Government Cloud Workshop 2026

Level 200-300

Step Functions で実現する
フルマネージド・ジョブ開発

ガバメントクラウド開発における
設計/開発・運用時の「理想と現実」のギャップ

自己紹介

杉元 大一

Sugimoto Taichi **課長**

Project Manager（PM）

マネジメント規模 50名

所属： NTT DATA 第二公共事業本部 ワークライフセキュリティ事業部

経 歴

2013～2022	オンプレ更改プロジェクトPM／業務AP機能PM	中央省庁向けオンプレ基盤構築。業務AP機能PMを複数件担当。
2022～2024	Azureクラウド更改プロジェクトPM	Azure IaaSリフト更改プロジェクトPM（バックアップ/DR）。チーム45名を統括。基盤500台、AP：20KS。
2024～現在	**AWSガバメントクラウド移行プロジェクトPM**	フルマネージド基盤でのガバメントクラウド更改プロジェクトPM。チーム50名を統括。基盤10台、AP：500KS。

開発プロジェクト概要

K省庁システム ガバメントクラウド移行プロジェクト

50名	約2年	20本+	R1
チーム規模	プロジェクト期間	移行バッチ数	移行方式

背景と目的

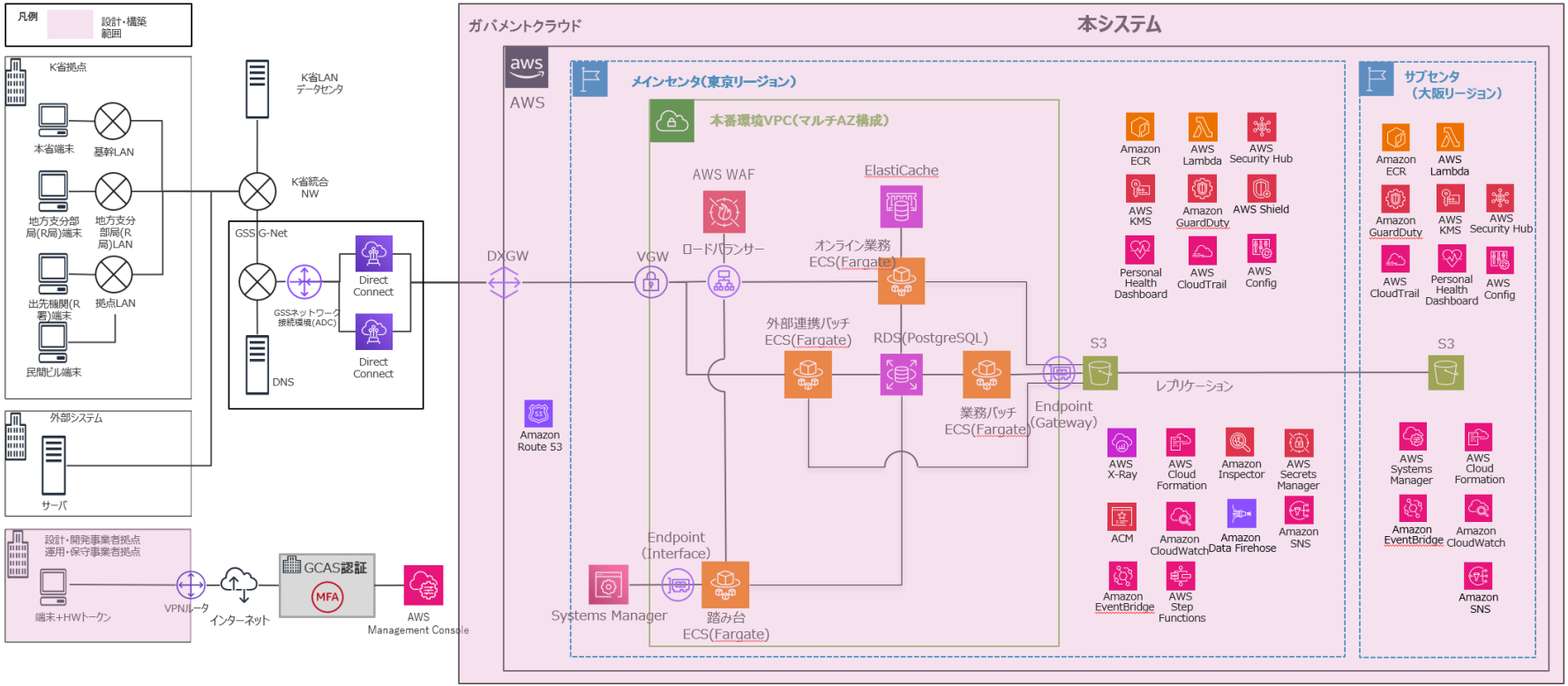
- 政府共通2期PF上のAWS（ジョブ管理ソフト）からガバメントクラウドへ移行（フルマネージド対応）。
- 2024年6月～2025年12月のPJ。※検討内容は当時の最新アップデートを前提とする。
- 20本以上のバッチジョブを Step Functions で再構築。

