Challenging Tomorrow's Changes

IoT-GWプロビシステムの裏側

伊藤忠テクノソリューションズ

渡部/栂崎

アウトライン

- 本日の議題とテーマ
- 自己紹介
- 自動化対象のサービス概要
- 現行サービスの課題と解決
- 今回開発した自動化システムについて
- 完成までの道のりと苦労したポイント
- 議論したいこと

本日の議題

- IoT-GWクラウドサービスのルーターのデリバリーの仕組みを自動化した経験から、設計から製造および試験での技術的な苦難を共有させていただきます
- 現在自動化と格闘している方々、またはこれから自動化に取り組む方々と議論をさせていただければ幸いです

登壇者自己紹介

- 渡部友也（わたなべ　ともや）

  2010年から現職にてサーバー/NW/新規商材発掘等を担当

  現在は通信キャリア様向けの
  NWおよびNW自動化/運用高度化開発を主務

  自動化できない作業はほぼないと思っています。
  今回参加させていただいた上で、次に繋がる議論ができたら嬉しいです

  ※写真はCisco NSO Developer Days in Swedenです

- 栂崎彩花（とがさき　あやか）

  2016年-2018年 通信キャリア様向け　VoIP/NWを担当

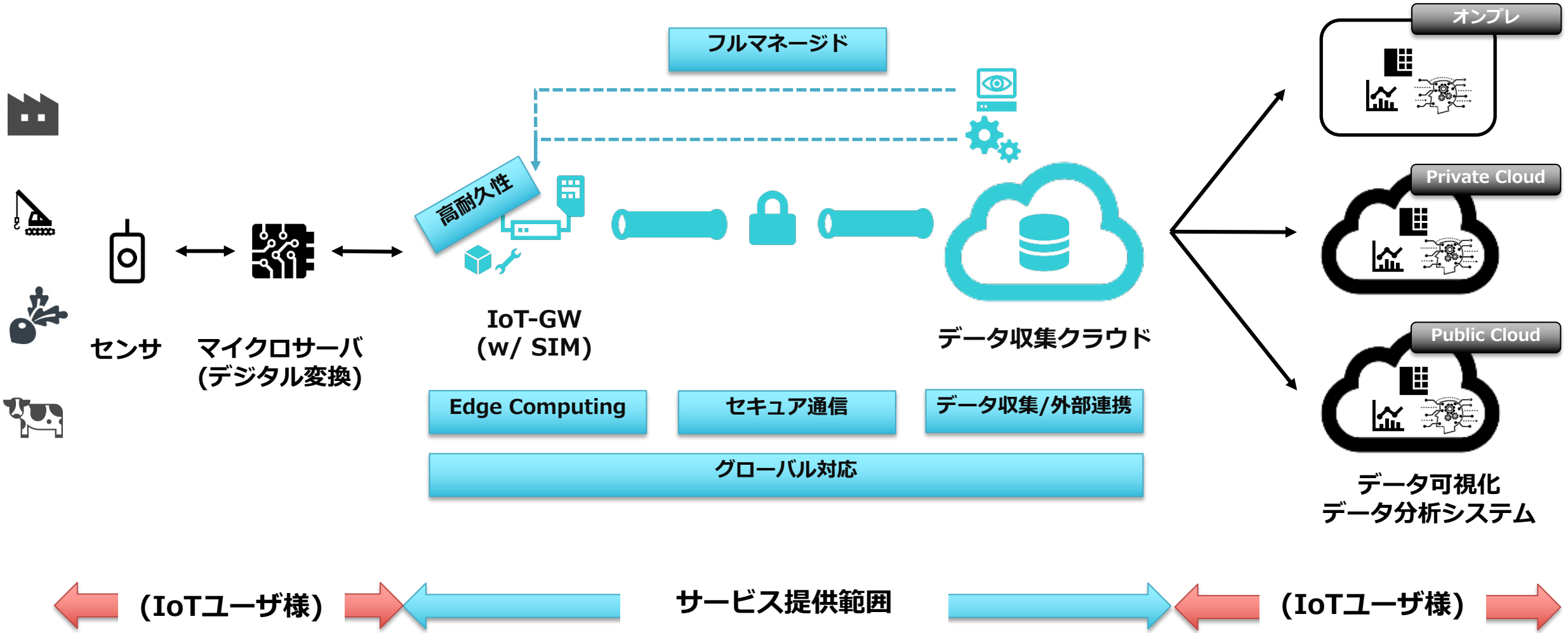
  2018年-現在　　同上　NWおよびNW自動化/運用高度化開発を主務

  - 当たり前になってしまっている既存のめんどくさい作業は
    なるべく自動化していきたいなと考えてます。
  - 今回の発表で苦労した点や同じ課題をもっている方達と
    さまざまな議論ができたら嬉しいです。

サービス概要

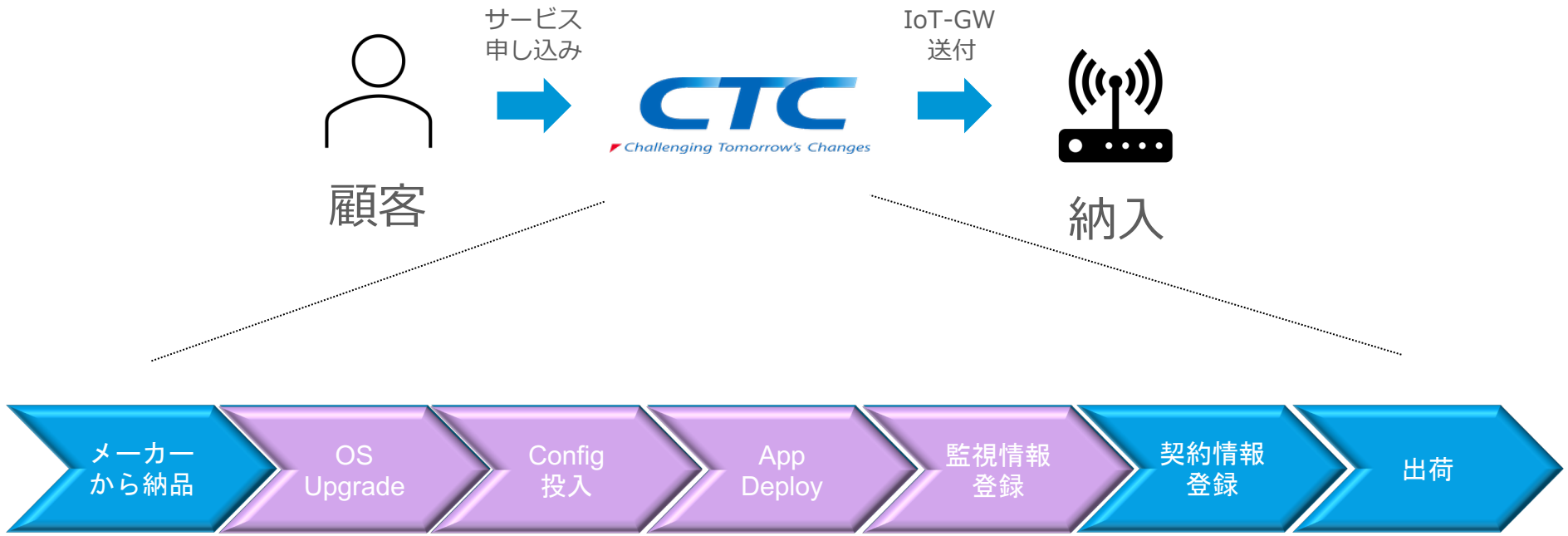
お客様は届いたルータの**電源をON**するだけで「簡単に」「安全に」IoT**データ収集可能**

