

AWS 秋の COST OPTIMIZATION 祭り 2025

コンテナに移行したらコストが増えた！？ ECS のコスト最適化に向けて 改めて確認したいポイント

Kenichi Azuma

Amazon Web Services Japan G.K.
Solutions Architect



自己紹介

東 健一（あずま けんいち）

アマゾンウェブサービスジャパン

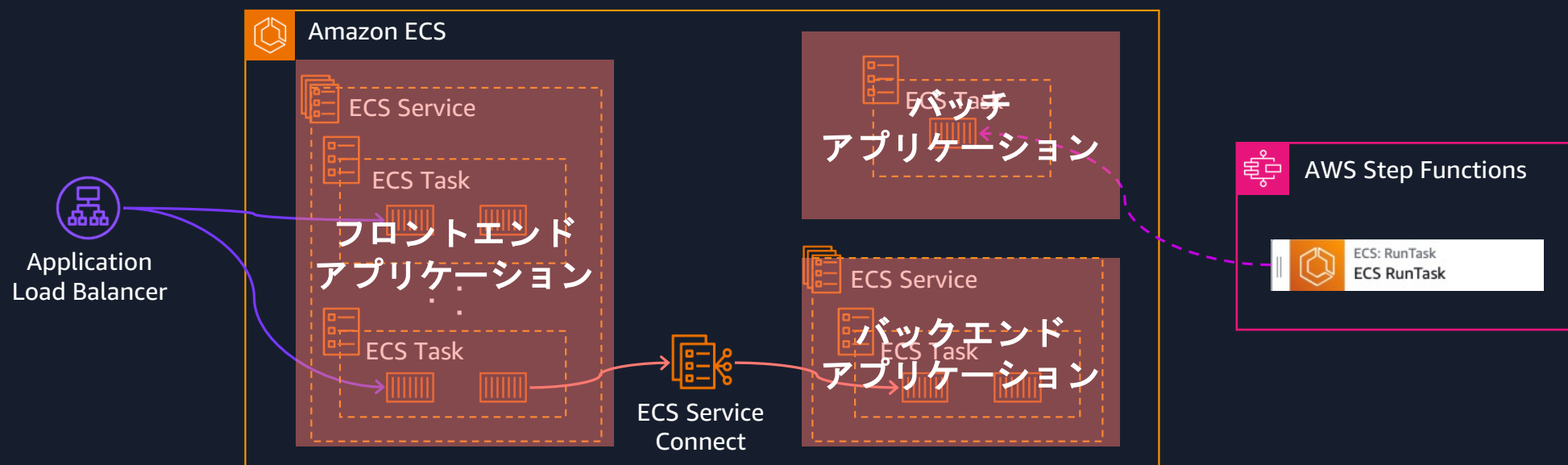
パブリックセクター ソリューションアーキテクト

- 中央省庁のお客様を中心に
ガバメントクラウド、医療DXに関するご支援
- 社内 Tech コミュニティ（[TFC](#)）における
コンテナサービスの Japan Lead（ECS Focus）



はじめに

- ECS を用いて様々なユースケースにおいてコンテナを実行することが可能です。
- コンテナの移行によってアジリティやスケーラビリティなどのメリットを享受している一方でコスト面で従来よりも課題を感じているというケースを聞きます。
- 本発表では直近の Amazon ECS のサービスアップデートも紹介しつつ、コストを最適化するためのポイントを一部ご紹介させていただきます。

