

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	3
2. 機能・非機能要件	4
2-1. 機能要件	4
2-2. 非機能要件	4
2-3. 画面イメージ	5
3. インフラ設計.....	7
3-1. アーキテクチャ.....	7
3-2. テクノロジースタック (App Runner)	8
3-3. ソーステクノロジースタック (AppRunner)	9
4. App Runner インフラ構築手順.....	12
4-1. 作業者情報	12
4-2. 作業実績.....	12
4-3. 構築手順.....	12
5. ECS(Fargate)インフラ構築手順.....	18
5-1. 作業者情報	18
5-2. 作業実績.....	18
5-3. 構築手順.....	18
5-4. 今後の予定と課題	26

1. 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。

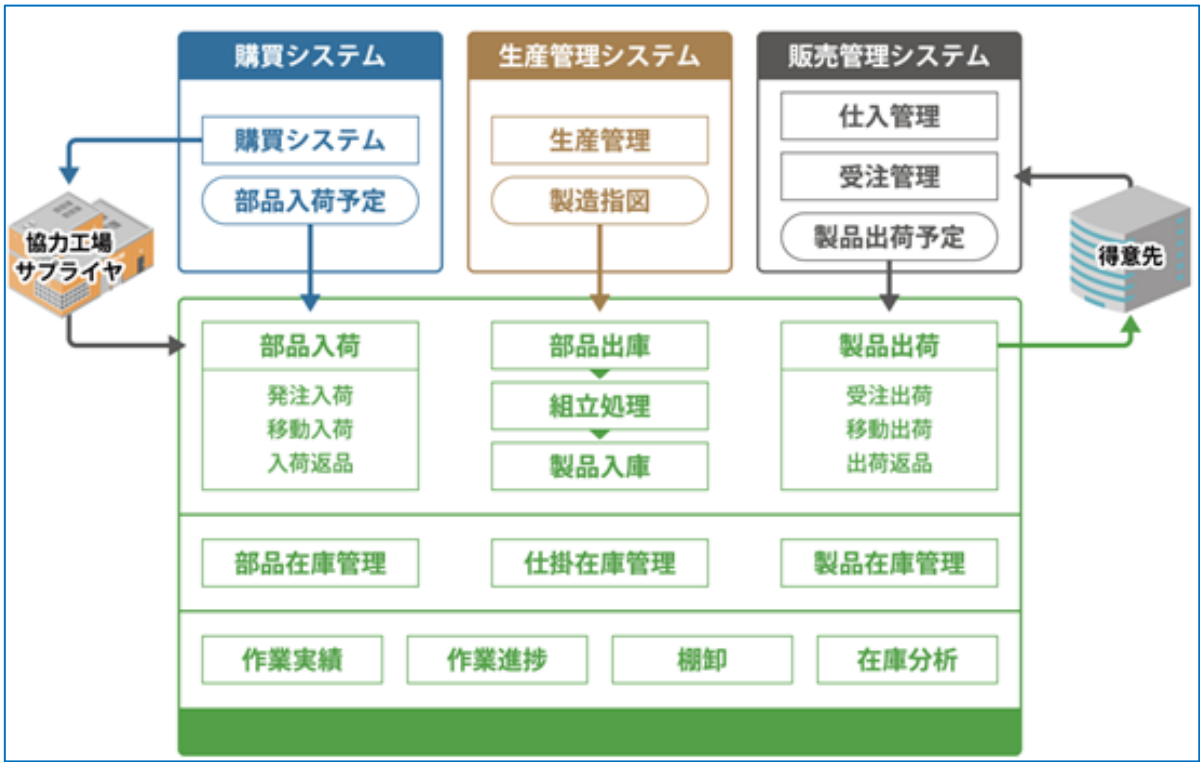


Fig. 1 システムの全体像

2. 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

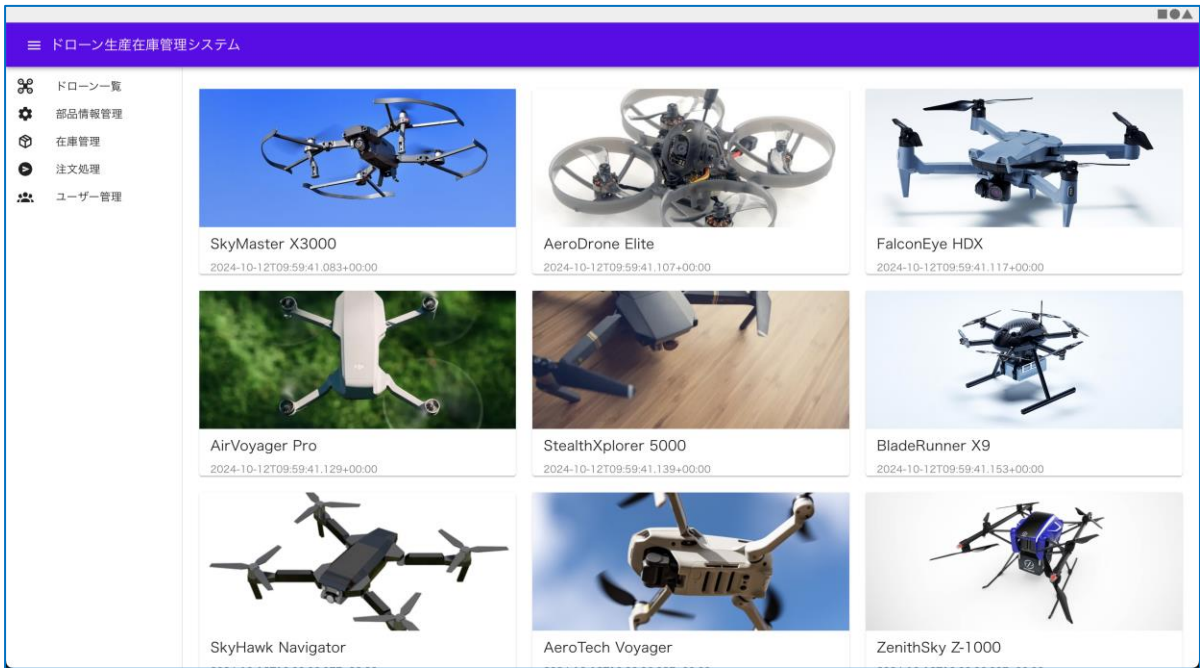


