インフラ構築手順書

第1.1版　2025年01月31日

改定履歴

|  |  |  |  |
| --- | --- | --- | --- |
| **版数** | **日付** | **改定内容** | **項番・ページなど** |
| 1.0 | 2024年12月22日 | 初版作成 |  |
| 1.1 | 2025年02月06日 | ECS構築手順を追記  その他、全体の修正 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

[1. 業務要件 2](#_Toc1621267908)

[2. 機能・非機能要件 3](#_Toc748398822)

[2-1. 　機能要件 4](#_Toc1589610749)

[2-2.　非機能要件 4](#_Toc1039301568)

[2-3.　画面イメージ 5](#_Toc996553285)

[3. インフラ設計 6](#_Toc604454083)

[3-1.　アーキテクチャ 7](#_Toc754877847)

[3-2.　テクノロジースタック（App Runner） 8](#_Toc299382185)

[3-3.　ソーステクノロジースタック（AppRunner） 8](#_Toc102686209)

[4. App Runner インフラ構築手順 11](#_Toc216071741)

[4-1.　作業者情報 12](#_Toc2092743634)

[4-2.　作業実績 12](#_Toc1546648250)

[4-3.　 構築手順 12](#_Toc1189808611)

[5. ECS(Fargate)インフラ構築手順 17](#_Toc432970510)

[5-1.　作業者情報 18](#_Toc473569545)

[5-2.　作業実績 18](#_Toc1096423776)

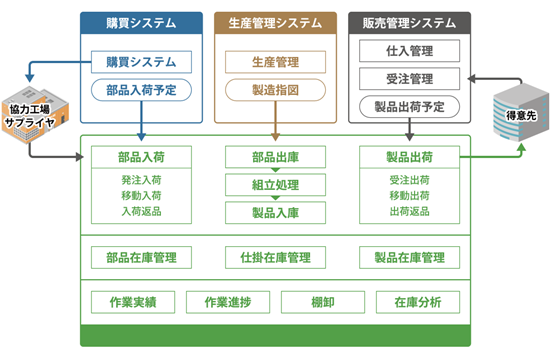
[5-3.　 構築手順 18](#_Toc209430723)

[5-4.　 今後の予定と課題 26](#_Toc508048032)

## 業務要件

産業用ドローンの需要拡大に伴い、各分野で異なるニーズに応えるためのカスタマイズ要件が求められています。しかし、従来のパッケージ型生産システムでは対応が困難であると判明したため、新たに独自のドローン生産システムを開発することが決定されました。本プロジェクトでは、その生産システムの一部である在庫管理システムを構築します。

本手順書では AWS App Runnerの構築、ロギングとモニタリングの設定、 インフラコスト管理の設定、 ECSを用いたサーバ構築を行います。

  
Fig. 1 システムの全体像

## 機能・非機能要件

### 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）

### 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。

### 2-3.　画面イメージ



Fig. 2 画面イメージ

## インフラ設計

### 3-1.　アーキテクチャ

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。

または、ECS(Fargate)を利用する。App Runnerでは OSやランタイムの自由度が低く リソース設定やネットワーク設定が限定的なため、ECSを利用することでより柔軟性の高い構築が可能。