お客様は「どんなデータを」取得し、その「データをどう活用したいか」の検討/開発に専念



IoT-GWデリバリーの流れ

- サービスの申し込みからIoT-GWの納入までのながれ



- 納入までの作業はそれなりに多い
- 顧客ごとの情報等も考慮する必要がある
- 保守交換も考える必要がある

現行サービスの現状課題

- サービスは既にローンチされていたため、自動化の仕組みは既存で存在していた。
  しかし、課題が大きく２つ存在していた

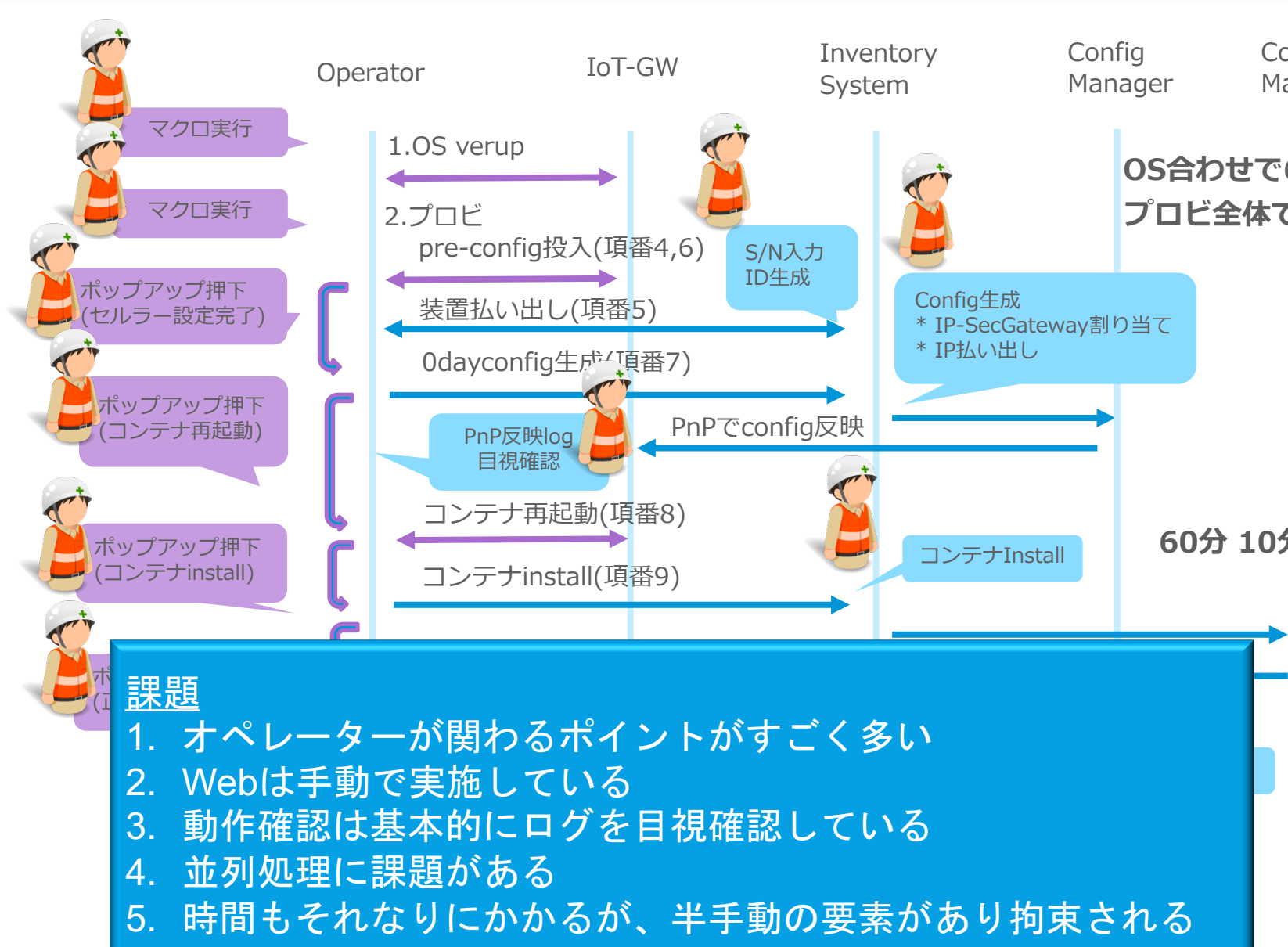
作成したマクロの機能不足

既存はTeratermマクロを利用してOS upgrade/Config投入を実施
作業が成功/失敗は基本的にマクロ終了時に人間が判断
最後の判断は人間になるため、ミスを誘発する可能性もあり

Web部分は手動作業

契約情報/監視情報はwebシステムとなるため、手動で実施していた
数十台を一度に作業するため、オペレーターへの負担が大きい

既存フローと課題（実際の資料）



課題

1. オペレーターが関わるポイントがすごく多い
2. Webは手動で実施している
3. 動作確認は基本的にログを目視確認している
4. 並列処理に課題がある
5. 時間もそれなりにかかるが、半手動の要素があり拘束される

