クラウドネイティブな
新しいネットワークコントローラをつくる

2022年10月28日
NTTコミュニケーションズ 奥井 寛樹

自己紹介

NTTコミュニケーションズ
Software Engineer

奥井 寛樹
@HirokiOkui
hrk091

略歴

- 伝送ネットワークの設定自動化システム開発
- DevOpsプラットフォーム開発
- IoTデータ収集基盤のモダナイゼーション

前置き

ネットワークコントローラとは

- サービス・NW仕様に基づいて、APIリクエストからコンフィグを導出し装置に設定するもの

以下はスコープ外

- SDNによるコントロールプレーン
- 構成管理機能（連携先として考慮）
- システム間連携をするワークフロー機能
- 監視機能（現時点は。いずれは考えたい）
- ZTP（装置へのReachabilityが確保されている前提）

本発表の構成

1. ネットワークコントローラ開発の現状と課題
2. クラウドネイティブ技術の紹介
3. クラウドネイティブ技術を用いてつくってみた

ネットワークコントローラ開発の現状と課題

ネットワークシステム開発も高度化しつつある

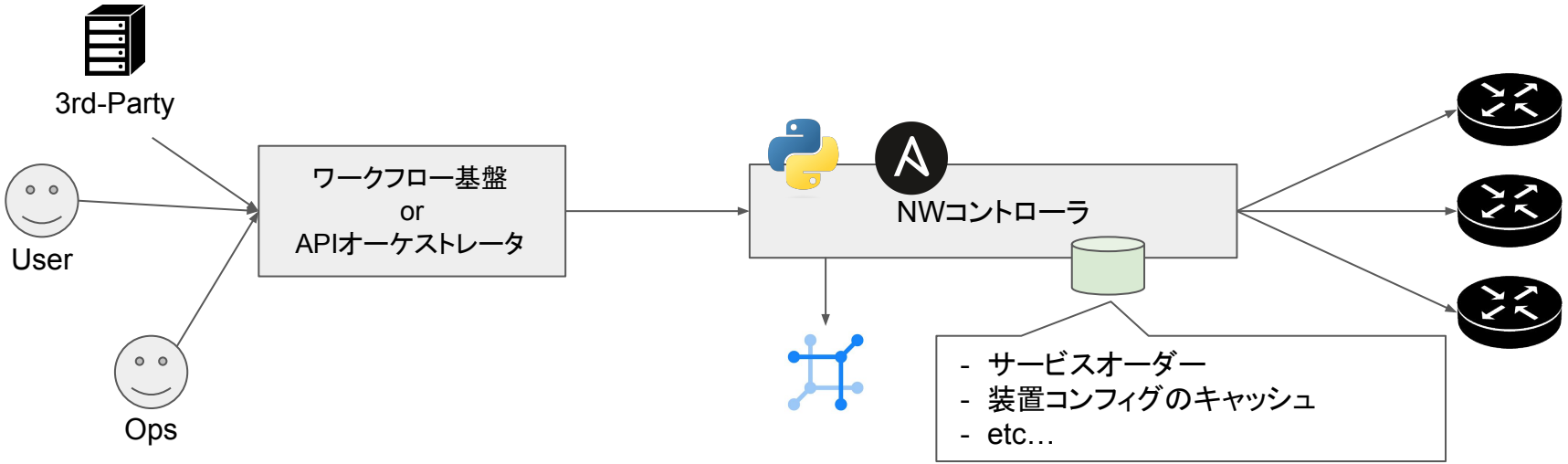
1. API連携での自動化

2. GitOps

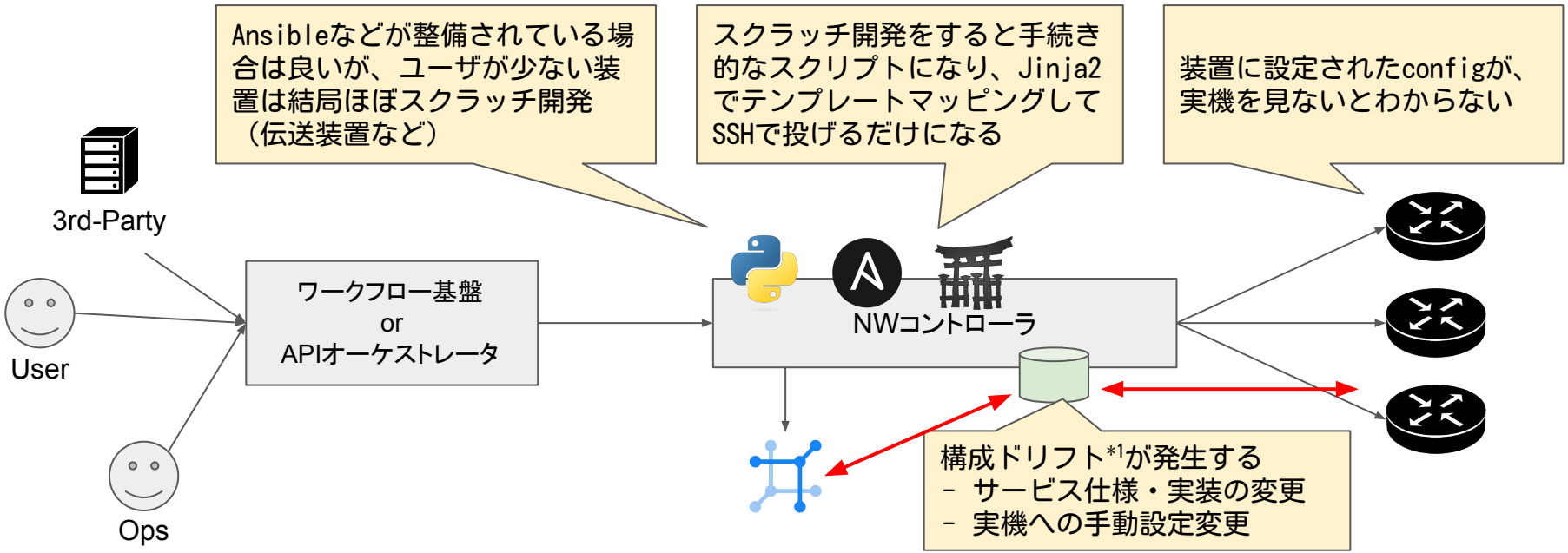
API連携での自動化

オーケストレータと組み合わせてon-demandサービス化

- 装置ベンダEMS or 装置自体がREST APIを開けているケースが増え、自動化しやすくなった
- Ansibleを用いた自動化事例が増え、Pythonライブラリなども充実