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

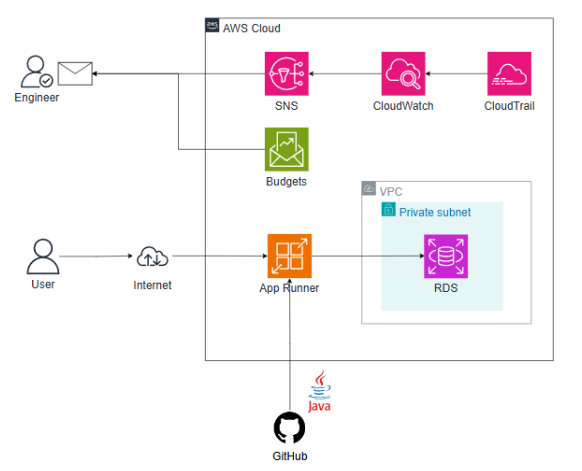
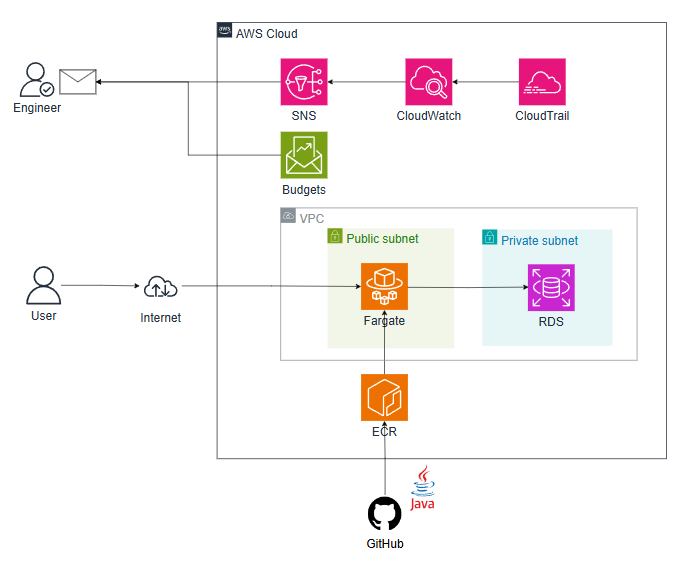


Fig. 3 App Runner構成図

Fig. 4 ECS(Fargate)構成図

### 3-2.　テクノロジースタック（App Runner）

Table 1 テクノロジースタック（ App Runner）

|  |  |  |
| --- | --- | --- |
| カテゴリ | サービス名 | 用途 |
| アプリケーション | App Runner | コンテナビルド・コンテナイメージ生成・デプロイ |
| リポジトリ | GitHub | ソースコードの管理 |
| データベース | RDS/Aurora (PostgreSQL) | データベースとして利用 |
| 監視 | CloudWatch | メトリクス監視・ログ管理 |
| 監査 | AWS CloudTrail | アクティビティログの記録 |
| アラート通知 | Amazon SNS | リソース超過時の通知 |

### 3-3.　ソーステクノロジースタック（AppRunner）

* アプリケーション
  + Java 11
  + Spring 2.7.15
  + Docker
* アカウント設計

Table 2 アカウント設計（AppRunner）

|  |  |
| --- | --- |
| 役職 | IAMロール |
| 開発者 | ReadOnlyAccess + 限定的なデプロイ権限 |
| 管理者 | AdministratorAccess |
| 運用者 | ReadOnlyAccess + CloudWatchFullAccess + AWSAppRunnerFullAccess |

* CloudWatch 監視項目
  + システムメトリクス
    - CPU
    - Mem
  + ログメトリクス
    - ログカウント
  + サービスメトリクス
    - HTTPリクエスト数
    - HTTPステータスコード別レスポンス数
    - レイテンシー
  + CloudWatch logs
    - エラーログ抽出
* メトリクスアラーム
  + CPU 使用率 が80%を超過した場合 、SNS で通知

3-4.　テクノロジースタック（ ECS ）

Table 3 テクノロジースタック（ECS）

|  |  |  |
| --- | --- | --- |
| カテゴリ | サービス名 | 用途 |
| アプリケーション | ECS (Fargate) | コンテナのデプロイ・管理 |
| リポジトリ | ECR  GitHub | コンテナイメージの保存  ソースコードの管理 |
| データベース | RDS/Aurora (PostgreSQL) | データベースとして利用 |
| シェル | CloudShell | ソースコードからコンテナをビルドしイメージをプッシュ |
| ネットワーク | VPC | 仮想ネットワーク |

3-5.　ソーステクノロジースタック（ ECS ）

* アプリケーション
  + Java 11
  + Spring 2.7.15
  + Docker
* アカウント設計

Table 4 アカウント設計（ECS）

|  |  |
| --- | --- |
| 役職 | IAMロール |
| 開発者 | ReadOnlyAccess + 限定的なデプロイ権限 |
| 管理者 | AdministratorAccess |
| 運用者 | ReadOnlyAccess + CloudWatchFullAccess + AmazonECS\_FullAccess |

