**Confidential（秘密情報）**

**ドローン生産部品在庫管理システム**

**インフラ構築手順書**

第1.1版　2024年6月1日

株式会社ドットライフ

**改定履歴**

| **版数** | **日付** | **改定内容** | **項番・ページなど** |
| --- | --- | --- | --- |
| 1.0 | 2024年6月1日 | 初版作成 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**目次**

[**1. 業務要件 3**](#_2qtun4sa9pkb)

[1-1. システム化の背景と目的 3](#_ajo1gbuf5tjm)

[1-2. システムの全体像 3](#_8rii6xjfqoz5)

[1-3. システム化の範囲 3](#_jea2njk963va)

[**2. 機能・非機能要件 4**](#_52rsdcaglww7)

[2-1. 機能要件 4](#_bou08ugwn3xa)

[2-2. 非機能要件 4](#_msp1j4oy8o51)

[**3. インフラ設計 6**](#_su8kyf2cgo6x)

[3-1. アーキテクチャ 6](#_8l27r8q7c6np)

[3-2. テクノロジースタック 7](#_pll14arvo016)

[3-3. ソーステクノロジスタック 7](#_7afuy5he4f95)

[**4. インフラ構築手順 8**](#_x0x74dxlc704)

[4-1. 前提条件 8](#_qm7ckxz1ilmt)

[4-2. 作業完了条件 8](#_7fqsnl5uaoty)

[4-3. 作業者 8](#_rlz5odyf14pt)

[4-4. 作業場所 8](#_rsc76at5jsuw)

[4-5. 作業端末 8](#_5fyhzse6cpev)

[4-6. 制約事項・備考 8](#_3ti7qwteyf5)

[4-7. 作業実績 8](#_6xzma1tqxfpy)

[4-8. 手順 9](#_hj7avcr2p68c)

[4-8-1. デプロイ環境の構築 9](#_4bhgegz31a17)

[4-8-2. ロギングとモニタリングの設定 10](#_1ikro3mbmf3h)

[4-8-3. インフラコスト管理の設定 12](#_yjgagm2eg3l0)

[4-8-4. セキュリティテストの実施 14](#_9xwxl6ybl2h8)

# 

# **1. 業務要件**

## **1-1. システム化の背景と目的**

　クライアント企業は、「ドローンが空を飛び交い、さまざまな社会課題を解決し、人々の生活や社会を豊かにする」というミッションを掲げ、2016年に創業した、主に産業用途のドローンを開発しているベンチャー企業である。クライアント企業の産業用ドローンの活用の幅は年々広がっており、現在ではインフラ、巡回、物流、農業、災害対策、セキュリティ対策分野で利用され、順調に売上が拡大している。

　その一方で、各分野でドローンに求められるニーズが異なり、カスタマイズ要件が増えている。ドローンの生産では、プラットフォーム（基本的な構成）をベースに追加部品を組み込むことで、各分野のニーズに応える生産方式を採用している。従来導入していたパッケージ型の生産（購買・生産管理・販売管理・在庫管理）システムでは、この生産方式に耐えれない、パッケージをカスタマイズするにしてもコストが莫大にかかることが事前調査で判明し、独自のドローン生産システムを開発することが、クライアント企業内で決定した。

## **1-2. システムの全体像**

## 

## **1-3. システム化の範囲**

本システムの開発範囲は、産業用ドローンの生産システム全体像（購買システム・生産管理システム・販売管理システム・在庫管理システム）のうち、在庫管理システムを新規に開発する。