Fig. 2 画面イメージ

3. 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。

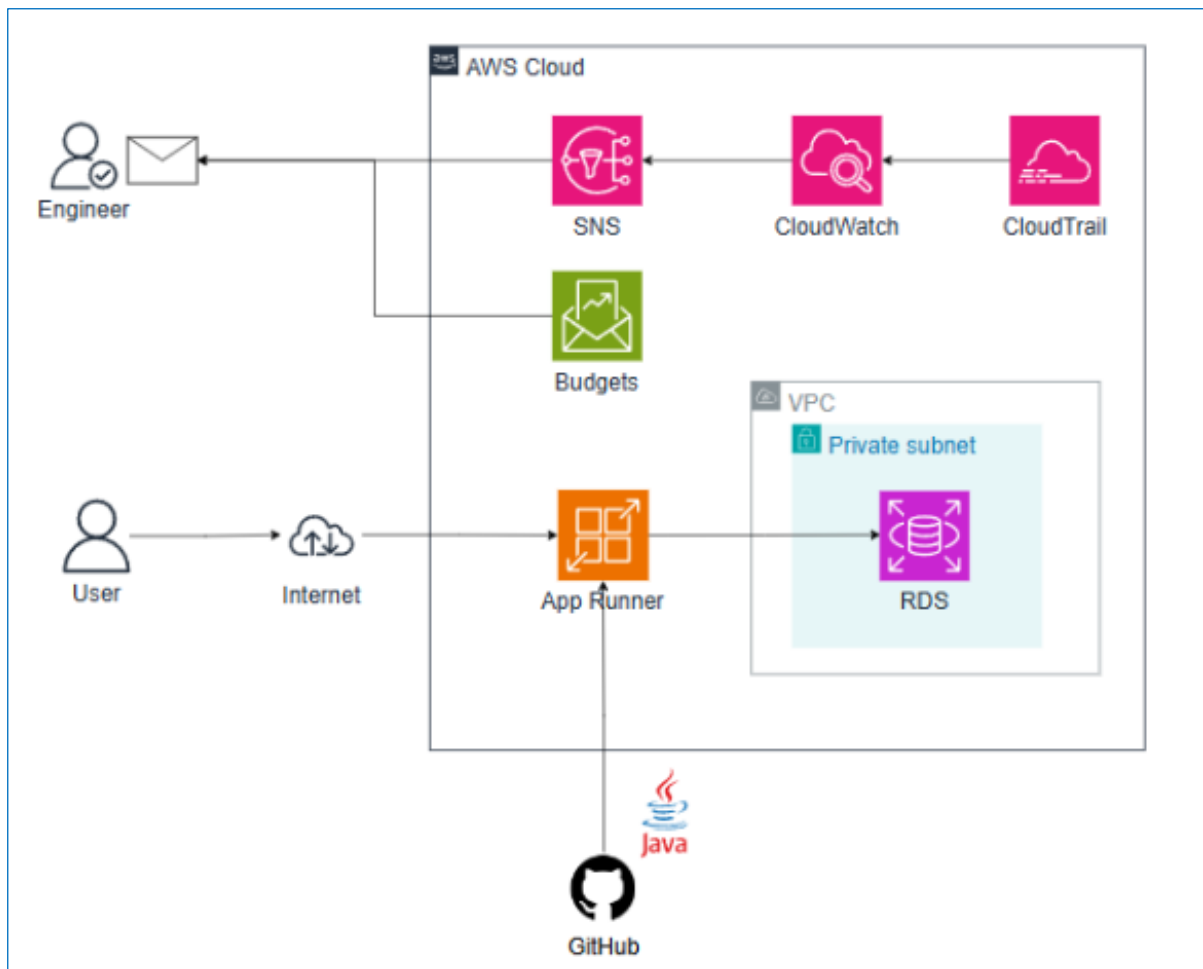


Fig. 3 App Runner 構成図

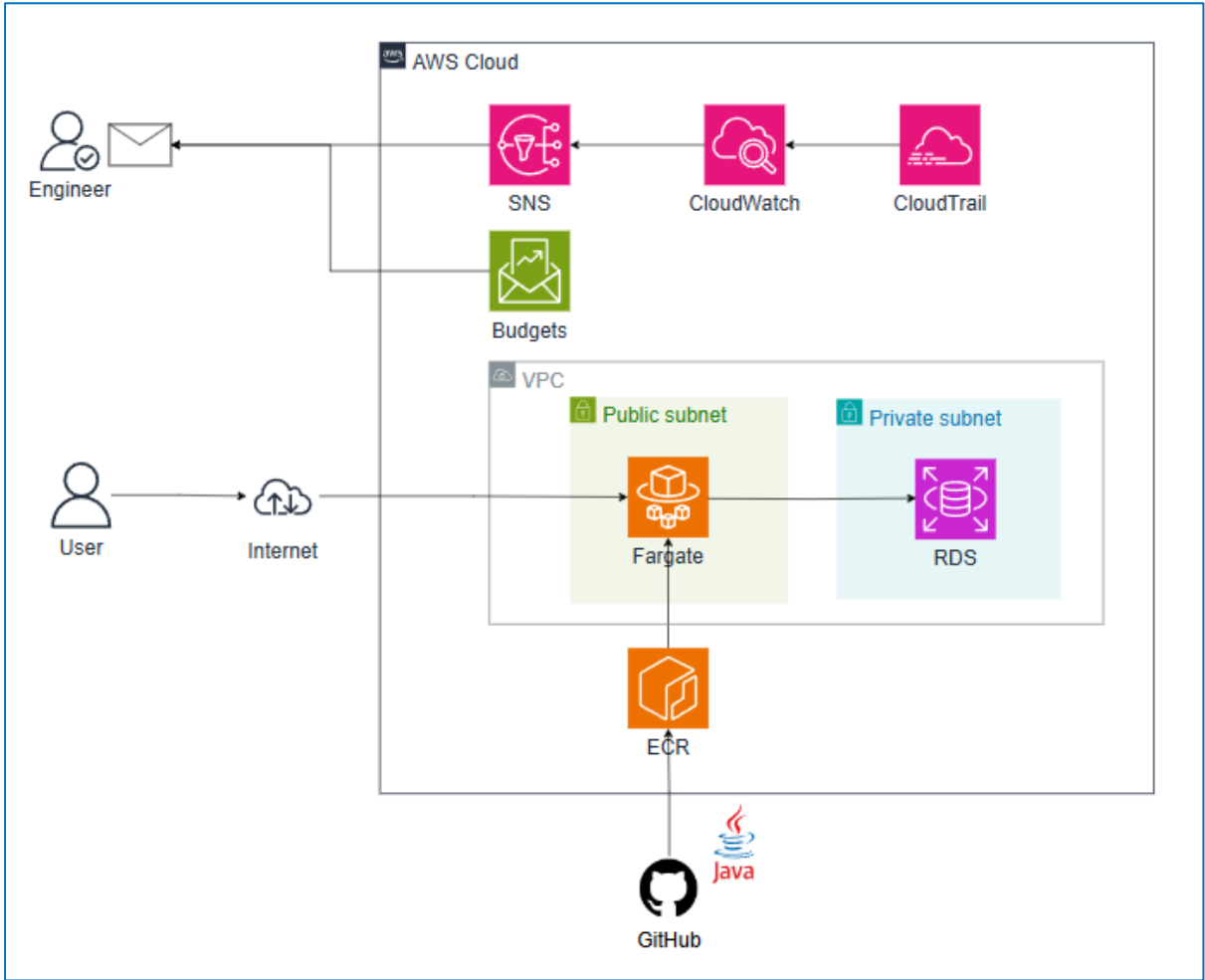


Fig. 4 ECS(Fargate)構成図

3-2. テクノロジスタック (App Runner)

Table 1 テクノロジスタック (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)

4. App Runner インフラ構築手順

4-1. 作業情報

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

3. 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
```

設定ファイル

☒ ここですべての設定を構成する
App Runner コンソールで、サービスのすべての設定を指定します。

☐ 設定ファイルを使用
