CUEとKubernetesカスタムオペレータを用いた新しいネットワークコントローラをつくってみた

NTTコミュニケーションズ
イノベーションセンター
奥井 寛樹

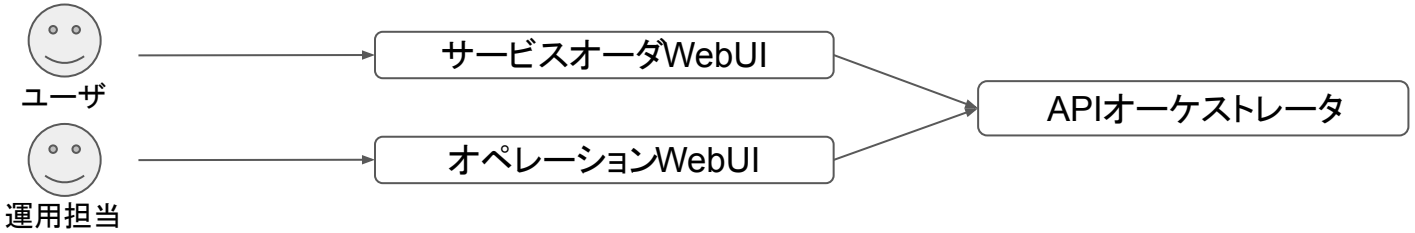
自己紹介

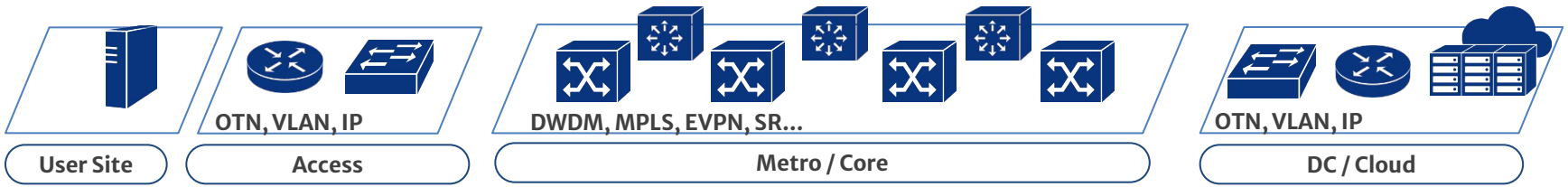
- NTTコミュニケーションズ
- Software Engineer

- 奥井 寛樹 @HirokiOkui
  hrk091

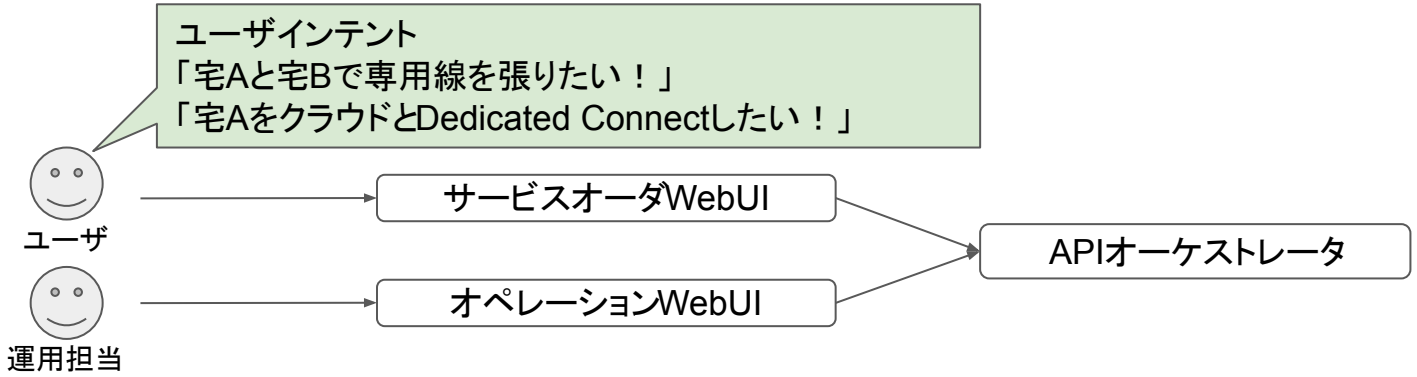
- 略歴
  - 伝送ネットワークの設定自動化システム開発
  - DevOpsプラットフォーム（Qmonus Value Stream）開発
  - IoTデータ収集基盤のモダナイゼーション