自動化の課題

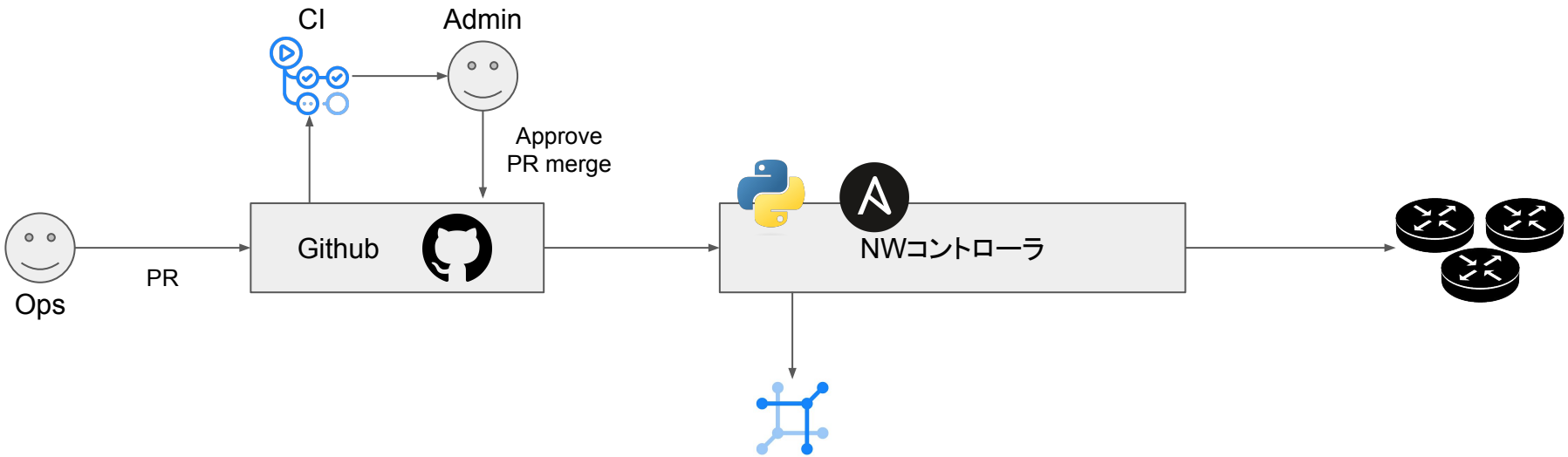


*1: NWコントローラで導出したコンフィグと、実機に入っているコンフィグに差異があること

GitOpsの例

Gitでコンフィグ管理し、CIでテストし、PR mergeトリガでデリバリ

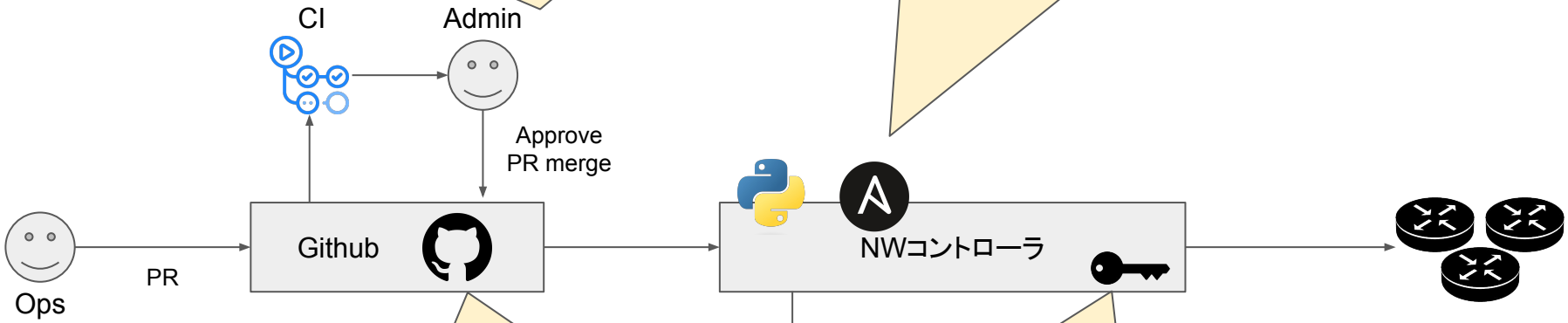
- Netboxから構成情報を取得しつつ、Ansibleでデリバリ



GitOpsの課題

モデル・スキーマ情報がなく、ゴールデンファイルとの差分確認テストしか出来ない。
検証は全てReviewerの目視確認に委ねられる。

PR reviewでCIの結果を見て判断しているが、CIで確認したコンフィグが必ずしも装置に入るとは限らない
- 外部(Netboxなど) に依存している
- CIで検証した版と実環境で動いている版が異なる



外部(Netboxなど)に依存しており、Gitで過去の版に戻しても同じconfigを必ずしも再現できない

Push型であり、装置のシークレットがここに集約される。セキュリティリスクが高い

クラウドネイティブ技術を活用した
現代的なソフトウェアデリバリ

クラウドネイティブなCI/CDの技術・プラクティス
（ネットワークコントローラ開発に役立つものを抜粋）

Infrastructure as Code

Reconciliation Loop

GitOps

Secret管理

構成テスト

IaCのプログラマビリティ

CUE言語の台頭

まだ普及していませんが、
今回の開発の重要なコンポーネントなので紹介

Infrastructure as Code（aka IaC）

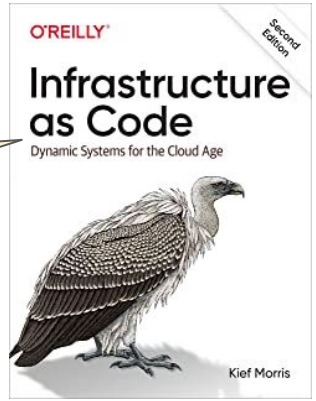
インフラをコードで記述し、ソフトウェア開発のプラクティスを活かす

- 多くのツールが生み出され、大多数の企業が採用している
- Terraform、Cloud Formation、Kubernetes YAML、etc…

ただツールを使ったり、宣言的コードで表現すればよいわけではない

- 自動テスト、CI/CD、シフトレフト、モニタリング、高速なリリースサイクル …

IaCを効率的に使う
システム・プロセス・慣習が大事



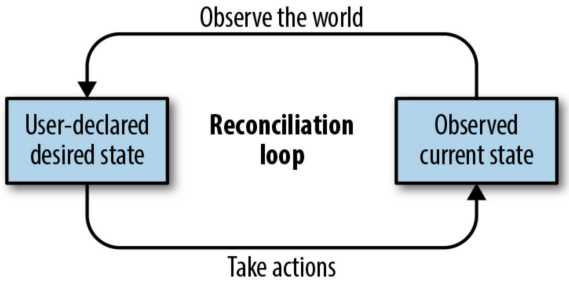
宣言状態に収束させるReconciliation Loop

IaCで宣言された状態に収束させるには、実行エンジンが必要

- 非常駐型：TerraformなどのIaCツール全般
- 常駐型：KubernetesのReconciliation Loop ← 昨今のトレンド

k8s Reconciliation Loop

- 観測されたインフラ状態と宣言状態が一致するまで、
  デプロイ処理を繰り返すk8sの仕組み
- k8sのリソースは全てこの仕組みで制御される



k8sカスタムオペレータ

- k8s上で、ユーザが独自の定義を持ち込み、
  任意のリソースをIaCとして管理するk8s拡張方式
- CNCFの多くのOSSは、k8sカスタムオペレータとして実装

今回の開発でも使用しています

GitOps

Gitで構成管理をして、PR mergeをトリガとしてデプロイする手法

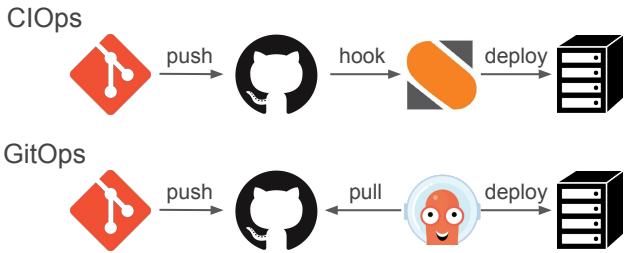
- 2017年にWeaveworks CEOによって提唱された概念 *1
- Single Source of Truth（SSoT）が重要で、Pull型が選択される
- ArgoCD、FluxCDなど

インフラにデリバリするコンフィグが
一箇所に集約されている状態

今回の開発で
使用しています

GitOpsのメリット

- Single Source of Truth(SSoT)であり外部依存がなく、
  Git Version指定だけで任意の状態を再現可
- Pull型のため、構成・運用の両面で Secretを秘匿しやすい



