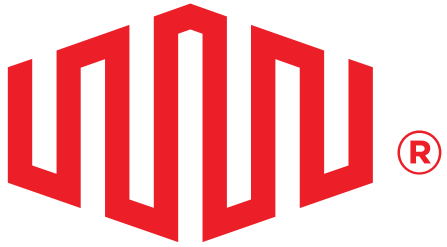
IaC文化を広く根付かせる



EQUINIX

丸野　達矢
(エクイニクス・ジャパン株式会社)

自己紹介

名前：丸野　達矢　(MARUNO Tatsuya)
出身：熊本
趣味：バイク、キャンプ、スキー、写真

JANOGとの関わり：

スタッフ：　JANOG42 / 43 / 45
登壇：
[JANOG36] **お絵かきのお話 ～NW構成図ってどんな感じで書いてます？～**
→ **[**Qiita] ネットワーク図の書き方(PowerPoint編)
[JANOG43] **パブリッククラウド3社のプライベート接続サービス比較**
[JANOG45] **JANOGerとクラウドのワクワクする関係**

業務経歴

- 2013年～2017年：
  → L2-L3メインのネットワーク設計/構築/運用 （DCラック内のマネージドサービス）

- 2018年後半～現在：
  → クラウド（主にAWS）を用いた設計/構築
    ・お客様サービスの基盤作り
    ・業務効率化/自動化の取り組み

2020年2月のJANOG45「JANOGerとクラウドのワクワクする関係」
“ネットワークエンジニアの自分がクラウドインフラとどう向き合ってきたか”

本題

本発表のテーマ

「IaC(Infrastructure as Code)文化を広く根付かせる」

- チームで始めたいとは思ってるけど、進め方がイメージできない
- 実際に取り組みを始めたけど、なかなか業務フローに落とし込めてない
- 一部の担当者のモチベーションに依存してる（救われてる）状態

などで悩んでる人がまだまだ多くいるのではないか。

あらためて、
IaCの文化をチーム全体で定着させるためにはどうすればいいかという
“最初の文化作り“にフォーカスして
議論＆共有できる場になればと思います。

時間配分

- 自身の取り組み紹介（15分ほど）
- IaC文化を広く根付かせるために必要なことを考える（15分ほど）
- 議論質疑（15分ほど）

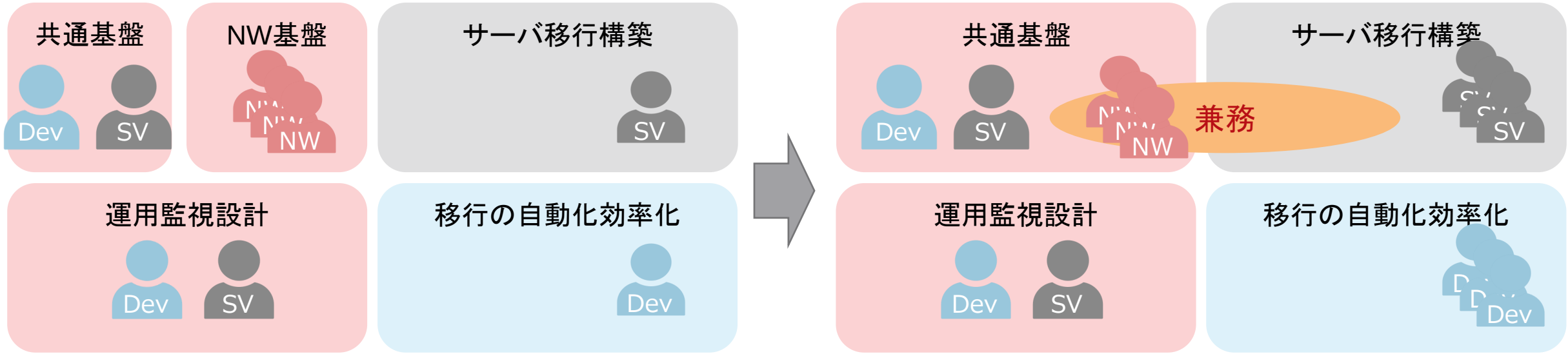
計45分（14:45 ~ 15:00まで）

※　時間が余れば、付録の内容説明

IaCの文化作りに関わる背景

当時のチームアサイン図

※人数は実際と異なる（Dev：開発、SV：サーバ、NW：ネットワーク）



【初期】

AWS＋物理基盤でのハイブリット環境を設計構築

NWメンバーはNW基盤設計対応

※サーバ移行を走らせながら自動化効率化のツールを並行で増やしていく状況、エクセル設計書＋GUIでの構築も数多く実施してる

【中期】

ある程度設計や基盤作りが固まったところで、本格的に始まる移行構築に向けて、兼務対応する形にシフト

そのときの移行構築 / 自動化効率化のチーム課題

■課題
- ツールの使い方がうまく伝わらない、流れがいまいちわからない
- ツールの利便性が認知されてない
- 課題解決のお願いをしたけど、依頼側は依頼しっぱなし、開発側は課題の優先度が分かりづらい
- お互いの距離がある