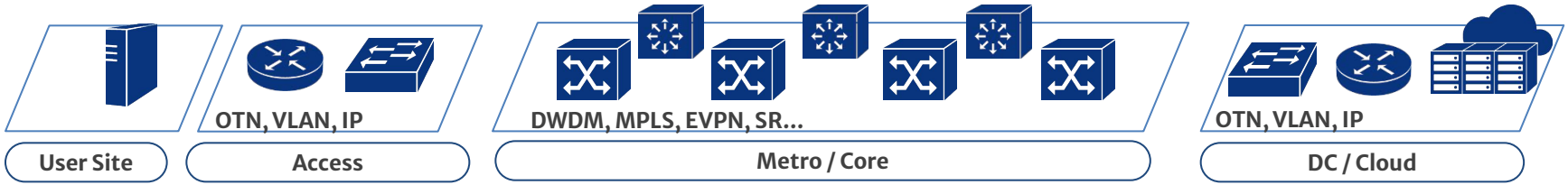
この取組で目指しているもの



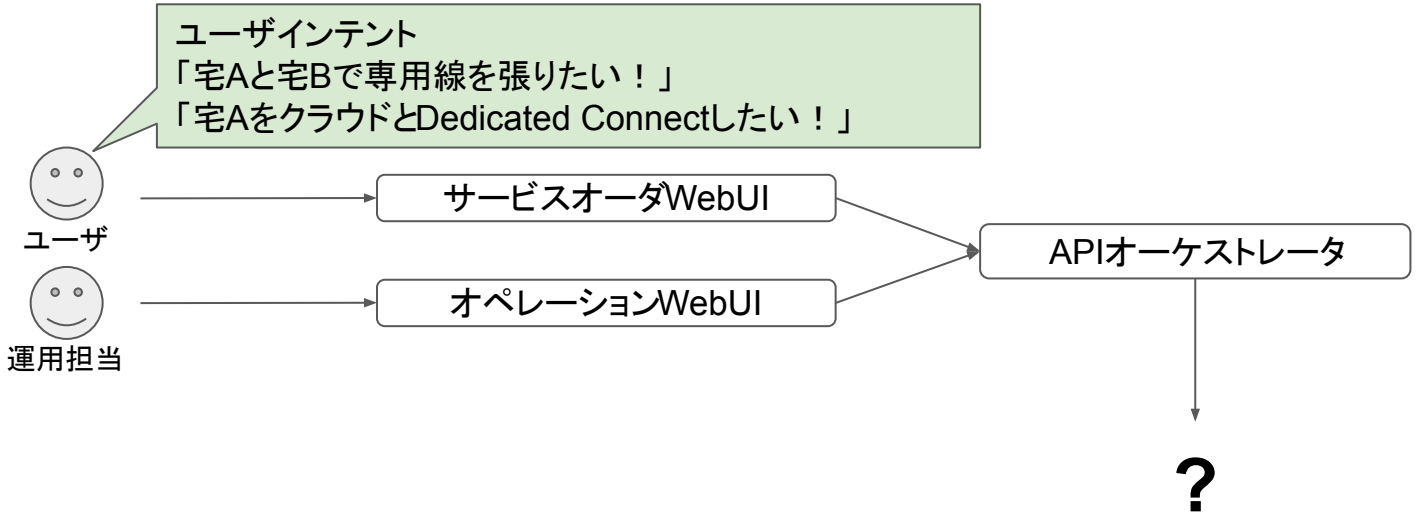


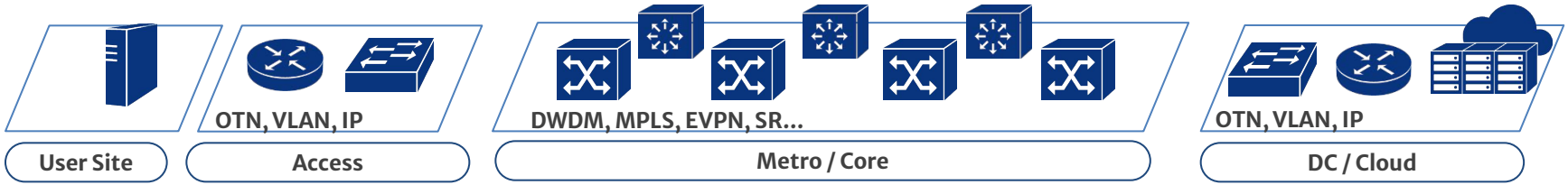
この取組で目指しているもの



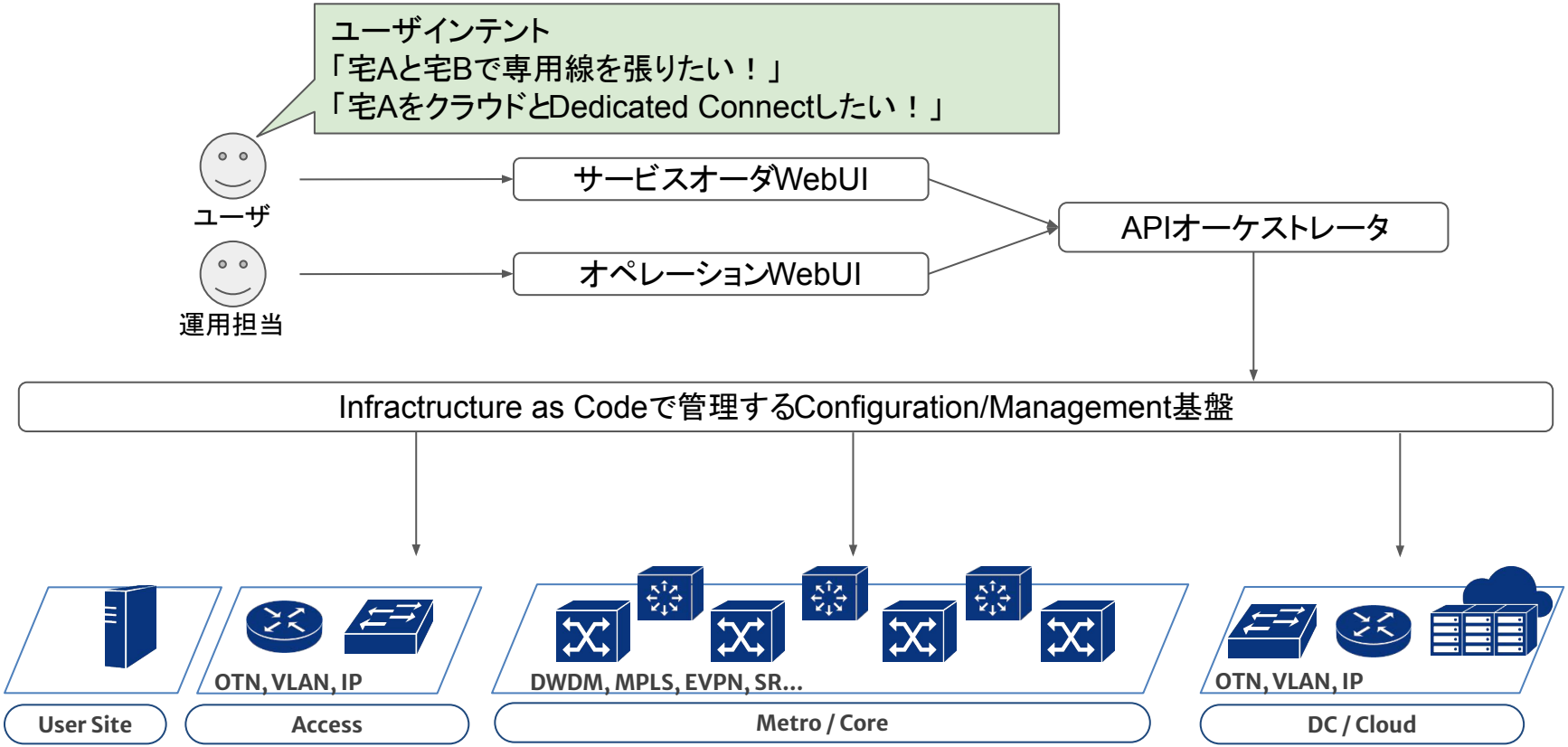


この取組で目指しているもの

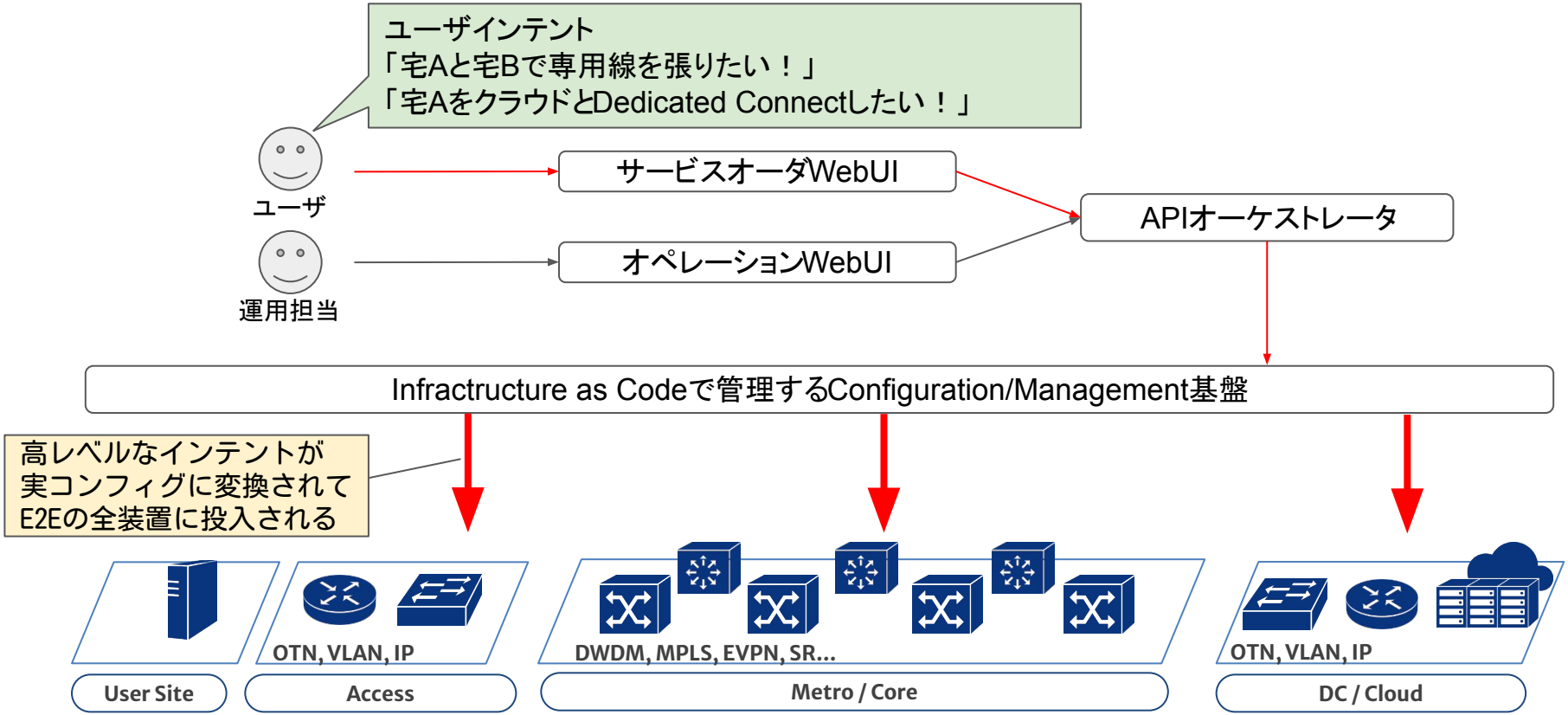




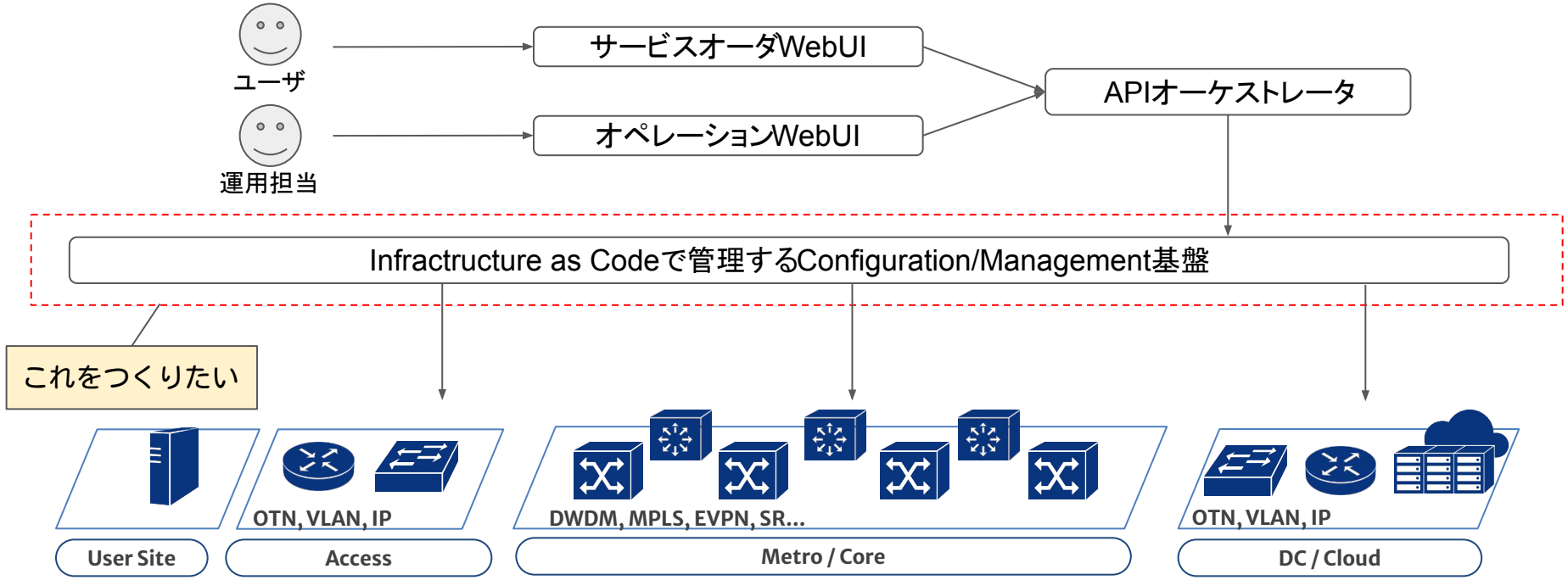
この取組で目指しているもの



この取組で目指しているもの

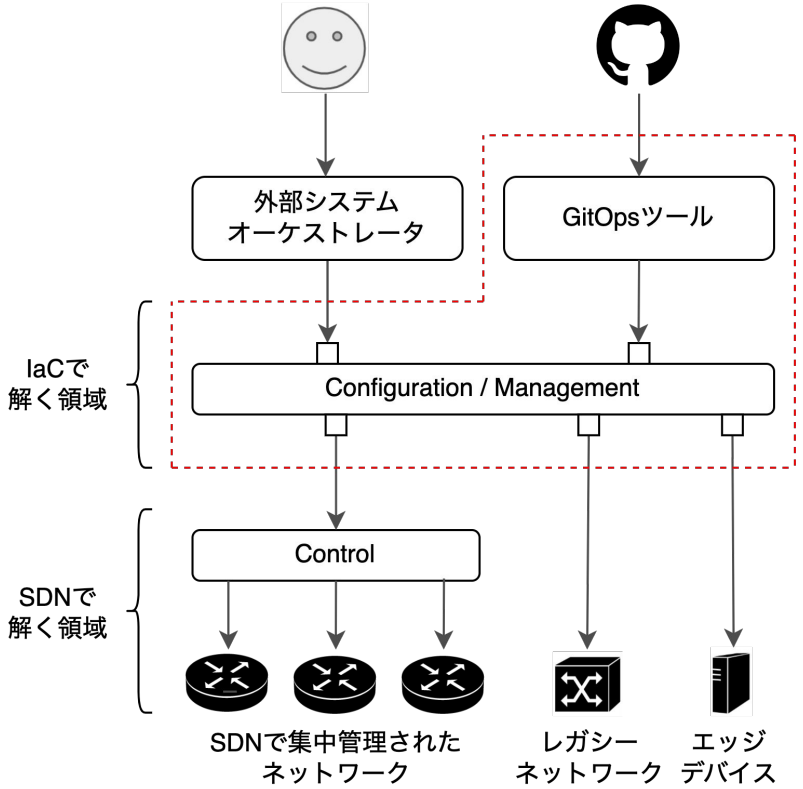


この取組で目指しているもの



本発表のスコープ

- スコープ
  - 高レベルのユーザインテントからのコンフィグ生成
  - 宣言的なコンフィグ管理
  - GitOps
- スコープ外
  - SDNによるコントロールプレーン
  - システム間連携をするワークフロー機能
  - 監視
  - ZTP
  - etc…



発表の流れ

- IaCとは？GitOpsとは？
- クラウドネイティブなアプリ開発におけるIaCとGitOps
- ネットワーク管理におけるIaCとGitOpsの現在
- クラウドネイティブ技術を用いて新しいコントローラをつくってみた
- デモ
- 検証結果と課題

IaCとは？GitOpsとは？

IaCとGitOps

- IaC(Infractructure as Code)やGitOpsを実践する例がネットワークでも増えている
- IaCやGitOpsは、コードを宣言的に表現してGitで管理をすればよいわけではない
  - 正しく活用しないと、本質的な価値を享受できない
- IaCとGitOpsについて、あらためておさらいします

IaCで大事なこと（特に開発目線で）

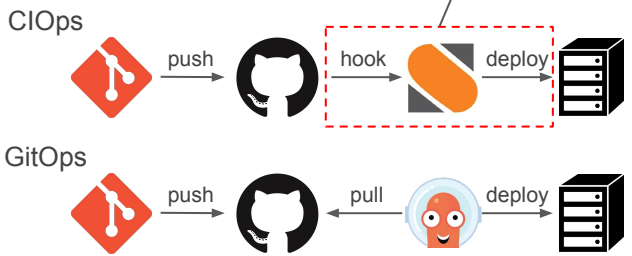
- インフラをコードで宣言的に記述し、ソフトウェア開発のプラクティスを活かす
  - ツールを使うだけでなく、システム・プロセスが大事
  - 自動テスト、CI/CD、モニタリング、高速なリリースサイクル ...
- 高いプログラマビリティ
  - モジュール化により複雑な具象を隠蔽し、インテントを記述しやすい
    高抽象なインターフェースを提供する
  - 保守性の高い記述ができる（モジュール化、凝集化、型指向、テスタブル）
- CIでテストできる
  - CIでデプロイ前に問題を検出できる（シフトレフト）
  - 静的解析・構成テスト・ポリシーエンフォースメント

GitOpsで大事なこと

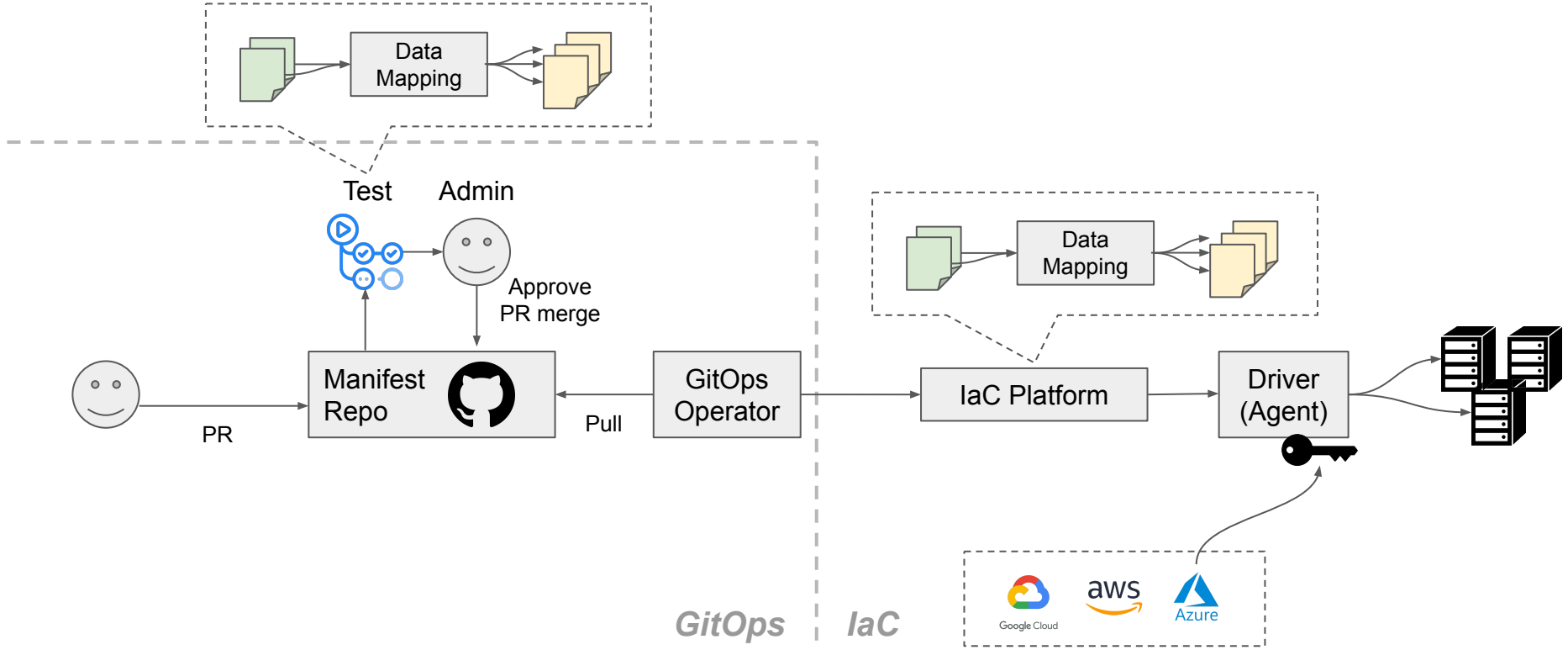
- SSoT(Single Source of Truth)
  - 単一のGitレポジトリでコンフィグ全体が管理されており、唯一の情報源となっている
- Gitのbranch headを変えることで、適用するコンフィグの版を変更できる
  - PRマージをトリガとして、マージされた版がデプロイされる
  - 古いrevisionに戻すと、rollbackできる
- Pullベースのアプローチ
  - SSoTのGitレポのマニフェストを、
    外部から値注入せずにそのままデプロイする
  - Pullベースにすることで、シークレットを
    デリバリ先に近いところに置ける