# **2. 機能・非機能要件**

産業用ドローンを開発するベンチャー企業の生産部品の在庫管理システム新規開発プロジェクトの機能要件と非機能要件である。

## **2-1. 機能要件**

**部品情報管理**

* 部品の一覧表示と詳細情報の閲覧
* 部品の追加、編集、削除機能
* 部品カテゴリーの設定と管理

**在庫管理**

* 部品の入庫および出庫の記録
* 在庫数量の管理
* 在庫の閲覧と検索機能（カテゴリー、部品番号、在庫状況などでのフィルタリング）

**注文処理**

* 部品の注文と受け取りのトラッキング
* 注文履歴の表示と管理
* 注文ステータスの更新
  + 在庫が一定数以下になった場合の自動発注

**ユーザー管理とアクセス制御**

* 管理者と一般ユーザーの役割の設定と管理
* ユーザーごとのアクセス権限の設定（閲覧、編集、削除など）

これらの機能要件を満たすことで、生産部品の在庫管理システムが効率的に運用され、企業の生産プロセスを支援し、品質向上やコスト削減に貢献する。

## **2-2. 非機能要件**

**パフォーマンス要件**

* システムの応答時間：ユーザーの要求に対するシステムの応答時間は2秒以内であること。
* 同時アクセスのサポート：システムは最大100人の同時アクセスをサポートすること。
* データ処理速度：在庫データの更新や検索などのデータ処理は高速かつ効率的に行われること。

**セキュリティ要件**

* アクセス制御：ロールベースのアクセス制御（RBAC）を実装し、ユーザーごとに適切なアクセス権を付与すること。
* データの暗号化：重要なデータはトランジットおよびアットレストで暗号化すること（AES256など）。
* ログと監査：システムへのアクセス、変更、操作などのアクティビティをログとして記録し、適切に監査可能な形式で保持すること。

**可用性と耐障害性**

* システムの可用性：システムは99.9％の可用性を維持すること。
* バックアップと復元：定期的なデータバックアップと災害復旧計画を実施し、データの損失を最小限に抑えること。

**拡張性と保守性**

* システムの拡張性：将来的なシステムの拡張性を考慮し、新しい機能やユーザーの追加が容易に行えるアーキテクチャを採用すること。
* コード品質とドキュメント：コードは適切にコメントされ、保守性が高く、新しい開発者が迅速に理解できるようにすること。

**ユーザビリティ**

* インターフェースの直感性：ユーザーが簡単に操作できる直感的なインターフェースを提供すること。
* エラーハンドリング：エラーが発生した場合には、ユーザーに分かりやすいエラーメッセージを表示し、適切な対処方法を提供すること。