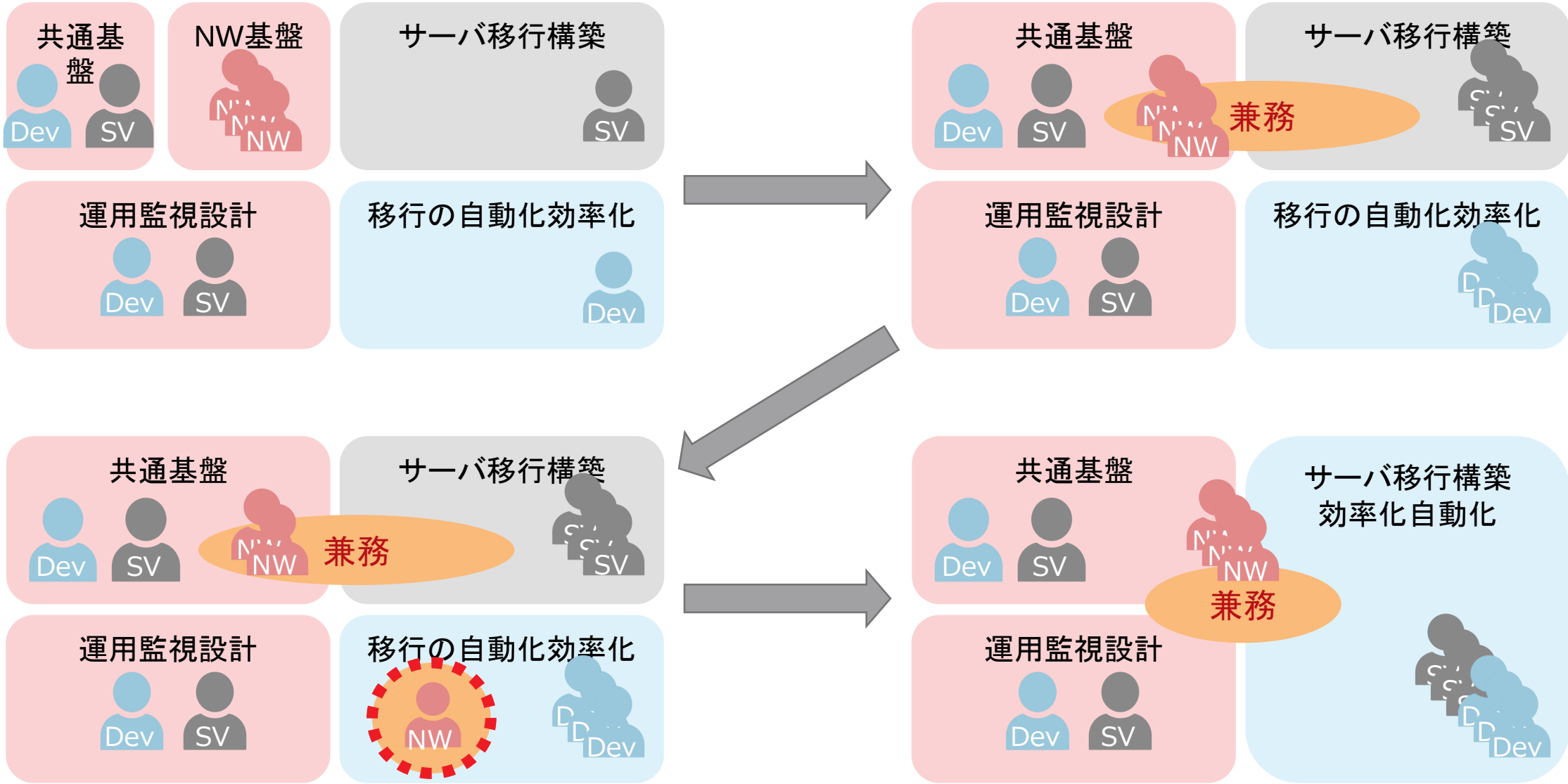
■チームの将来の向かう先
- 将来的に、自動化効率化チームと構築チームの垣根をなくしたい