Git操作だけでデリバリが
完結する = GitOps

Push → 値の加工・注入が容易にできてしまう
Pull → 値の注入処理をはさみにくい

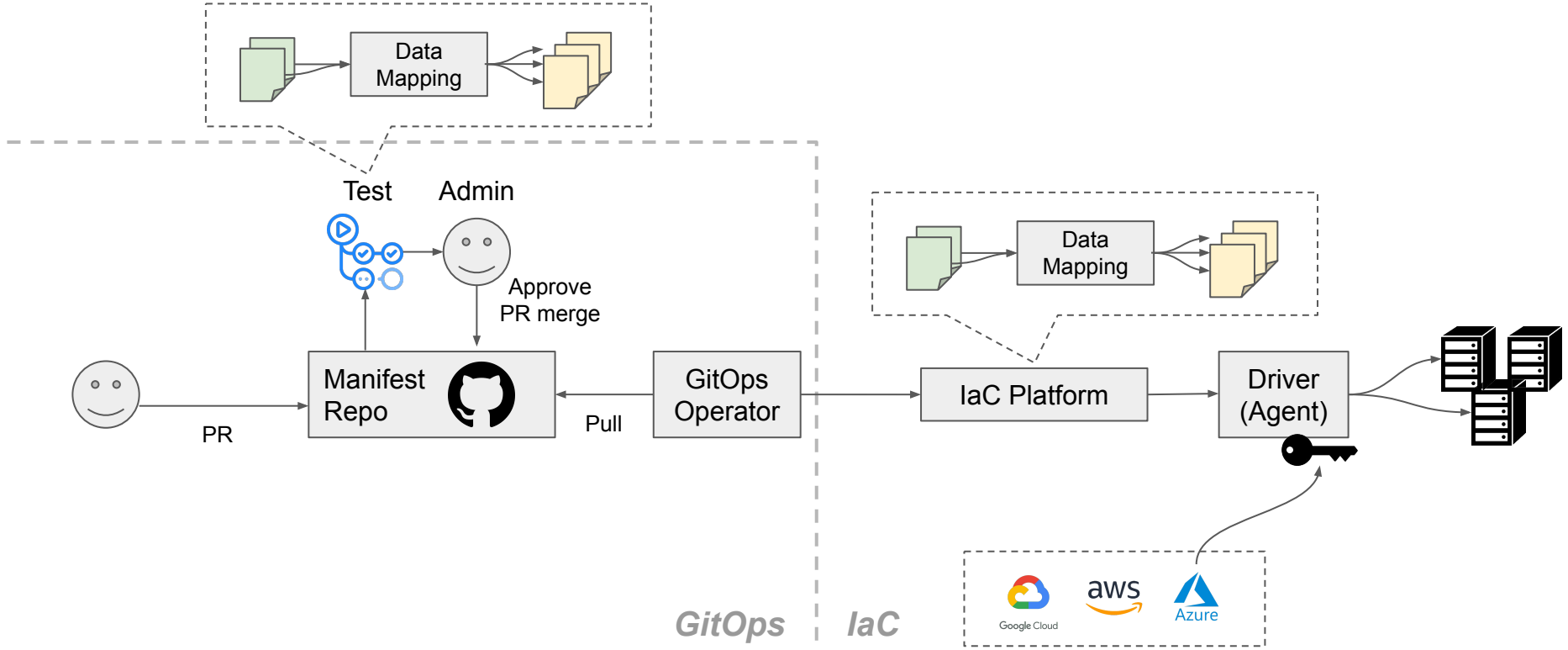


IaC・GitOpsの構成例

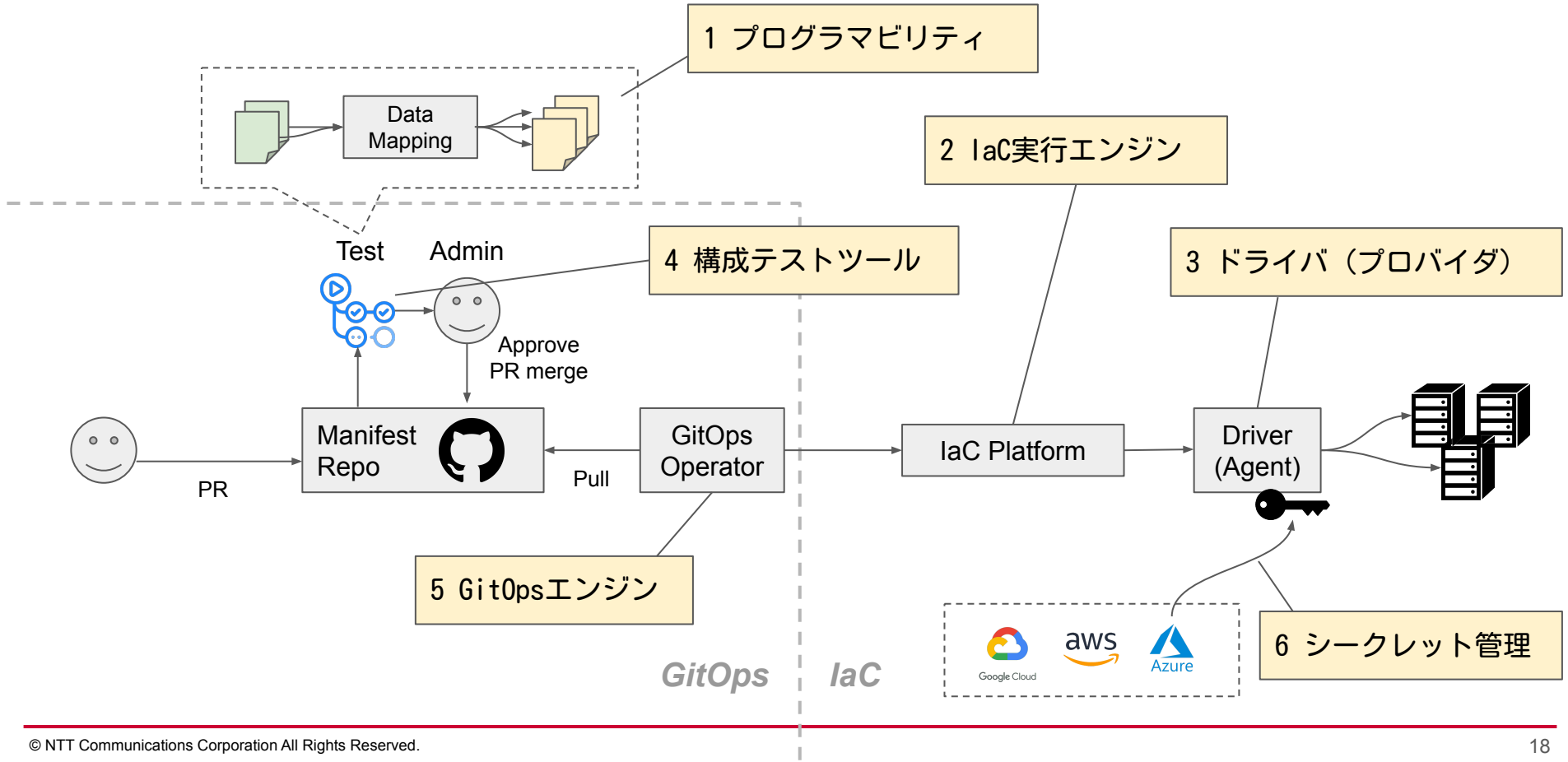


クラウドネイティブなアプリ開発における IaCとGitOps

IaC・GitOpsの構成例



IaC・GitOpsで重要な6つの機能



アプリリリースにおけるIaCとGitOpsのトレンド

機能	概要	ツール・OSS	
		Terraformの場合	Kubernetesの場合
1 プログラマビリティ	- モジュール化、高抽象化 - 型指向でテスタブルな記述	HCL CDKTF	kustomize Helm, CUE
2 IaC実行エンジン	- 制御対象リソースをIaCの宣言状態に冪等収束させるエンジン	Terraform	Kubernetes Reconciliation Loop
3 ドライバ (プロバイダ)	- リソース毎のプロトコル・インターフェース差異を吸収	Terraform Provider	Kubernetes Custom Operator
4 構成テスト	- 静的解析、ポリシーチェックによりデプロイ前に問題を検出	Sentinel	Admission Webhook OPA, Kyverno
5 GitOpsエンジン	- PullベースでGitのコンフィグをそのままデプロイ(SSoT)	Terraform Cloud Atlantis	ArgoCD FluxCD
6 シークレット管理	- シークレットの漏洩防止 - SecretManagerとの連携	Vault	External Secrets Secrets Store CSI Driver

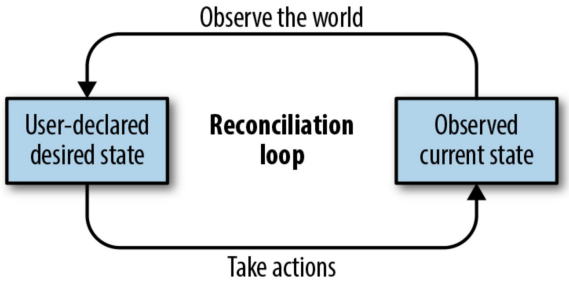
アプリリリースにおけるIaCとGitOpsのトレンド

機能	概要	ツール・OSS	
		Terraformの場合	Kubernetesの場合
1 プログラマビリティ	- モジュール化、高抽象化 - 型指向でテスタブルな記述	HCL CDKTF	kustomize Helm, CUE
2 IaC実行エンジン	- 制御対象リソースをIaCの宣言状態に冪等収束させるエンジン	Terraform	Kubernetes Reconciliation Loop
3 ドライバ (プロバイダ)	- リソース毎のプロトコル・インターフェース差異を吸収	Terraform Provider	Kubernetes Custom Operator
4 構成テスト	- 静的解析、ポリシーチェックによりデプロイ前に問題を検出	Sentinel	Admission Webhook OPA, Kyverno
5 GitOpsエンジン	- PullベースでGitのコンフィグをそのままデプロイ(SSoT)	Terraform Cloud Atlantis	ArgoCD FluxCD
6 シークレット管理	- シークレットの漏洩防止 - SecretManagerとの連携	Vault	External Secrets Secrets Store CSI Driver

IaCの宣言状態に収束させる実行エンジン

- 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



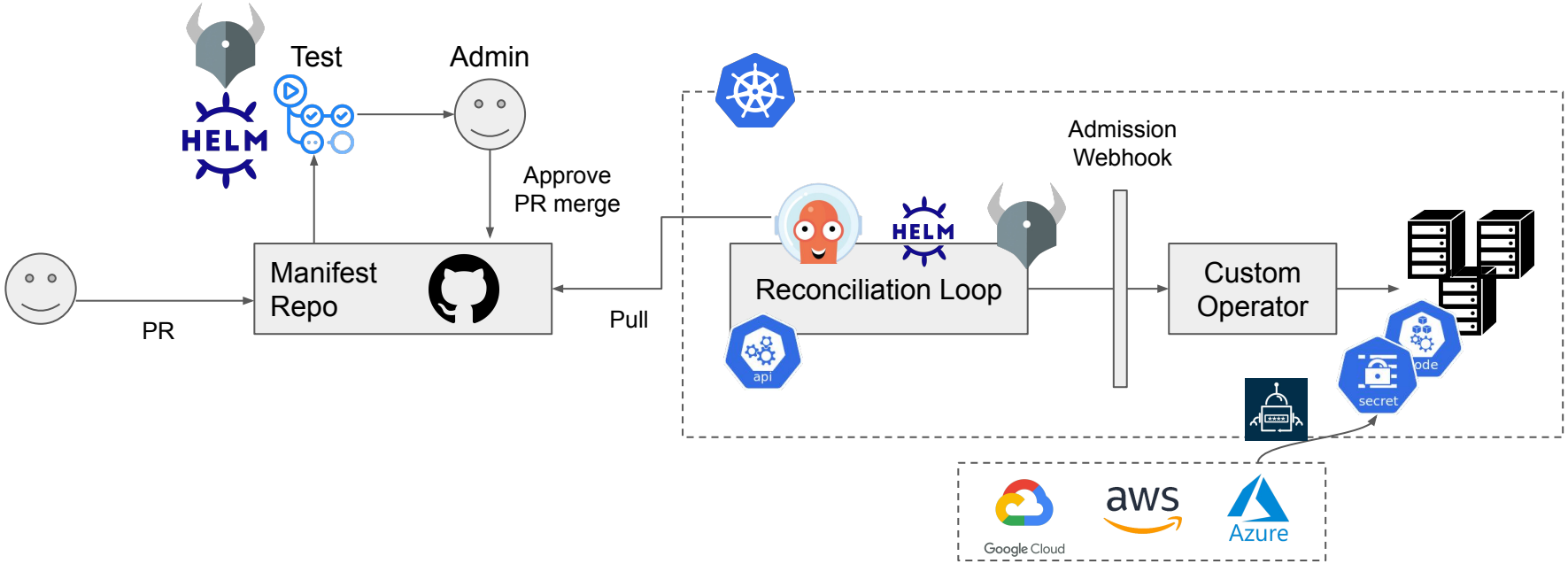
- k8sカスタムオペレータ
  - k8s上で、ユーザが独自の定義を持ち込み、任意のリソースをIaCとして管理するk8s拡張方式
  - CNCFの多くのOSSは、k8sカスタムオペレータとして実装