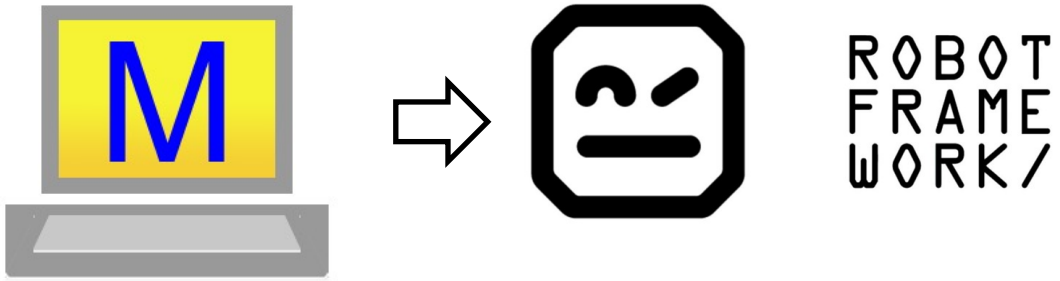
1台あたりのオペレーション × 台数分

吹き出しの数分入力の手間がかかる

マクロは設定で10台分走らせられても
コンテナは3台ずつとかとにかく手間
工程の途中にシステム入力があるのも手間

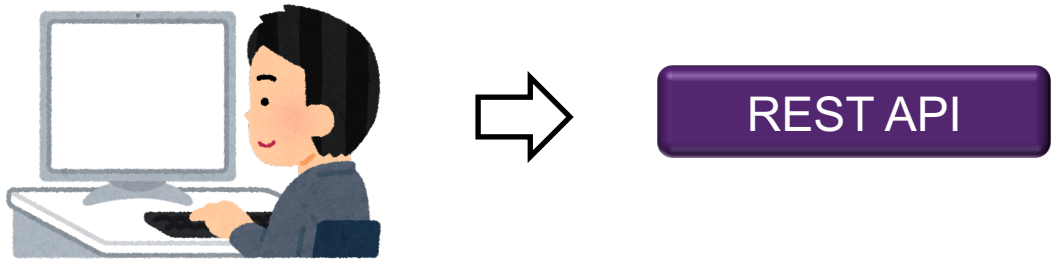
現状の課題を解決するために

- TeratermマクロのRobot Framework化



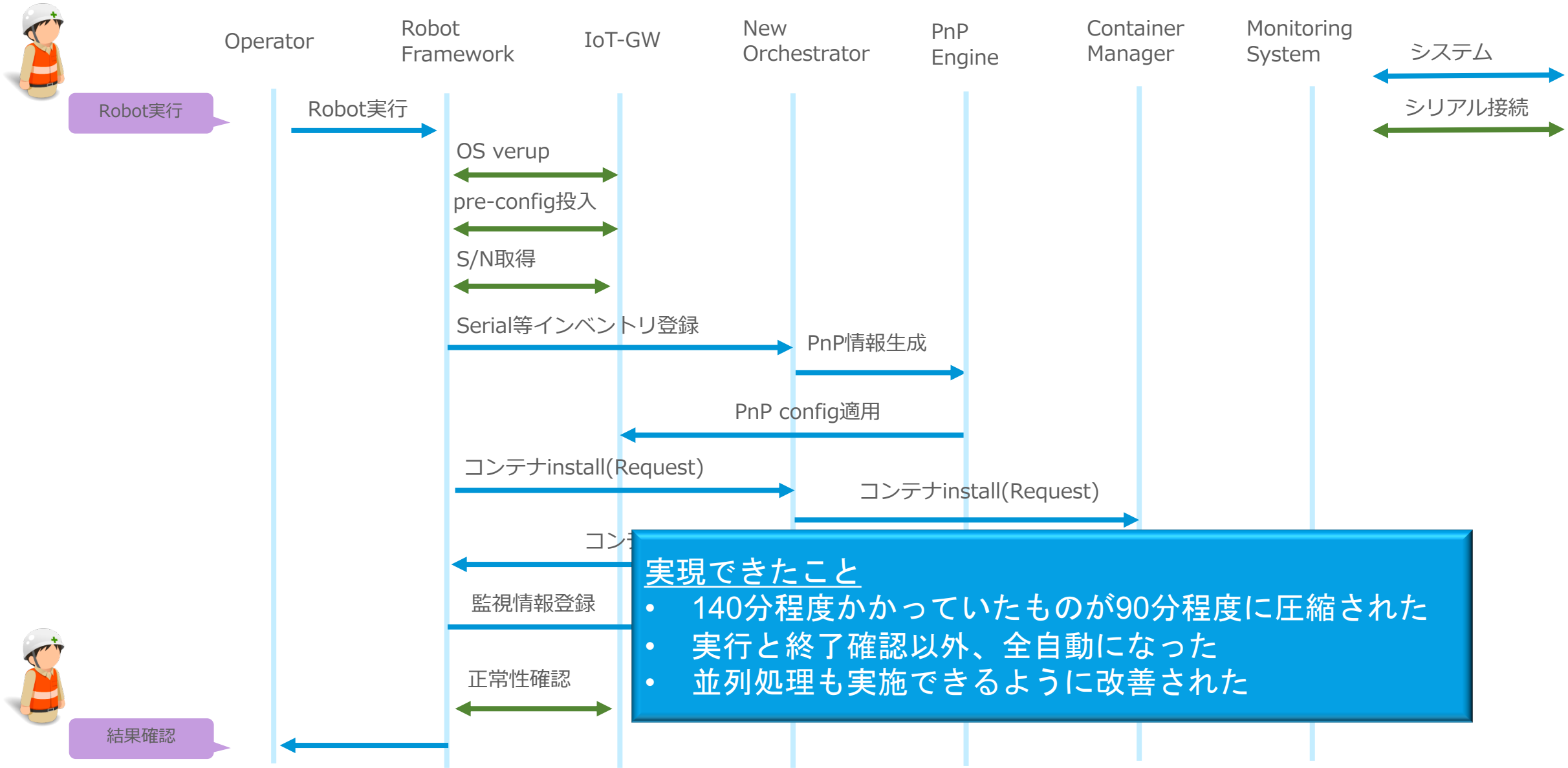
- メリット
  - ✓ 試験自動化Frameworkなので Assertion(合否判定が自動で可能)
  - ✓ 実行結果がhtmlのレポートとして自動生成
  - ✓ PythonをDriverとして利用できるので、色々と小回りが効くので並行処理とかもうまくやれば可能