↓↓↓↓

チーム課題を解決＆将来にむけ、自動化効率化チームにジョイン

初期 → 中期 → “今回の取り組み” → 目指す先

※人数は実際と異なる（Dev：開発、SV：サーバ、NW：ネットワーク）



取り組み

取り組んだことサマリー

① 課題管理の整備＆ルール作り
② 開発者とのブリッジ役を担う
③ 開発にコードレビュアーとして加わる

---

④ その他、細かな取り組み → 付録に記載
- 定例の簡略化
- Gitの展開＆運用方針
- 後回しにしたこと

①課題管理の整備＆ルール作り

①課題管理の整備＆ルール作り

- ツールの使い方がうまく伝わらない、流れがいまいちわからない
- ツールの利便性が認知されてない
- 課題解決のお願いをしたけど、依頼側は依頼しっぱなし、開発側は課題の優先度が分かりづらい
- お互いの距離がある

↓

課題のチケット管理、1つのバケツにリスト化されてる感じで、他のメンバーが分かりづらい

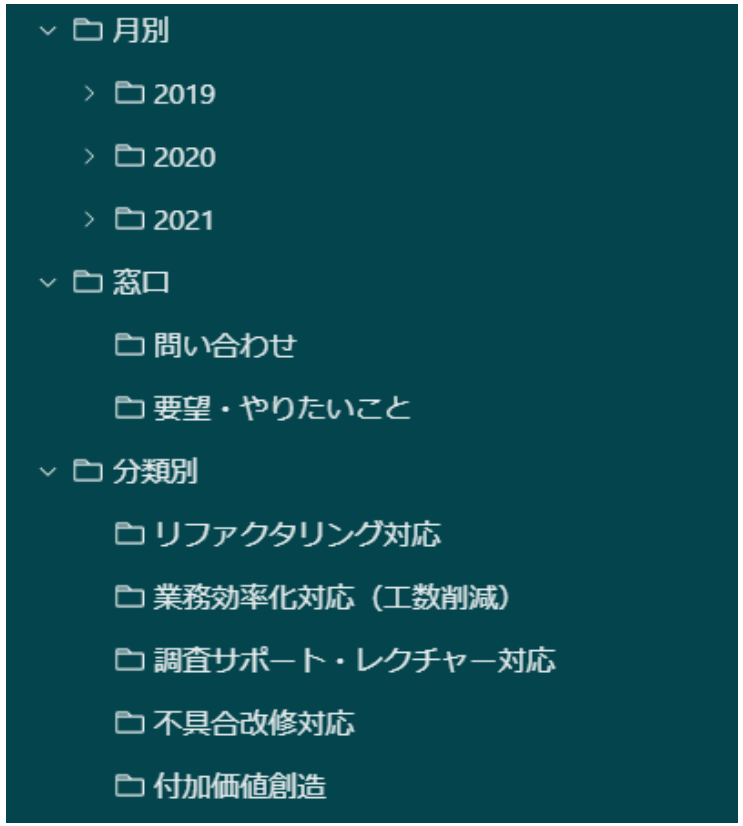
↓

現状整理＆ルール化して誰から見ても見やすい管理状態にしよう

課題管理の整備＆ルール作り実施

課題管理の整備

“wrike”という有料ツール（無料トライアル版あり）を利用



■タスク管理の種類

← 時間軸で管理するためのフォルダ
この配下に”202101”, ”202102”…みたいなフォルダがある

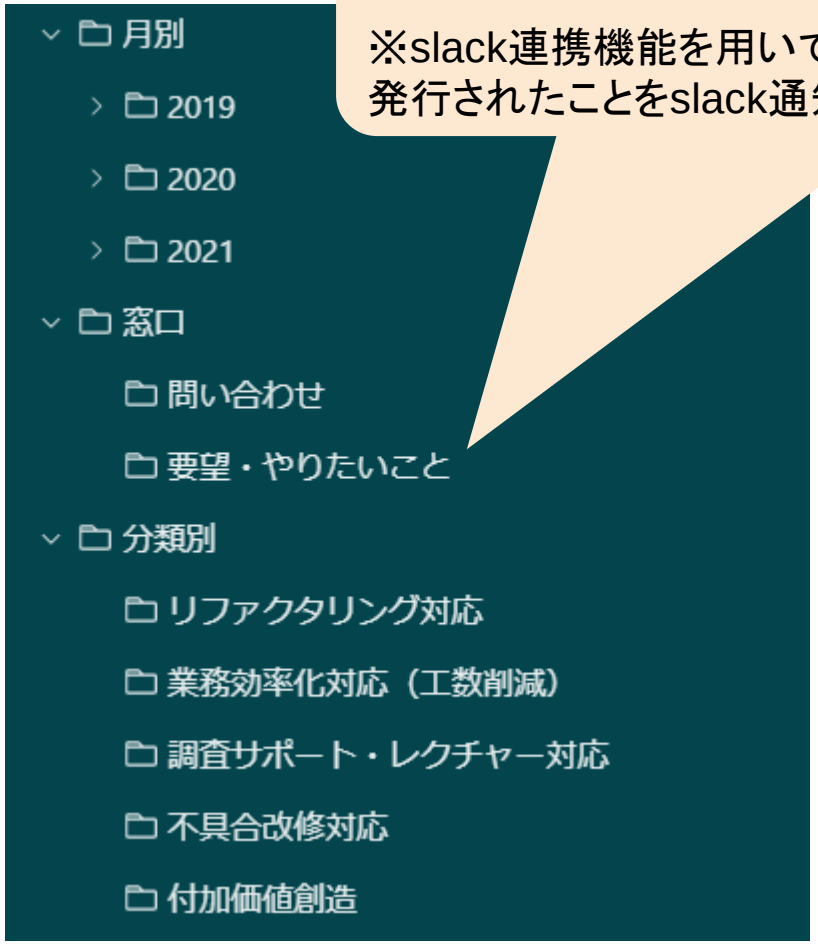
← 課題のチケット発行窓口となるフォルダ
【問い合わせ】　：すでに存在するものに対する不明箇所を改善する依頼
【要望やりたいこと】：新たな要望ややってほしいことの依頼