App Runner がコードリポジトリのソースディレクトリにある `apprunner.yaml` ファイルから設定を読み取るようにします。前のステップでソースディレクトリが指定されていない場合、App Runner はデフォルトでルートディレクトリになります。

ランタイム
サービスの App Runner ランタイムを選択します。

Corretto 11 ▼

構築コマンド
このコマンドは、新しいコードバージョンがデプロイされると、リポジトリのソースディレクトリで実行されます。これを依存関係のインストールやコードのコンパイルに使用します。前のステップでソースディレクトリが指定されていない場合、App Runner はデフォルトでルートディレクトリになります。

mvn clean package

開始コマンド
このコマンドは、サービスのソースディレクトリで実行され、サービスプロセスを開始します。このコマンドを使用して、サービス用の Web サーバーを起動します。このコマンドは、App Runner とユーザーが定義した環境変数にアクセスできます。前のステップでソースディレクトリが指定されていない場合、App Runner はデフォルトでルートディレクトリになります。

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

Fig. 5 App Runner

4. 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

(2) 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

2. 「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

3. 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
```

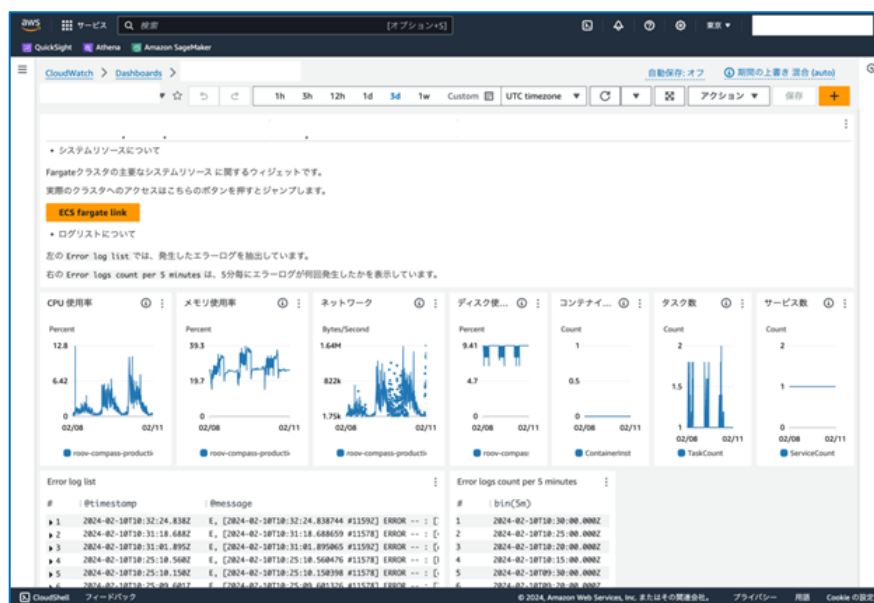


Fig. 6 ダッシュボード

(3) 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

3. 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

(4) 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

(5) 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

2. 「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
3. 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