- Web系SysとのREST API結合による完全自動化



- メリット
  - ✓ 手動Web操作の排除による効率化
  - ✓ etc…

課題解決後のシーケンスのコンセプト

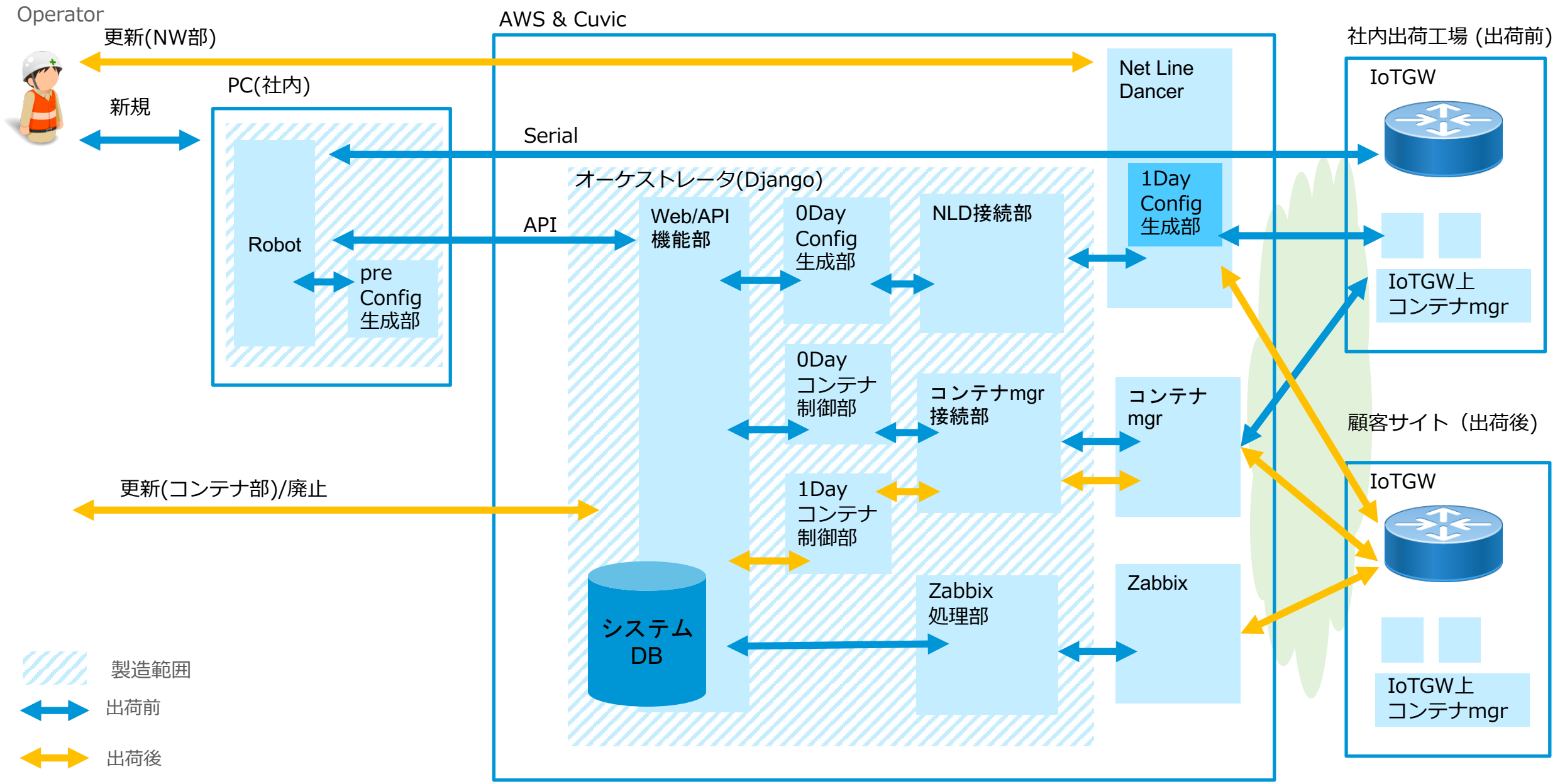


実現できたこと

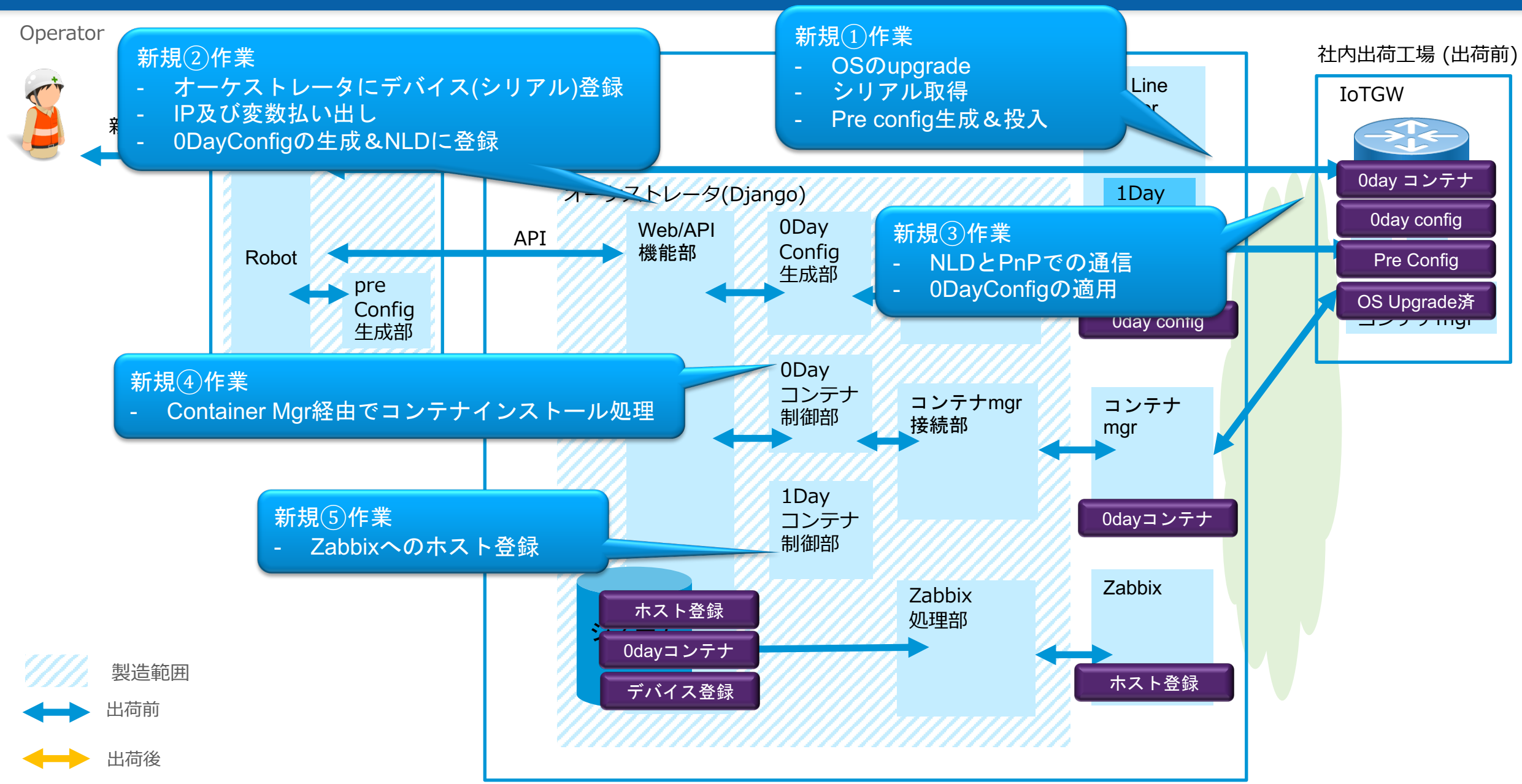
- 140分程度かかっていたものが90分程度に圧縮された
- 実行と終了確認以外、全自動になった
- 並列処理も実施できるように改善された