← チケットそのものの種類を管理するためのフォルダ
【リファクタリング対応】　：コードの再設計の対応
【業務効率化対応（工数削減）】：作業の効率化を目指す
【調査サポート・レクチャー対応】：使い方がわからない、思ったとおりに動いてくれないなどの対応
【不具合改修対応】　：展開したツールにて不具合が発生した際の対応
【付加価値創造】　：お客さん要望で発生したシステム開発　例）サーバの電源管理システム

チケット処理の流れ

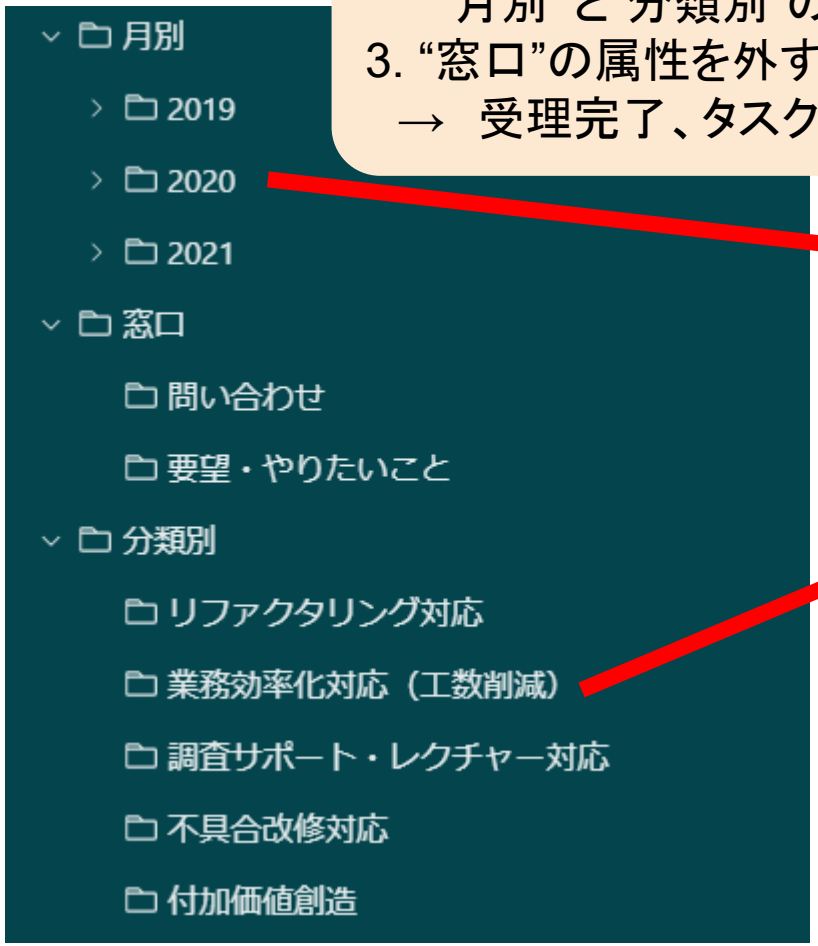
1. “窓口”のどちらかにチケット発行してもらう

※slack連携機能を用いて、
発行されたことをslack通知するようにしてる



チケット処理の流れ

2. 通知が来たタスクを確認し、その月に対応する際に
　“月別”と”分類別”の属性を紐付ける
3. “窓口”の属性を外す
　→　受理完了、タスク対応スタート



タスク管理のポイント

時間軸でのシンプルな管理

→フィルタの効いた管理は、どの立場のひとが見てもわかりやすい

→月に何件何時間やったなどの集計も楽

分類分けを行い、優先度を明確化＆タスクの属性を把握する

- そのコード変更は、不具合対応なのか/リファクタリングなのか
- そのトラブルは、不具合によるものなのか/本人が慣れてないだけなのか

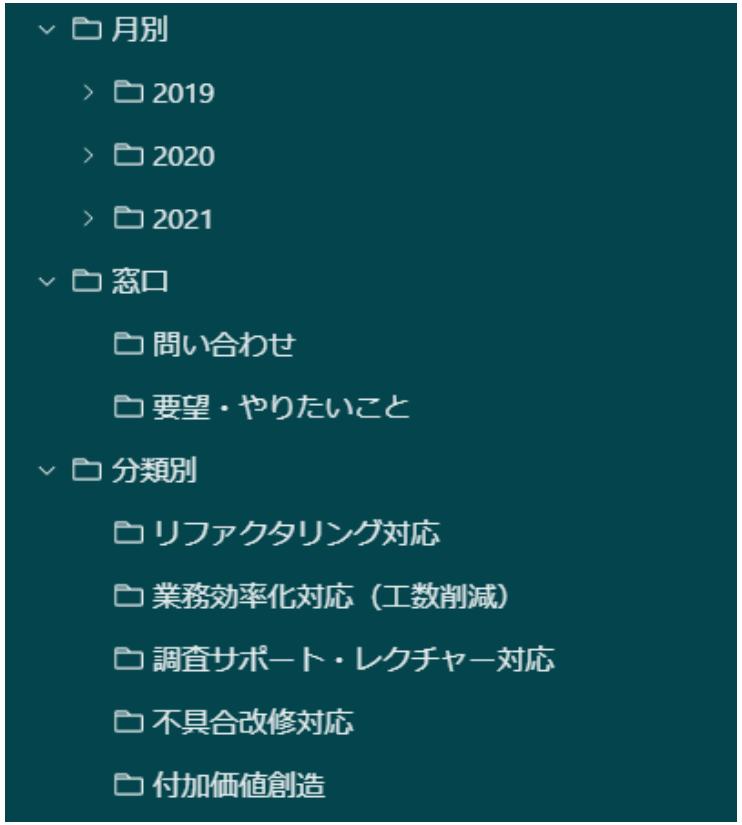
→ 分類分けを行うことで、優先度付けがしやすくなる

窓口を設置＆チャットツールをうまく使い、楽に気づける状態にする

→ “発行”と”受理”の管理を明確化

→ 依頼が放置されてるような状態を防ぐ

※有料ツールだと連携やら分析が楽



②開発者とのブリッジ役を担う