カスタムオペレータで管理する対象リソースは、Custom Resource(CR)と呼ばれます

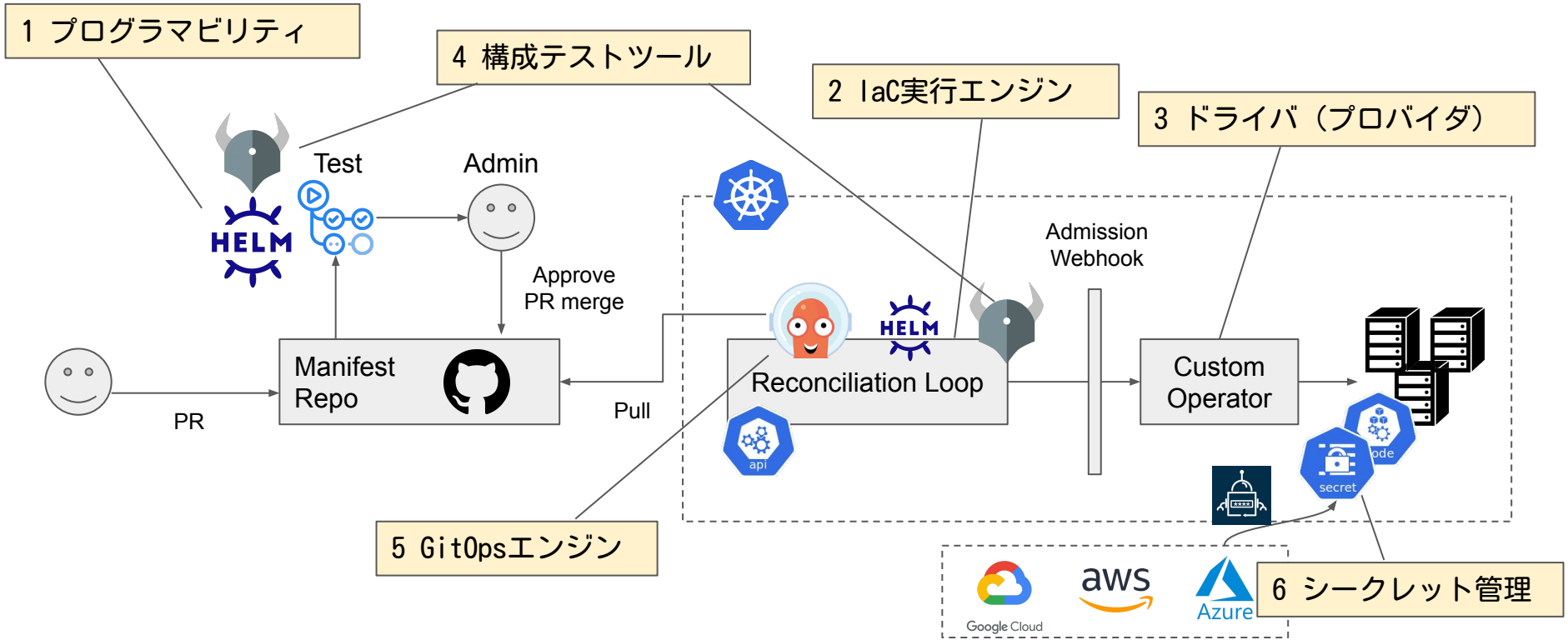
アプリリリースにおけるIaCとGitOpsのトレンド

機能	概要	ツール・OSS	
		Terraformの場合	Kubernetesの場合
1 プログラマビリティ	- モジュール化、高抽象化 - 型指向でテスタブルな記述	HCL CDKTF	kustomize Helm, CUE
2 IaC実行エンジン	- 制御対象リソースをIaCの宣言状態に冪等収束させるエンジン	Terraform	Kubernetes Reconciliation Loop
3 ドライバ (プロバイダ)	- リソース毎のプロトコル・インターフェース差異を吸収	Terraform Provider	Kubernetes Custom Operator
4 構成テスト	- 静的解析、ポリシーチェックによりデプロイ前に問題を検出	Sentinel	Admission Webhook OPA, Kyverno
5 GitOpsエンジン	- PullベースでGitのコンフィグをそのままデプロイ(SSoT)	Terraform Cloud Atlantis	ArgoCD FluxCD
6 シークレット管理	- シークレットの漏洩防止 - SecretManagerとの連携	Vault	External Secrets Secrets Store CSI Driver

現代的なGitOps



現代的なGitOps



ネットワーク管理における
IaCとGitOpsの現在

ネットワーク管理におけるIaCとGitOpsの現在

機能	概要	ツール・OSS	
		Ansible Module/Roleがある	Module/Roleがない
1 プログラマビリティ	- モジュール化、高抽象化 - 型指向でテスタブルな記述	Ansible Module/Roleを加工	汎用言語でスクラッチ (主にPython + Jinja2)
2 IaC実行エンジン	- 制御対象リソースをIaCの宣言状態に冪等収束させるエンジン	Ansible Module/Roleで手続き的に実装	スクラッチ
3 ドライバ (プロバイダ)	- リソース毎のプロトコル・インターフェース差異を吸収	Ansible Moduleで実装	スクラッチ
4 構成テスト	- 静的解析、ポリシーチェックによりデプロイ前に問題を検出	別途作り込み	スクラッチ（難しい）
5 GitOps	- PullベースでGitのコンフィグをそのままデプロイ(SSoT)	WebhookでCIOps or Agentでpull化	WebhookでCIOps
6 シークレット管理	- シークレットの漏洩防止 - SecretManagerとの連携	Ansible Vault	スクラッチ or 外部ツール

ネットワーク管理におけるIaCとGitOpsの現在

機能	概要	ツール・OSS	
		Ansible Module/Roleがある	Module/Roleがない
1 プログラマビリティ	- モジュール化、高抽象化 - 型指向でテスタブルな記述	Ansible Module/Roleを加工	汎用言語でスクラッチ
2 IaC実行エンジン	- 制御対象リソースをIaCの宣言状態に冪等収束させるエンジン	Ansible Module/Role手続き的に実装	
3 ドライバ (プロバイダ)	- リソース毎のプロトコル・インターフェース差異を吸収	Ansible Moduleで実装	
4 構成テスト	- 静的解析、ポリシーチェックによりデプロイ前に問題を検出	別途作り込み	
5 GitOps	- PullベースでGitのコンフィグをそのままデプロイ(SSoT)	WebhookでCIOps or Agentでpull化	WebhookでCIOps
6 シークレット管理	- シークレットの漏洩防止 - SecretManagerとの連携	Ansible Vault	スクラッチ or 外部ツール