*1: https://www.weave.works/blog/gitops-operations-by-pull-request

Secret管理、構成テスト

Secret管理：IaCでの大きな懸念事項

- Secretをmanifestに書かないことで、漏洩リスクなくGitOpsしたい
- パブリッククラウドのシークレットマネージャーを使うのが今の主流
- External Secrets Operator、Secrets Store CSI Driver など

構成テスト：IaCマニフェストのテスト

- リリースする前に、CIでマニフェストの正しさを検証する
- OPA conftest、Kyverno、Kubewardenなど



ネットワークコンフィグのプロビジョニングでも
取り入れたいプラクティスです

IaCのプログラマビリティ

宣言的マニフェストを効率的に記述するために高度なアプローチが生まれている

- Terraform系：Pulumi、CDK（JSなど汎用言語で書ける）
- k8s系：Helm、kustomize（Helm chartやkustomizationによるモジュール化）

モジュール化するだけでなく、透明性と拡張性が重要

- Overlay（マニフェストの任意の拡張・変更）機能を各ツールサポート
- 内部を隠蔽せず、透明性を維持する

抽象化による内部の隠蔽・関心の分離が重要な
通常のソフトウェア開発とはアプローチが異なります

Overlayの例

base

```
kind: Deployment
spec:
  template:
    spec:
      containers:
        - name: test
          image: say-hello
```

+

patch

```
...
image: say-hello:v1.1.0
env:
  - name: TARGET
    vaule: world
```

=

base + patch

```
...
spec:
  template:
    spec:
      containers:
        - name: test
          image: say-hello:v1.1.0
          env:
            - name: TARGET
              value: world
```

CUE言語の台頭

今回の開発で使用しています

2018年に登場したデータ記述言語

- Marcel van Lohuizenが GCL[*1]での経験を元に作成
- 国内では、メルカリが使っていることで有名
- CUEを用いた新しいサービスや OSSなどが昨年から出てきている [*2]

特徴

- better JSONとしてライトに使える
- データの中にロジックを書ける（チューリング完全）
- モジュール化など、ソフトウェア開発のプラクティスが適用可
- 型と値と制約を同一のものとして扱う型システム
- 交換法則と結合法則が成り立ち、結合順序・階層によらない任意のOverlay（Composite）が可能
- 構成テストもできる

ネットワークコントローラ開発で重要な特性です

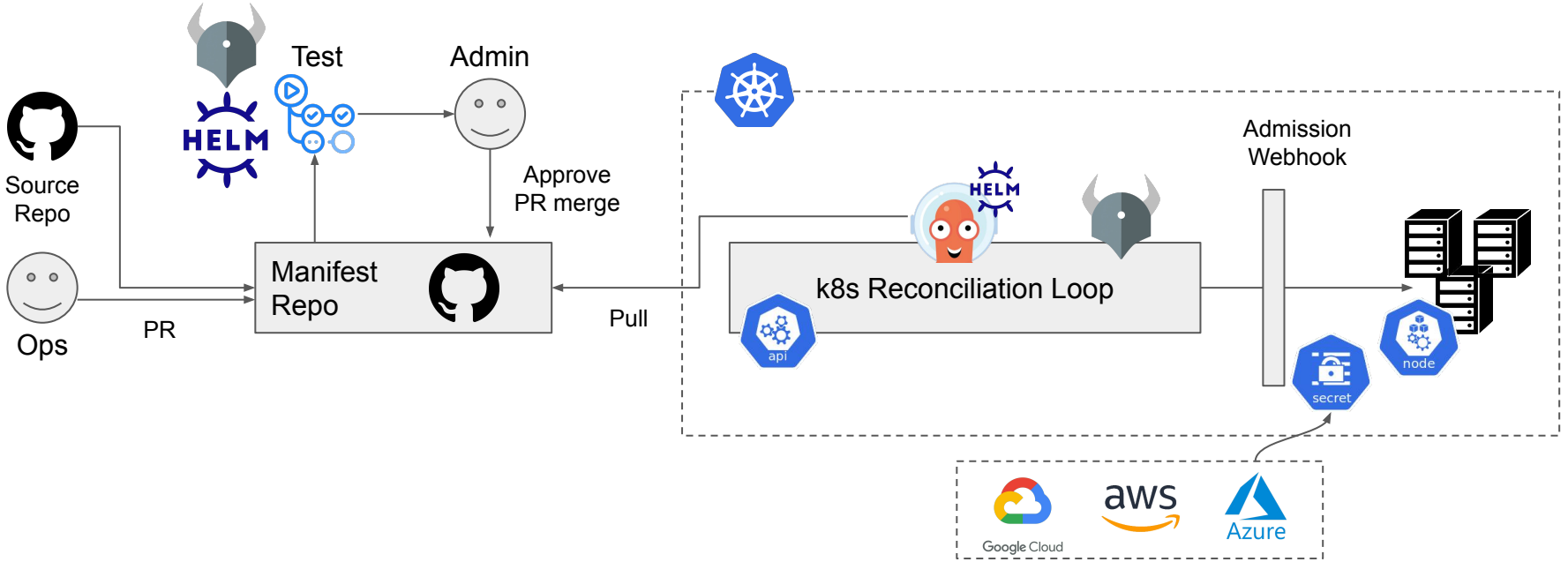
型と値と制約を同様に扱う

```
// Value
Alice: age: 20

// Type
People: age: int

// Constraint
Member: age: > 18

// Validate
Alice & People & Member
```