開発者とのブリッジ役を担う

- ツールの使い方がうまく伝わらない、流れがいまいちわからない
- ツールの利便性が認知されてない
- 課題解決のお願いをしたけど、依頼側は依頼しっぱなし、開発側は課題の優先度が分かりづらい
- お互いの距離がある

↓

うまくコミニュケーションがとれてない（会話、手順、などなど）

↓

間に立って、距離を詰めてくしかない

↓

以下取り組みを実施

- 依頼内容の深堀り
- 手順書整備、説明会実施
- 開発メンバーにも構築対応してもらう

ブリッジ役のポイント

依頼内容の深堀り

→「〇〇を自動化してほしい」 みたいな、抽象的な依頼でチケット化されてることが意外とある

→抽象部分を深堀りした形（次ページ参照）で開発者と会話すると、やりとりがスムーズ

手順書整備、説明会実施

→構築経験の目線を活かして、自動化ツールの利用手順を作る

→ハマりやすいポイントとかに気づける

→ツールごとの手順だけではなく、流れを意識した手順を軸に展開することができる

開発メンバーにも構築対応してもらう

→現状のやり方を知ってもらう、実際に触らないとわからないことは多い

※ 逆に自分は、開発状況を理解してもらうアプローチを積極的にする

※抽象部分を深堀り

NW作業の展開	ansibleの展開
・ 導入する設定を準備 ↓ ・ 作業対象にアクセス ↓ ・ 作業前確認、showコマンド ↓ ・ 設定導入 ↓ ・ 作業後確認、showコマンド	・ vars準備（設定情報） ↓ ・ playbookの準備(実行対象の情報、varsの指定など) ↓ ・ ansible-playbook実行 ↓ ・ 実行中（すでに設定されているか確認） ・ 実行中（設定導入） ・ 実行中（設定後確認） ↓ ・実行完了