プロジェクト概要

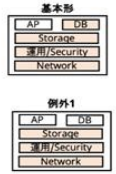
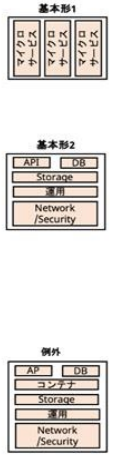
No	項目	説明
1	プロジェクト説明	K省システムのガバメントクラウド移行プロジェクト
2	利用ユーザ数	約3300名
3	利用拠点数	380拠点
4	システム構成	次スライド「システム構成概要」を参照
5	SLA(稼働率)	オンライン・バッチ機能の稼働率：99.5%
6	オンライン稼働時間	開庁日の8:00～21:00
7	案件期間	設計・構築～リリースまで：約1年7か月
8	ガバクラ移行方式	R1:Replatform 基本形（R2への二段階移行前提）

システム構成概要

- 本番環境はメインセンタ内の冗長化を取り、サブセンタにデータのバックアップのみ行い、障害発生後の復旧方式を採用



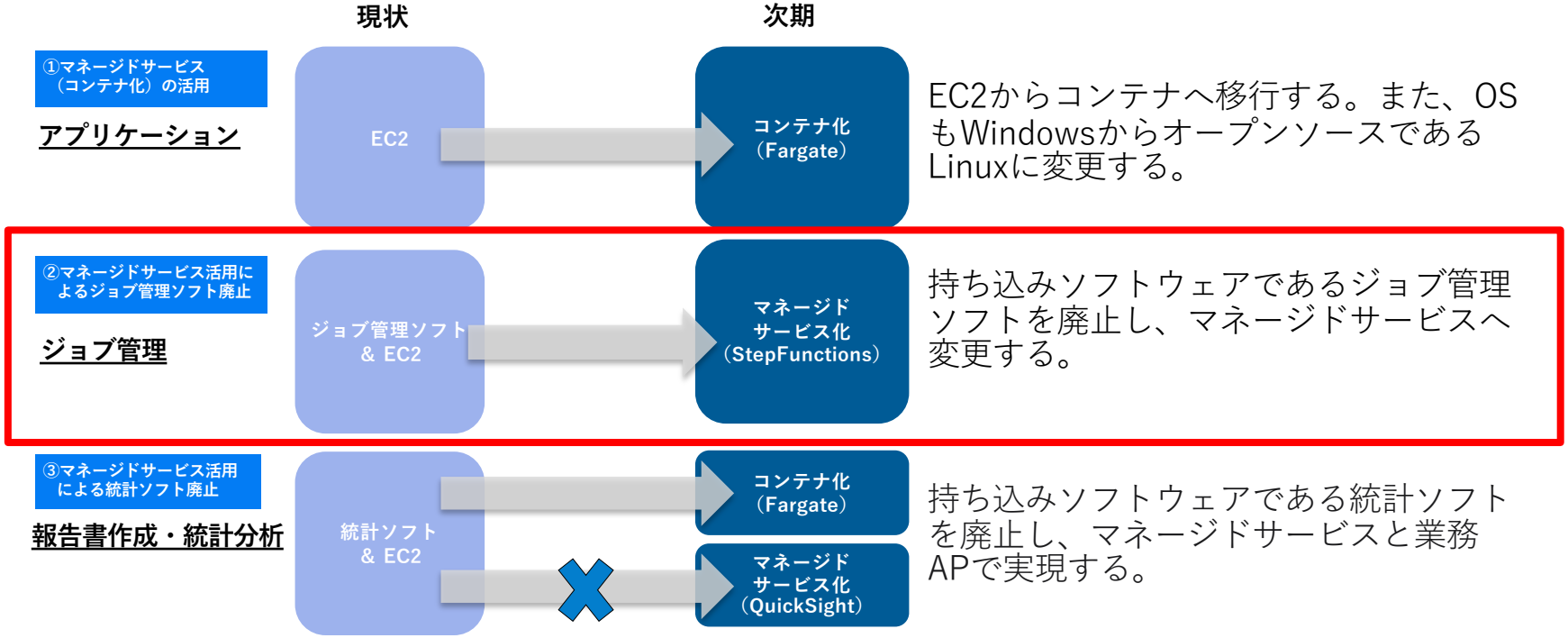
ガバメントクラウド移行方式とは？

移行パターン		内容	移行パターン詳細
R1	**Replatform** アプリー部変更移行 2段階移行の第1段階の場合のみ	アプリケーションの変更を最小限にクラウドのマネージドサービスを活用 **基本形** ・運用／セキュリティ、RDB、ストレージをマネージドサービス化 ・共有ストレージについては可能な範囲でオブジェクトストレージを利用 **例外1** ・既存DBMSがマネージドサービスに対応していない等RDBをマネージドサービス化できない場合、RDBをサーバインスタンスにインストールして構成することを例外的に受け入れる。	
R2	**Rebuild** アプリ再構築移行	アプリケーションを変更してクラウドサービスをフル活用 **R2の前提** ・R2におけるコンピュートサービスは、サーバレスまたはコンテナを用いることを前提とする **基本形1** ・アプリケーションをマイクロサービスにアーキテクチャ変更して移行 ・基本形1の詳細は、「ガバメントクラウド概要解説_6 必須検討事項」の6.1.4の「（1） 基本形1（マイクロサービスアーキテクチャ）」を参照 **基本形2** ・アプリケーションをフロントアプリ化、バックをAPI化し、イベントドリブンアーキテクチャでバッチ数を極小化して移行 ・DBもRDB前提とするのではなく、ドキュメントDBやHadoop＋オブジェクトストレージ等最適なデータベース/ストレージへ変更 ・基本形2の詳細は、「ガバメントクラウド概要解説_6 必須検討事項」の6.1.4「（2） 基本形2（フロントエンド・バックエンド分離）」を参照 なお基本形1,2はどちらでもよく、結果的に同じアーキテクチャになる可能性もある。 **例外** ・長期塩漬けシステムや今後の運用コストが極小のシステムに関して、アプリケーションサーバをコンテナ化しての移行を例外的に受け入れる。ただし、アプリケーションのサイズをコンテナに最適な粒度まで機能分割してコンテナ化することを検討すること ・「長期塩漬け」と「運用コスト極小」の基準は、「ガバメントクラウド概要解説_6 必須検討事項」の6.1.4「（3）例外」を参照	
R3	**Repurchase** SaaS利用移行	既存アプリをoff-the-shelfで使えるSaaSへ置き換え	n/a

引用元：
システム移行ガイド(AWS編) | テーマ別技術ガイド | ガバメントクラウド AWS 技術ガイド | GCASガイド
https://guide.gcas.cloud.go.jp/aws/technology-guides/how-to-migrate-system

R1対応の内容

基盤構成はフルマネージド化・IaC・CI/CD・コンテナ化によりR2相当の構成。アプリのフロント/バック分離は行わない（R1相当）。



大規模システムの奇妙な現実

1. 技術のアップデートサイクルに対して、
システム刷新のタイミングは**長期（4～5年）**である。

2. 本番で問題なく動作している、
非機能を改修アップデートする**リスクは甚大**である。

3. 大規模になるほどコミュニケーションハブが増加する。
エンドユーザー機能の拡張により、技術者の質にも
ばらつきがある。
～世にも奇妙な複雑構造のシステムへ～

NTT DATA

Agenda

01	Step Functions 基礎・アーキテクチャ	2 min
02	ガバメントクラウド特有の制約・考慮点	2 min
03	設計・開発フェーズの 理想 vs 現実	7 min
04	運用フェーズの 理想 vs 現実	7 min
05	今後の展望と提言	4 min