**コスト最適化**

* インフラコストの最適化：インフラのコストを最適化することによりサービスの持続可能性を高めること。

これらの非機能要件を満たすことで、生産部品の在庫管理システムが高品質で信頼性の高いものとなり、ビジネスの運用を円滑に支援できるようになる。

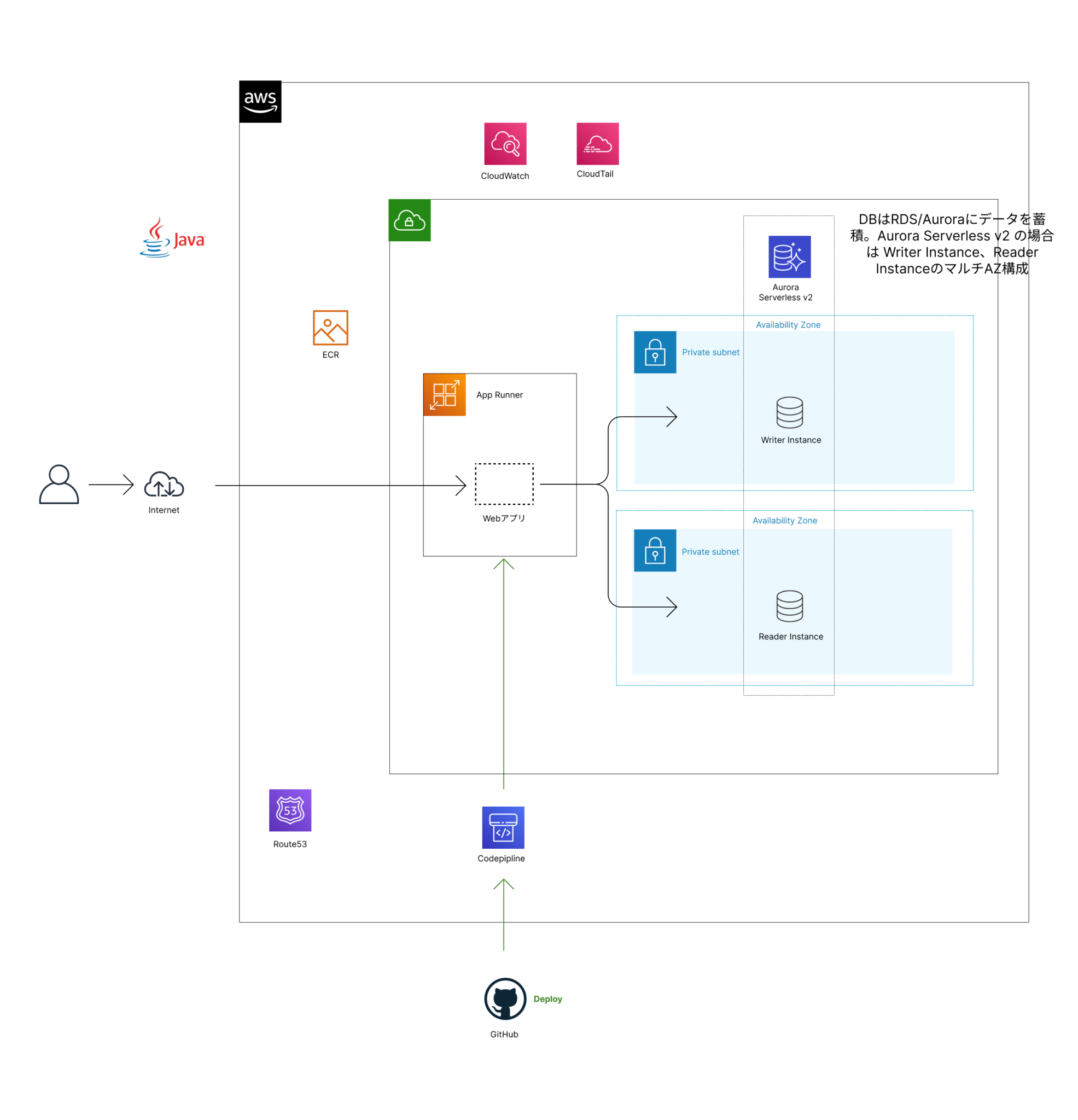
# 

# **3. インフラ設計**

## **3-1. アーキテクチャ**

ドローンベンチャー企業の生産部品の在庫管理システムのAWS構成図である。

Web アプリフルマネージドサービス AWS App Runner を用いて、 Java マイクロサービスをデプロイする。コンテナ管理にAmazon Elastic Container Registry (Amazon ECR) を使用する。



## **3-2. テクノロジースタック**

**App Runner**

サーバーとして、App Runner を用いる。AWS App Runner はフルマネージド型のコンテナアプリケーションサービスであり、インフラストラクチャやコンテナの経験がなくても、ウェブアプリケーションや API サービスを構築、デプロイ、実行できる。

**RDS / Aurora**

データベースとして、RDS / Aurora の PostgreSQL を用いる。

## **3-3. ソーステクノロジスタック**

Java マイクロサービス (Spring Boot で実装されたもの) と Docker にデプロイされたものである。

* Java 11
* Spring 2.7.15



**※ VPNなどのネットワーク周りの設計も記載すべきです。**

**※ アカウント設計もあれば記載すべきです。**

# **4. インフラ構築手順**

## **4-1. 前提条件**

* 作業者のAWSアカウントが発行されていること
* 適切なロールが適用されていること

## **4-2. 作業完了条件**

* 全てのリソースの作成が完了していること
* アプリケーションが正しく動くこと

## **4-3. 作業者**

* 〇〇 〇〇 (xxxxx@dotlife.co.jp)

