AWS構成意図ドキュメント（Step1〜4） - Hayata Cloud Portfolio

1. はじめに：このドキュメントの目的

このドキュメントは、私が構築したAWSクラウドインフラ（Step1〜Step4）について、

「なぜこの構成にしたのか？」「どう応用できるのか？」を言語化したものです。

単なる構築スキルにとどまらず、

技術選定理由・設計思想・再現性・実務応用力を整理することで、

採用時に「任せられる構成力」「説明できる力」を証明することを目的としています。

---

2. 構成と技術選定の理由

2-1. GitHub Actionsを選んだ理由

- 無料枠で使えるCI/CDサービスであり、GitHubとの親和性が高い
- `deploy.yml` による明示的な構成が書け、保守しやすい
- CodePipelineと比較して、**学習コストが低く即実装できる**ため個人開発に最適

2-2. Lambdaを選んだ理由

- サーバレスでインフラ管理不要
- API Gatewayと連携しやすく、**シンプルなバックエンド構成に適していた**
- スケーラブルかつ安価（無料枠）で、個人学習にも実務にも応用可能

2-3. CloudWatch Logs / Metrics

- 実行状況やエラーを**ログ・数値で可視化**できる
- DurationやInvocationsのメトリクスをもとに監視体制を組める
- Metricsを元にAlarmを作成し、**運用での異常検知体制を再現**

2-4. CloudWatch Alarm + SNS通知

- Errors >= 1 でアラートが発火 → メール通知
- チーム開発ではSlack通知にも拡張可能（SNS + Lambda連携）

---

3. 設計時に意識したこと

ポイント	内容
再現性	TerraformやGitHub Actionsで、手順を自動化・コード化
可観測性	CloudWatch Logs / Metrics / Alarmを通じて、**「気づける仕組み」**を構築
メンテナンス性	deploy.ymlでCI/CDルールを一元管理。構成の追加・変更がしやすい
拡張性	ECSやFargate、Slack通知にもスムーズに展開可能な構成

---

4. 他の構成との比較・選定理由

比較対象	採用しなかった理由／将来的な可能性
AWS CodePipeline	学習コストが高め／無料枠が少ないため、まずはGitHub Actionsで構築
GUI構築（マネジメントコンソール）	手軽だが再現性が低い。チーム運用ではIaCの方が望ましい
CloudTrail + EventBridge	より複雑で応用範囲広いが、**現段階ではまず基本構成の定着を優先**

---

5. 実務応用のイメージ

- 小〜中規模のAPI構成（静的Web + Lambda + DynamoDB）にそのまま転用可能
- チーム開発では、SNS通知先をSlackにすることでリアルタイム障害検知体制に発展可能
- deploy.ymlを拡張すれば、**複数関数・ECS/Fargateデプロイにも応用できる**

---

6. まとめ

本構成を通じて、私は以下を実現しました：

- GUIだけでなく、**Terraform + GitHub Actions** によるIaC/CI/CDを習得

- CloudWatchを使った**監視・アラーム通知体制**を実装
- なぜこの構成にしたか、**言語化し説明できる力**を身につけた

今後は、ECS構成・企業別カスタム構成・実案件再現などに取り組み、
「構築 × 自動化 × 監視 × 説明 × 応用」を持つクラウドエンジニアを目指します。