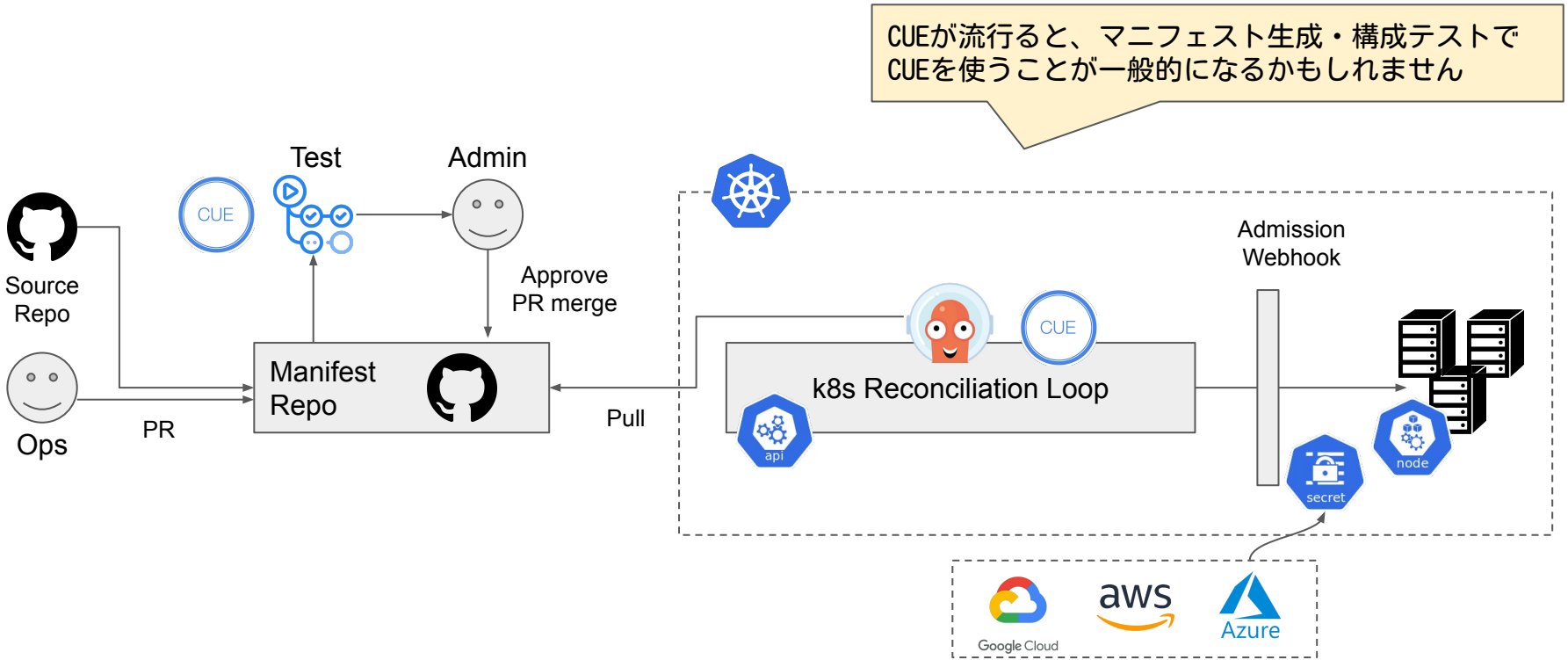
*1: Google/Borg の中で使われているデータ記述言語

*2: dagger.io, KubeVelaなど

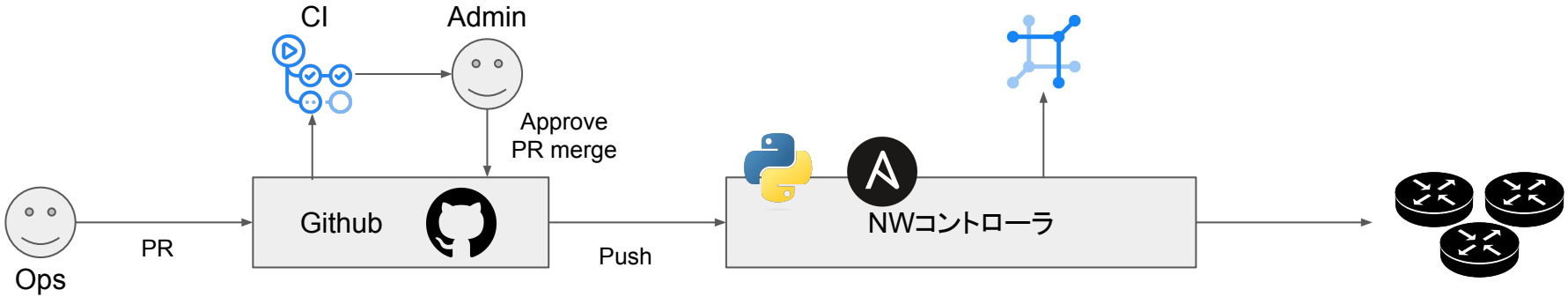
現代的なCDパイプライン

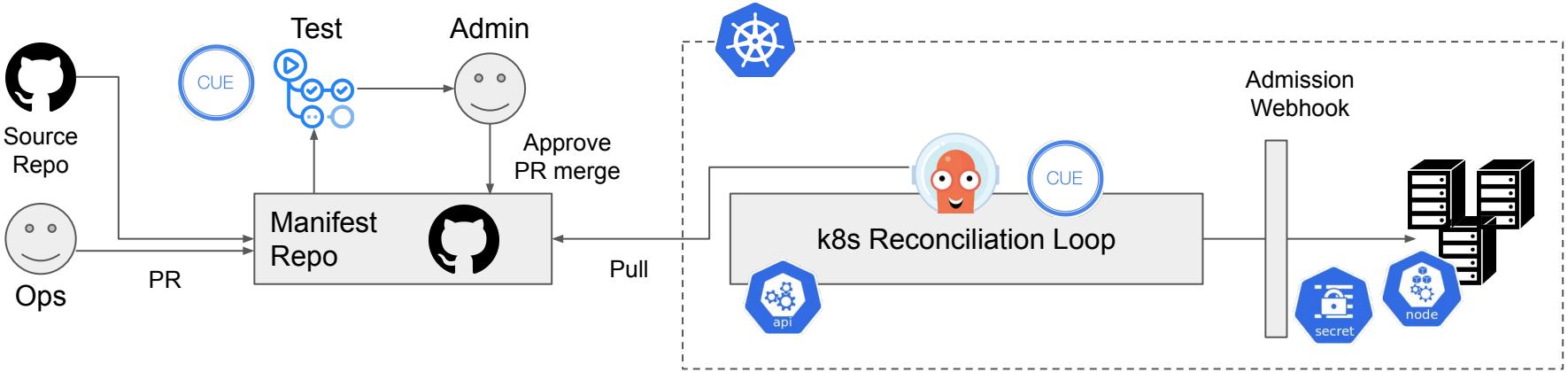


これから来るCDパイプライン（？）

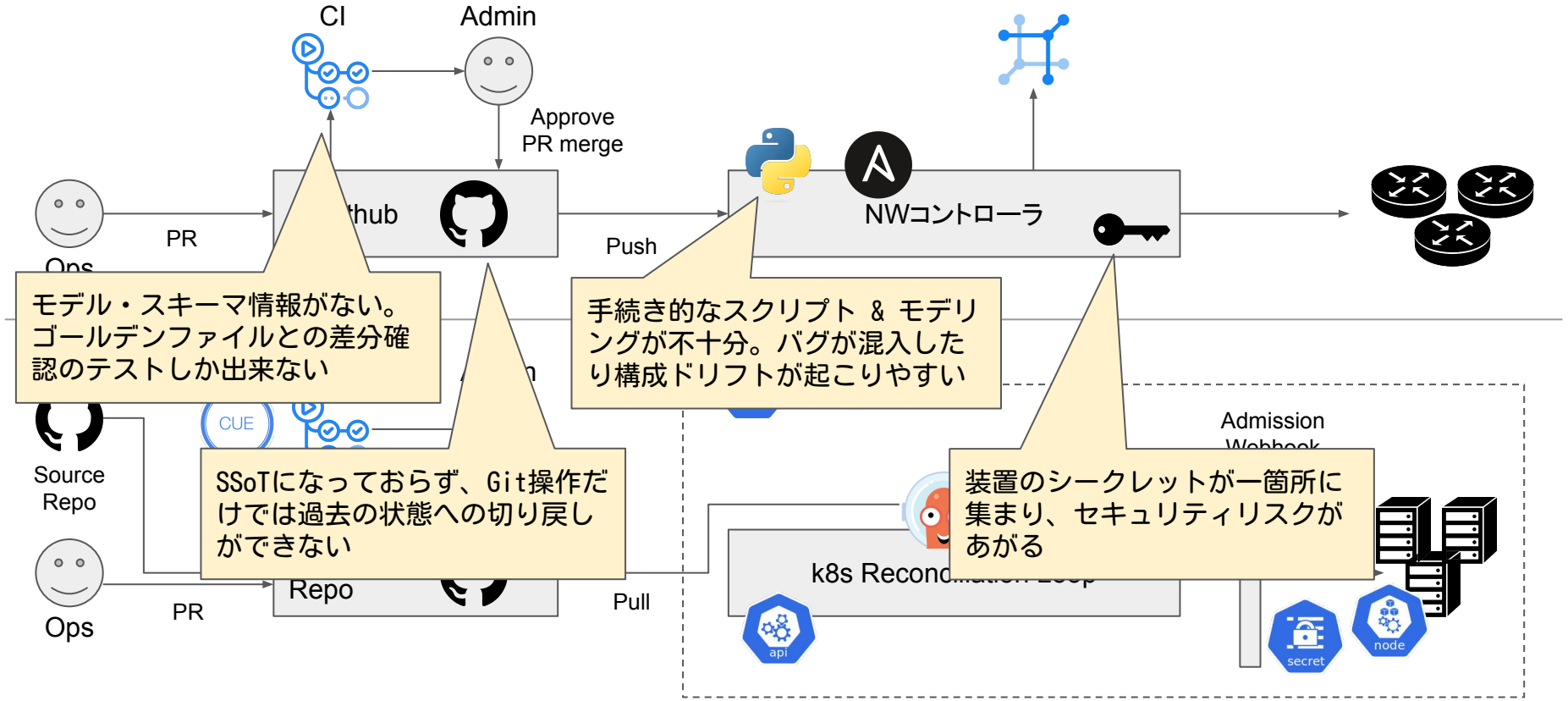


ネットワークプロビジョニングとの比較





ネットワークプロビジョニングは遅れている



クラウドネイティブ技術を用いて
つくってみた

要件

ネットワークコンフィグを、テキストではなくモデルで扱う能力

- ドメイン駆動・オブジェクト指向で開発できるプログラマビリティ
- 型・スキーマに基づいたバリデーション・構成テスト、 Overlayによる拡張機能
- そのためには、型付きドキュメントツリーの合成能力が必要