→既存の手順作成でも無意識にやっているようなこと

各要素の前提条件や順番、確認してる観点、どこの処理が自動的だといいのか、
などを処理の流れで共有するだけでも違う

③開発にコードレビュアーとして加わる

開発にコードレビュアーとして加わる

- 将来的に、自動化効率化チームと構築チームの垣根をなくしたい

↓

- まずは自分がその代表にならなくては、、、

↓

- とはいえコーディングの学習コストは高い、、、
- すぐ動けないし、書くスピードは遅いし、生み出すコードは可読性が低い

↓

自動化していく作業内容自体は理解してるんだから、レビューをやっていくほうが覚えが早いのでは？
※最悪トラブったら自分で対応すればいいや

↓

自家発電でのコーディングを一旦諦め、レビューから関わる
とにかく人のコードを読むとこからスタート

ポイント

IaCの自動化の多くは「もともとの構築手法が存在する」ということ

→元のやり方でも対応できる準備をしつつやれば、リスクは意外と小さい

→元のやり方を理解してるおかげで、コードの理解が早い、応用がしやすい

ツールを使う側ならではの目線（初心者の気持ち）もレビューには活かせる

- Readmeファイルの文言
- コメントアウト

→もともとコードに馴れてないからこそ可読性が自然と上がる

※ IaCにおいては、処理速度よりも、可読性を高める方が大切と思う

レビュアーという立場/責任が人を強くする

→腹をくくる

取り組みの整理

取り組みの整理（現状課題）

課題（タスク）

課題（タスク）

解決方法1

解決方法n

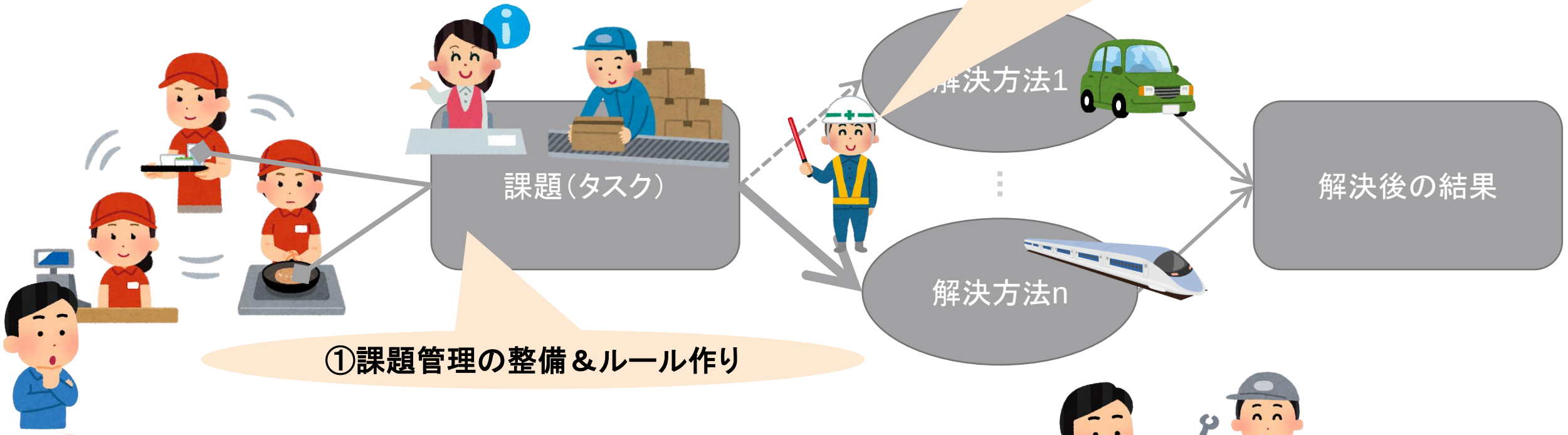
解決後の結果

■現状課題

- ツールの使い方がうまく伝わらない、流れがいまいちわからない
- ツールの利便性が認知されてない
- 課題解決のお願いをしたけど、依頼側は依頼しっぱなし、
  開発側は課題の優先度が分かりづらい
- お互いの距離がある