* VPC

Table 5 VPC

|  |  |
| --- | --- |
| 項目 | 設定内容 |
| VPC CIDR | 10.0.0.0/20 |
| パブリックサブネット | 10.0.0.0/24、10.0.1.0/24 |
| インターネットゲートウェイ | あり |
| セキュリティグループ設計 | 80、8080 (全開放) |

## App Runner インフラ構築手順

### 4-1.　作業者情報

氏名：佐藤

連絡先：satoushouta1205@gmail.com

### 4-2.　作業実績

工数：7h

結果：正常完了

### 4-3.　 構築手順

* AWSリソースの命名規則

下記の命名規則に従って構築する。

{sysname}-{env}-{user}-{service}-{予備}

Table 6 リソースの命名規則

|  |  |
| --- | --- |
| 要素 | 詳細 |
| sysname | 固定値として **drone** を使用 |
| env | 固定値として **dev** を使用 |
| user | （個別）**IAM user名** |
| service | （個別）**対象サービス** |
| 予備 | 一意にできない場合に使用 |

* タグの命名規則

作成したリソースに以下の命名規則でタグ付けする。

Table 7 タグの命名規則

|  |  |
| --- | --- |
| キー | 値 |
| Cost | drone\_**IAM user名** |
| Project | （固定）**infra-course-drone** |
| Name | **リソース名** |
| createdBy | （個別）**IAM user名** |

1. AWS App Runner の構築
   1. App Runner で「サービスの作成」をクリック
   2. ソースおよびデプロイ

「ソースコードリポジトリ」を選択してGitHubと連携

GitHub連携の「ソースディレクトリ」は「/dev」を指定

デプロイ設定は「自動」を選択

* 1. 構築を設定

ランタイムは「Corretto 11」を選択

「構築コマンド」を入力

|  |
| --- |
| $ mvn clean package |

「開始コマンド」を入力

|  |
| --- |
| $ java -Xms256m -jar target/dev-0.0.1.jar |



Fig. 5 App Runner

* 1. サービスを設定

命名規則に従い「サービス名」を入力し、「タグ」を設定

1. CloudWatch ダッシュボードの作成
   1. CloudWatch > ダッシュボード から「ダッシュボードの作成」をクリック

「ダッシュボード名」を入力して作成

* 1. 「ウィジェットの追加」から「線またはスタックされたエリア」を選択

必要なメトリクスを選択し、ダッシュボードにウィジェットを追加

下記の各メトリクス分、操作を繰り返す

AppRunner >インスタンスメトリクス＞ CPUUtilization

AppRunner >インスタンスメトリクス＞ MemoryUtilization

ログ > ロググループメトリクス> IncomingLogEvents

AppRunner >サービスのメトリクス＞ Requests

AppRunner >サービスのメトリクス＞ 2xxStatusResponses

AppRunner >サービスのメトリクス＞ 4xxStatusResponses

AppRunner >サービスのメトリクス＞ 5xxStatusResponses

AppRunner >サービスのメトリクス＞ RequestLatency

* 1. ログに関する情報をダッシュボードに掲載する

「ウィジェットの追加」からデータ型「ログ」を選択

クエリを下記に変更し、ダッシュボードに追加

|  |
| --- |
| fields *@timestamp*, *@message*, *@logStream*, *@log*  | filter *@message* like /PAUSED/ or *@message* like /pause/  | sort *@timestamp* desc  | limit 100 |

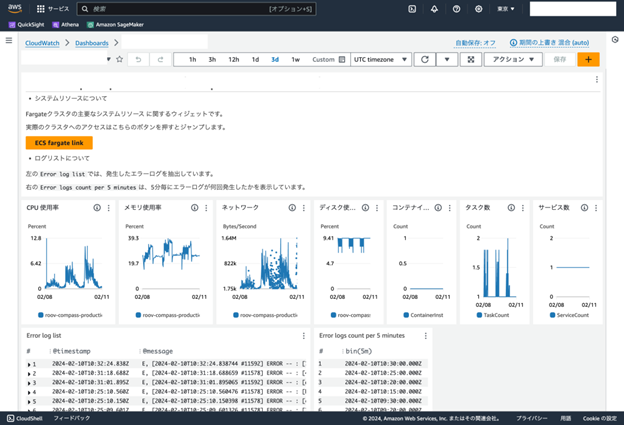


Fig. 6 ダッシュボード

1. CloudWatch Alarmの作成
   1. CloudWatch > すべてのアラーム から「アラームの作成」をクリック
   2. 監視するCPUメトリクスを選択しアラートを作成する

CPUの閾値設定は以下とする

Table 8 CPU閾値設定

|  |  |
| --- | --- |
| 項目 | 設定内容 |
| 閾値 | 80%より大きい |
| データ取得間隔 | 5分 |
| アラームを発生させるデータポイント数 | 3 |
| 評価期間数 | 3 |