## **4-4. 作業場所**

* 〒153-0063 東京都目黒区目黒1-4-16

## **4-5. 作業端末**

* Macbook Pro, 2020

## **4-6. 制約事項・備考**

* 特になし

## **4-7. 作業実績**

**作業結果**

* AWS環境上で、正常にアプリケーションが動作することを確認した
* CloudWatch のダッシュボード画面が制作できている
* ダッシュボードには、ECRやRDS/Auroraのデータベースのメトリクスが表示されている
* Cost Explorer でコストを監視、顧客からの要望があればレポート・CSVデータで出力できること
* 予算超過したら管理者にアラートがメールで通知されること
* OWASP ZAPの実施レポートでセキュリティを担保していること

## **4-8. 手順**

### **4-8-1. デプロイ環境の構築**

アーキテクチャに従い、AWSにインフラを構築、App Runner を用いて Java マイクロサービスをデプロイした。App Runnerはフルマネージド型のコンテナアプリケーションサービスであり、簡単に ECS・Fargate で Web アプリを構築できる。

**ソースおよびデプロイ**

1. AWSの管理画面を開いて、App Runnerで「サービスの作成」をする
2. 「ソースコードリポジトリ」を選択してGitHubと連携する
3. GitHub連携の「ソースディレクトリ」は「/drone/dev」を指定する

**デプロイ設定**

1. 「自動」を選択する

**構築を設定**

1. ランタイムは「Corretto 11」を選択する
2. AppRunner 上ではビルドコマンドとスタートコマンドを設定する。

*$ ./gradlew bootJar && cp build/libs/dev-0.0.1.jar ./*

*$ java -jar ./dev-0.0.1.jar*



**期待する成果**

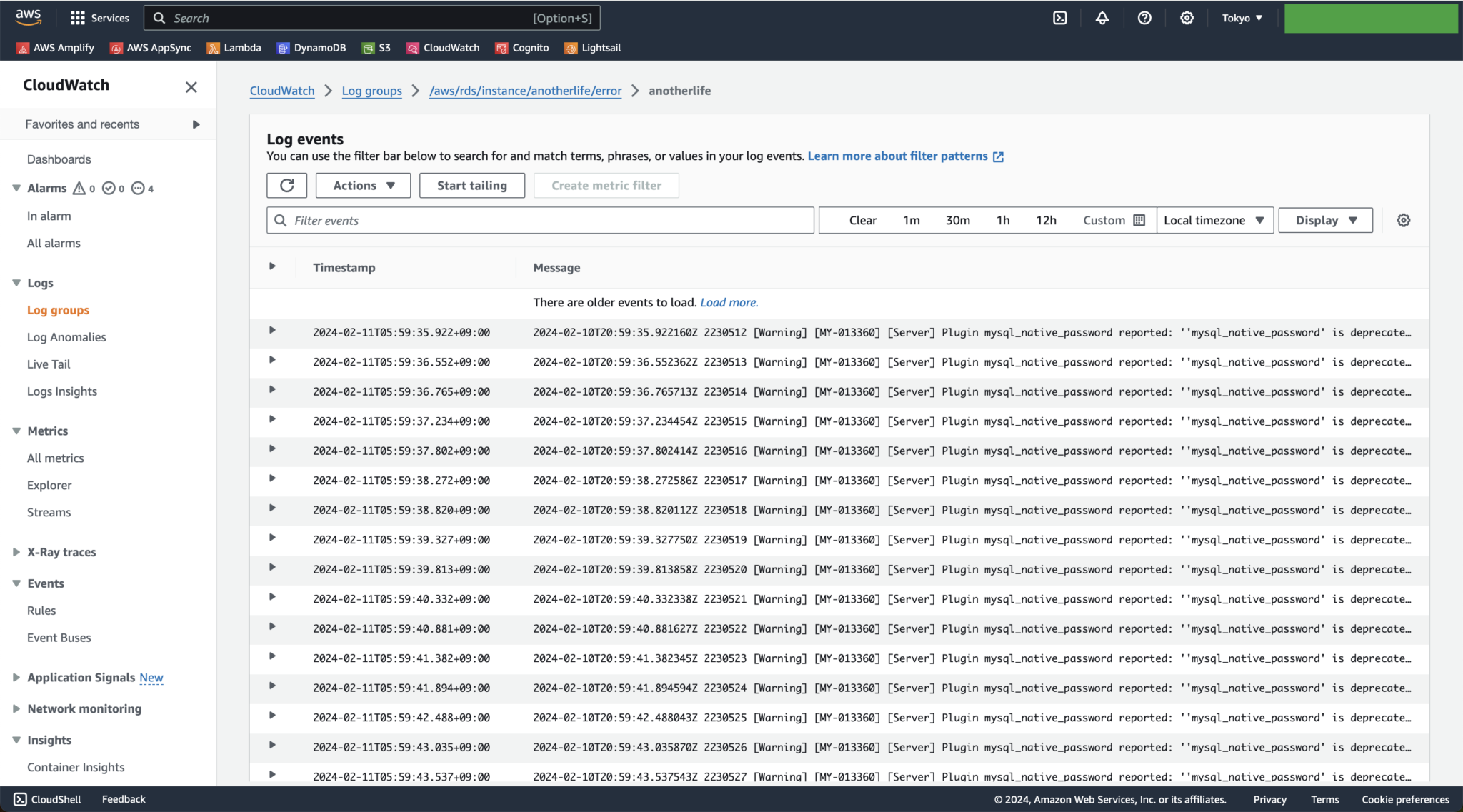
* Java マイクロサービスをデプロイするために、App Runner が構築できていること。
* VPCなど必要なネットワーク設定ができていること。