GitOpsの実践

- SSoTの徹底
- Pull型で、シークレットの漏洩リスクを減らす

そもそも、ネットワークコントローラとしてほしいもの

- マルチデバイスの分散トランザクション
- マルチベンダー・マルチバージョン対応
- 外部システムとのAPI連携のためのNorthbound Interfaceの自動生成

要件

ネットワークコンフィグを、テキストではなくモデルで扱う能力

- ドメイン駆動・オブジェクト指向で開発できるプログラマビリティ
- 型・スキーマに基づいたバリデーション・構成テスト、Overlayによる拡張機能
- そのためには、型付きドキュメントツリーの合成能力 が必要

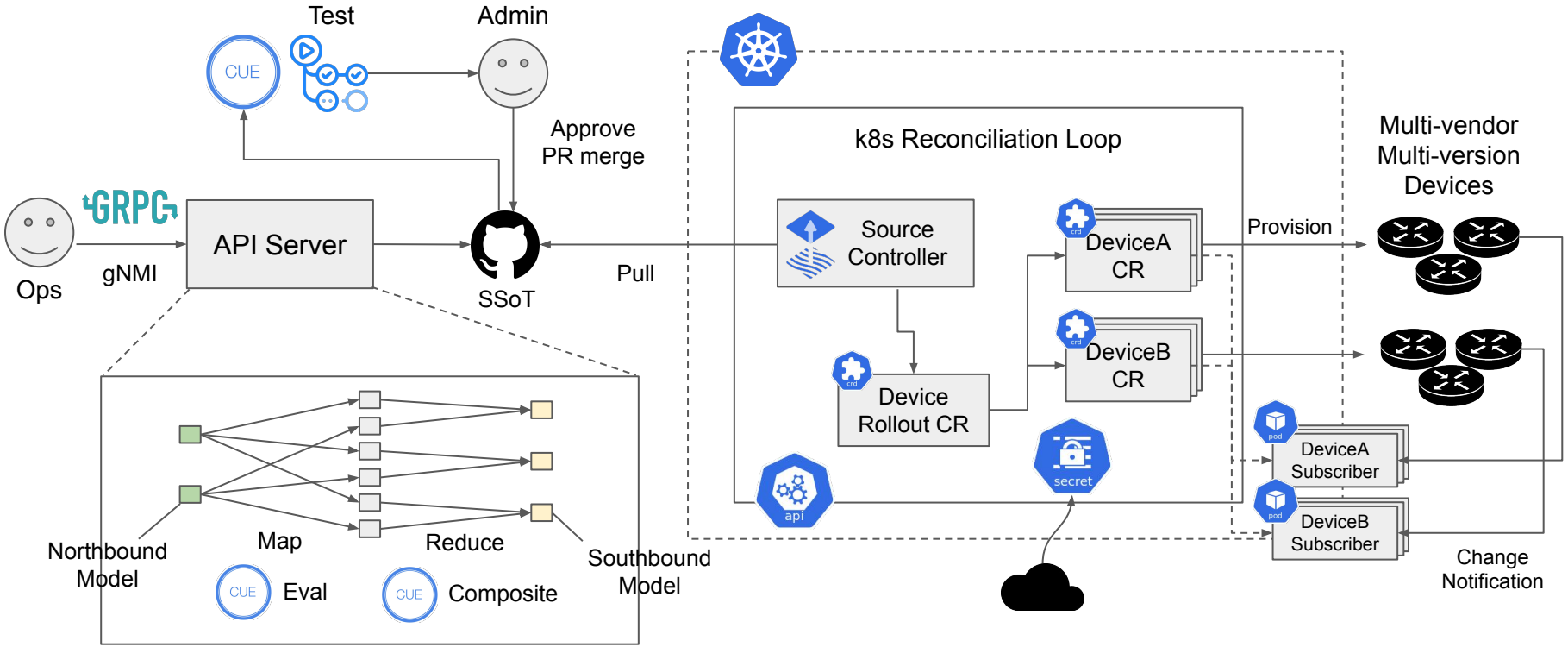
GitOpsの実践

- SSoTの徹底
- Pull型で、シークレットの漏洩リスクを減らす

そもそも、ネットワークコントローラとしてほしいもの

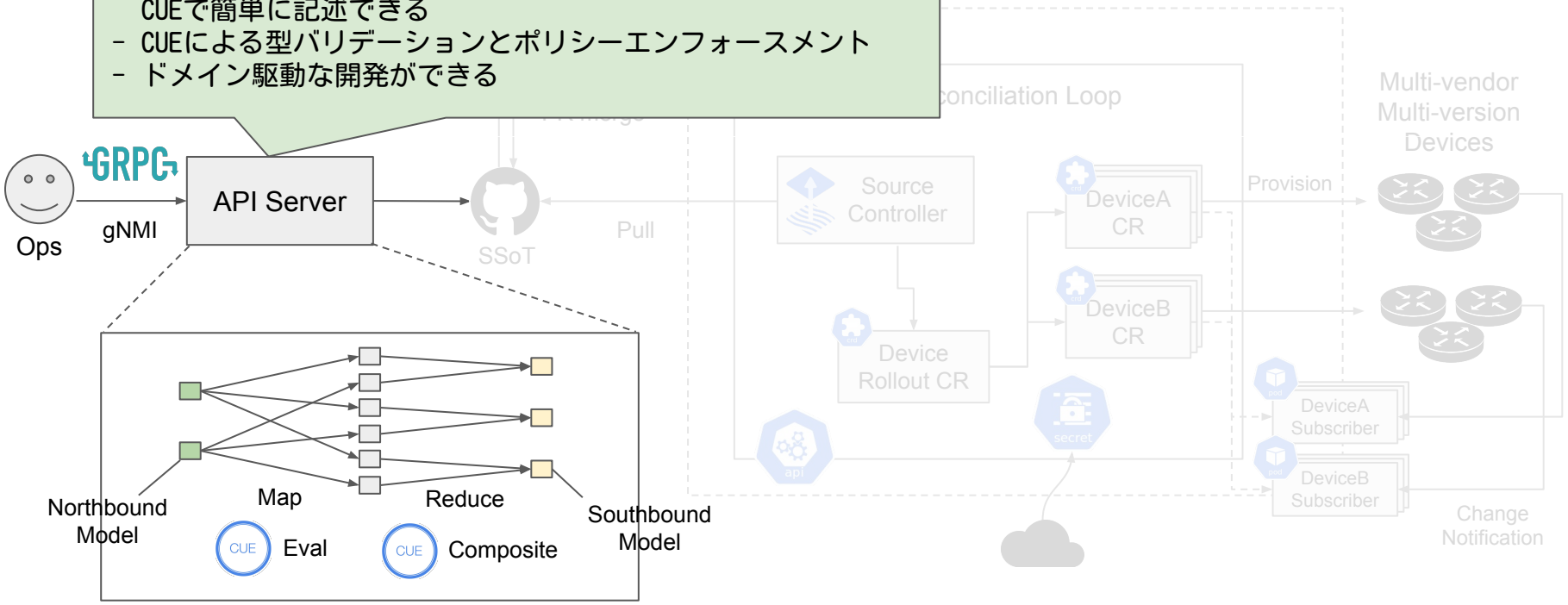
- マルチデバイスの分散トランザクション
- マルチベンダー・マルチバージョン対応
- 外部システムとのAPI連携のためのNorthbound Interfaceの自動生成