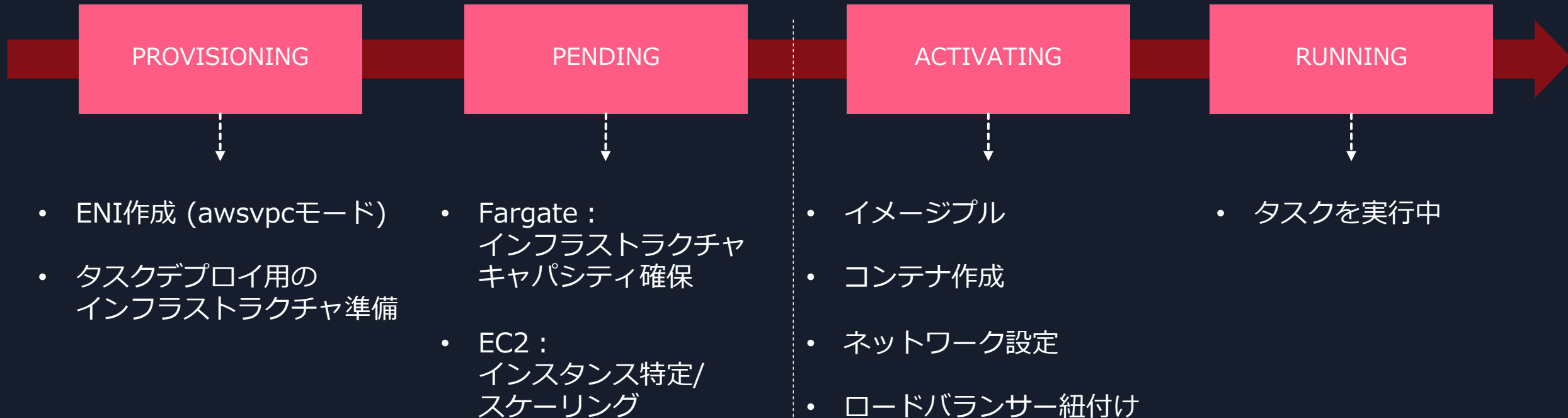


目次

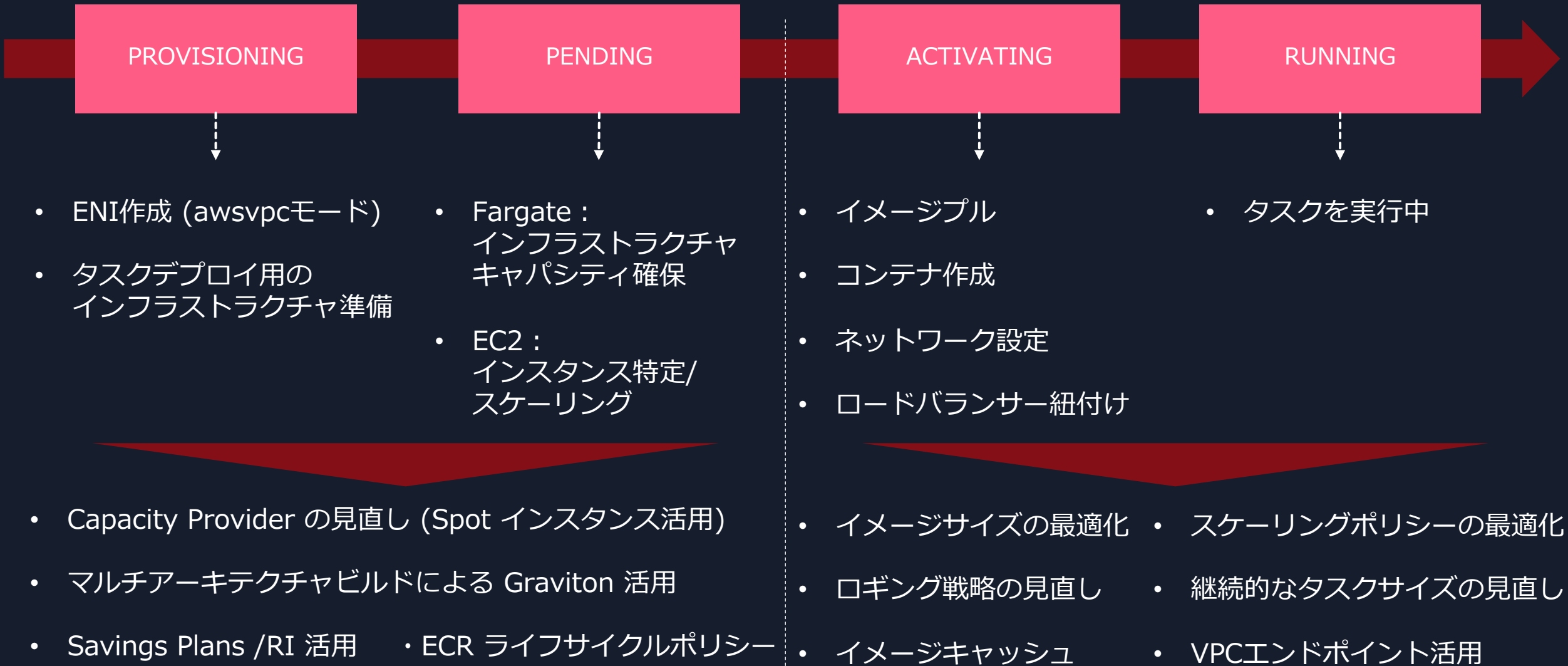
- ECS のライフサイクルとコスト最適化のポイントの全体像
- 各コスト最適化のポイント
- 参考) ECS Managed Instance
- まとめ

ECS のライフサイクルと コスト最適化のポイントの全体像

ECS タスクのライフサイクル（～Running）



各ライフサイクルにおけるコスト最適化ポイント



各ライフサイクルにおけるコスト最適化ポイント

[ECS CFM Tips](#) に掲載されている内容を除き、以下5点のポイントを今日はご紹介

- **Capacity Provider の見直し (Spot インスタンス活用)**
- マルチアーキテクチャビルドによる Graviton 活用
- Savings Plans /RI 活用
- ECR ライフサイクルポリシー
- **イメージサイズの最適化**
- **イメージキャッシュ**
- **スケーリングポリシーの最適化**
- **ロギング戦略の見直し**
- **継続的なタスクサイズの見直し**
- VPCエンドポイント活用

