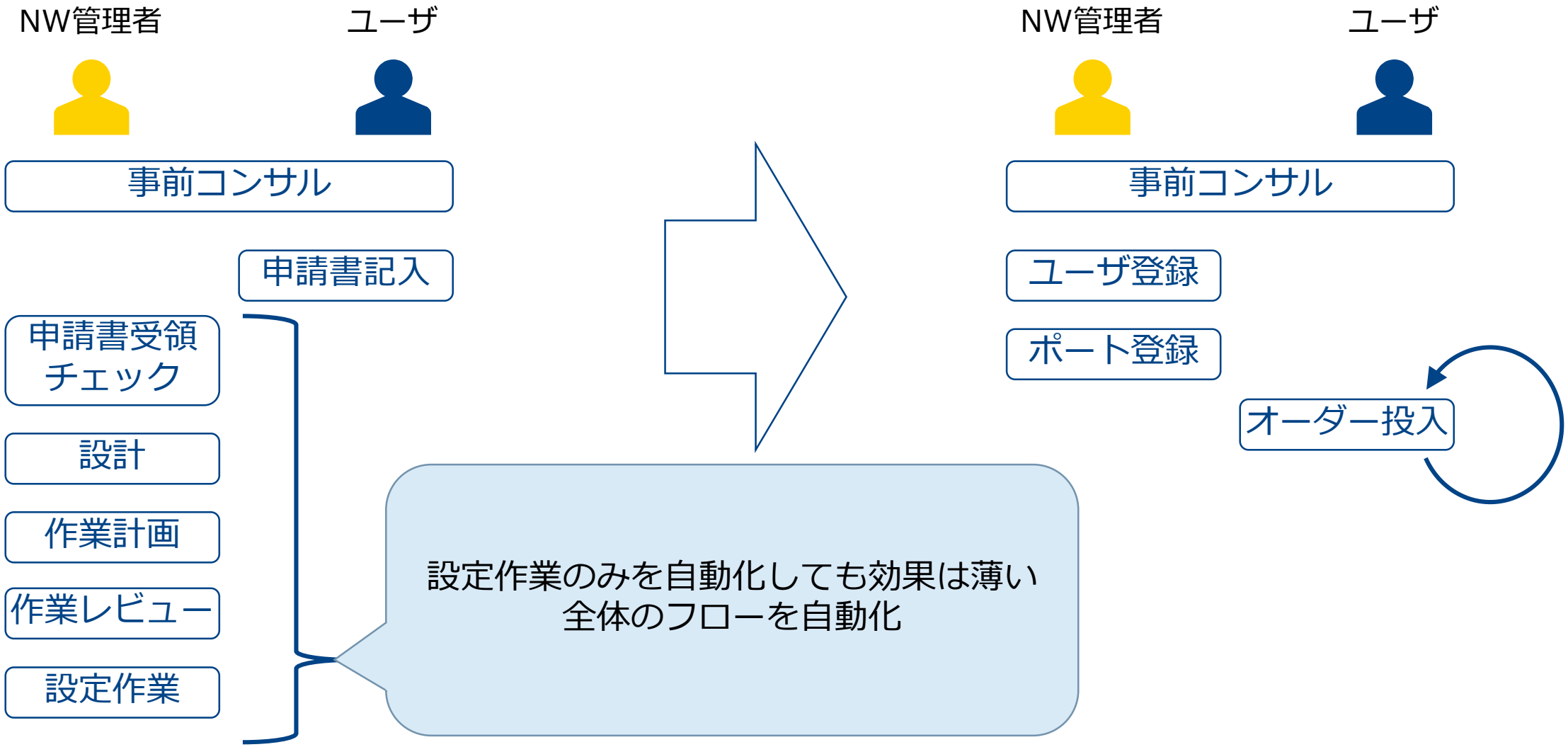
■ 各モデルの実装時にレコードのCRUDをどう扱うか

論理削除　＝　削除フラグ	物理削除　＝　レコード削除
メタな情報を含む、モデル 削除履歴もモデル内に保存可 キーが値重複できることが大事 • オーダーの情報 • ユーザーの情報	物理層 現実世界と一対一対応なもの • 拠点情報 • 機器情報 • 物理インターフェースの情報