設計



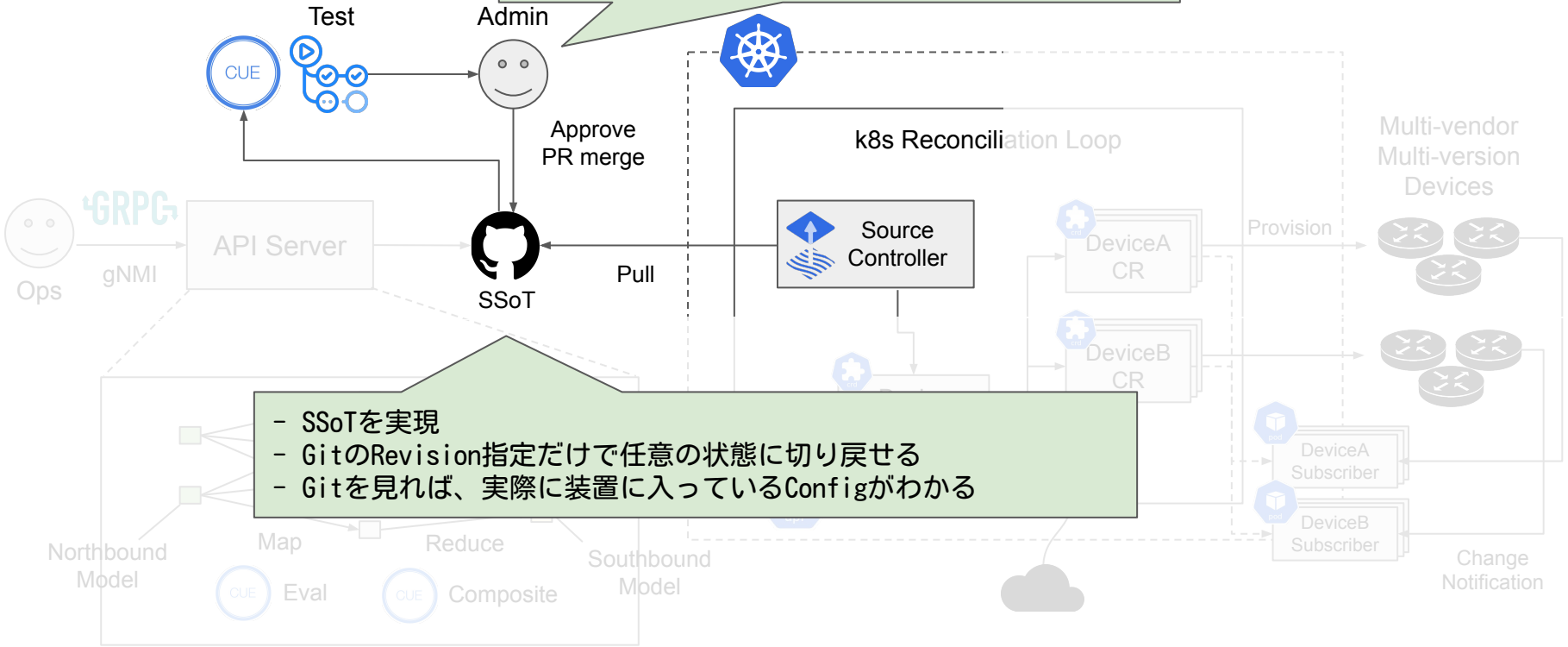
設計

- 上位ドメインモデルから装置コンフィグへのマッピングを CUEで簡単に記述できる
- CUEによる型バリデーションとポリシーエンフォースメント
- ドメイン駆動な開発ができる