ここからは実際の実装の話と苦労の話となるので、開発者の栂崎にスピーカーチェンジします

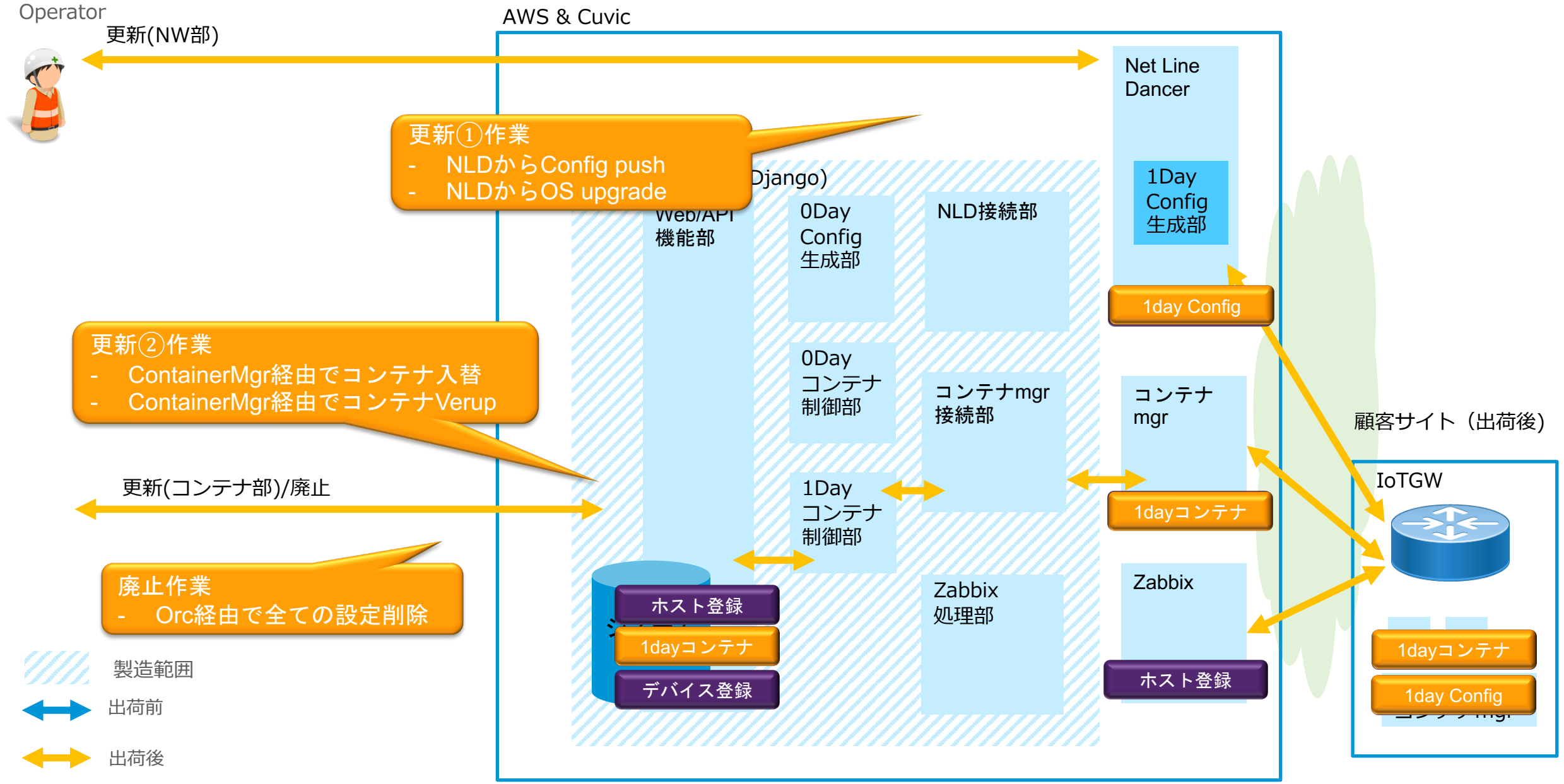
自動化システム構成



シーケンス（出荷前 新規）



シーケンス（出荷後 更新/廃止）



完成までの道のりと苦労ポイント

- 完成までの道のり

設計 ~ 環境構築 ~ 製造 ~ 自動試験製造 ~ 総合試験(手動) ~ doc類整備 ~ …

- 苦労ポイント

設計

- 設計がFixしない

製造

- CI/CD
- APIの冪等性

試験

- Serialポートの罠
- NW/自動化チームの連携
- 環境差分に悶絶

その後

- 引き継ぎ
  運用者、設計者共に…

苦労した点：設計フェーズ

- 設計がFixしない

なぜなのか

苦労した点：設計フェーズ

- ステークホルダーからの要望

	Operator	設計者(既存課題)	設計者(新規要望)
課題	装置ごとに払い出したInventory情報をwebに手入力するのが手間(装置シリアルとIPの紐付けとか)	現行システムの運用コストがかかりすぎる	Inventry情報を自動で払い出して欲しい (IP,シリアル情報の自動紐付け)
	キッティング中目視確認するタイミングが常にあるので離席できない(半自動)	納入後のconfig,コンテナの変更 (verup等も含む)ができない	他システムとの連携 (監視システムや契約管理システム)
	複数台の同時作業が手間	-	将来的に機能拡張性が欲しい

そもそもの要望ははっきりしていた
基本設計は簡単に固まった。
しかし、詳細設計Fixは難航した。

苦労した点：設計フェーズ

- NW詳細設計のFixが遅れた
  - ✓ NWの設計が固まらなかった(pre config/0day config/1day config)
  - ✓ configのfixが試験直前までできなかったため、試験中に自動化側で改修が必要だった

- 商用/検証環境のインフラ構成（AWS)が流動的だった
  - ✓ NWチームと自動化チームで役割分担が曖昧だった
  - ✓ 環境の設計fixが試験直前までできなかったため、
    オーケストレータを動作させるnginx等のconfigが急務になった