01

SECTION 01

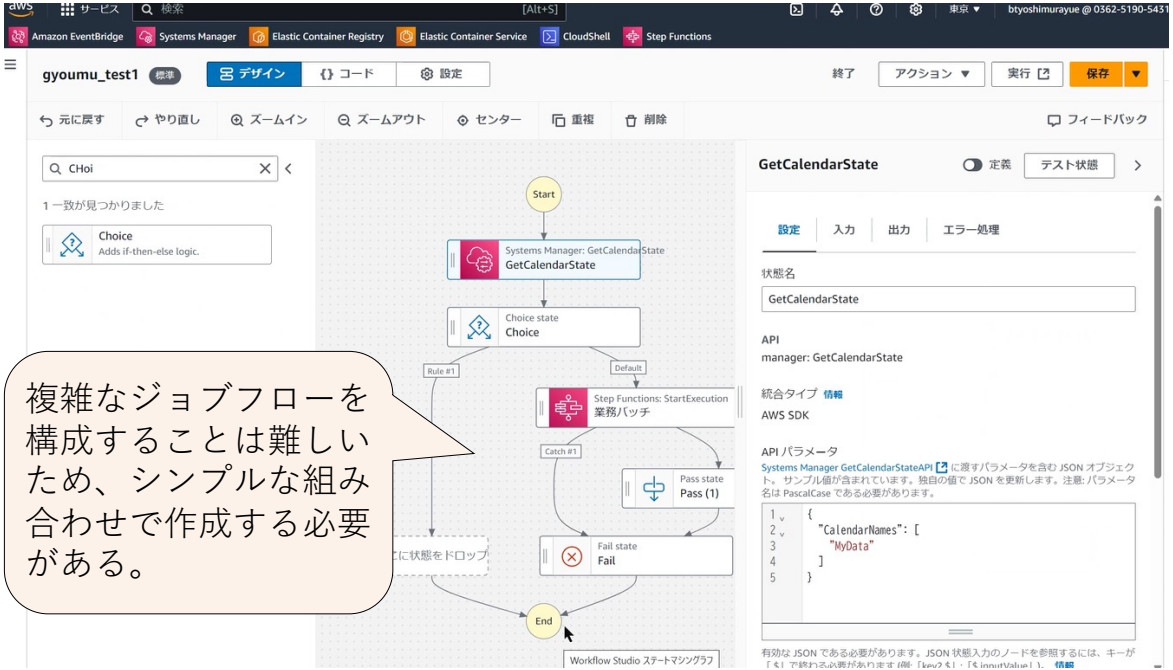
Step Functions 基礎・アーキテクチャ

State Machine によるサーバーレスワークフロー管理

AWS Step Functions とは？

フルマネージド ワークフローサービス

- ✦ AWS Lambda / ECS / Glue などをステートマシンで連結
- ✦ ビジュアルワークフロー設計（Workflow Studio）
- ✦ 自動リトライ・エラーハンドリングを組み込み標準装備
- ✦ サーバーレス：インフラ管理不要
- ✦ CloudWatch / X-Ray との統合



複雑なジョブフローを構成することは難しいため、シンプルな組み合わせで作成する必要がある。

SECTION 02

ガバメントクラウド特有の制約・考慮点

デジタル庁認定クラウドサービスにおけるStep Functions活用

ガバメントクラウドの全体像

セキュリティ要件

- ISMAPクラウドサービス登録
- FISCガイドライン準拠
- 必須適用テンプレートの適用義務

ネットワーク要件

- 閉域網接続（専用線/VPN）
- GCAS認証必須（ハードトークン）
- GSS Gnetを用いた接続必須

運用・監査要件

- CloudTrail 完全有効化
- 定期的な脆弱性診断
- 変更管理プロセスの厳格化

SECTION 03

設計・開発フェーズの理想 vs 現実

設計時の想定と実際の開発で直面するギャップ

設計フェーズの「理想」

💡 IDEAL

✓ ワークフローの完全可視化

Workflow Studio でドラッグ&ドロップ。
全ジョブフローを非エンジニアにも説明可能である。

✓ エラーハンドリングの標準化

Catch / Retry を ASL に定義。
個別スクリプトでの try-catch は不要である。

✓ 並列処理の容易な実装

Map / Parallel ステートで自然に並列化。
ジョブの依存関係を宣言的に管理できる。

✓ サーバーレスで運用コスト削減

サーバー管理ゼロ・自動スケール。
ジョブ専用EC2インスタンスが不要である。

設計フェーズの「奇妙な現実」①

ジョブ管理ソフトで実現していた機能

AWS アップデート

問題点

✗ AWS Step Functions では異常終了した処理からのリスタートしかできない。

ジョブネットは、成功・失敗を問わず途中から確実に再実行できるリスタート機能を標準装備している。
Step Functions には失敗時の再実行（Redrive）の概念しかなかった。

未対応

RedriveExecution API
失敗ステートからの再実行はサポート（Redrive）。成功ステートからは再実行できない。

✗ 複雑なジョブフローを設定できない

ジョブネット多重起動・待ち合わせ・OR条件・時刻監視など、複雑な依存関係をGUIで設定可能である。Step FunctionsはChoice / Waitの条件表現やステート数上限（2,500状態）に制約があり、複雑度を再現しにくい。