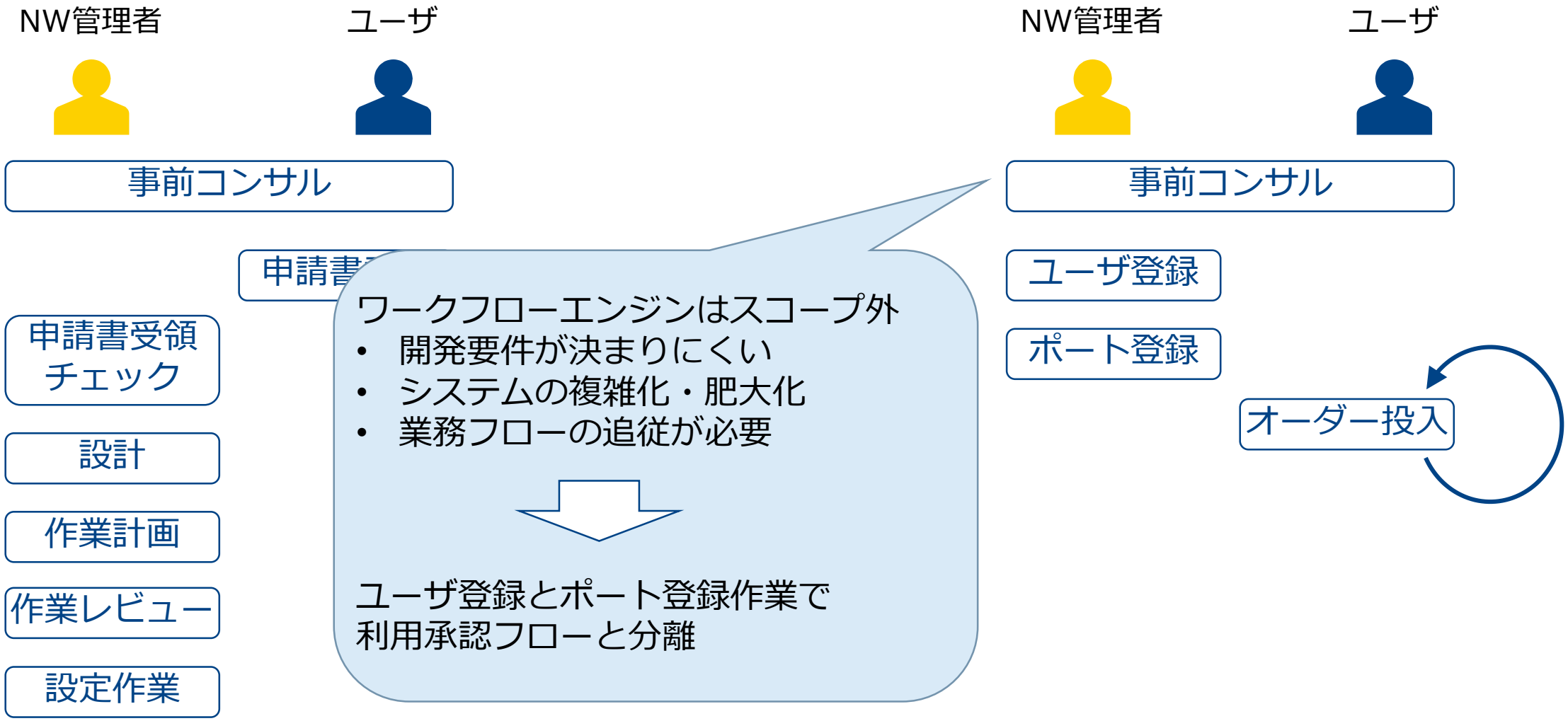
アプローチ

1. サービスモデル定義
2. 全自動化
    - 工程の一部ではなく、ツールで完結させる
3. GUI作成
4. API連携
5. システムのCI/CD

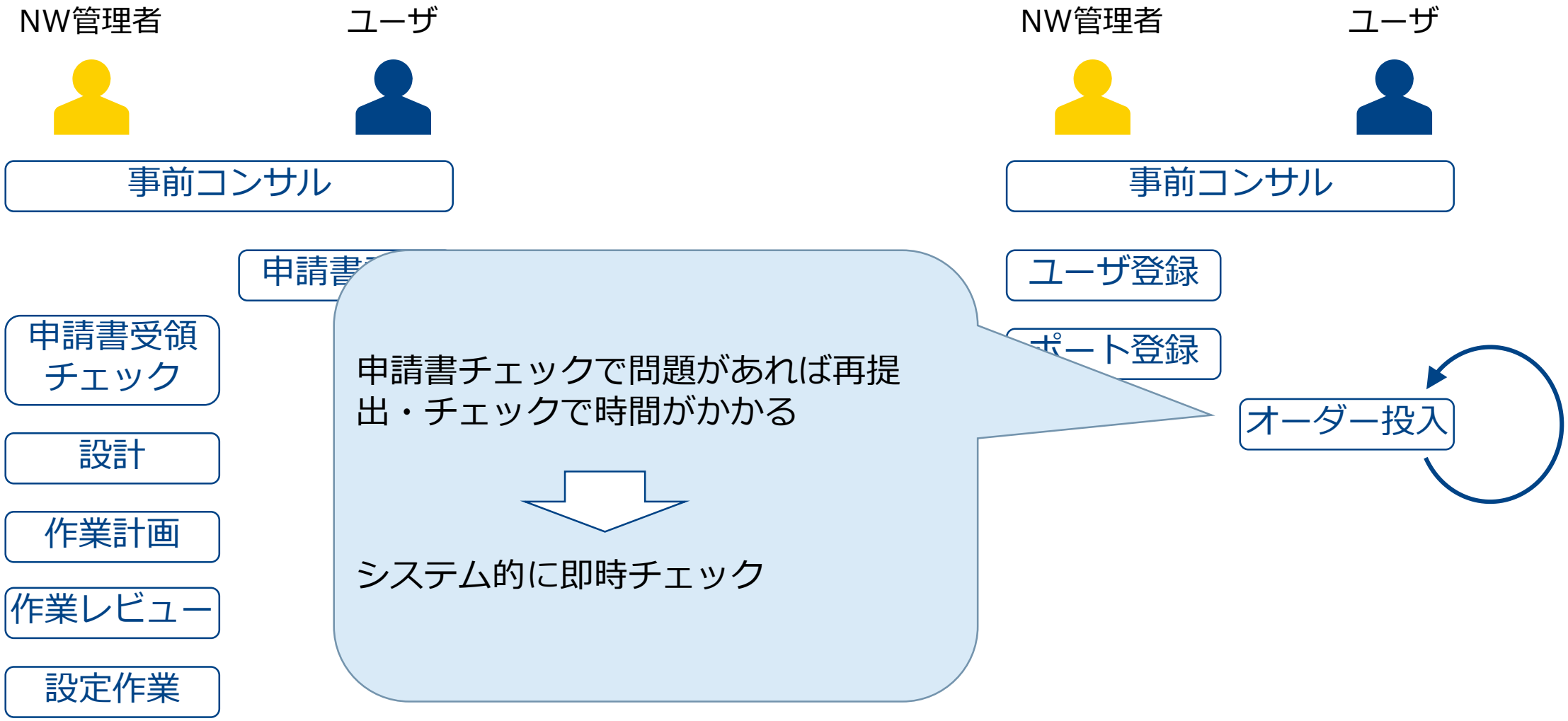
全自動化：新旧のフロー比較



全自動化：新旧のフロー比較



全自動化：新旧のフロー比較



自動化することとしないこと

自動化　する	自動化　しない
サービスオーダー全て • 頻度とリードタイム 故障復旧 • 定型作業化が重要 • 既存configを再投入なので影響範囲限定	拠点増設 • 台数と頻度が(今のところ)少 • 影響範囲が大きい オーダーのキューイング • 開発規模を大きくする • オーダー頻度に依存