各コスト最適化のポイント

1. スケーリングポリシーの最適化
2. イメージサイズの最適化
3. ロギング戦略の見直し
4. Capacity Provider の見直し (Spot インスタンスの活用)
5. 継続的なタスクサイズの見直し

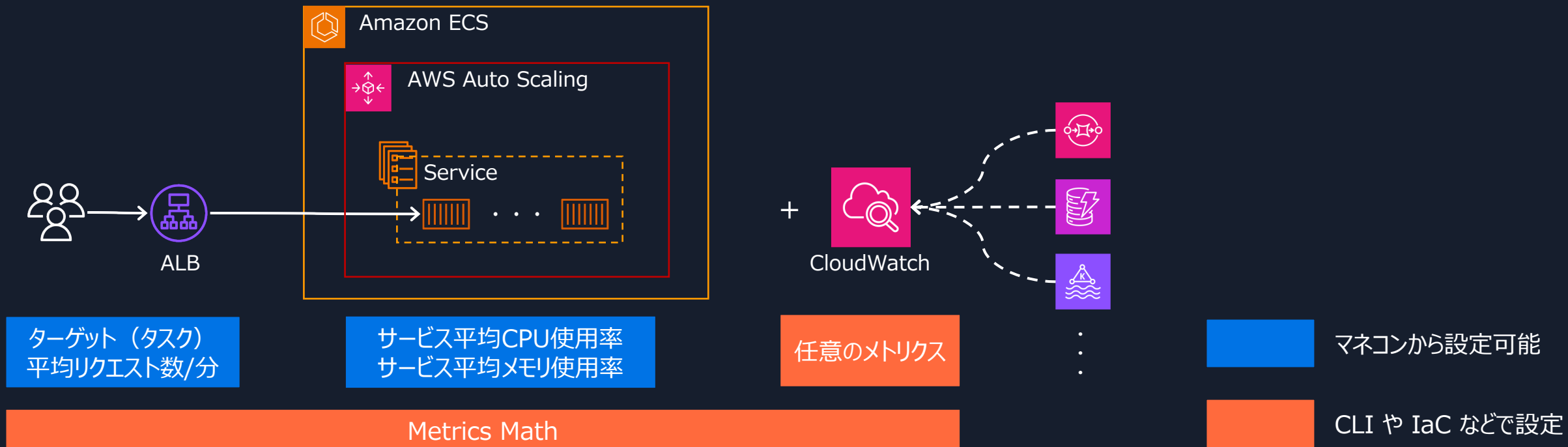
各コスト最適化のポイント

1. スケーリングポリシーの最適化
2. イメージサイズの最適化
3. ロギング戦略の見直し
4. Capacity Provider の見直し (Spot インスタンスの活用)
5. 継続的なタスクサイズの見直し

カスタムメトリクスとMetrics Mathの対応

2023

- 2023年にステップスケーリングにおいて、カスタムメトリクス（Metrics Math）に対応
※マネコンからは設定できないため注意
- ワークロードにあわせて、より適切なスケーリングポリシーが設定可能に
- 例）SQS のコンシューマであるECSサービスにて「キューの長さ/タスク数」を利用



カスタムメトリクスとMetrics Mathを使うパターン

- 「SQSキューの長さ/タスク数」 スケーリングの設定例

```
1  {
2    "TargetValue": 10.0,
3    "CustomMetricSpecification": {
4      "MetricName": "BacklogPerTask",
5      "Namespace": "AWS/ApplicationELB",
6      "Statistic": "Average",
7      "Metrics": [
8        {
9          "Id": "m1",
10         "MetricStat": {
11           "Metric": {
12             "MetricName": "ApproximateNumberOfMessages",
13             "Namespace": "AWS/SQS",
14             "Dimensions": [
15               {
16                 "Name": "QueueName",
17                 "Value": "your-queue"
18               }
19             ],
20             "Stat": "Average"
21           }
22         }
23       ],
```

```
24     {
25       "Id": "m2",
26       "MetricStat": {
27         "Metric": {
28           "MetricName": "RunningTaskCount",
29           "Namespace": "AWS/ECS",
30           "Dimensions": [
31             {
32               "Name": "ServiceName",
33               "Value": "your-service"
34             },
35             {
36               "Name": "ClusterName",
37               "Value": "your-cluster"
38             }
39           ],
40           "Stat": "Average"
41         }
42       }
43     },
44     "Expression": "m1/MAX([m2,1])"
45   ]
46 }
47 }
48 }
```

スケールアウト・インのデータポイント

- ターゲット追跡スケーリングポリシーは以下の設定になっており、最低3分以上スケールに必要
 - スケールアウト：3分内の3個のデータポイントで評価
 - スケールイン：15分内の15個のデータポイントで評価
- [ドキュメント](#)の通り、設定の変更は非推奨