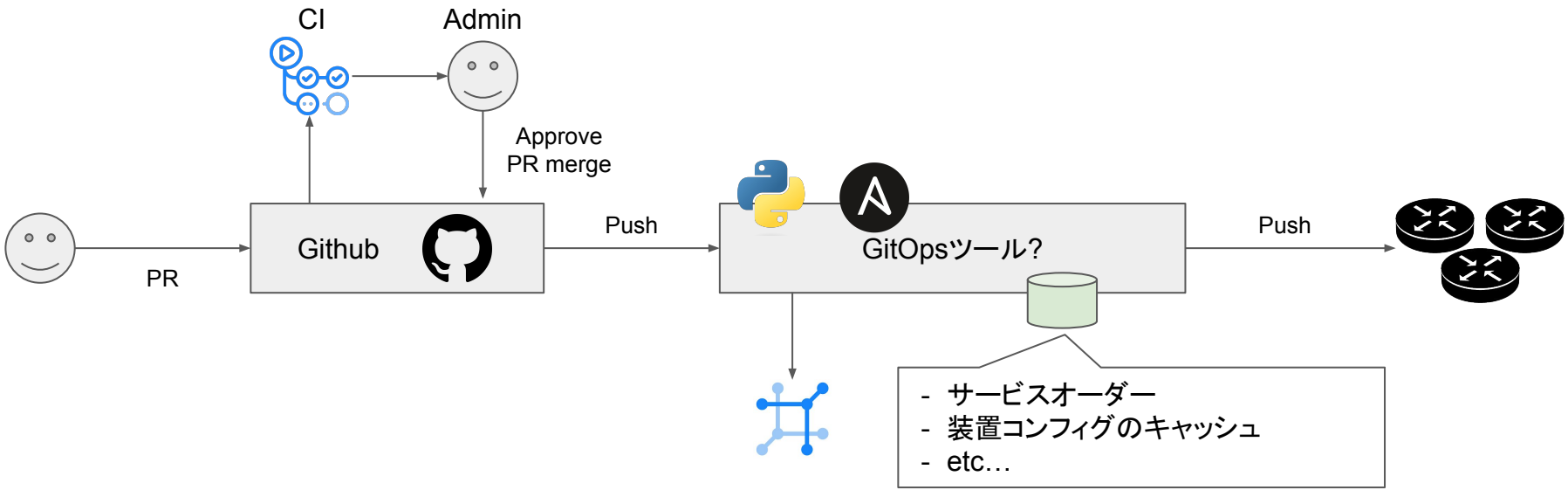
ユーザが少なくAnsibleが未整備な装置は結局スクラッチ開発（伝送装置など）

スクラッチ開発をすると手続き的なスクリプトになり、Jinja2でテンプレートマッピングしてSSHで投げるだけになる

IaCの実践はほぼ不可能

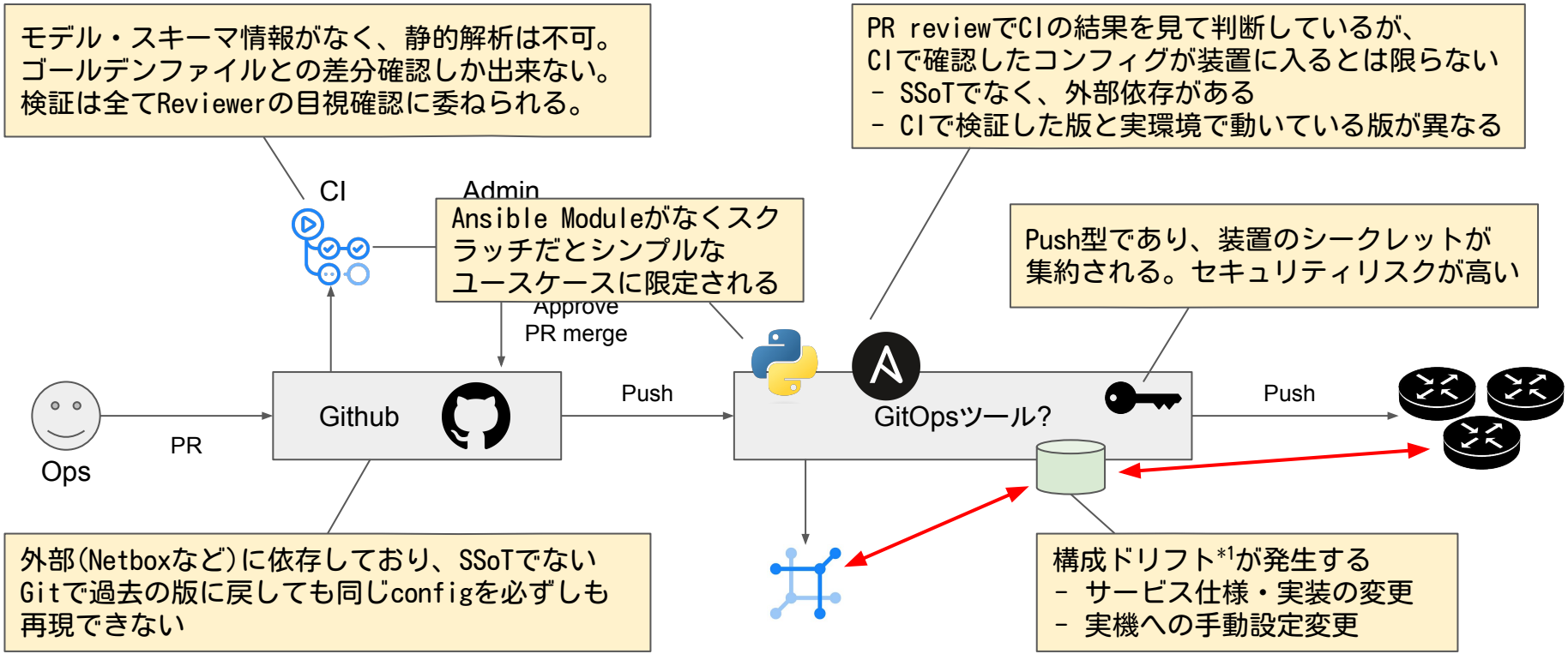
ネットワークのGitOps*1の例

*1: 正しくはCIOps



ネットワークのGitOps*1の例

*1: 正しくはCIOps



*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

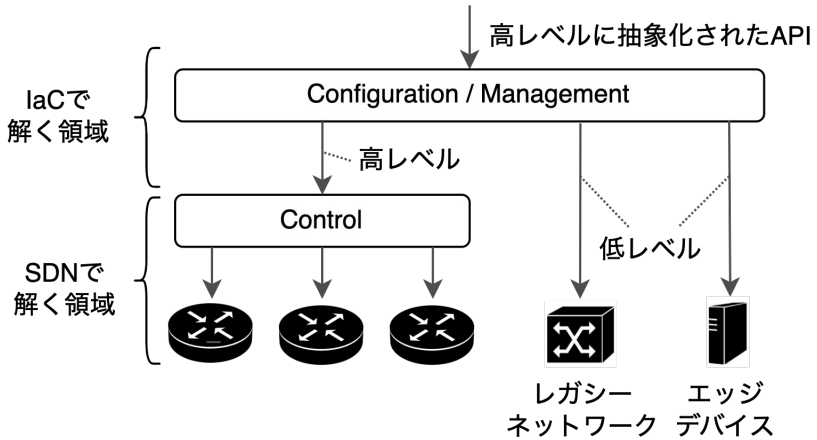
クラウドネイティブ技術を用いて
新しいコントローラをつくってみた

つくりたいもの

- **任意のネットワークコンフィグを高レベルなモデルに抽象化しIaCとして制御するもの**
  - ネットワークのユースケース・プロトコルスタック非依存、伝送も転送も何でも表現可能
  - 装置コンフィグは低レベルすぎるので、高レベルへの抽象化が必要
    - インテントベースになっているSDNはそのままで良いが、それ以外の既存設備など

- **GitOps**で、Git操作だけでデプロイを完結させる
  - CIテスト → PRレビュー&マージ でデプロイ
  - SSoTでありGit操作だけでロールバック

- その他、基本機能としてほしいもの
  - **マルチベンダー・マルチバージョン** のサポート
  - **マルチデバイス** の分散トランザクション
  - アーキテクチャの柔軟性・開発容易性

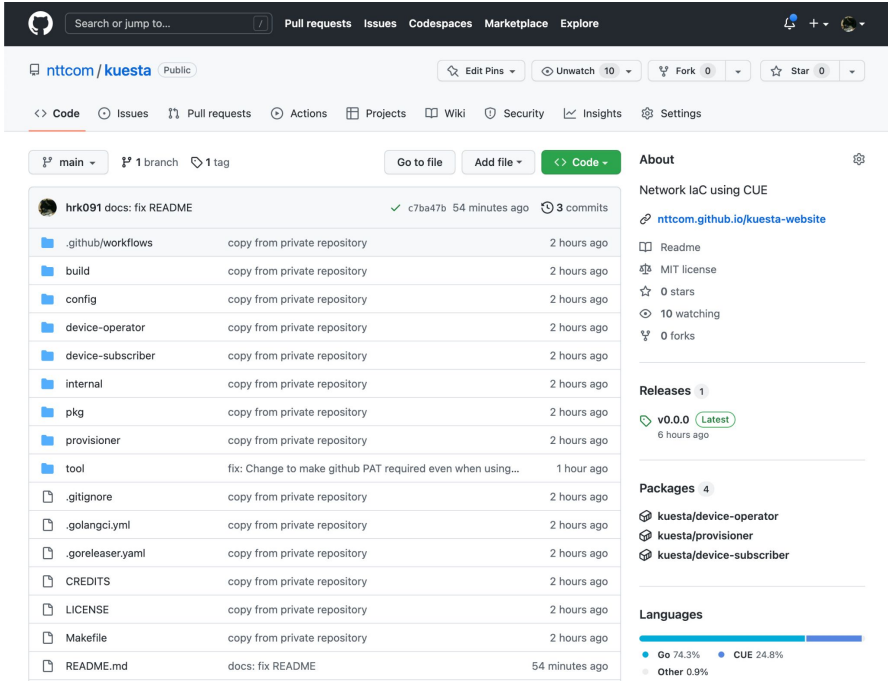


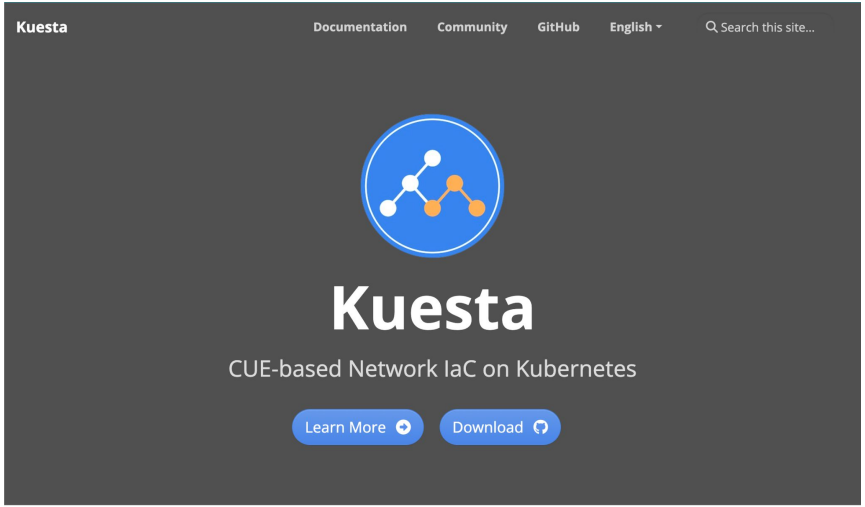
Kuesta

CUE-based Network IaC on Kubernetes

OSS化しました

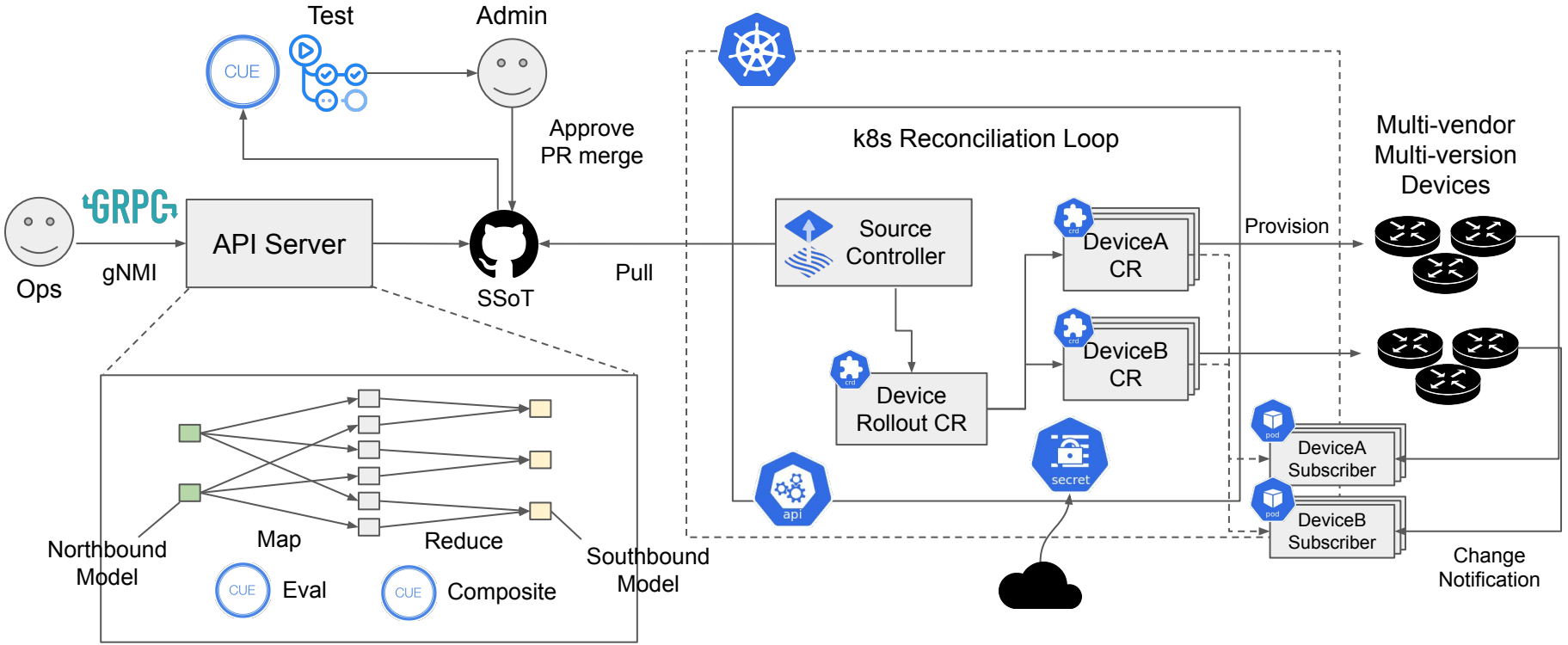
- https://github.com/nttcom/kuesta





Kuesta is an open source framework for network configuration, enabling operators Infrastructure-as-Code of network equipments. It provides us a rich network automation by adopting cloud-native best practices of IaC and GitOps.

kuestaのアーキテクチャ



ネットワークIaC実現のための8つの技術要件

1. ドキュメント型を定義できるスキーマ言語
2. 抽象モデルからのマッピングを記述する言語
3. ManyToManyのリソースマッピング
4. IaCをデプロイするエンジン
5. GitOpsエンジン
6. マルチデバイス間の分散トランザクション
7. 多様な装置に対するドライバープラグイン
8. デバイスシークレット

1ドキュメント型を定義できるスキーマ言語

WHY

- CIでテストを行うには、ネットワークコンフィグの型情報は必須
  - 型情報があることで、デプロイ前の静的解析で問題を検出できる
- コーディングの品質が大きく改善
  - 汎用言語の構造体を出力することで、型安全なコーディングができる
  - マッピングの実装でも、ポリシーの実装でも

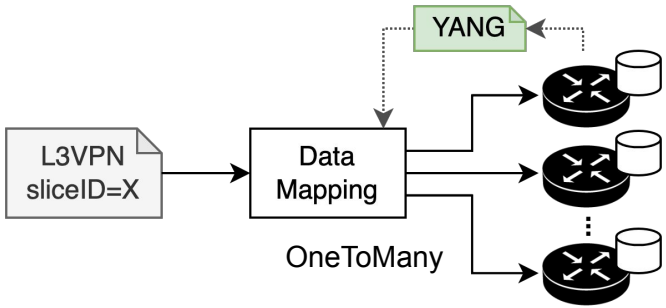
HOW

- YANG
  - ネットワークコンフィグのドキュメントツリースキーマを定義できる
    - 最近は多くの装置がYANGのモデルを提供している
  - pyang, goyangなどを使えば、汎用言語の構造体を出力できる
  - 他の静的解析ツールと組み合わせて、コンフィグの検査ができる

2 抽象モデルからのマッピングを記述する言語

WHY

- IaCでは、インテントを可読性高く記述したい
  - 装置コンフィグは低レベルすぎるので、高レベルへの抽象化が必要
- 高レベルなモデルから低レベルな装置コンフィグへの
  マッピング処理の実装が必要
- YANGの生成型から、型安全にマッピング処理を
  実装できるのが望ましい



HOW

- Python, Goなど、YANGから型生成できる言語なら何でも良い
  - ただし、テンプレートライブラリとの連携がシームレスな言語が良い
- kuestaでは**CUE**を使用（理由は後述）