アプローチ

1. サービスモデル定義
2. 全自動化
3. GUI作成
4. API連携
5. システムのCI/CD

UI/API開発

■ フレームワークで標準的に（魔改造せずに）実装

- Python Django
  - ✓ フロントエンド開発よりフレームワーク維持が楽


```
$ curl  -H 'Authorization: JWT xxx' http://xxx/so-portal/api/so/l2vpn | jq .
[
  {
    "uuid": "a6532db9-34a6-4434-93b3-6c25a848f2dc",
    "tenant": 3,
    "vlan": 1000,
    "be_membership": [
      {
        "uuid": "4679e43f-f166-414f-beb9-6699f56bd6ce",
        "bundle_ether": {
          "uuid": "96eebba4-233c-4a57-b2fc-388db5827267",
          "per_pair": "OCNWAKBEGRT01-OCNWAKBEGRT02",
          "bundle_id": 1008
        },
        "qos": null
      }
    ],
```

アプローチ

1. サービスモデル定義
2. 全自動化
3. GUI作成
4. API連携
5. システムのCI/CD

サービス拡張に伴う改修

- NW技術の発展で次々新しい機能が利用可能に
- 運用システムもそれに追従する必要あり

- 例：QoSサービス追加
  1. サービスモデルに追加、config作成
  2. UI作成
  3. テスト
  4. デプロイ

サービス拡張に伴う改修例：QoSサービス追加

1. サービスモデルに追加、config作成
   - L3VPN
     - 接続拠点とポート
       - ✓ QoS
   - QoSプロファイル
     - 各cos/tosの上限値

2. UI作成
3. テスト
4. デプロイ

```
policy-map test-L3VPN-ToS-POLICY-2
 class MATCH-L3-ToS-67
  set precedence 0
 !
 class MATCH-L3-ToS-5
  police rate 100 mbps
  !
  priority level 1
 !
 class MATCH-L3-ToS-4
  bandwidth 80 mbps
 !
 class MATCH-L3-ToS-1
  bandwidth 50 mbps
 !
 class class-default
  bandwidth 10 mbps
 !
 end-policy-map
```