- ターゲット追跡スケーリングポリシーのためにサービスの自動スケーリングが管理する CloudWatch アラームを編集または削除しないでください。スケーリングポリシーを削除するときに、サービスの自動スケーリングはアラームを自動的に削除します。

- ターゲット追跡スケーリングポリシーでスケール要件を満たすことが難しい場合は、ステップスケーリングや独自のスケーリングメカニズム（Lambda等）の利用などを検討。
 - 例えばスケールインの要件が厳しい場合はスケールインを「無効化」した上で別のポリシーで制御するというアプローチも

予測スケーリングポリシーが利用可能に

2024

- 既存のターゲット追跡、スケジューリング、ステップスケーリングポリシーに加えて予測スケーリングポリシーが利用可能になった
- 「予測のみ」の利用も可能、複数の予測スケーリングポリシーからベストなものをレコメンドしてくれる機能もある

ポリシー名	▲	ステータス	▼	メトリクス	レコメンデーション	可用性への影響	▼	コストへの影響	▼	チャート
pc-cpu-10		🕒 予測のみ		ECSServiceCPUUtilization (10%)	最善の予測でポリシーを使用	✅ やや増加		✅ やや削減		チャートを表示
pc-cpu-30		🕒 予測のみ		ECSServiceCPUUtilization (30%)	最善の予測でポリシーを使用	✅ やや増加		✅ やや削減		チャートを表示
pc-req-100		✅ 予測およびスケール		ALBRequestCount (100)	予測結果を待機	⋯ さらにデータが必要		⋯ さらにデータが必要		チャートを表示
pc-req-50		🕒 予測のみ		ALBRequestCount (50)	予測結果を待機	⋯ さらにデータが必要		⋯ さらにデータが必要		チャートを表示
ps-cpu-60		🕒 予測のみ		ECSServiceCPUUtilization (60%)	[予測およびスケール] に切り替える	✅ やや増加		✅ やや削減		チャートを表示
最善の予測										

予測スケーリングポリシーが利用可能に

2024

- チャートにより実際の負荷やキャパシティとポリシーによる推測が確認可能



各コスト最適化のポイント

1. スケーリングポリシーの最適化
2. イメージサイズの最適化
3. ロギング戦略の見直し
4. Capacity Provider の見直し (Spot インスタンスの活用)
5. 継続的なタスクサイズの見直し

マルチステージビルドの採用

- アプリケーションの実行に使用される最終イメージとビルドを分割する
- 最終的なコンテナイメージにはビルドプロセスで必要なコンポーネントが不要となり、**コンテナイメージを小さく**することが可能となる
- コンテナレジストリからのプルも**データ転送量の削減**により高速化し、**コンテナ起動も高速化**可能

ビルドステージのイメージ

FROM maven:3.9-amazoncorretto-17 AS builder

Javaアプリケーションのビルドアセットのコピー

WORKDIR /build

COPY pom.xml .

COPY src ./src

アプリケーションのビルド

Javaアプリケーションのビルド

RUN mvn clean package -DskipTests

実行ステージのイメージ

FROM amazoncorretto:17-alpine

#ビルドステージのアウトプット（JAR）をコピー

WORKDIR /app

COPY --from=builder /build/target/*.jar app.jar

最終イメージの作成

#アプリケーションの実行

EXPOSE 8080

CMD ["java", "-jar", "app.jar"]

ワンライナーによるイメージレイヤーの設定

- Dockerfileでは1命令につき、1レイヤーとなるため、例えば左下図のようにRUNコマンドを2つ実行した場合はコンテナイメージサイズの削減に寄与しない可能性がある
- パッケージをインストールした後に不要となったダウンロードファイルはワンライナーで削除することを推奨する

rm コマンドを使用してキャッシュされたファイルを
削除したとしても、1つ目のレイヤーにキャッシュされたファイルを含む

```
RUN apt-get install -y some-package  
RUN rm -rf /var/lib/apt/lists/*
```

ワンライナーで1つのコマンドで全て実行することで
レイヤーのサイズを小さくし、
コンテナイメージ全体のサイズも削減可能

```
RUN apt-get install -y some-package && rm -rf /var/lib/apt/lists/*
```

イメージレイヤー の数・順序性の意識

- なるべく、変更が少ない命令を前半に持つてくることでキャッシュ効率が向上し、CI/CDパイプライン（CodeBuild等）のイメージビルド時間とコスト（通信量含む）削減可能

アプリケーションの変更のたびにCOPYコマンドの実行内容が変わり、その後のRUNコマンドのキャッシュが無効に

```
FROM amazonlinux:2023
RUN yum update -y
COPY . /app
RUN yum install -y python python-pip wget
CMD [ "app.py" ]
```

RUNコマンドを1行としてまとめ、変更頻度の高いCOPYコマンドの前に実行する

```
FROM amazonlinux:2023
RUN yum update -y && yum install -y python python-pip wget
COPY . /app
CMD [ "app.py" ]
```