CUE

3 ManyToManyのリソースマッピング（1）

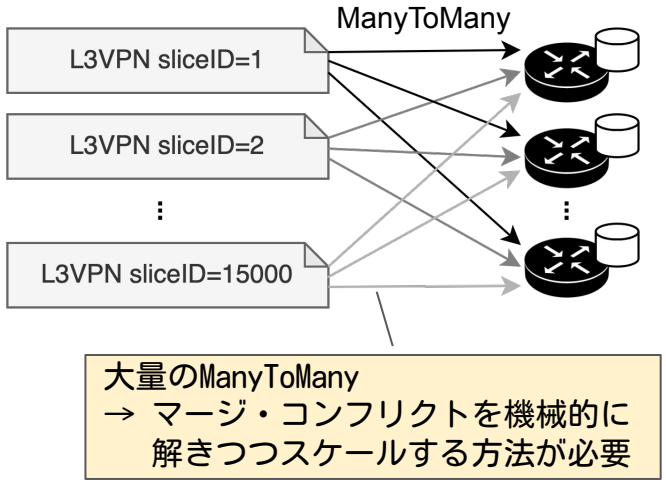
WHY

- 装置コンフィグは、装置ごとに
  独立した1つのドキュメントとみなせる

- 面・スライスでの抽象化をすると
  上位・下位モデル間がManyToManyに

- ManyToOne/ManyToManyは難しい
  - ある上位モデルの更新時に、他の親から設定された
    コンフィグを保護しつつ、コンフリクトも避ける必要がある

- **ネットワークIaCはManyToManyと戦わなければならない**
  - 装置コンフィグを直接扱うIaCでは、ManyToManyは避けられない



クラウドIaCでは、ManyToManyは
特定のユースケースに限定するなど
うまく避けています

CUE

- データの中にロジックを記述する新しいデータ記述言語
  - GCL(Google/Borgの設定言語)の作者が作った OSS
- **データのマージに強い**
  - 任意の数・任意のレイヤのドキュメントツリーをマージ化
  - 評価順に依存せず、一意の結果に収束する(交換律と結合律)
- 型と値と値制約を同一視して一緒に扱う型システム
  - 値が収束すればOK、値が不定なら検査エラー
  - シンプルで可読性が高い
- 高いプログラマビリティ
  - ドキュメントツリーを宣言的プログラミングで記述
  - テンプレーティング、モジュール化などを CUEだけで実現
  - Go API、OpenAPI、Protobufなどから型を自動生成

交換律:

$$a+b = b+a$$

結合律:

$$(a+b)+c = a+(b+c)$$

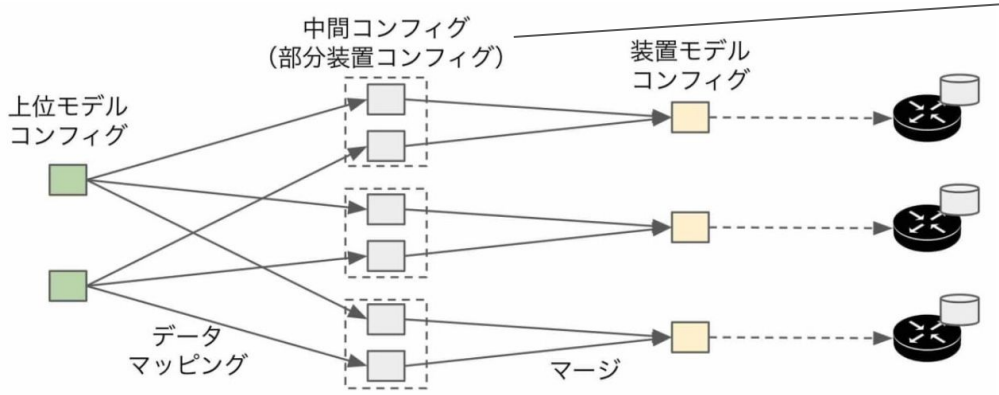
Types and Values

```
// Value
Alice: age: 20

// Type
People: age: int

// Constraint
Member: age: > 18

// Validate
Alice & People & Member
```

3 ManyToManyのリソースマッピング（2）

HOW

- CUEを使う
  - ManyToManyに強い、型安全、コンフィグのプログラマビリティが高い
- ただし、大量のリソースを一気に評価するとパフォーマンス問題がでるので、
  以下の2つに処理ステップを分ける
  1. 上位モデル → 下位モデル（装置コンフィグ）へのマッピング
  2. 1の処理結果をマージして、実際の装置コンフィグを生成

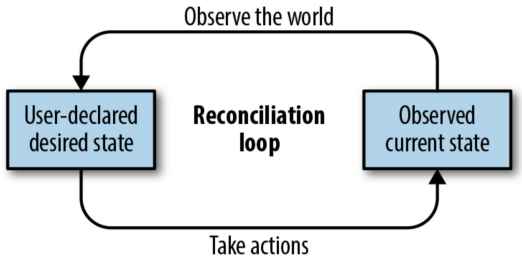


中間コンフィグもgitにコミットしてキャッシュとして使うことでperformanceを改善

4 IaCをデプロイするエンジン

WHY

- デプロイ先の状態を観測しつつ、IaCで宣言された状態に一致するようにデプロイ処理を進めるエンジンが必要
- 全体を駆動する基盤となるため、拡張性が高いものが望ましい



HOW

- Kubernetes(k8s)
  - k8sカスタムコントローラを用いることで、ネットワーク装置の制御もできる
  - ArgoCD、FluxCDなど、k8sネイティブなGitOpsツールがある
  - 必要に応じて、任意のオペレーション用の Podを起動し処理を実行できる
  - 成熟していて開発しやすい（情報・ツール・開発者が多い）

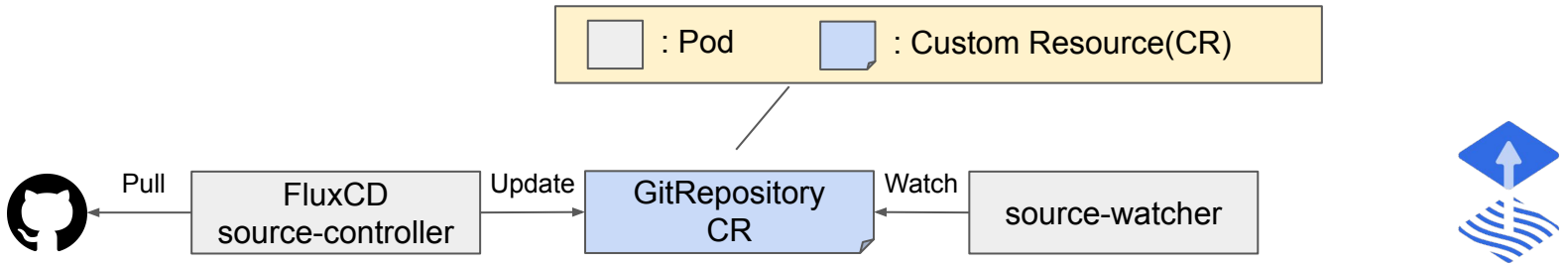
5 GitOpsエンジン

WHY

- SSoTなGitレポジトリからコンフィグを取得して、加工・注入せず装置に流し込む
- Gitの操作だけで完結するために必要

HOW

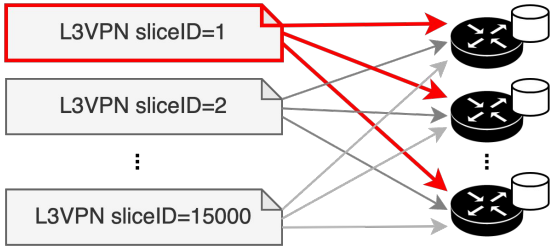
- FluxCDのsource-controllerを使う
  - FluxCDのgitからpullする機構を使いつつ、k8sへのapply以外の用途に拡張できる



6 マルチデバイス間の分散トランザクション

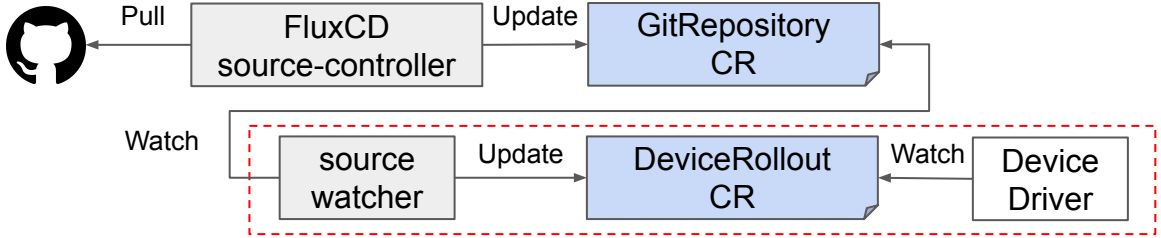
WHY

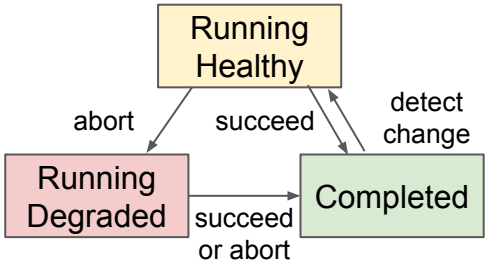
- 上位モデルを作成・更新することで複数の装置に設定が入るため、装置間のトランザクションが必要
  - all or nothingを保証して、一部にのみ設定が入るのを回避



HOW

- DeviceRollout CRとコントローラを実装し、トランザクションの管理をさせる
  - ArgoRolloutを参考にした設計





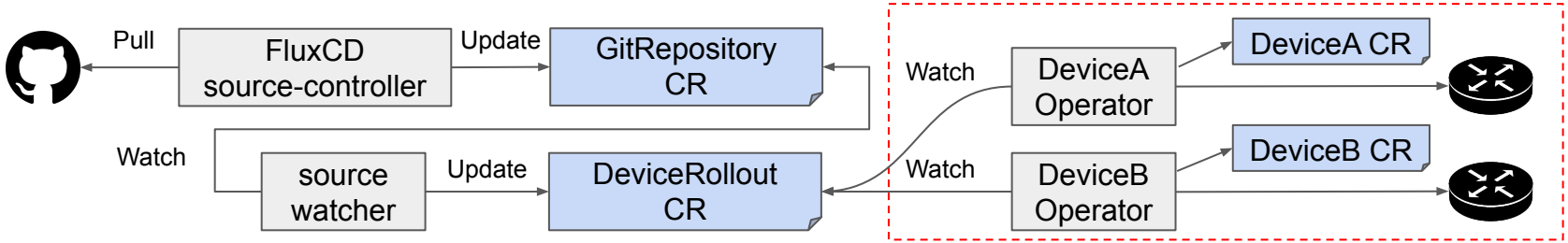
7 多様な装置に対するドライバプラグイン

WHY

- マルチベンダ・マルチバージョンの各デバイスを制御したい
  - デバイスごとに、SSH/REST/NETCONF/gNMIと異なるプロトコルを使用しており、コンフィグのsyntax/semanticsも異なる

HOW

- k8sカスタムオペレータを使用する
  - カスタムコントローラ：コンフィグを実行するドライバ処理の実装
  - カスタムリソース：装置のアドレスなどの接続情報、投入されたコンフィグを保持
- 装置・バージョンごとに**実装すれば**、k8sの仕組みを活用していくらでも拡張可能



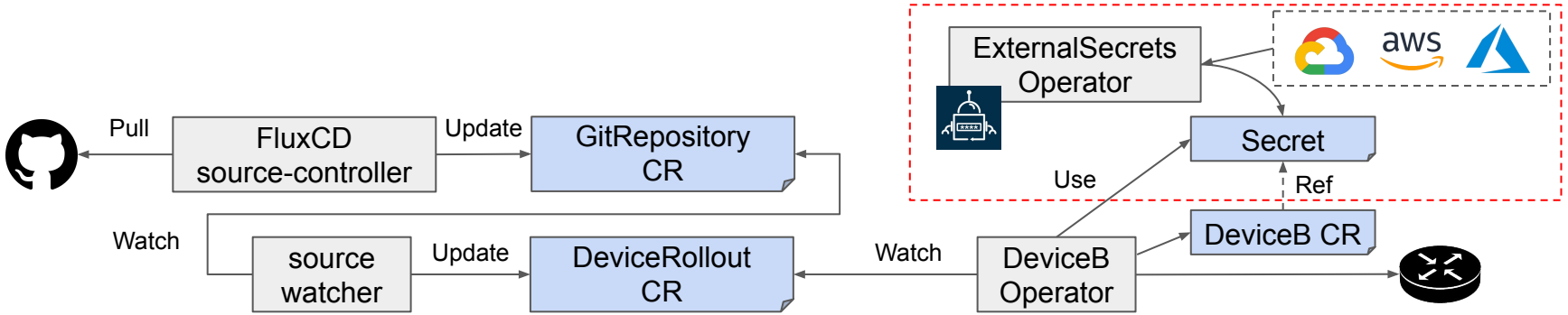
8 デバイスシークレット

WHY

- 装置のアクセスに使用するパスワードを安全に管理したい
- 漏洩リスク・漏洩時の影響を最小化したい

HOW

- k8sのSecretを使って管理し、k8sで盛んに開発が進められている Secret管理プラクティスに則って管理する
- External Secrets Operatorを使って、パブリッククラウドのSecretManagerで管理する