取り組みの整理（現状課題）

②開発者とのブリッジ役を担う
- 手順書整備、説明会実施

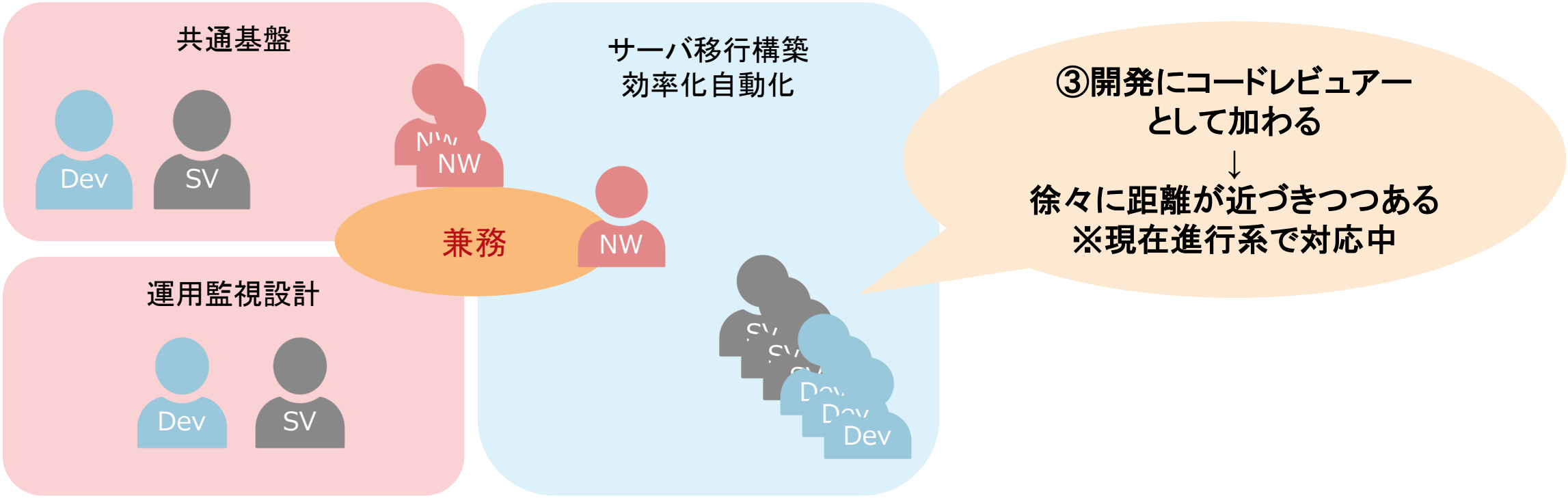


①課題管理の整備＆ルール作り

②開発者とのブリッジ役を担う
- 依頼内容の深堀り
- 開発メンバーにも構築対応してもらう

取り組みの整理（チームの将来の向かう先）

・ 将来的に、自動化効率化チームと構築チームの垣根をなくしたい



IaC文化を広く根付かせるために必要なことを考える

～全部で4つ～

IaC文化を広く根付かせるために必要なこと、その1

“全体のプロセス”を考えることがまず重要

悩み） 実際に取り組みを始めたけど、なかなか業務フローに落とし込めてない
→課題整理と優先度付けを行い、無理なく継続的に動く仕組みを作れていない
→整理できていないことで、時間的把握ができない状態

コードで業務をすることが基本となる仕組みを組み上げることが最優先

IaC文化を広く根付かせるために必要なこと、その2

“1から自前でのコード学習”を避ける

悩み） チームで始めたいとは思ってるけど、進め方がイメージできない
→今の業務をしながらのコード学習、十分ハードル高い

なんとかしてでも開発者をチームに招き、異文化を持ってきてもらう方が進みがいい
→経験者が1人いるだけで全く違う（例：IDE(統合開発環境)、モジュールとの付き合い方、など）
※IDE（Integrated Development Environment）
→今は柔軟な雇用体系がある（曜日で雇うなど）
※「①課題管理の整備＆ルール作り」に関して、別案件で来ていたSREコンサルエンジニアの方に技術支援として協力頂いた

※現チームの開発者のバックグラウンドはこんな感じ
→元々、基幹系のシステム開発（Java/ SVN）、ansible/python/Gitは今回初めて触る

※自前でのコード学習が良くないと言いたいわけではない、限りある時間をもっと課題解決のアイデア出しや設計に使うべき

IaC文化を広く根付かせるために必要なこと、その3

コードに関わる入り口、書くことからよりも”読む”ことから （の方がいいかも）

悩み） 一部の担当者のモチベーションに依存してる（救われてる）状態
→課題解決の過程自体のナレッジが蓄積されないリスクを抱えてる
→コーディングできる人を増やす ＜ まずは理解できる人を増やす

コーディングからではなく、コードレビュアーから関わる
→ネット記事抜粋：「開発における初心者上級者と比較して、コーディングは10倍の差、レビューは6倍の差」
→レビューの方が影響が少ない

IaC文化を広く根付かせるために必要なこと、その4

“ブリッジ役”を作り、異文化を混ぜる

とにかくやり方を理解し合うことが大切、そして間に立つ役割を担う人がいた方がいい
→互いにわかりやすい形で、情報のやりとりをする
→ときにはインフラ構築を開発者にも対応してもらう（逆も重要）

【まとめ】IaC文化を広く根付かせる

- “全体のプロセス”を考えることがまず重要
- “1から自前でのコード学習”を避ける
- コードに関わる入り口、書くことからよりも”読む”ことから （の方がいいかも）
- “ブリッジ役”を作り、異文化を混ぜる