* 1. SNS(Simple Notification Service)を利用し、自身のメールへ通知が来るように設定を行う

設定を作成後、「AWS Notification - Subscription Confirmation」という件名でAWSより認証メールが届くため承認する

1. CloudTrail での監査
   1. CloudTrail で特定ユーザーの利用ログを確認する

CloudTrail > イベント履歴 から、「[ルックアップ属性] ： ユーザー名」として、利用ログを確認する

1. インフラコスト管理の設定
   1. AWSの利用状況を監視する

AWS Billing and Cost Management を利用する

* 1. 「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、保存する
  2. 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する

「請求とコスト管理」 > 「予算」 でアラートを月次コスト予算を設定する



Fig. 7 Cost Explorer

## ECS(Fargate)インフラ構築手順

### 5-1.　作業者情報

氏名：佐藤

連絡先：satoushouta1205@gmail.com

### 5-2.　作業実績

工数：8h

結果：正常完了

### 5-3.　 構築手順

* AWSリソースの命名規則

4-3と同じとする

* タグの命名規則

4-3と同じとする

1. ECRリポジトリ作成
   1. AWSマネジメントコンソールからECSを検索
   2. 左側のメニューから「Amazon ECR」を選択
   3. 右側の「リポジトリの作成」をクリック
   4. プライベートリポジトリ作成画面

命名規則に従い「リポジトリ名」を入力し、「作成」をクリック

（※暗号化設定はデフォルトのAES-256とする）



Fig. 8 リポジトリ作成画面

* 1. 次の作業で使用するプッシュコマンドを表示

作成したリポジトリを選択し、「プッシュコマンドの表示」をクリック



Fig. 9 プッシュコマンド表示

1. CloudShellでコンテナビルドしイメージをECRにプッシュ
   1. AWSマネジメントコンソールから CloudShell を検索して起動
   2. GitHubリポジトリをクローンするコマンドを入力し、

Dockerfileがあるリポジトリに移動

|  |
| --- |
| $ git clone https://github.com/satoushouta1205/infra-course-drone.git  $ cd infra-course-drone |

* 1. ECRの「プッシュコマンドを表示」で表示された

macOS / Linux用コマンドの手順に従ってCloudShell へ入力

|  |
| --- |
| //1.認証とログイン $ aws ecr get-login-password --region ap-northeast-1 | docker login --username AWS --password-stdin 975050087718.dkr.ecr.ap-northeast-1.amazonaws.com  //2.Dockerイメージのビルド  $ docker build -t drone-dev-sato-ecr .  //3.イメージにタグ付け  $ docker tag drone-dev-sato-ecr:latest 975050087718.dkr.ecr.ap-northeast-1.amazonaws.com/drone-dev-sato-ecr:latest  //4.イメージをECRにプッシュ  $ docker push 975050087718.dkr.ecr.ap-northeast-1.amazonaws.com/drone-dev-sato-ecr:latest |

* 1. ECRでプッシュしたイメージのURIをコピーして控えておく



Fig. 10 イメージURI

1. VPCの作成
   1. VPC＞お使いのVPC から「VPCを作成」をクリック

下記の通り設定し、作成する

Table 9 VPC設定

|  |  |
| --- | --- |
| 項目 | 設定内容 |
| 作成するリソース | VPCなど |
| 名前タグの自動生成 | チェックを入れ、命名規則に従い入力 |
| IPv4 CIDRブロック | 10.0.0.0/20 |
| アベイラビリティゾーンの数 | 2 |
| パブリックサブネットの数 | 2 |
| プライベートサブネットの数 | 0 |
| パブリックサブネットCIDRブロック | 10.0.0.0/24  10.0.1.0/24 |
| NATゲートウェイ | なし |
| VPCエンドポイント | なし |