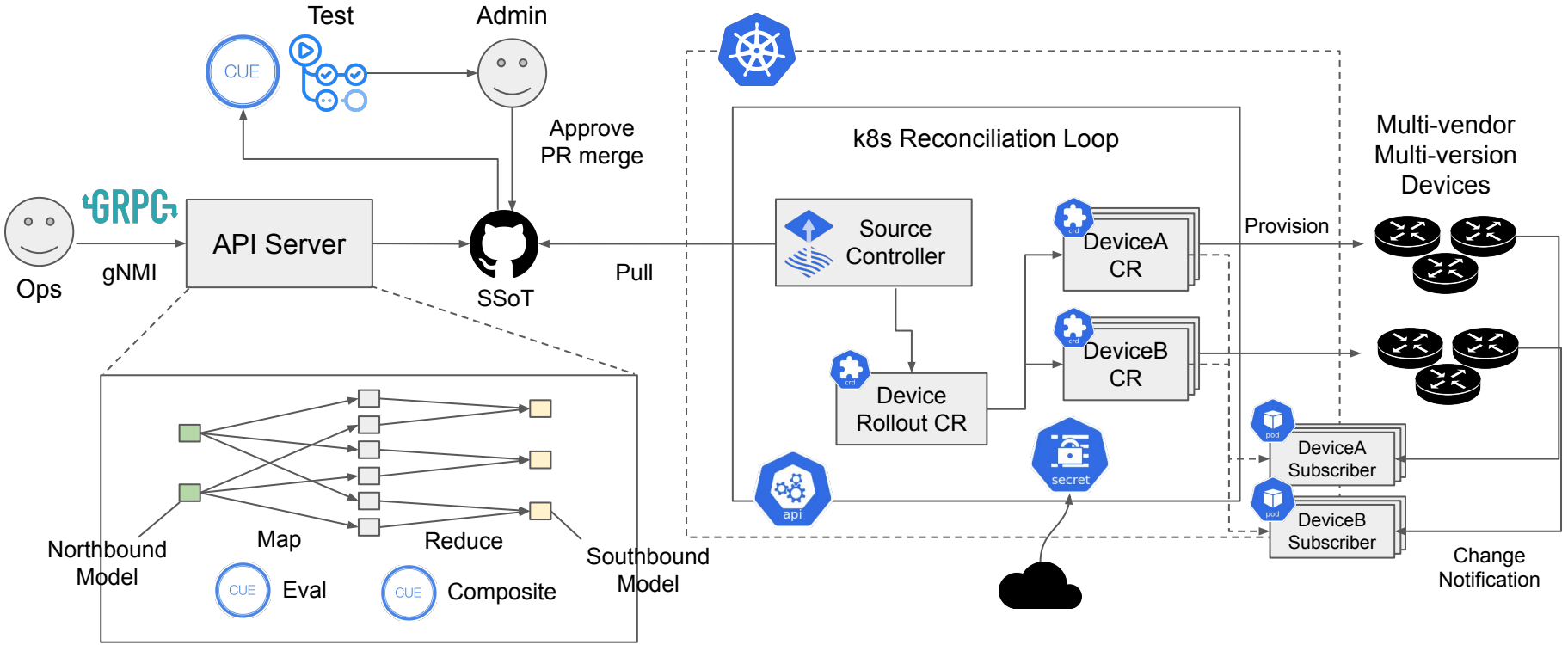


ネットワークIaC実現のための技術選定

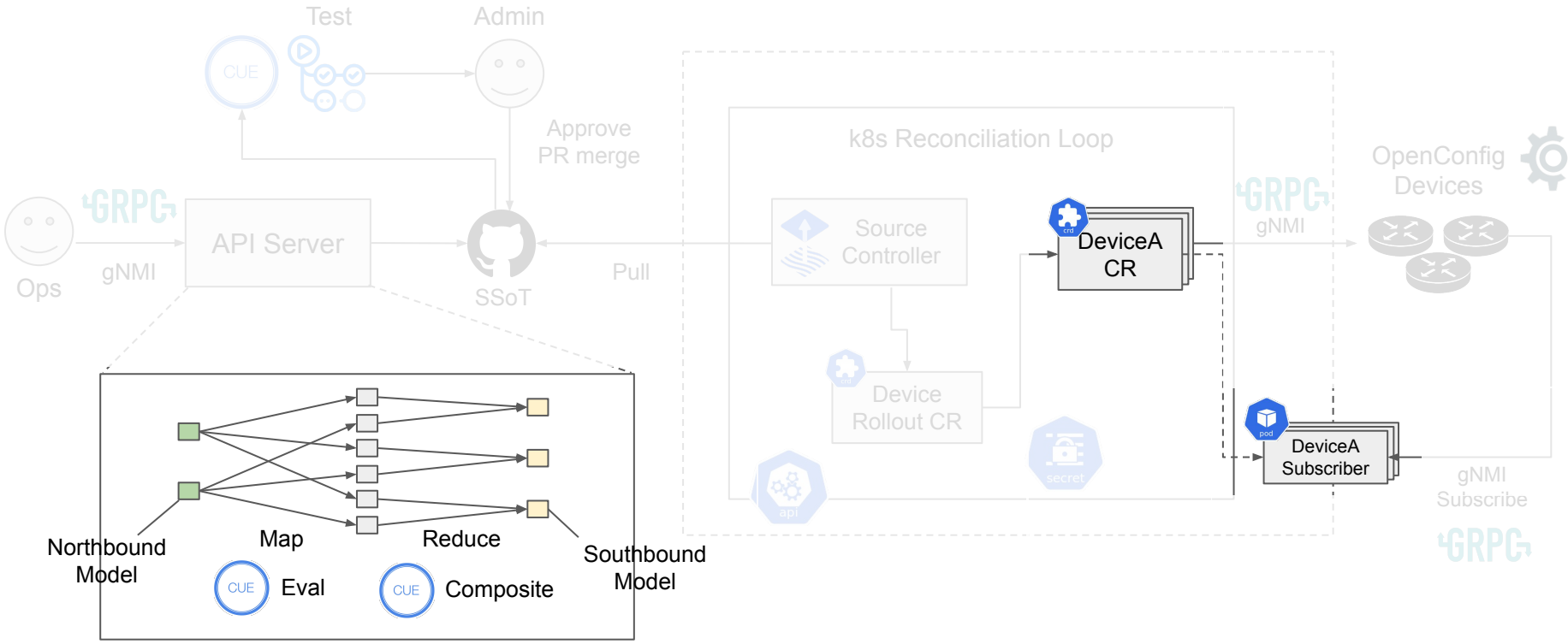
1.	ドキュメント型を定義できるスキーマ言語	➔ YANG
2.	抽象モデルからのマッピングを記述する言語	➔ CUE
3.	ManyToManyのリソースマッピング	➔ CUE
4.	IaCをデプロイするエンジン	➔ k8s
5.	GitOpsエンジン	➔ FluxCD
6.	マルチデバイス間の分散トランザクション	➔ DeviceRollout(独自)
7.	多様な装置に対するドライバープラグイン	➔ k8sカスタムオペレータ(独自)
8.	デバイスシークレット	➔ External Secrets Operator