→ → → IaCに関われる場面を、より早く増やすことが大切

→ → → インフラエンジニアとして関われる場面は、
コーディングだけではない

議論質疑

議論質疑

- 文化作り、うまくいった / いってない話
- こういうところで困ってる話
- 開発スキルの採用の話
  - どのような基準を設けてますか？、実際人集まってますか？

最後に

最後に

誰かのIaC文化作りのプラスになれば幸いです。

DAY1午後も運用自動化に関わるプログラムが続きます。
個人的にも楽しみにしてます！

以上「 IaC文化を広く根付かせる」の発表をおわります。
ご清聴ありがとうございました。

EQUINIX

【付録】その他細かな取り組み

その他1、定例の簡略化

- チーム会議を最小限にする
  →共有する系の時間を減らす
  →タスク全体の把握担当は1人、それ以外は自分のタスクに専念

- 会議のポイント
  →基本は進捗をチケットに書き出ししておいてもらうスタンス
  →定例では把握担当が担当者に深堀り、情報の質の均一化
  　関係ない人は別のことしててOK　＝　この時間に自分の別タスクの書き出ししてた印象

対応するかもわからない他人のタスクを把握しておくのは、全員がやる必要はないと思う
文字に起こしてもらうことを基本とすることで、必要なときに文字情報をみれば把握できる環境を目指す

※タスクが片付いた or 優先度の高いものがなくなったタイミングで、slackベースの定例に変更

その他2、Gitの展開＆運用方針

※リポジトリ管理はAWS-CodeCommitで管理してる前提

- 最初は1つのリポジトリ運用でいいと思う
  →「構築に必要なツール」というくくりで管理する
  →一部の機能でCI/CDを導入するタイミングで、リポジトリを分けだした

- 初期の頃Gitに対する理解がまだまだだったので、各所にREADME.mdファイルを設置し、
  そこに変更情報を残す運用で対応
  →READMEを積極的に使うのはチームに理解されやすかった

- ブランチ運用のルールは、GitHubフロー
  →とにかくシンプルな仕組みからスタート、すべてmaster(main)ブランチに集約
  →CI/CDがやりたいタイミングで、gitフローなどのルールに変更する

- master(main)ブランチにプッシュできないよう権限制御
  → master(main)ブランチは必ずプルリクエスト経由で変更する、安心できる環境作り

その他2、Gitの展開＆運用方針

※リポジトリ管理はAWS-CodeCommitで管理してる前提

- ブランチ名やcommitやPullRequest&Mergeのルール、そこまで厳格に決めなかった

ルール	内容
ブランチ名	・接頭語として”modify”,”create”,”delete”あたりのわかりやすい動詞をつける ・接尾語としてユーザー名(IAMユーザー名)をつける ・あとはおまかせ、迷ったら相談
コミット	・コミットメッセージは、日本語でいいので、やったことを簡潔に書く ・タスクのチケットのURLなど、紐づく情報のリンクをつけてもらえればよりよい
プルリクエスト マージ	・マージ方式は“3wayマージ（Non-Fast-fowardマージ）” →マージコミットも残したかったから ・Author nameは”「Committer: xxxx / Merge: yyyy」の命名規則” →マージコミットに2つの情報を載せるため ・マージ後のリモートリポジトリ側ブランチ（リモートブランチ）は残さず削除 →古いブランチを残さない、master(main)で原則管理

その他2、Gitの展開＆運用方針

※リポジトリ管理はAWS-CodeCommitで管理してる前提

- 抑えておくGitコマンド一覧

Gitコマンド	
---- ブランチ作成、編集、add、commit、pushするまで ----	---- ちょっとした操作系 ---
git clone “リポジトリのURL”	git fetch --all
git checkout -b “ブランチ名”	git checkout “ローカルブランチ名”
git status	git checkout origin/”リモートブランチ名”
git diff	git stash
git add .	git stash apply
git commit -m “コミットメッセージ”	git checkout “ファイル名”
git pull origin master　(または　git pull origin main)	git log “ファイル名”
git push origin HEAD	git show “コミットID”

その他3、後回しにしたもの

- CI/CD（Continuous Integration/Continuous Delivery）
  →作業頻度が高いものは効果あるが、低いものにCI/CD入れても費用対効果薄い
  →運用してて「いつもやってるこの作業めんどくさいなー、神経つかうなー」と思ったタイミングでいい

- 開発する環境、ツールを利用する環境は当初、Linuxサーバで展開
  →実行権限を整理が楽だったから（AWS環境においてIAMはEC2に適応できる）
  →現在は各メンバーのローカルPC環境で利用できるように、実行権限を整理

EQUINIX