* 1. VPC＞セキュリティグループ から

「セキュリティグループを作成」をクリック

下記の通り設定し、作成する

Table 10 セキュリティグループ設定

|  |  |
| --- | --- |
| 項目 | 設定内容 |
| セキュリティグループ名 | 命名規則に従い入力 |
| 説明 | any port 8080 |
| VPC | 作成したVPCを選択 |
| インバウンドルール | カスタムTCP/8080ポートを全開放 |



Fig. 11 セキュリティグループの作成画面

1. ECSでのデプロイ設定
   1. ECS > クラスター から、「クラスターの作成」をクリック

クラスター名を入力し、作成する

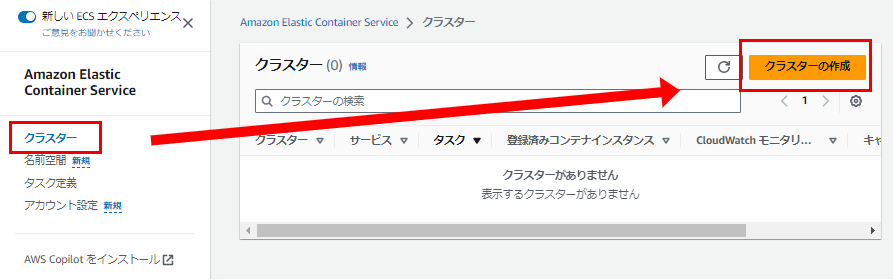


Fig. 12 クラスターの作成

* 1. ECS > タスク定義 から、「新しいタスク定義の作成」をクリック



Fig. 13 新しいタスク定義の作成

* 1. 新しいタスク定義の作成画面

以下の通り設定し、作成する

・タスク定義ファミリー：命名規則に従い入力

・インフラストラクチャの要件

Table 11 インフラストラクチャの要件の設定

|  |  |
| --- | --- |
| 項目 | 設定内容 |
| 起動タイプ | AWS Fargate |
| CPU | .5 vCPU |
| メモリ | 1GB |

・コンテナ

Table 12 コンテナの設定

|  |  |
| --- | --- |
| 項目 | 設定内容 |
| 名前 | 命名規則に従い入力 |
| イメージURI | ECRにプッシュしたイメージのURIを入力 |
| ポートマッピング | 80（デフォルト） |

* 1. タスク定義を元にサービスを作成

作成したクラスターを開き、サービスタブから「作成」をクリック



Fig. 14 サービスの作成

* 1. 作成画面

下記の通り設定し、作成する

・デプロイ設定

Table 13 デプロイ設定

|  |  |
| --- | --- |
| 項目 | 設定内容 |
| アプリケーションタイプ | サービス |
| タスク定義 | 作成したファミリーを選択 |
| サービス名 | 命名規則に従い入力 |

・ネットワーキング

Table 14 ネットワーキング設定

|  |  |
| --- | --- |
| 項目 | 設定内容 |
| VPC | 作成したVPCを選択 |
| セキュリティグループ | 作成したセキュリティグループのみ選択 |