### **4-8-2. ロギングとモニタリングの設定**

　非機能要件に「ログと監査：システムへのアクセス、変更、操作などのアクティビティをログとして記録し、適切に監査可能な形式で保持すること」とあり、こちらをAWSで実現する。

**CloudWatch による利用ログの監視**

過度な利用やサーバー・データベースの負荷を監視するために CloudWatch でモリタリングし、必要に応じてアラートを通知する。ECSやAurora/RDSなどのログは、左メニューの「ロググループ」で確認できる。

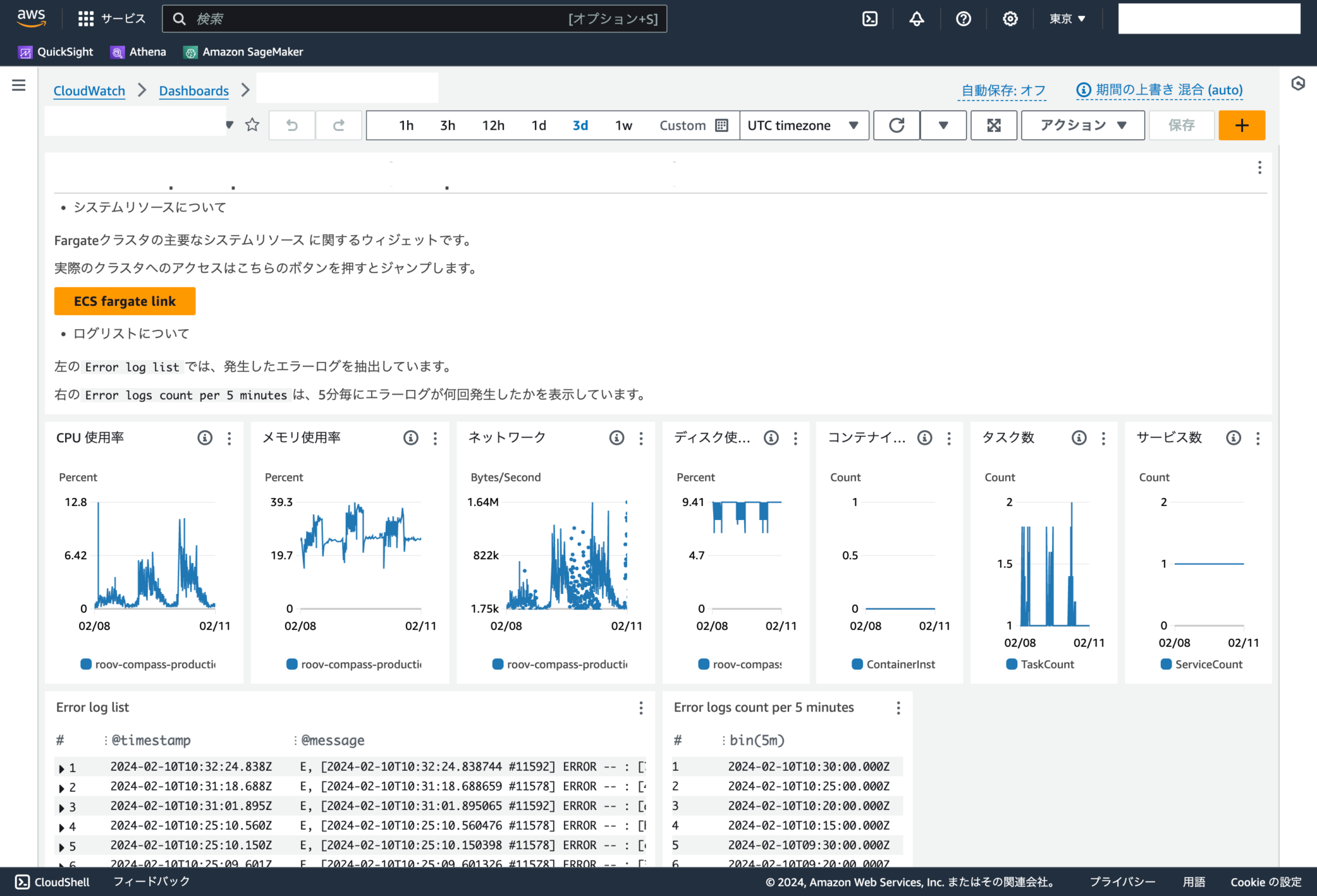


**CloudWatch のダッシュボード作成**

CloudWatch でメトリクスの監視ダッシュボードを作成した。ダッシュボードは、システムリソースメトリクスについてウィジェットに追加した。

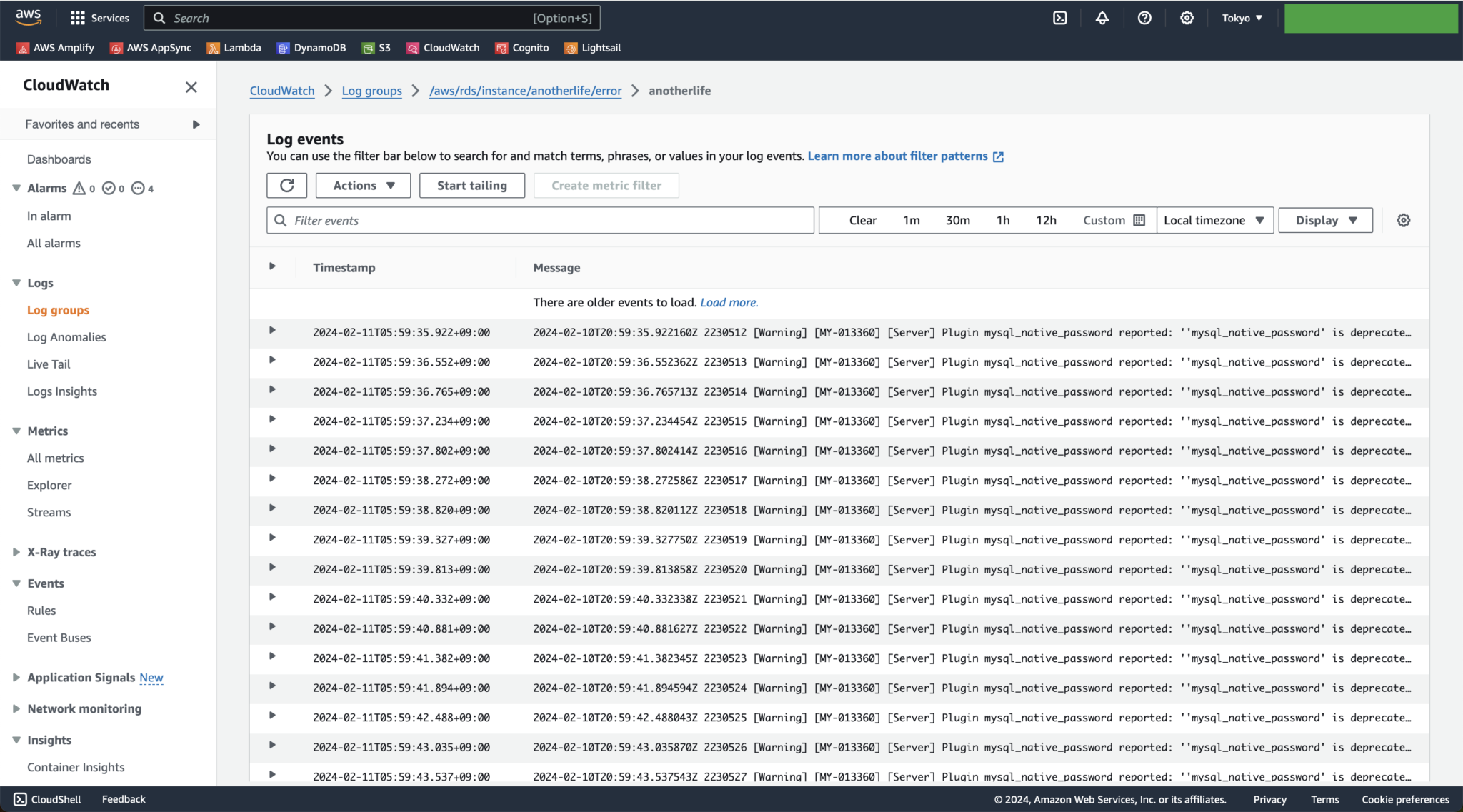