部分対応

Distributed Map /
ネスト State Machine
Map状態の大規模並列処理や子ステートマシン呼び出しで表現力は大幅に向上した。

設計フェーズの「奇妙な現実」②

ジョブ管理サービスで実現していた機能	AWS アップデート
問題点	
✗ 定められた日時でのスケジュール起動が困難 「独自カレンダー」を管理し、非稼働日のジョブスキップ・翌営業日繰り越しを自動制御できる機能が標準搭載されている。試験時に定められた日付（未来日）での稼働も可能である。 EventBridge Schedulerはcron/rate表現のみで、独自カレンダー管理の概念がない。	部分対応 SSM Change Calendar + Lambda 祝日マスターをDynamoDBで管理し、Lambdaで稼働日判定を行う代替パターンで対応可能である。
✗ プログラム言語の制限 — Lambda で .bat ファイルが利用不可 Windows環境で.bat / .exeをジョブとして直接登録・実行可能である。 LambdaはLinuxコンテナ環境のため、Windowsバッチファイルをそのまま移行できず、大規模なスクリプト改修コストが発生する。	部分対応 ECS/Fargate Windows Container Windows Serverコンテナで.bat / .exeを実行可能である。 LambdaはLinux環境のみである。

対策①：AWS Step Functions では異常終了した処理からのリスタートしかできない

先行処理からの再起動が必要とならないよう、該当処理のプログラム構成を見直す。

異常終了を示す

現行システム

ジョブA → ジョブB → ジョブC

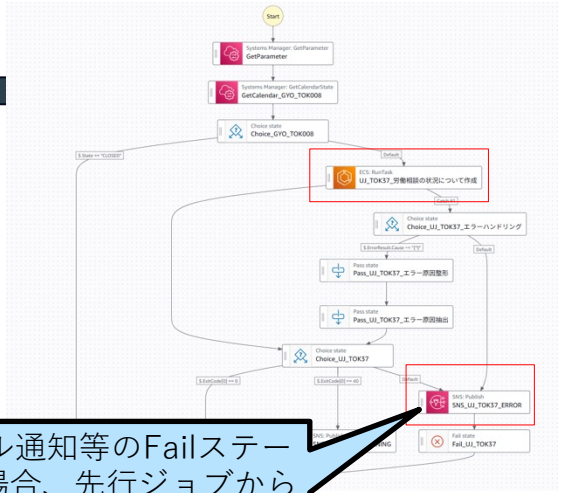
リランではワークフローの最初のジョブから再実行できる。

次期システム

ジョブA → ジョブB → ジョブC

リドライブでは異常終了したジョブからのみ再実行できる。

ジョブBから処理を開始しても問題なく処理可能となるようプログラムの構成を見直す。

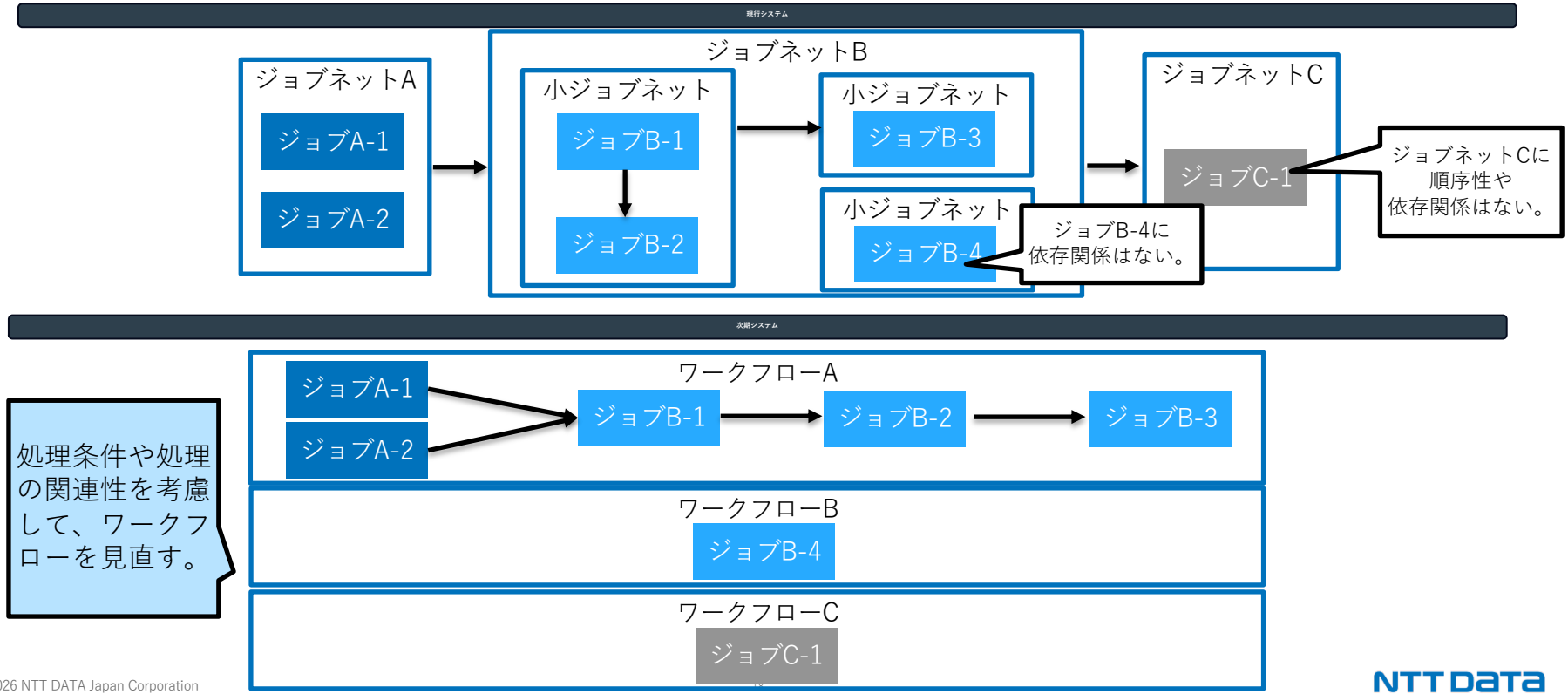