- 自分のスキル&経験不足
  - ✓ 開発面(DB Model設計) 、インフラ面の知識
  - ✓ 商用/検証/開発環境をfixさせられなかった

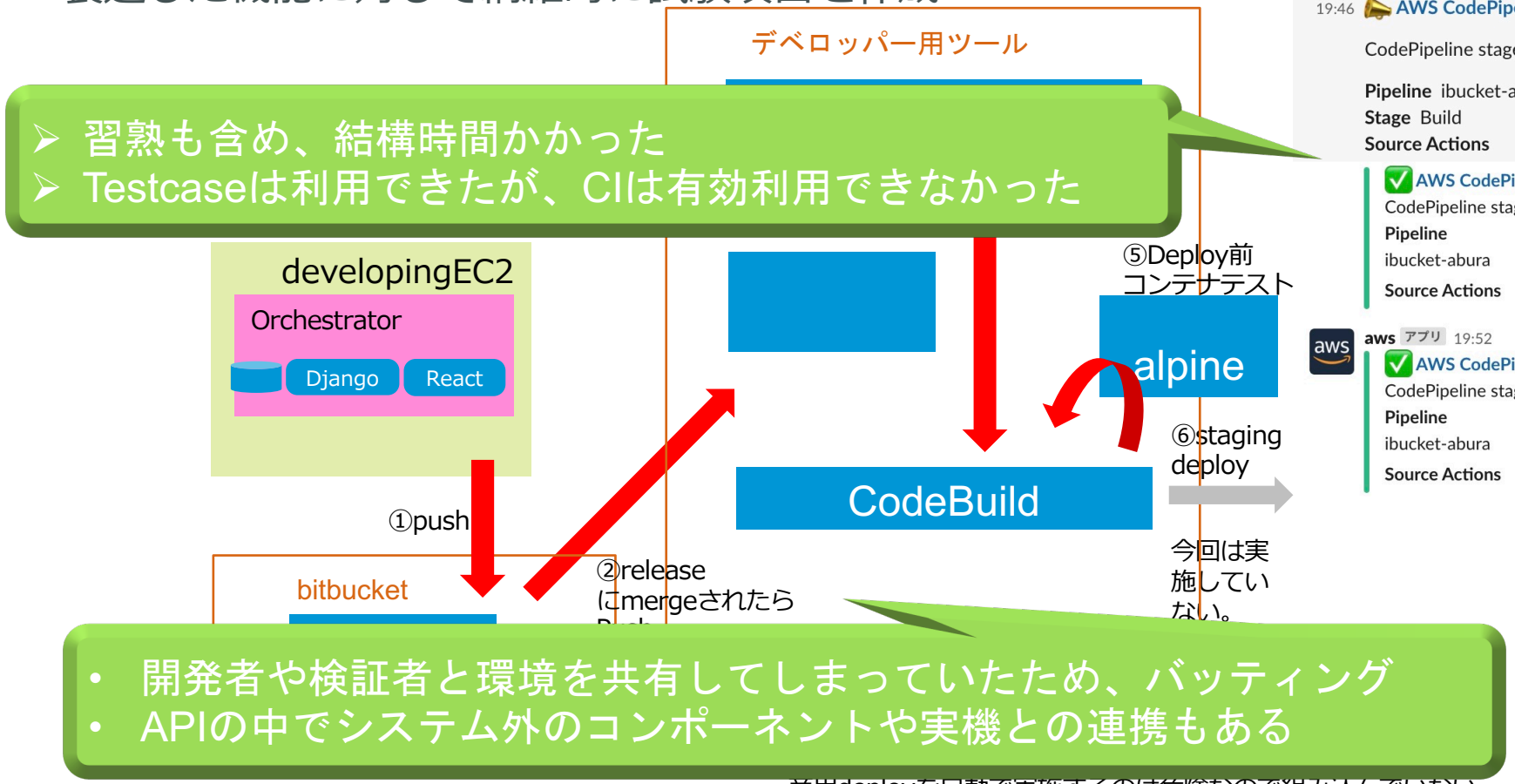
苦労した点：製造フェーズ

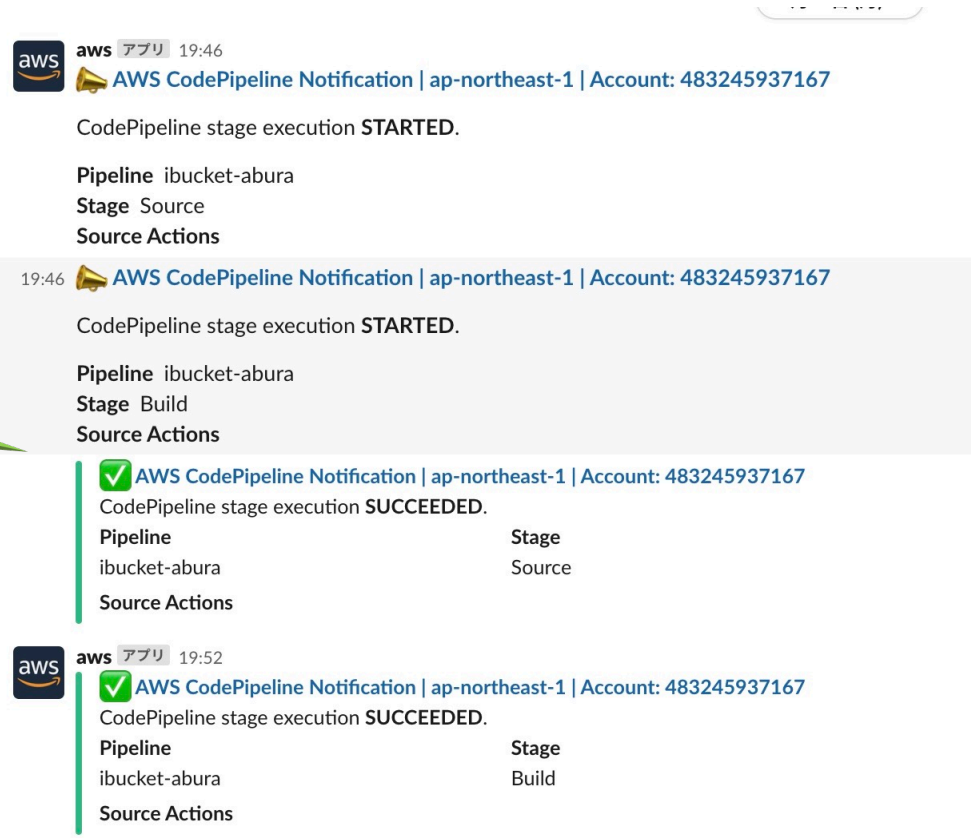
- 製造中から試験を考えてCIを実装していた
- APIの冪等性を考えて実装が必要だった

実装において幅広い技術知識が必要だった

苦労した点：製造フェーズ

- 製造中から試験を考えてCIを実装していた
  - ✓ オーケストレータのAPI製造部分をCIで自動試験
  - ✓ 自動試験の実装は、DjangoのTestcaseを使用
  - ✓ 製造した機能に対して網羅的に試験項目を作成





- ➢ 習熟も含め、結構時間かかった
- ➢ Testcaseは利用できたが、CIは有効利用できなかった

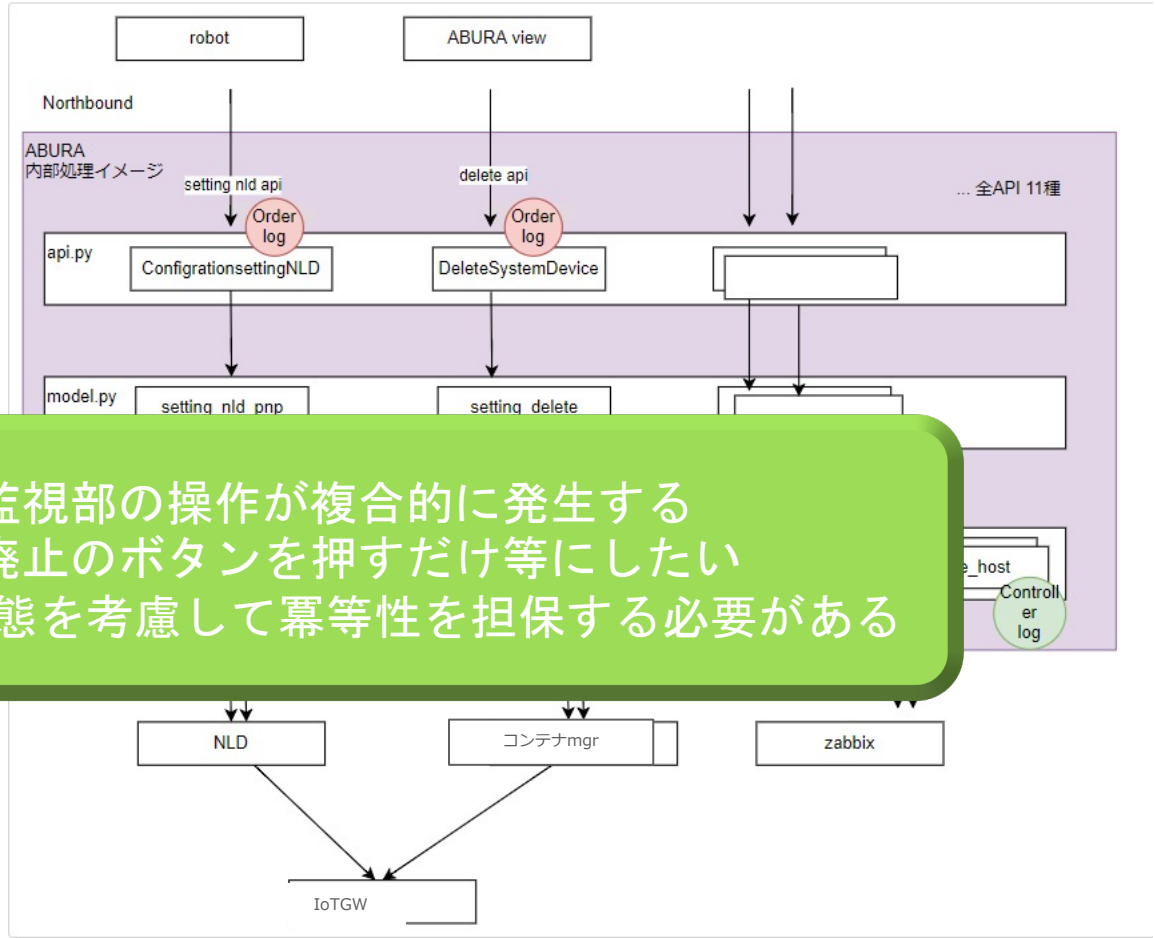
- 開発者や検証者と環境を共有してしまっていたため、バッティング
- APIの中でシステム外のコンポーネントや実機との連携もある

商用deployを自動で実施するのは危険なので組み込んでいない

苦労した点：製造フェーズ

- APIの冪等性を考えて実装が必要だった
  - ✓ 運用者向けに、シンプルなオペレーションにしたかった
  - ✓ 外的要因での失敗時に、切り分けを不要にしたい