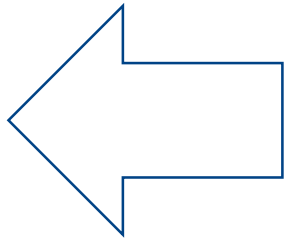
サービス拡張に伴う改修例：QoSサービス追加

1. サービスモデルに追加、config作成
2. UI作成

3. テスト
4. デプロイ

サービス拡張に伴う改修例

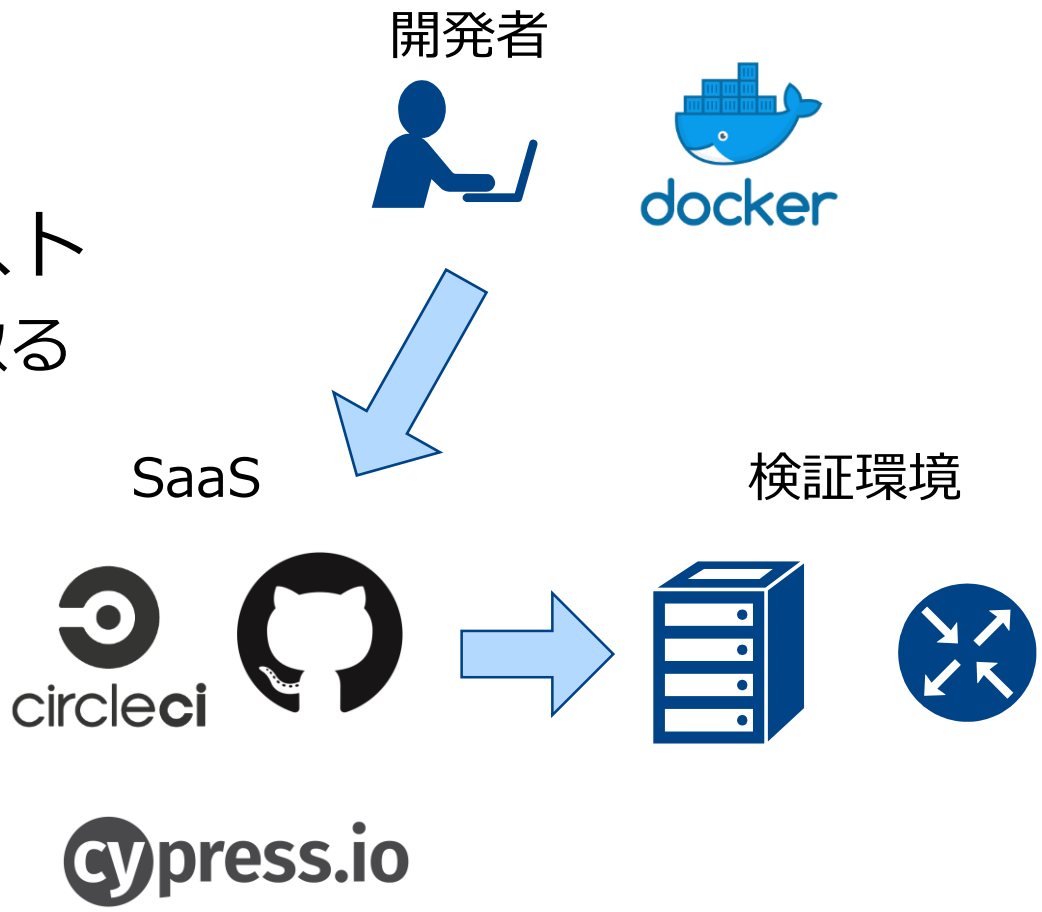
1. サービスモデルに追加、config作成
2. UI作成
3. テスト
4. デプロイ



?

運用システムのCI/CD

- アプリケーション自体のテスト
  - 例: flake8による型チェック
  - 例: cypressによる自動UIテスト
    - ✓ スクリーンショットを自動で取る
    - ✓ 手順書も自動で更新可能
- NW機器との統合テスト
  - 例: エミュレータ利用
    - ✓ yangのnetconf部分
  - 例: VNF利用



運用システム自体のデリバリ

- システムのデプロイも自動化
  - ansible
  - 作業者に依存しない、実施漏れ・設定齟齬の防止
    - ✓ 本番環境は開発したコンポーネント以外にいろいろ必要
    - ✓ proxy、ntp、syslog、dns・・・
- 自動化できない手順
  - 自動復旧しにくいインパクト大な手順
  - 例: モデル変更に伴う ALTER TABLE

サービス拡張に伴う改修（再掲）

- NW技術の発展で次々新しい機能が利用可能に
- 運用システムもそれに追従する必要あり

- 例：QoSサービス追加
  1. サービスモデルに追加、config作成
  2. UI作成
  3. テスト
  4. デプロイ

各手順をやりやすい
周辺ツールも同時に
開発することが鍵

まとめ

- ユーザーがセルフマネージできる運用システムを開発
- サービスモデルの定義で使用技術を隠蔽
- 全自動化でデリバリを短縮
- 技術アップデートに追従できるよう継続的な開発