エラーメール通知等のFailステートを入れた場合、先行ジョブからのリトライができず、運用が困難になる。

NTT DATA

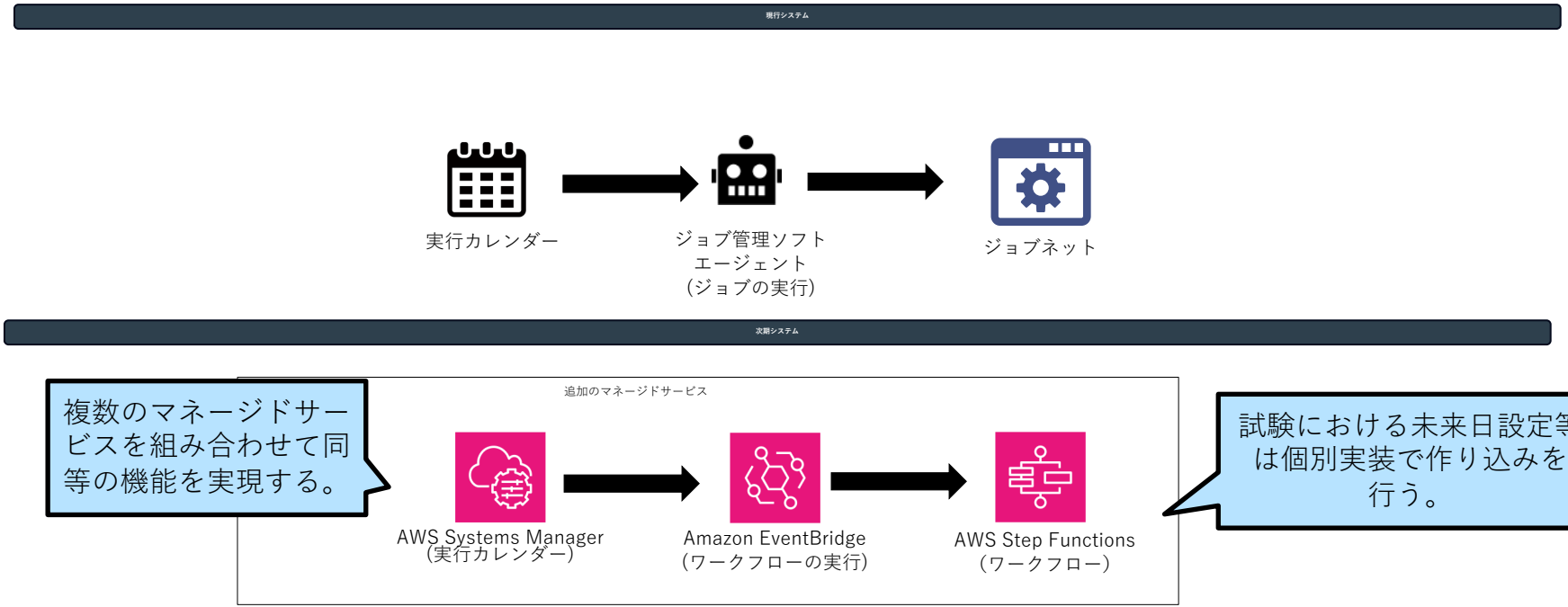
対策②：複雑なジョブフローを設定できない

現行システムとの互換性を確保するため、処理条件や処理の関連性を考慮して、ワークフローを抜本的に見直す。



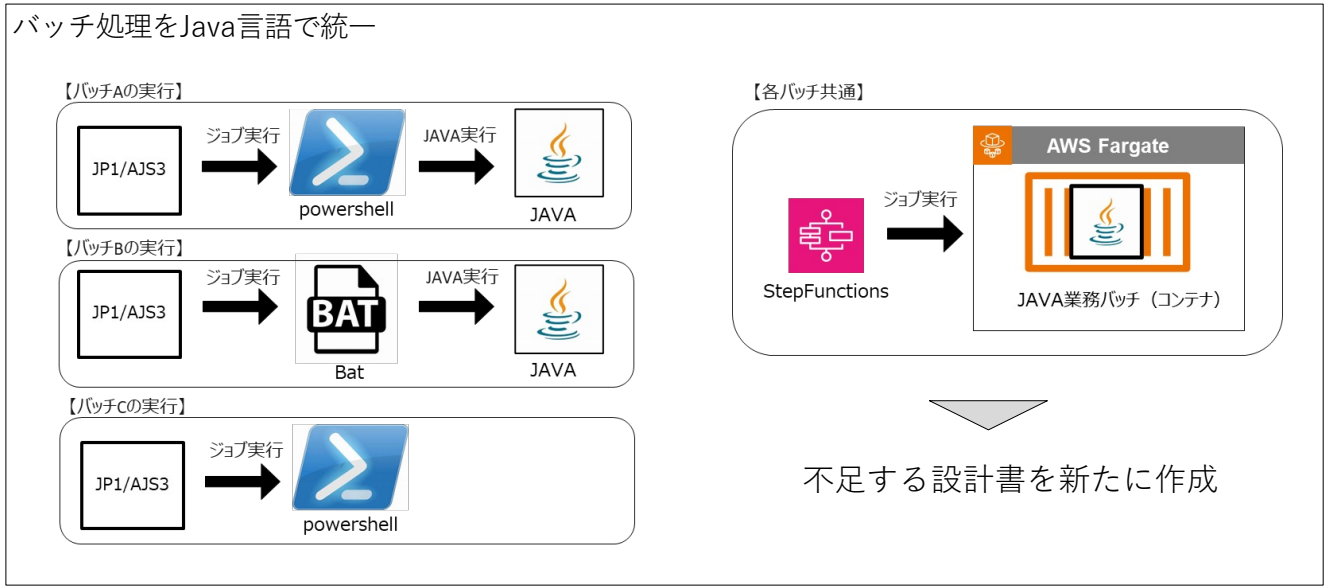
対策③：定められた日時でのスケジュール起動が困難

現行システムでは、ジョブ管理ソフトで設定したカレンダーに合わせてジョブをスケジュール実行しており、定められた日付で柔軟に実行できる。複数のマネージドサービスを組み合わせて同等の機能を実現する。



対策④：プログラム言語の制限事項による対応

今後の保守やメンテナンスを容易にすることも目的としてJavaに統一する。
これに伴い、PowerShellスクリプトや.batスクリプトの設計書を見直すとともに、既存のJava定義書の記載と合わせるため、不足する設計書を新たに作成する。