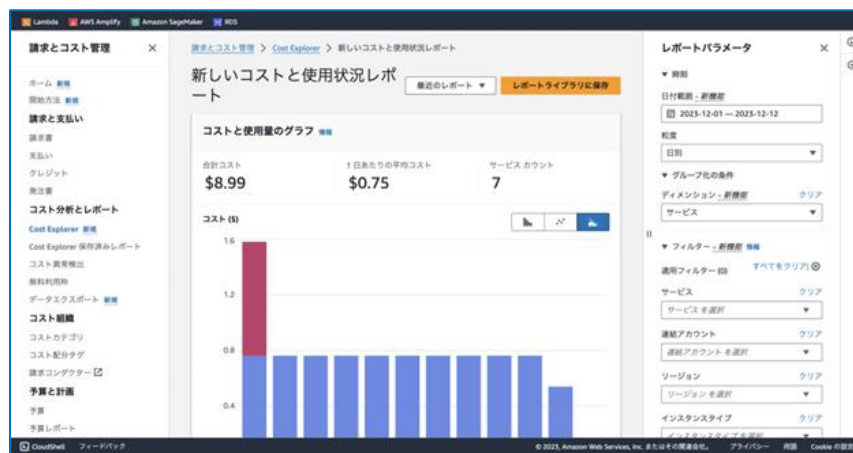


Fig. 7 Cost Explorer

5. 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)

Amazon ECR > プライベートレジストリ > リポジトリ > プライベートリポジトリを作成する

プライベートリポジトリを作成する

一般設定

リポジトリ名
既定の名前を生成していただくか、リポジトリ名は名前空間をリポジトリとして使用する。既定のリポジトリをグループ化する場合は推奨されます。
975050087718.dkr.ecr.ap-northeast-1.amazonaws.com/nameapp/app-name

最大文字数: 256 文字まで (256 文字以内) 必須は文字で指定する必要があります。小文字、数字、破折部、下線、およびハイフン (-) のみを含めることができます。

イメージタグのミュータビリティ [情報](#)

☒ Mutable
使用するタグのミュータビリティを設定します。リポジトリ内のイメージタグはリポジトリ内では変更可能で、タグは上書きされます。

☐ Immutable
イメージタグは上書きされません。

暗号化設定

リポジトリを作成すると、リポジトリの暗号化設定を変更することはできません。

暗号化設定 [情報](#)

デフォルトでは、リポジトリは業界標準の Advanced Encryption Standard (AES) 暗号化を使用します。オプションで、AWS Key Management Service (KMS) に提供されているキーを使用して、リポジトリ内のイメージを暗号化することもできます。

☒ AES-256
業界標準の Advanced Encryption Standard (AES) 暗号化

☐ AWS KMS
AWS Key Management Service (KMS)

▶ イメージのスキャン設定 - deprecated

キャンセル 作成

Fig. 8 リポジトリ作成画面

5. 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

プライベートリポジトリ (1)

[プッシュコマンドを表示](#) [削除](#) [アクション](#) [リポジトリを作成](#)

Q リポジトリの部分文字列で検索

リポジトリ名	URI	作成時刻	タグのイミュータビリティ	暗号化タイプ
drone-dev-sato-ecr	975050087718.dkr.ecr.ap-northeast-1.amazonaws.com/drone-dev-sato-ecr	2025年1月20日, 22:00:49 (UTC+09)	Mutable	AES-256

Fig. 9 プッシュコマンド表示

(2) CloudShell でコンテナビルドイメージを 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
```

3. 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
```