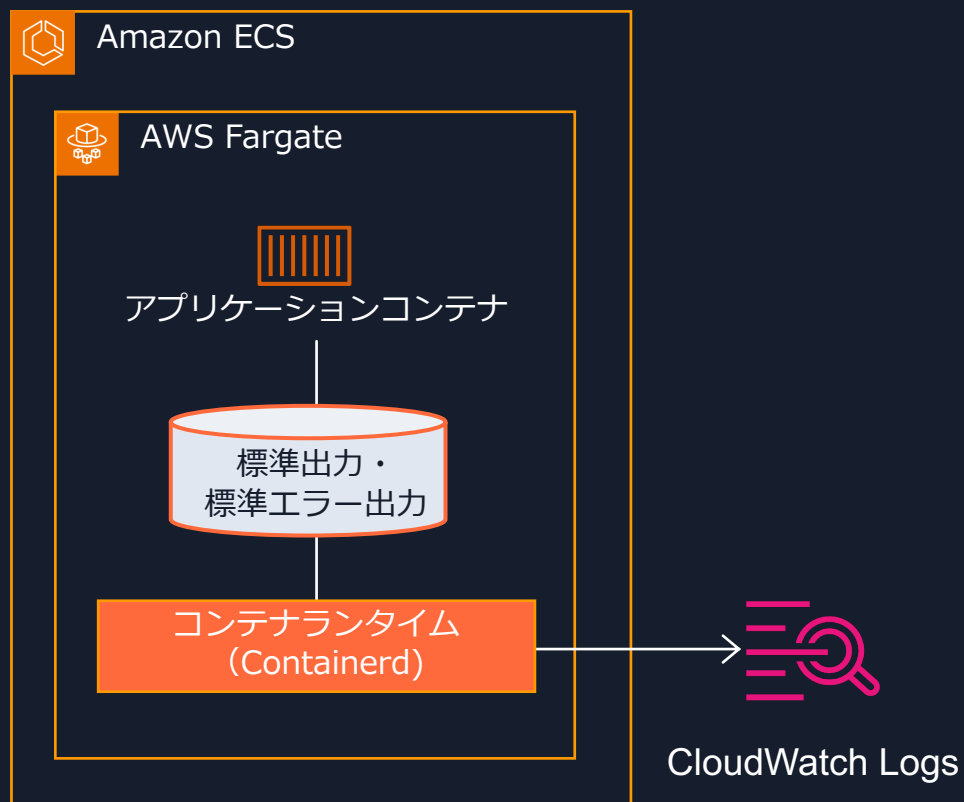
アプリケーションのコピーコマンドは後半で実行

各コスト最適化のポイント

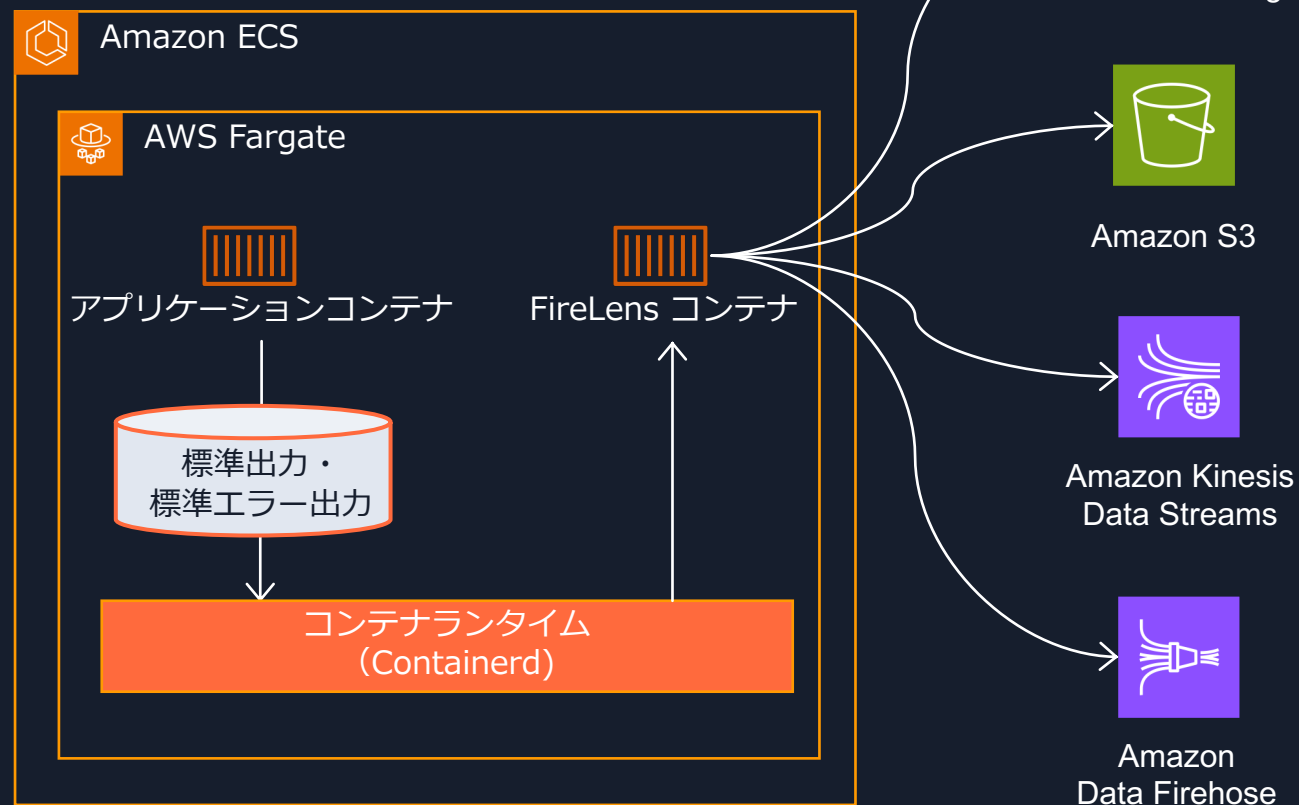
1. スケーリングポリシーの最適化
2. イメージサイズの最適化
3. ロギング戦略の見直し
4. Capacity Provider の見直し (Spot インスタンスの活用)
5. 継続的なタスクサイズの見直し