04

SECTION 04

運用フェーズの理想 vs 現実

Step Functions 稼働後の監視・障害対応における課題

運用フェーズの「理想」

IDEAL

✓ ダッシュボードで一元管理

全ジョブの実行状態をコンソールで確認
ビジュアルな実行履歴で原因特定が容易

✓ アラートの自動通知

CloudWatch Alarm → SNS → メール/Slack
失敗ジョブの即時検知と通知自動化

✓ 失敗ジョブの再実行

コンソール/API から特定ステートを
再実行でき、手動リカバリが容易

✓ 運用人員の削減

再実行処理の自動化、フルマネージド化により
夜間監視の人員削減

運用フェーズの「奇妙な現実」

⚡ REALITY

✗ CloudWatch Logs 爆発問題

大量バッチ実行でログ量が爆増。
コスト増・検索性が大幅悪化。
（特に顧客エラー操作が1対1にならず追跡が困難でログフィルタ設計が必須である。）

✗ 再実行時のべき等性

中途失敗からの再開が困難である。
部分的な再実行でデータが二重処理になる危険性がある（特にDB更新系）。

✗ 閉域空間へのアラート通知

Slack/Teams連携は工夫が必要である。
（VPC Endpoint経由等）
個人情報要件等で組織のセキュリティ要件が強いとデータ運搬工数を下げられない。

✗ SLA要求改善

プロジェクトのSLA要求を下げられないため、失敗時は人員による夜間リカバリが必要である。
現実には張り付き要員が必要で、現場側でSLA要求の改善を検討しないと運用工数は抜本的には下がらない。

運用改善：ギャップを埋めた運用例

1

課題：ログ爆発対策

Log Group に保存期間 (Retention) + Subscription Filter を設定。

2

課題：アラート通知改善

セキュリティルームでSNS通知を閲覧可能にする環境の穴あけを行う（情報システム部と調整）。

3

課題：べき等性の確保

DynamoDB に実行ID + ステータスを記録。
再実行前チェックで二重処理を防止 (Idempotency Token)。

4

課題：SLA要求

プロジェクトのSLA要求を再度交渉する。夜間人員の配置を再度見直す。また、DB起動時間を常時起動とし、AWSアップデート等による運用作業自体を削減する。

なぜこの理想と現実の乖離が生まれるか

積み上がった歴史・文化の違いにあると考える（どちらが良い悪いではない）。

それぞれの積み上げてきた歴史・文化・価値観

大規模システムという複雑な現実

大規模レガシーになるほど理想への道のりは遠い。
0 or 1ではなく、**折衷案の選択肢**を増やす必要がある。

SECTION 05

今後の展望と提言

レガシージョブ管理からの移行と次世代活用提言

公共SaaS、ISMAPの認可・拡張提言

フルマネージドによる運用経費削減の目的を達成するためには

1

ジョブ管理SaaS、運用SaaSの公共SaaS,ISMAP登録・認可の拡大

フルマネージドなジョブスケジューリングSaaSをガバメントクラウド利用可能サービスとして追加登録する。
Step Functionsの補完として、省庁システムへの導入障壁を下げる。Hinemos SaaS等。

2

最新技術、フルマネージド運用SaaSの提案障壁を下げる

最新技術のアップデートサイクルに対して、公共SaaS、ISMAP認定サイクルは追いつかない。
最新技術やフルマネージド運用SaaSの提案時の承認障壁を下げる。（大規模レガシーからの移行ほど柔軟に対応する）

まとめ

Step Functionsはガバメントクラウドにおける「ジョブ管理のフルマネージド化」を実現する強力なツールである。

01 **基礎を押さえれば強力**
既存のジョブフローを、ベストプラクティスを踏まえて初期段階で再整理することが鍵である。

02 **閉域空間での運用設計を先に固める（社内情報システム部と連携）**
セキュリティルームへの運用監視を踏まえ、VPC Endpointの穴あけ等はアーキテクチャ初期段階で必ず実施する。

03 **異常終了時の再実行フローは設計段階で整理**
べき等性や異常時の再実行フローは、無人で実行できる工夫を設計段階で整理する。

04 **運用設計は開発前から**
ログ戦略・アラート通知・再実行設計を先行して計画する。

05 **大規模レガシーほど、移行は段階的アプローチで**
全置換ではなくハイブリッド運用から始め、リスクを最小化する。

最後に

一番大事にしたいこと

エンドユーザーへの付加価値の最大化

手段：非機能変換・運用にかかるコスト削減等

目的の達成のためには**あらゆる選択肢を迅速かつ柔軟**にとれるようにしたい

NTT DATA

Q & A

ご清聴ありがとうございました

杉元 大一 | ガバメントクラウドワークショップ 2026
協力者　伊藤 慎吾、増田 邦光、吉村 優貴　等

AWS Step Functions ・ Amazon EventBridge ・ AWS Lambda ・ Amazon ECS