- Provisioning実施/更新/廃止の３つのボタン
  - ✓ 実施の途中でどこで失敗しても、
    廃止を押すだけで最初の状態に戻る設計



- 各APIごとにNW部、コンテナ部、監視部の操作が複合的に発生する
- 運用を考えると、どこでfailしても廃止のボタンを押すだけ等にしたい
- 実装上は外部システムAPI仕様や状態を考慮して冪等性を担保する必要がある

苦労した点：試験フェーズ

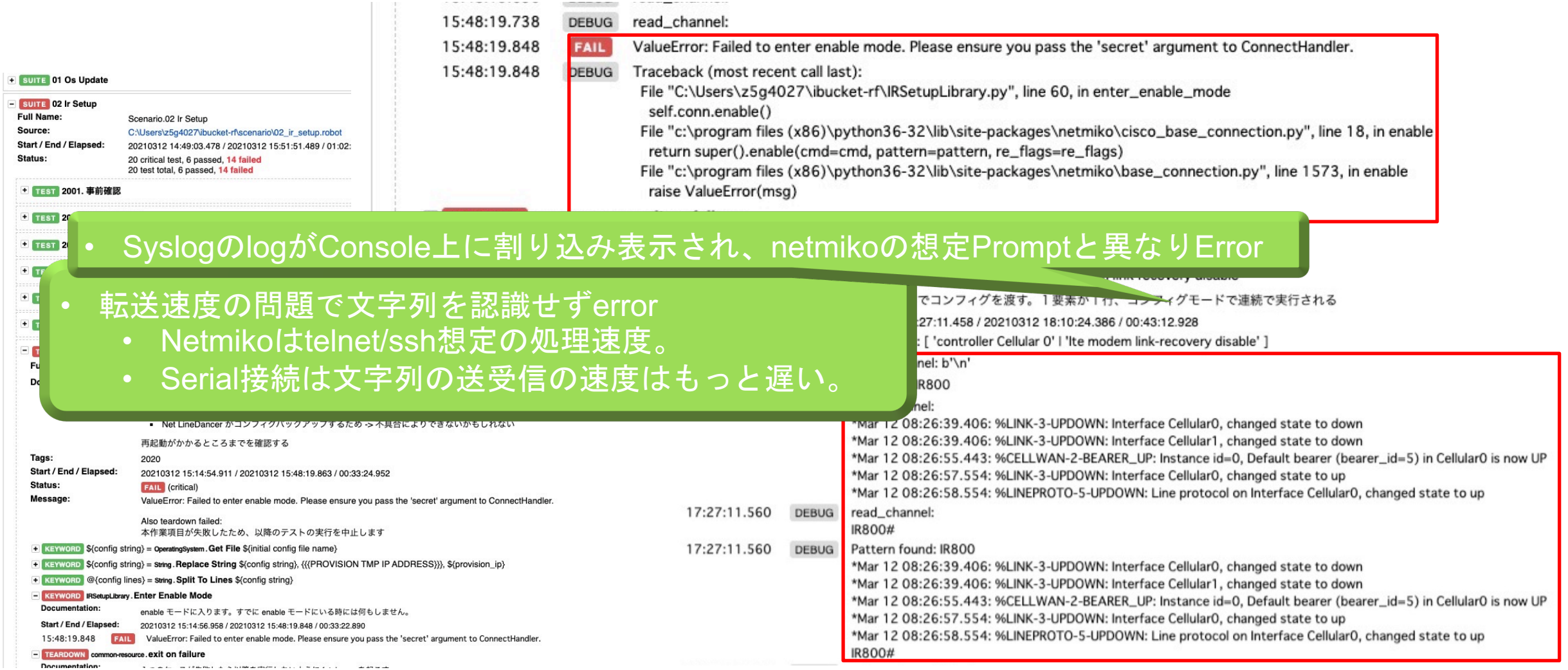
- Serialポートの罠で総合試験1000本ノックになった
- NWチームと自動化チームの連携がうまくいかず、試験実施が長引いた
- 環境差分に悶絶した

試験はやっぱり大変

苦労した点：試験フェーズ

- Serialポートの罠で総合試験1000本ノックになった

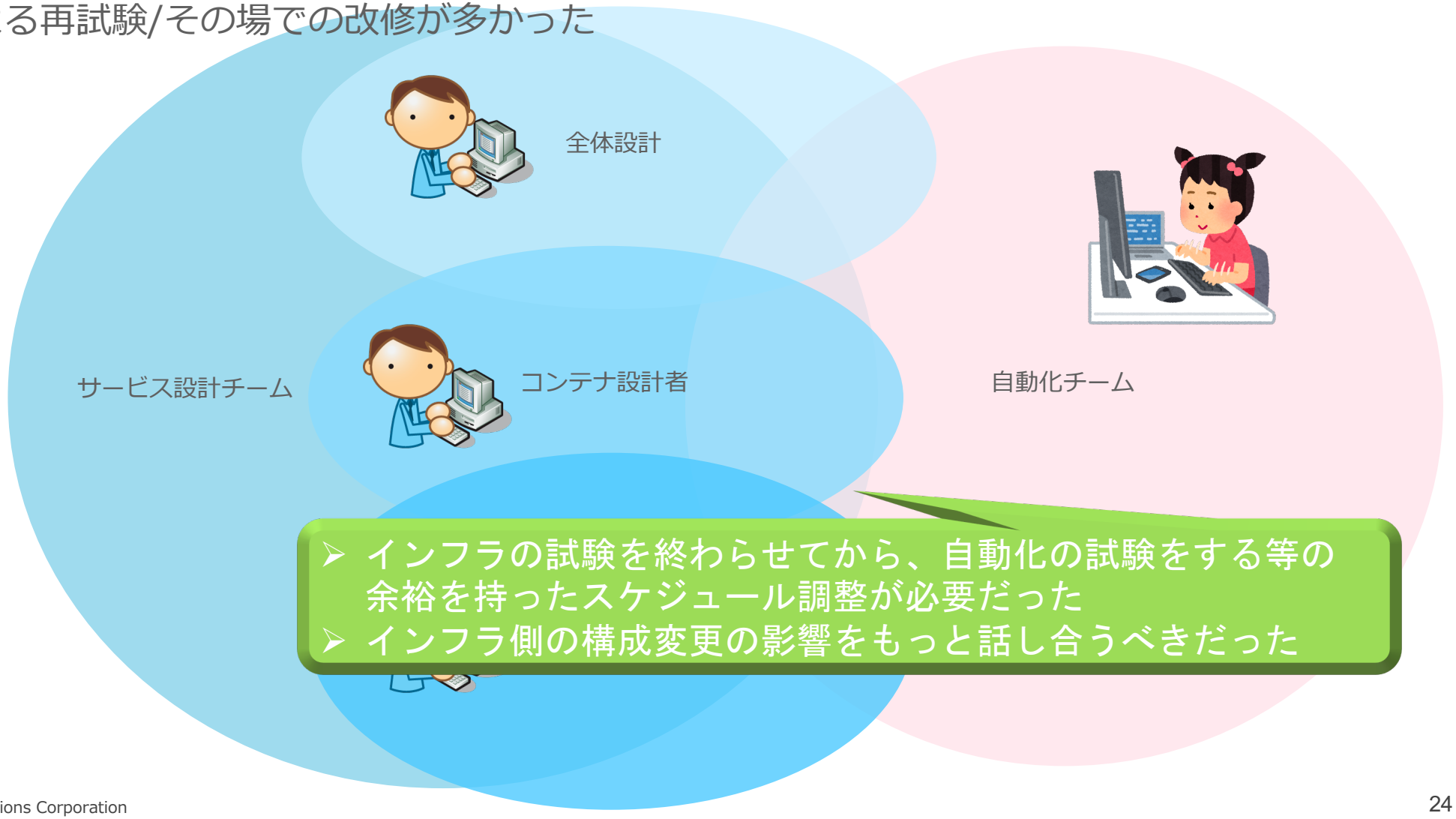
Serial接続で装置仕様の制約があった



- SyslogのlogがConsole上に割り込み表示され、netmikoの想定Promptと異なりError
- 転送速度の問題で文字列を認識せずerror
  - Netmikoはtelnet/ssh想定の処理速度。
  - Serial接続は文字列の送受信の速度はもっと遅い。