ECS における代表的なロギング戦略

awslogs



awsfirelens (Fluent Bit)



ロギング戦略

Pros.

Cons.

awslogs ログドライバー

- 設定がシンプル
- サイドカーコンテナ不要

- 送信先は CloudWatch Logs のみ
- ログの加工やフィルタリングは不可

FireLens ログドライバー

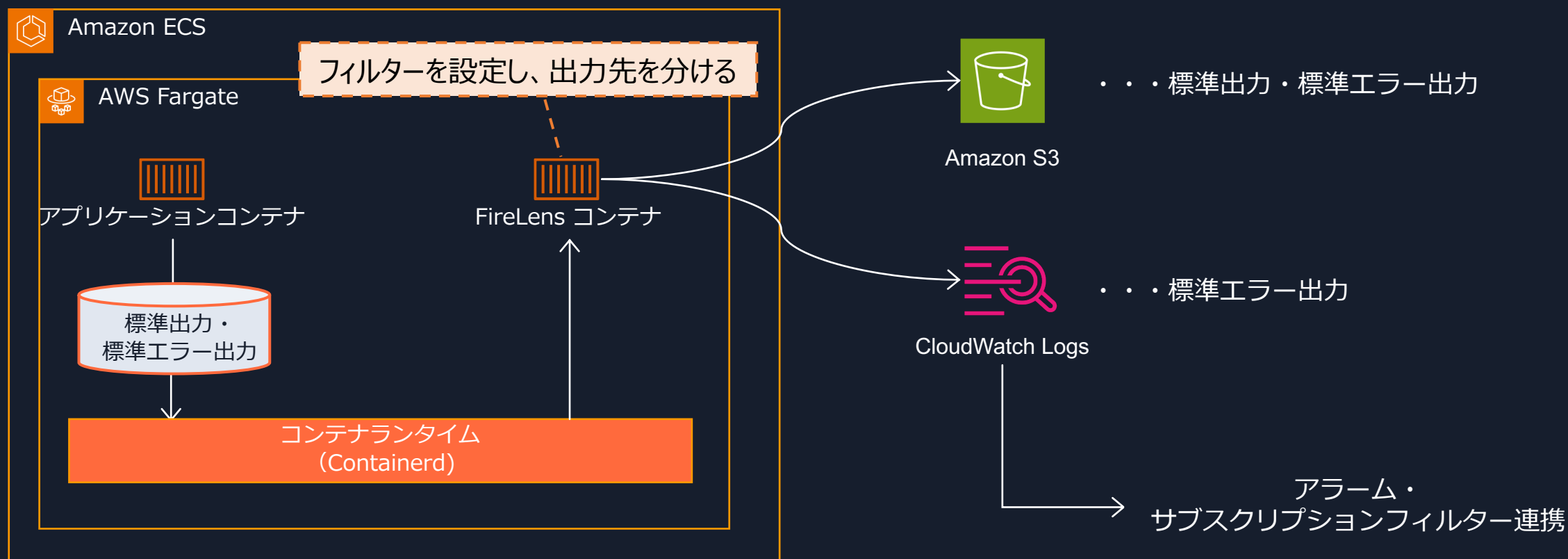
- 柔軟性が高い
- ログルーターとの接続が容易

- Fluentd または Fluent Bit が必要
- 設定ファイルの管理が発生

CW Logs の Ingest のコストが気になる場合は Firelens を利用してコスト最適化

FireLens を利用したコスト最適化例

- FireLens の設定については [GitHub 上にサンプル](#)が掲載されているので、そちらを参照に設定
- 標準出力はS3に出力し、重要なエラーログのみ CloudWatch Logs に出力することでコスト削減