* システムメトリクス
  + CPU
  + Mem
* ログメトリクス
  + エラーログカウント（サービスの健全性の指標になりえる）
  + サービスメトリクス
  + 稼働時間

ログに関する情報もダッシュボードに掲載した。Cloudwatch logsのクエリを使って、例えばエラーログの数を集計できる。



**CloudTrail での監査**

CloudTrail で特定ユーザーの利用ログをモニタリングした。



### **4-8-3. インフラコスト管理の設定**

非機能要件に「インフラコストの最適化：インフラのコストを最適化することによりサービスの持続可能性を高めること」とあり、こちらをAWSで実現した。

**AWS利用費の監視**

サービス運用中、多くのユーザーが利用します。AWSの利用状況を監視するために、AWS Billing and Cost Management を利用した。

**Cost Explorer でレポーティング**

「請求とコスト管理」 > 「コスト分析とレポート」 > 「Cost Explorer」でレポートを作成、保存した。その際、サービス、リージョンで絞り込んだ。データは、レポートとしてCSVで出力できる。



**予算超過アラート**

AWSの予算を設定して、予算超過のタイミングで管理者にメールでアラートを通知した。「請求とコスト管理」 > 「予算」 でアラートを設定した。

※今回は100円など少額にしてください。実務では顧客から予算をヒアリングして設定します。 予算は利用想定ユーザー数などから計算します。



### **4-8-4. セキュリティテストの実施**

品質保証として、セキュリティテストを実施することになりました。Javaのアプリケーションに脆弱性が存在し対策がされているか検証した。

今回は、セキュリティ診断ツール「OWASP ZAP（オワスプ・ザップ）」を用いて、Webアプリケーションの脆弱性をチェックした。