この4つはすべて
k8sカスタムオペレータ

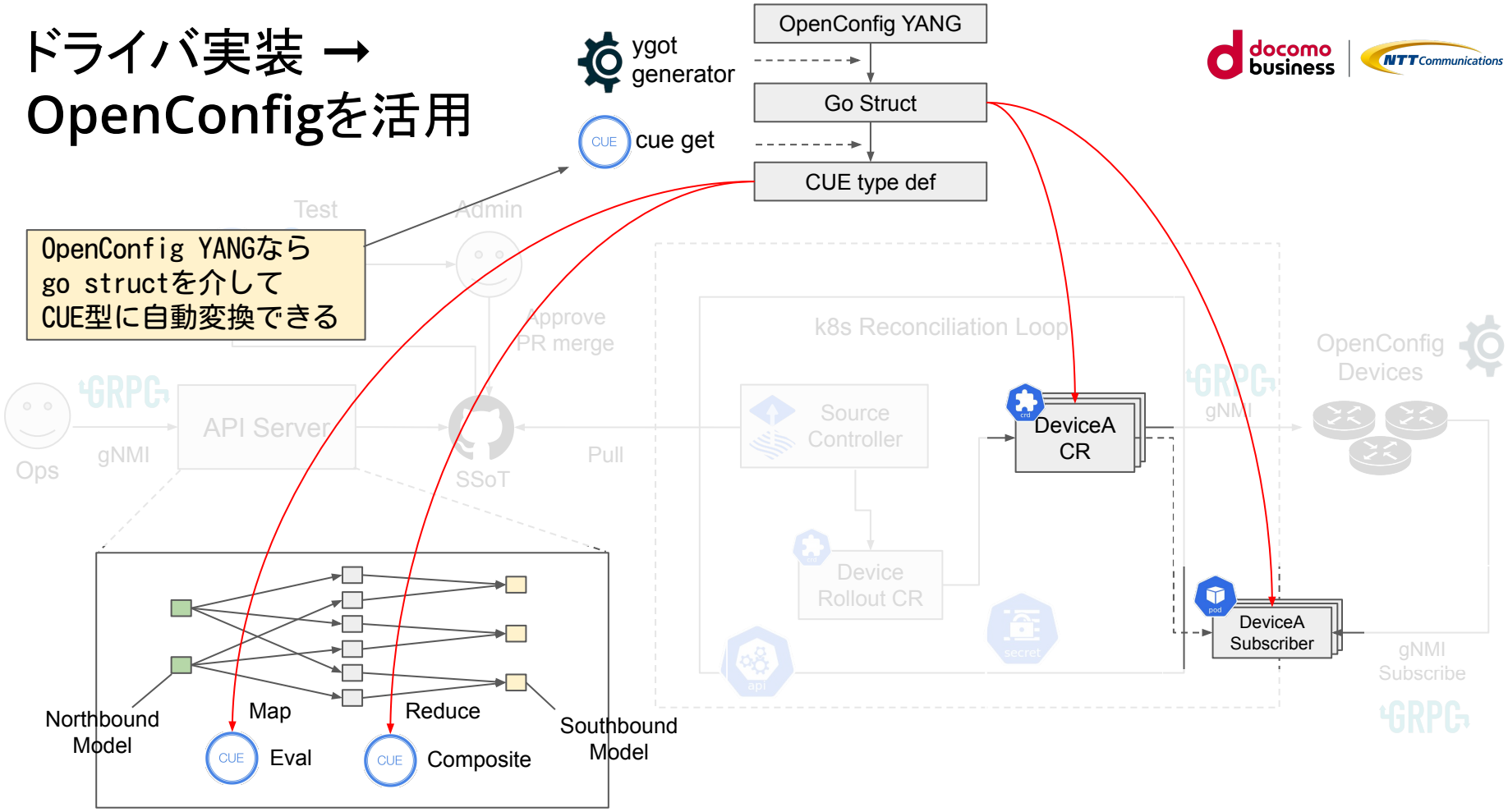
kuestaのアーキテクチャ



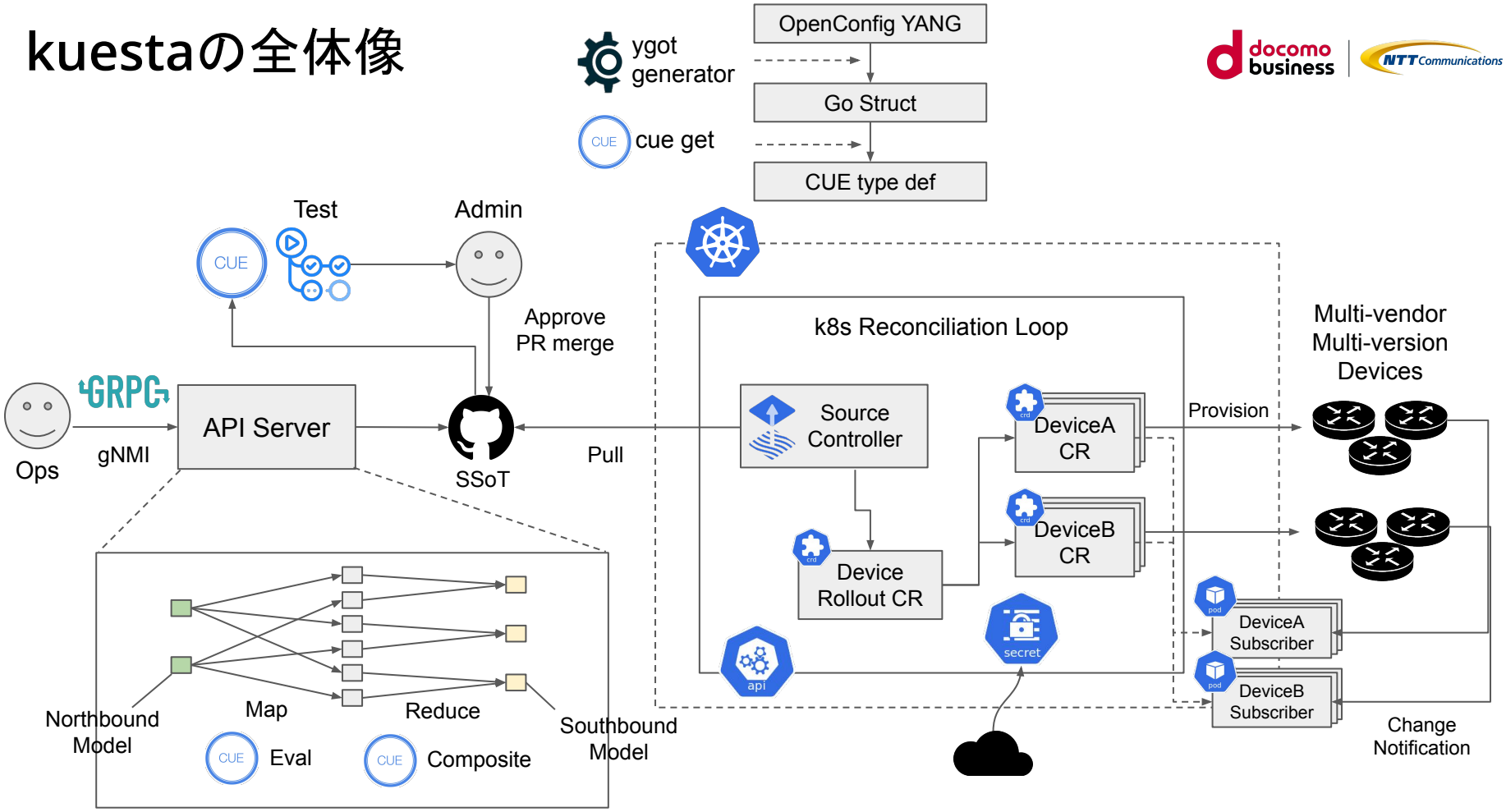
ドライバ実装はどうする？



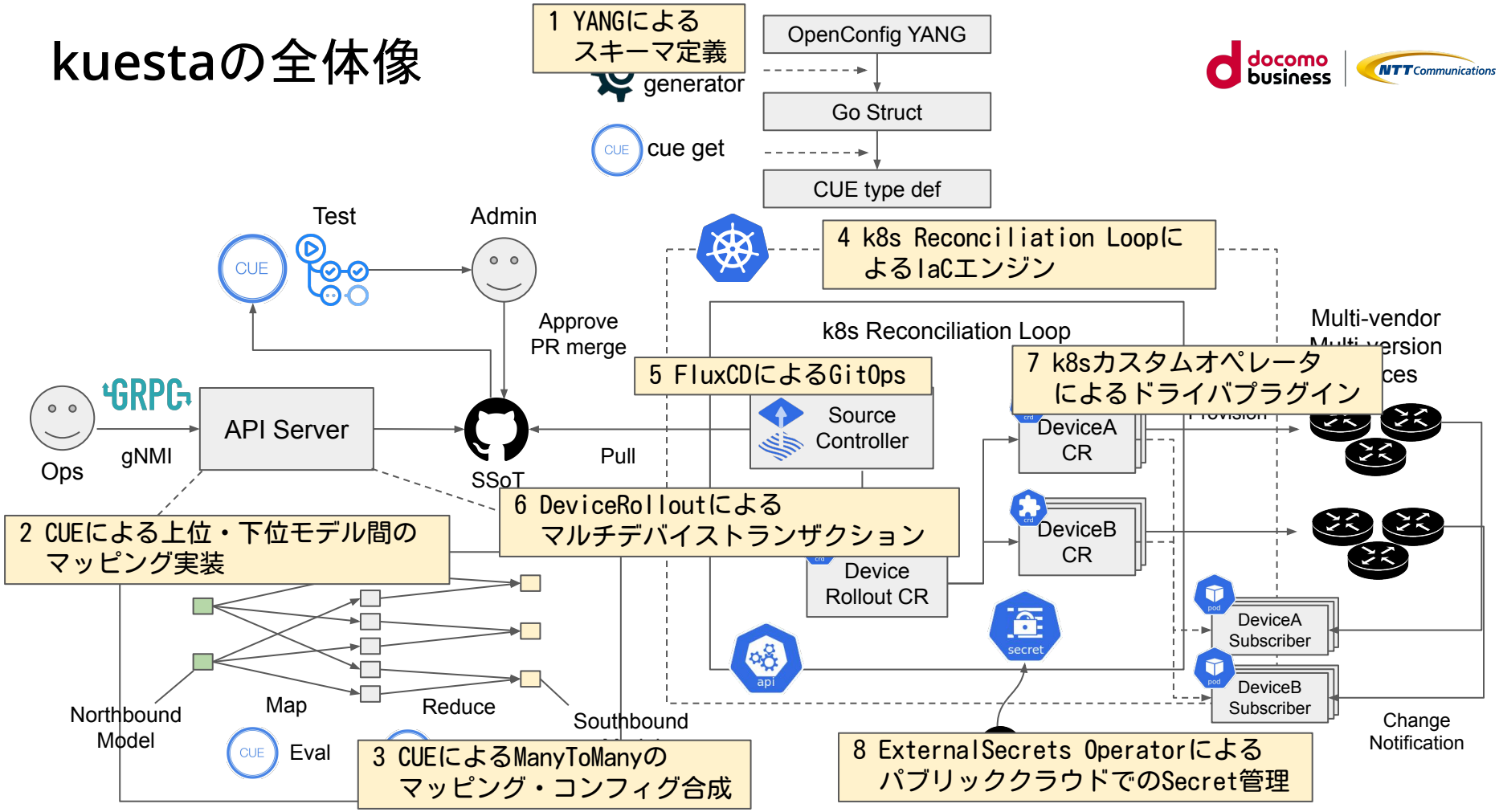
ドライバ実装 → OpenConfigを活用



kuestaの全体像



kuestaの全体像



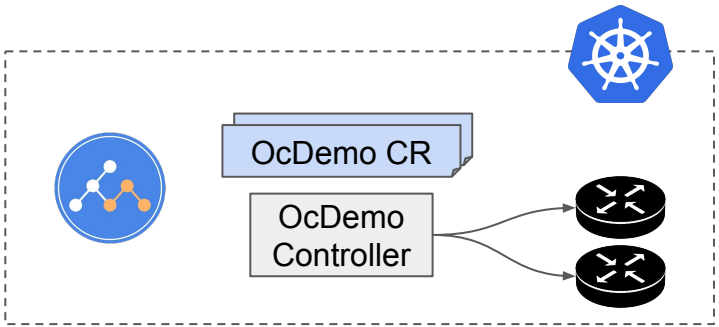
デモ

デモ

- kuestaのドキュメントサイトのgetting-startedをやります
  - https://nttcom.github.io/kuesta-website/docs/getting-started/

- デモの流れ
  - kindを用いてローカルにk8sクラスタを立ち上げる
  - kuestaをinstall
  - サンプルのサービスを作って、GitOpsできていることを確認

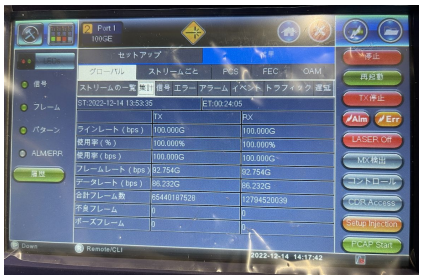
- デモの構成
  - ローカルのk8sクラスタ と kuesta一式
  - OpenConfig/gNMI対応のエミュレータ2台
  - エミュレータに対応したドライバ
    （Kuberenetes カスタムオペレータ）

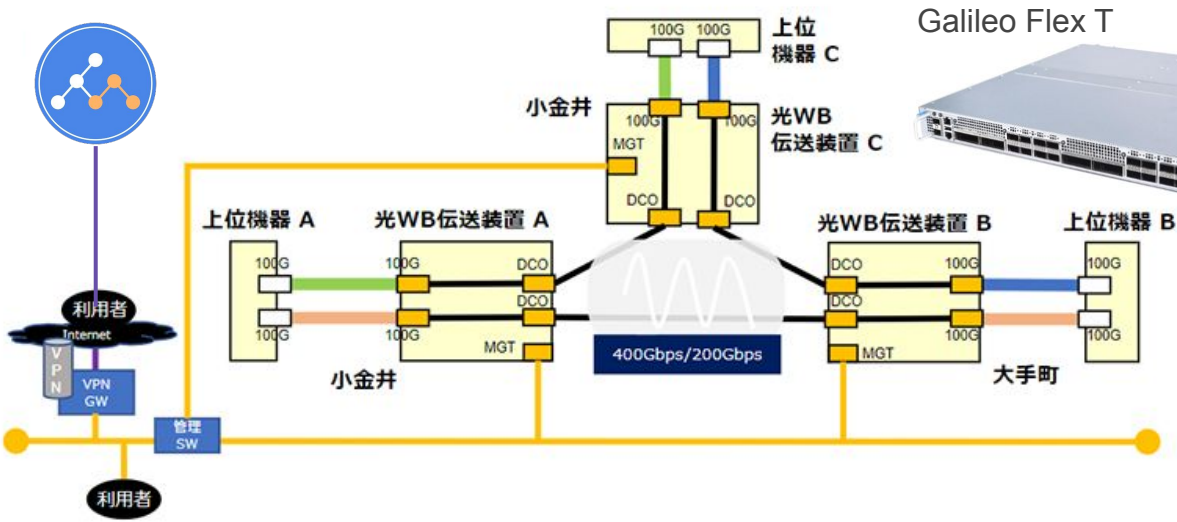


検証結果と課題

実機との接続検証

- 対象装置：伝送ホワイトボックス（Galileo Flex T）
- サポートしているOpenConfig YANGからカスタムオペレータを作成し、kuestaから GitOpsすることで、ライン側を導通しクライアント間を疎通させることに成功[1]





*1: 本研究は、国立研究開発法人　情報通信研究機構（NICT）が運用する NICT 総合テストベッド「B5G高信頼仮想化環境」を用いて行われました。

これまでの成果

- クラウドネイティブなアプリ開発で用いられているIaC・GitOpsのプラクティスを
  ネットワークIaCに持ち込んで、最低限ではあるがE2Eでの動作検証ができた
  - OpenConfig/gNMIな装置であれば、ドライバを作って実機接続検証できた

- 方式の妥当性・有用性を検証し、ある程度の確信を得た

- 一方で、課題も
  - Reconciliation Loopの外（CUEでのマッピング）までユースケース実装を追い出しているので、ネットワークの状態を観測しながら管理をすることが難しい
    - カスタムオペレータに特定ユースケースを実装すれば可能だが、レイヤ分離構造がおかしくなる
    - コンフィグだけに特化する案もあるが、システム全体の複雑さの低減につながらない
  - 素直に、特定のユースケース・デバイス向けのカスタムオペレータを作るほうが、
    実装は簡単だし責務の凝集もできる

ここからが大変

- まだPoCの域を出てない
- 本格的に使えるようにするには、必要なものがたくさん
  - OpenConfig/gNMI以外のプロトコルのサポート
  - ドライバを作りやすくするための Plugin Frameworkの作り込み
  - OpenConfig以外のYANGのCUE変換
  - Pre/Post-check、監視、その他EMS/NMSとしての機能（自装 or 連携）

Kubebuilderでは
不十分で、専用のものが必要

- 1～2人では到底開発できない
  - コミュニティで開発したい → OSS化してみた
  - 装置のドライバに関しては、ベンダにコントリビュートしてもらえるとありがたい
    - そのためには、ビジネス インセンティブが必要。。。

議論: お訊きしたいこと

- この取組、どう思いますか？
  - 👍ネットワークモデル非依存で任意の抽象モデル化して、 IaC・GitOpsできるところに新規性はある。コンフィグが複雑なサービスを開発するときには有用なはず
  - 👎ただし、この状態に持っていくまでのハードルが高い（ドライバ開発など）
  - 👎監視や、装置状態を観測しながら連携するのが難しい

- 「ネットワークモデルとベンダ非依存での任意コンフィグ抽象化」は欲しいですか？
  - このユースケースで成功している某プロダクトがあるので、ニーズはある認識
    - ツールセットが充実してきたら、楽にシステム開発できるようになる（はず）
  - 伝送のシステム開発やってるとかなり欲しいのですが、みなさんどうですか？

- ニーズがあるとして、作りたいと思いますか？
  - CUEとKubernetesカスタムオペレータに可能性を感じたので、PoC作ってみました
  - 「動くものファースト」で Ansibleで良くね？という考え方も全然あると思ってます

まとめ

- CUEとKubernetesカスタムオペレータを用いて、ネットワークaCを行うkuestaというOSSを作りました
  - https://github.com/nttcom/kuesta/
- すぐに試せるので、getting-startedをやってみてください！
  - 面白いと思ったら、ぜひ Starを付けてください
  - IssueやDiscussionに思ったことをコメントください
- この取組の是非について、忌憚ないご意見をください
  - 気に入った！私もやるぜ！ という方は、ぜひお声がけください 🙇🙇🙇