AWS for Fluent Bit 3.0.0

2025

- 悲願 の [AWS for Fluent Bit 3.0.0](#) が 2025年10月にリリース
- 内部的に利用している Fluent Bit のバージョンが 1.9.10 => 4.1.1 にアップグレード
- ベースイメージが AL2 から AL2023 に

Releases / v3.0.0

AWS for Fluent Bit 3.0.0 Latest

 ShelbyZ released this last week · [5 commits](#) to mainline since this release  v3.0.0  a09ada1 

This release includes:

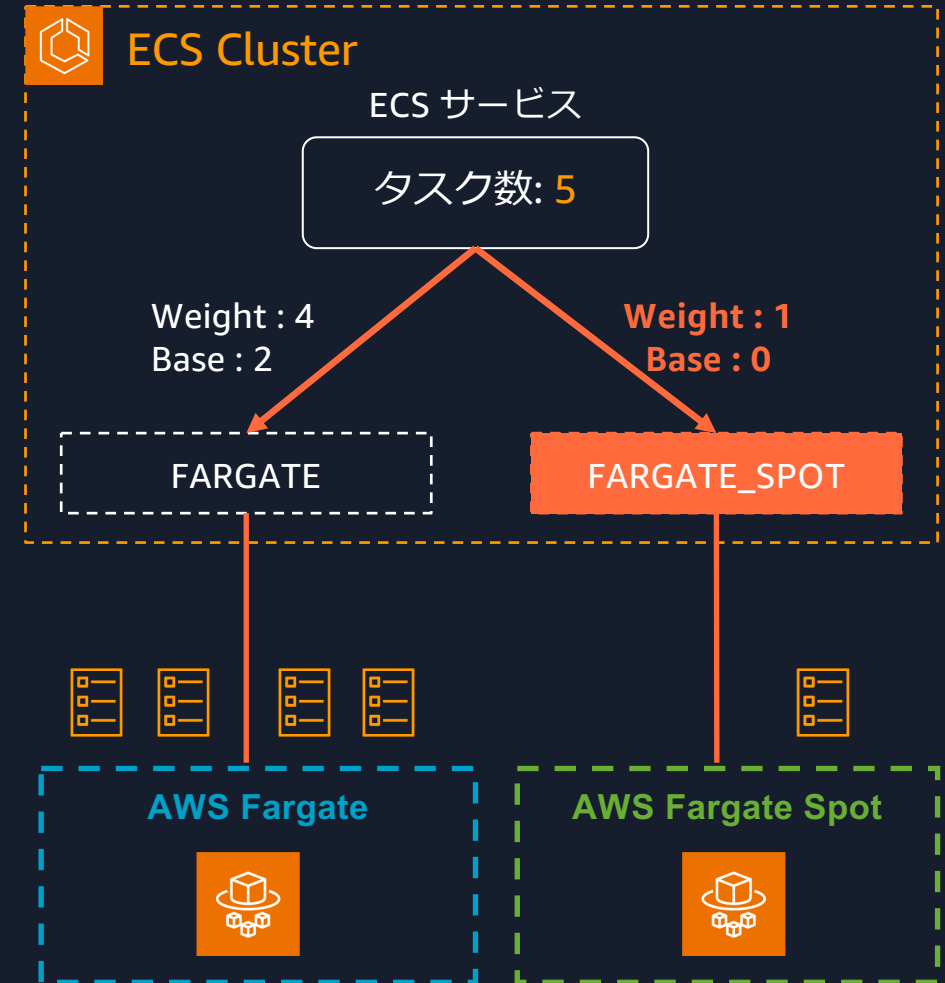
- Fluent Bit [4.1.1](#)
- Amazon CloudWatch Logs for Fluent Bit 1.9.4
- Amazon Kinesis Streams for Fluent Bit 1.10.3
- Amazon Kinesis Firehose for Fluent Bit 1.7.2
- Amazon Linux 2023 base container image version: [2023.9.20250929](#)

各コスト最適化のポイント

1. スケーリングポリシーの最適化
2. イメージサイズの最適化
3. ロギング戦略の見直し
4. Capacity Provider の見直し (Spot インスタンスの活用)
5. 継続的なタスクサイズの見直し