4. 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

(3) 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

2. 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

(4) 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

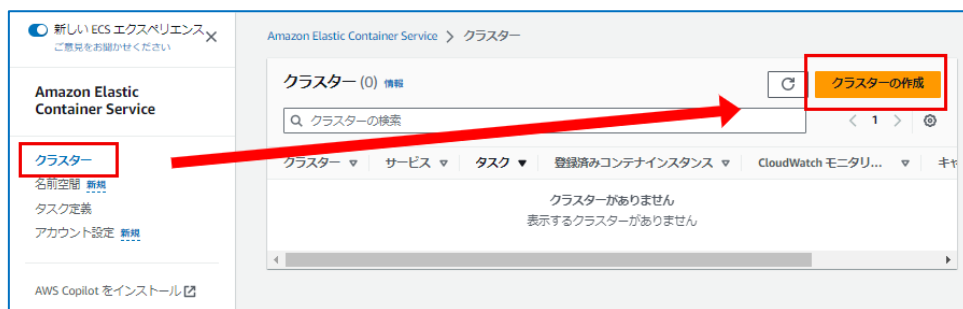


Fig. 12 クラスターの作成

2. ECS > タスク定義 から、「新しいタスク定義の作成」をクリック

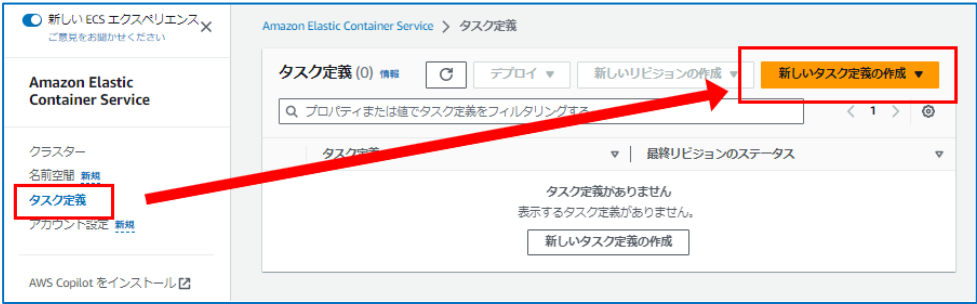


Fig. 13 新しいタスク定義の作成

3. 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)

4. 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

5. 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

(5) 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

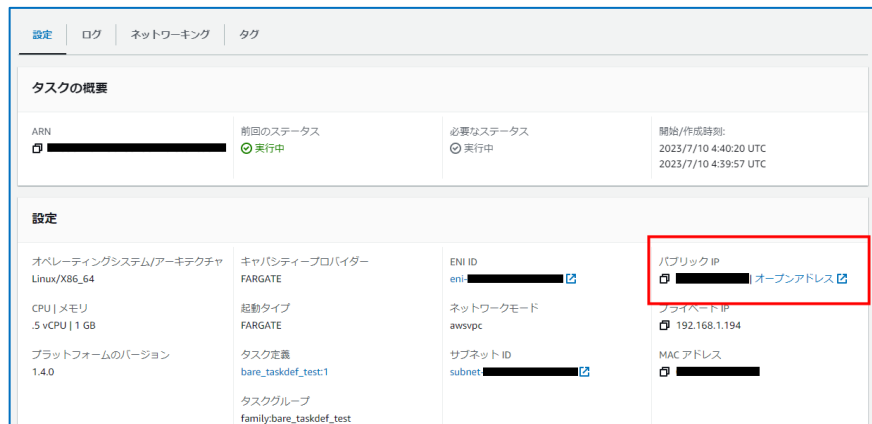


Fig.15 パブリック IP

2. 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/

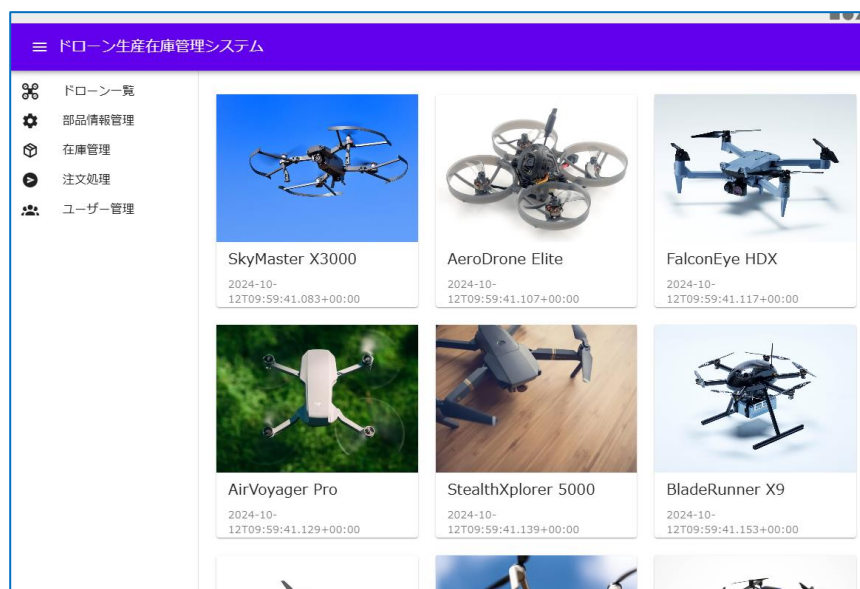


Fig. 16 ブラウザ画面

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