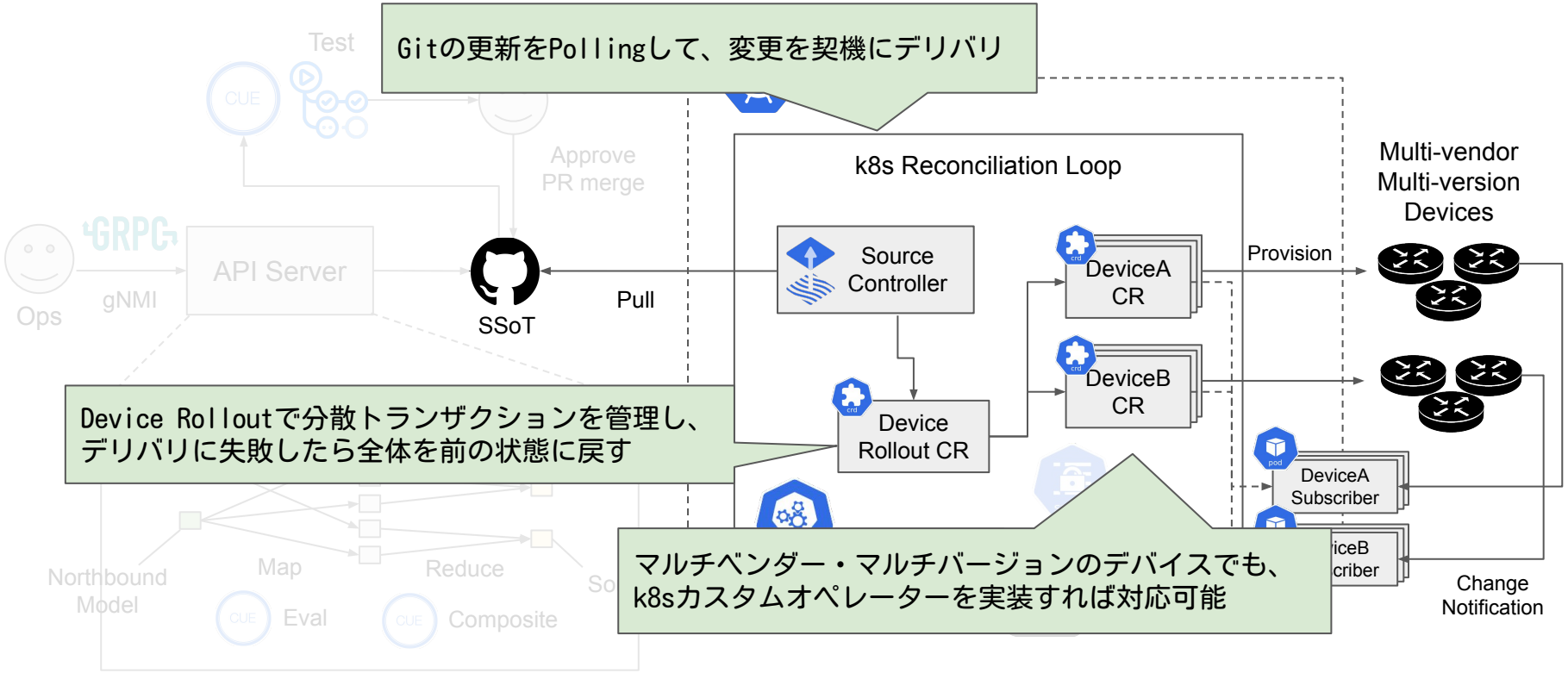
設計

- 実際のコンフィグを用いて、CIのテストを回せる
- 機械的な判定を自動化して、Reviewの負担を軽減

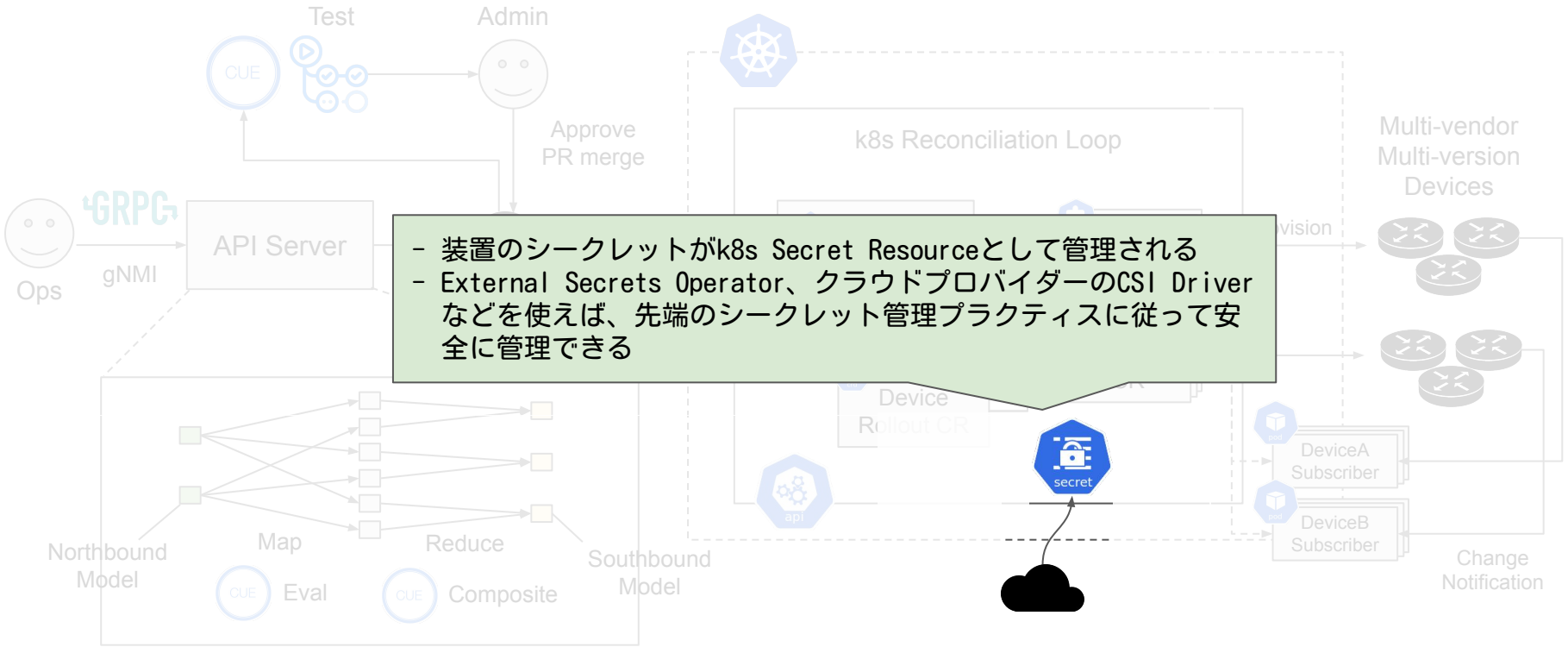


- SSoTを実現
- GitのRevision指定だけで任意の状態に切り戻せる
- Gitを見れば、実際に装置に入っているConfigがわかる

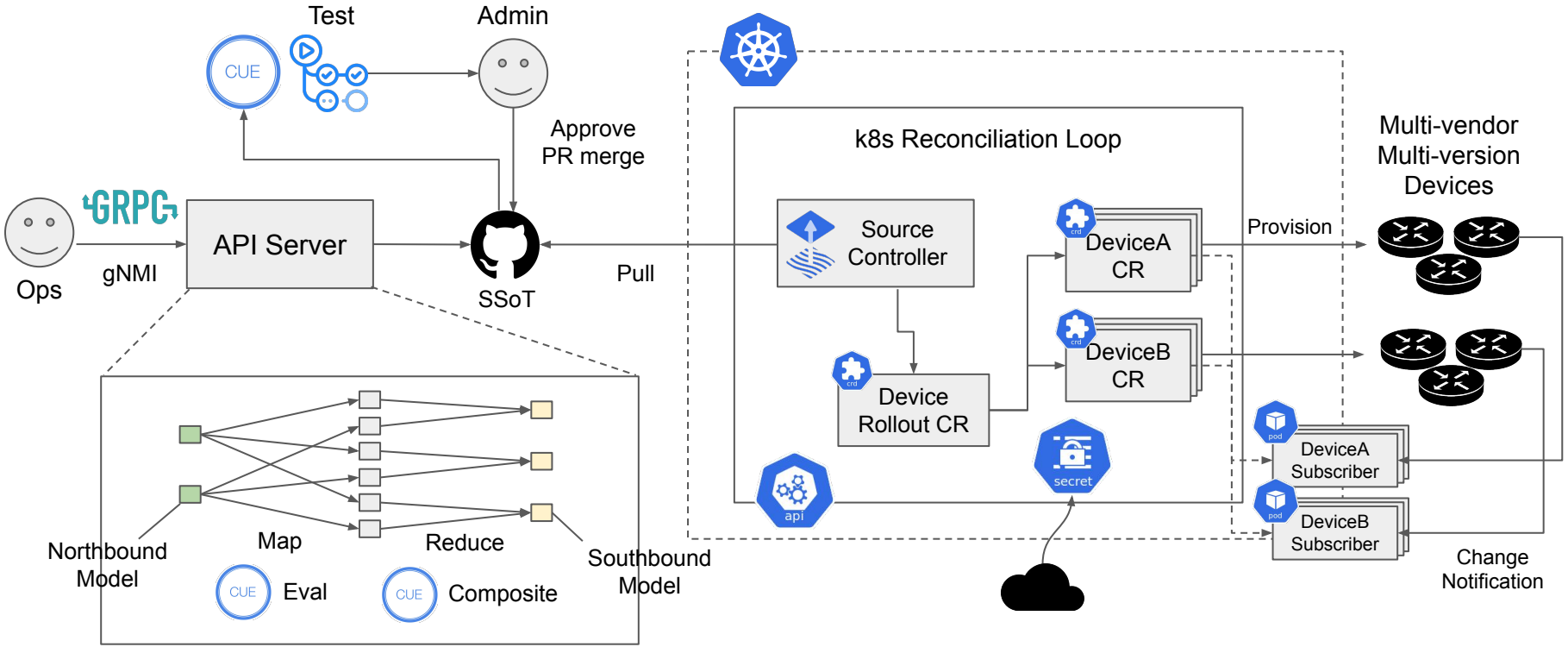
設計



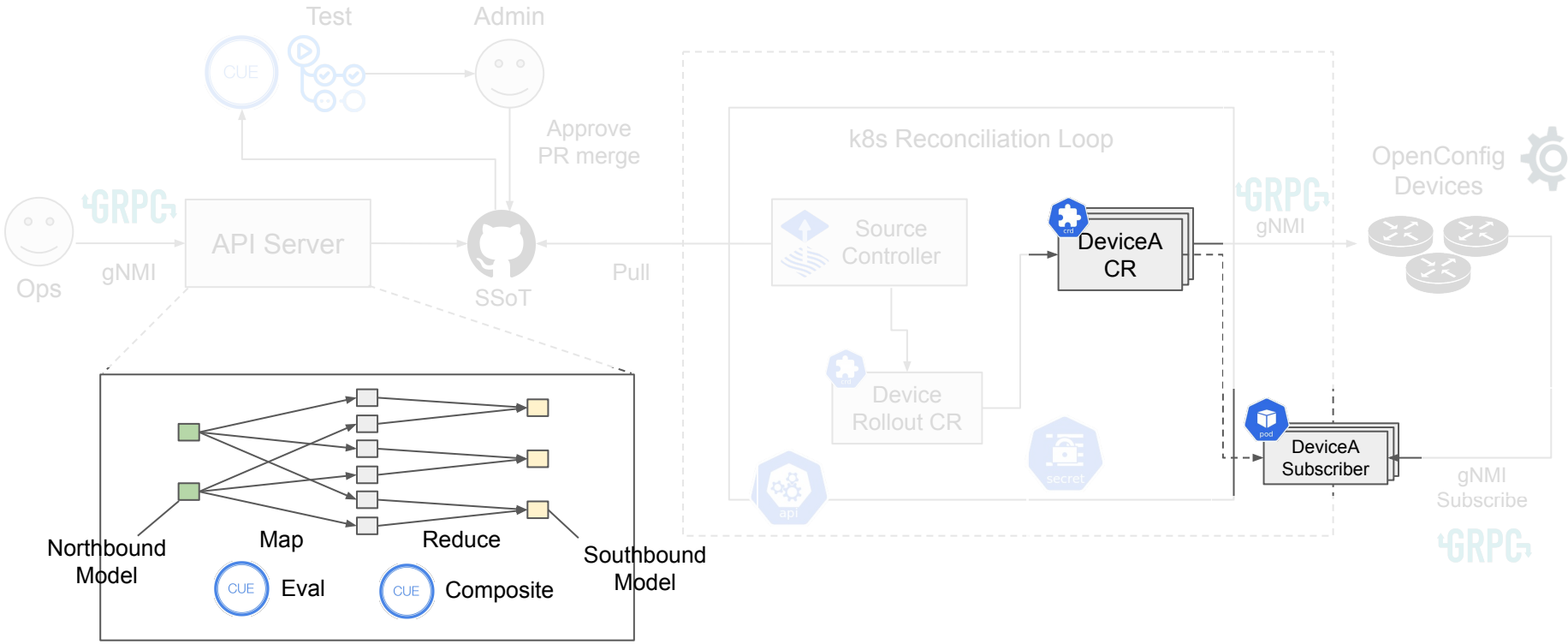
設計



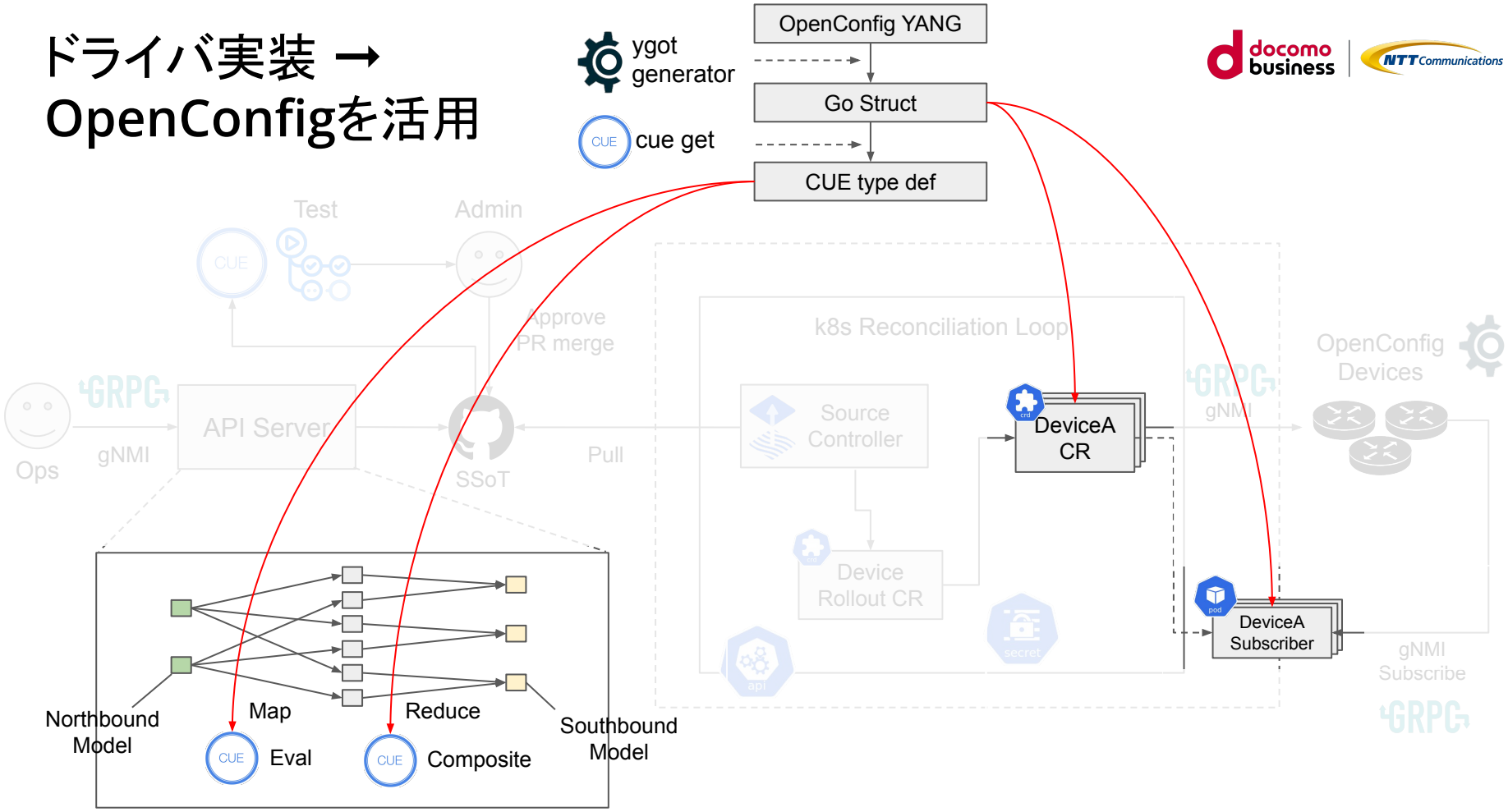
設計



ドライバ実装はどうする？



ドライバ実装 → OpenConfigを活用



ほしいものはできたのか？

ネットワークコンフィグを、テキストではなくモデルで扱う能力

- CUEで実現できた
- （CUEを記述する必要があり、敷居があがっている問題がある）

GitOpsの実践

- Flux CD Source-Controllerで簡易にできた
- k8s Secret関連のプラクティスや最小権限の原則を活かせる

そもそも、ネットワークコントローラとしてほしいもの

- マルチデバイスの分散トランザクション => k8s カスタムオペレータで実現
- マルチベンダー・マルチバージョン対応 => k8s カスタムオペレータで実現
- 外部システムとのAPI連携のためのNorthbound Interfaceの自動生成 => gNMIで実現

まとめと今後の予定

クラウドネイティブ技術を用いたネットワークコントローラを開発した

- OpenConfig/gNMI emulator対向での動作確認済

実機結合検証中

- 伝送：OpenConfig/gNMIを具備する伝送Whiteboxとのフィールドトライアル
- 転送：予定なし（検証ユースケース・検証パートナー募集中）

OSS化を目指して社内調整中

- 早ければ12月頃予定
- UX改善、パフォーマンス改善、ハーデニングなど推進中