1. ECSのアプリケーションにアクセス
   1. 作成されたサービスのタスクからパブリックIPを確認



Fig .15 パブリックIP

* 1. 下記URLにアクセスし、アプリケーションが表示されることを確認

http://<パブリックIP>:8080/

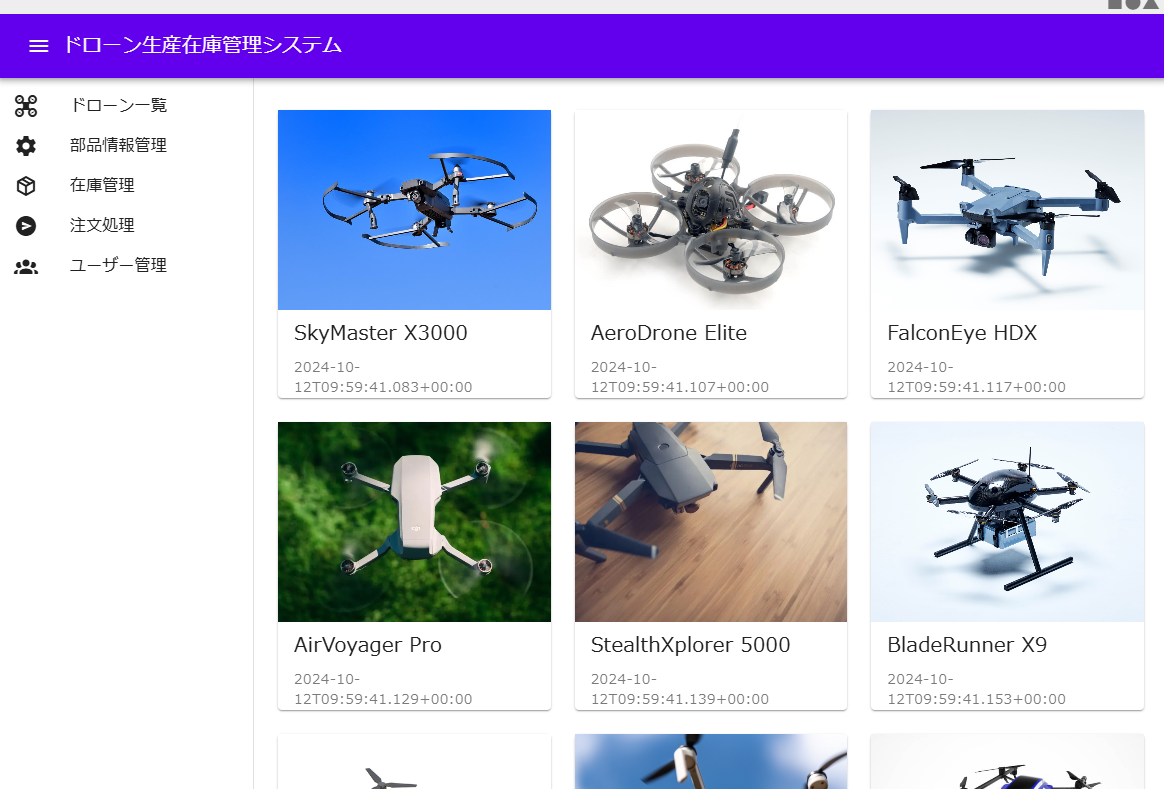


Fig. 16 ブラウザ画面

### 5-4.　 今後の予定と課題

今後の予定

* CloudWatch にて適切な監視運用の設定する
* 監査ログは CloudTrailを確認する
* Cost Explorerでのコスト分析を行う
* セキュリティ強化のため HTTPS/SSLの設定
  + AWS Certificate Manager (ACM) を利用してSSL/TLS証明書を取得し、ロードバランサーでHTTPSを有効にする
* 任意のドメイン名を取得し、DNSの設定
  + ドメインを取得し、Amazon Route 53 でDNSホスティングを設定
  + ドメインまたはサブドメインに対して、AレコードまたはCNAME（/エイリアス）レコードを作成し、ECSサービスのパブリックIPまたは、ロードバランサーのDNS名を指定する

課題

* パフォーマンス要件を満たす適切なリソース設定の調査が必要
  + シミュレーション環境や負荷テストツール（例：Apache JMeter、Locustなど）を用いて、実際のユーザーアクセスを模したテストを実施し、応答時間やスループットを測定する
* 99.9％の可用性を維持するための運用整備が必要
  + 自動スケーリング設定や、適切な監視設定、アラート設定を行う
  + 定期的な障害復旧テストを行う
* バックアップとDR計画の策定が必要
  + AWS Backupや、各サービス（RDS、EBS、DynamoDB、S3など）が提供するバックアップ機能を活用し、スケジューリングする
  + 必要に応じて別のリージョンへバックアップをレプリケートする
* データの適切な暗号化が必要
  + Githubのソースコードは非公開リポジトリ上に配置する
  + CloudShell上の一時データは明示的に暗号化されているか不明瞭のため、利用に問題がないか検討が必要
  + 今後他サービスを利用する場合においても適切な暗号化設定を行う

以上