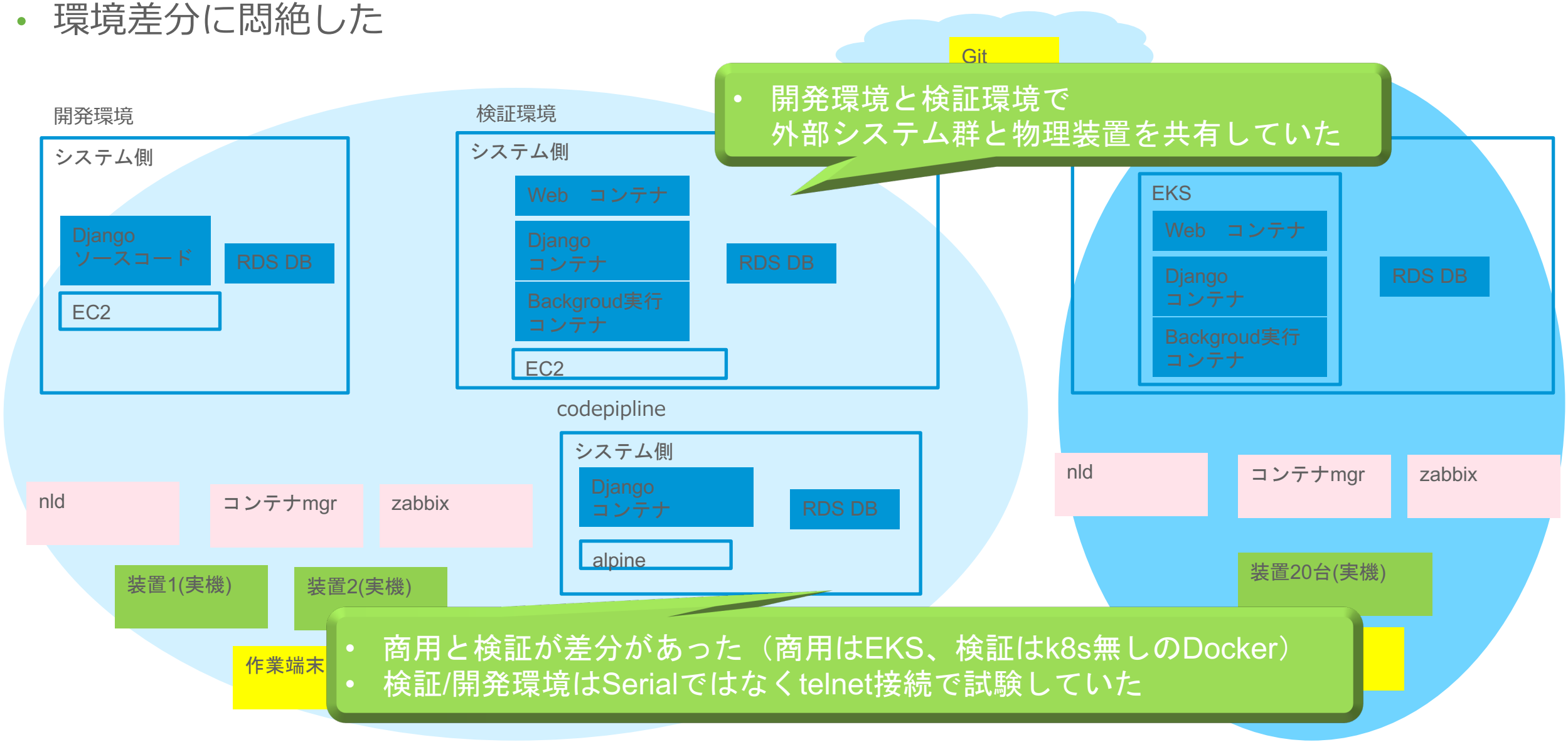
苦労した点：試験フェーズ

- NWチームと自動化チームの連携がうまくいかず、自動化システム試験が難航した
  - 自動化チームの試験実施のタイミングで、インフラ要因で試験が失敗して進まない事象が多かった
  - 環境構成変更による再試験/その場での改修が多かった



苦労した点：試験フェーズ

- 環境差分に悶絶した



苦労した点：その後(これから…)

- 引き継ぎ
  運用者、設計者共に…

- 製造者と運用していくメンバーが異なる
- Docで伝わりきるのかどうか

セットアップ(キッティング)手順書

ABURA、NLD、Fogd、Zabbixの各コンポーネントでデータが登録されているか確認する。

- ABURA

1. `Device一覧`を開く
2. 作成したデバイス(作業対象のIR829のS/N)があることを確認。成功した場合は赤枠
   ※この際に青枠内に記載してある、IR829のS/Nに紐づく「デバイスip」を控えてお

- セットアップ(キッティング)手順書
  - 前提条件
  - 物理的事前準備
    - 1.システム障害状況確認連絡

Device一覧

検索する

	id	シリアル番号	デバイスip	order_status	pro
☐	280	FGL2351LFLF	172.30.5.9	運用中	nor
☐	279	FGL2351LFMW	172.30.5.8	運用中	nor
☐	278	FGL2351LFLQ	172.30.5.7	運用中	nor

ABURA API

Updated by Hir
2021/04/19 19:1

▷ View　編集　プレゼンテーション　更新履歴

ABURA API

API一覧

APIは全部で11種類。
プロビ実行時のapiだけでなく、画面からの更新、削除の際も全て内部的にはAPIを実行している。

api実行時のurlやbody、内容についてはソースコード側のabura_api.restをmasterにして管理しているため割愛。
詳細は、下記URLを参照してください。
https://bitbucket.org/ctc-repo/ibucket-abura/src/af814d7e9fa67a2ee104e0a8cb22da041f10e85f/api_rest/abura_api.rest?at=develop

APIエラー一覧

エラー一覧

全部で下記5種類

- AburaSystemError
- ZabbixAPIError

- ABURA API
  - API一覧
  - APIエラー一覧
    - エラー一覧
    - 各エラー詳細
    - 参考ソースコード
  - APIテストケース
    - テスト項目作成観点
    - テストケース一覧
    - テストケース回し方

GROWI 4.1.7

議論したいこと

- NWチームと自動化チームの連携方法
  - 役割分担
  - サービス設計者へシステム運用改修の引き継ぎなど
  - そもそもチームが分かれてない？等

- 開発、検証、商用環境
  - 環境差分をどこまで許容しているか
  - CI/CDどこまでやるのがベストプラクティスか