Capacity Provider の見直し（Spot インスタンス活用）

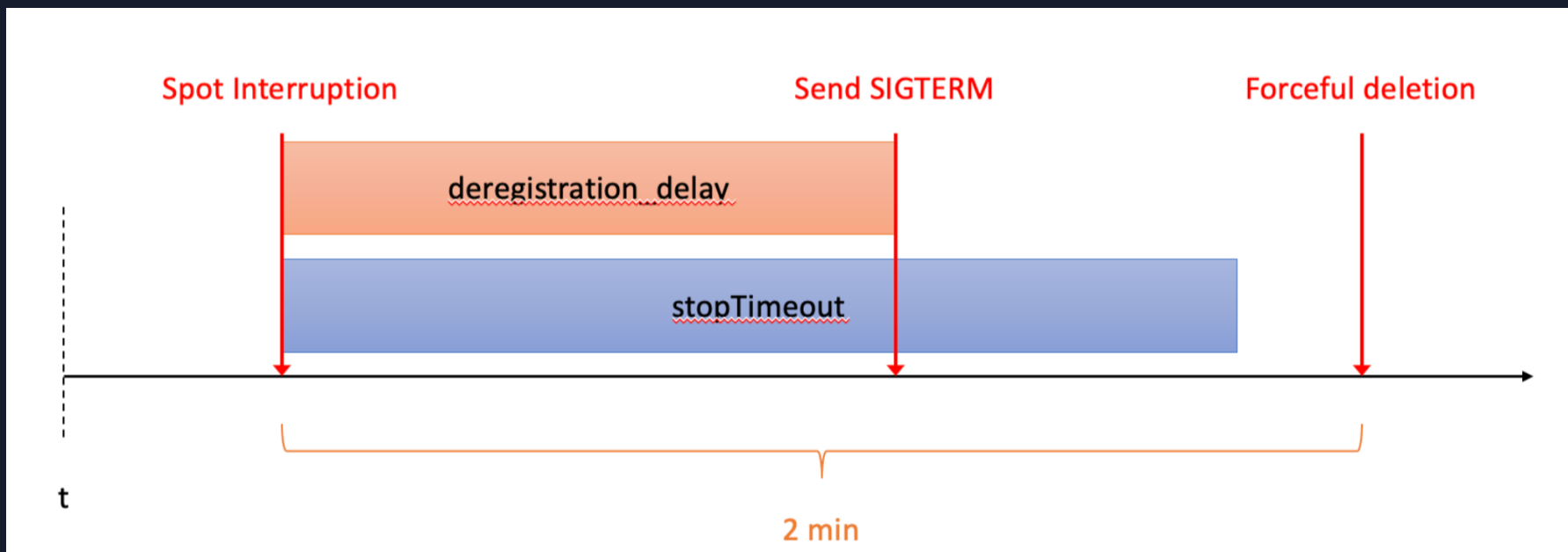
- ワークロードの特性に応じて Capacity Provider の設定を見直して、Spot インスタンスを活用する。
- 例えば、Fargate のワークロードにおいて、全体の20パーセント（Ondemand : Spot = 4:1）を Spot インスタンスに置き換え、ワークロードを管理
- Weight と Base の設定をワークロードに合わせて再調整
- 場合によっては別 ECS サービスとして Spot インスタンスを利用（サービス並列稼働）
※詳細は後述



Fargate spot 活用時の注意事項

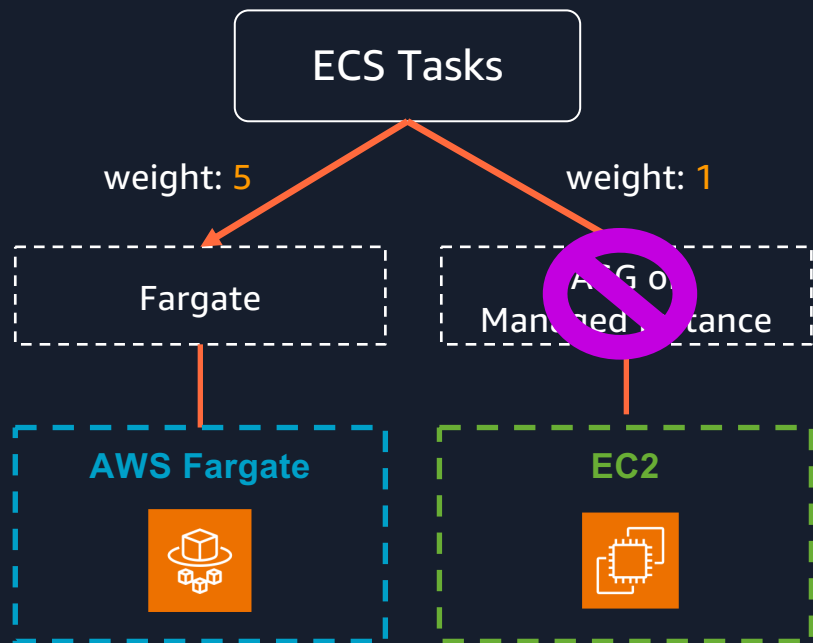
2023

- [2023年のアップデート](#)でタスクが ALB から登録解除されてから SIGTERM が送信されるように
- Fargate Spot において SIGTERM によるシグナルハンドリングを行うためには、
 $\text{deregistration_delay_timeout_seconds} < \text{stopTimeout} < 120 \text{ (sec)}$ とする必要がある
 - Fargate Spot は終了通知が2分前のため
 - $\text{deregistration_delay_timeout_seconds}$ のデフォルトは5分のため注意

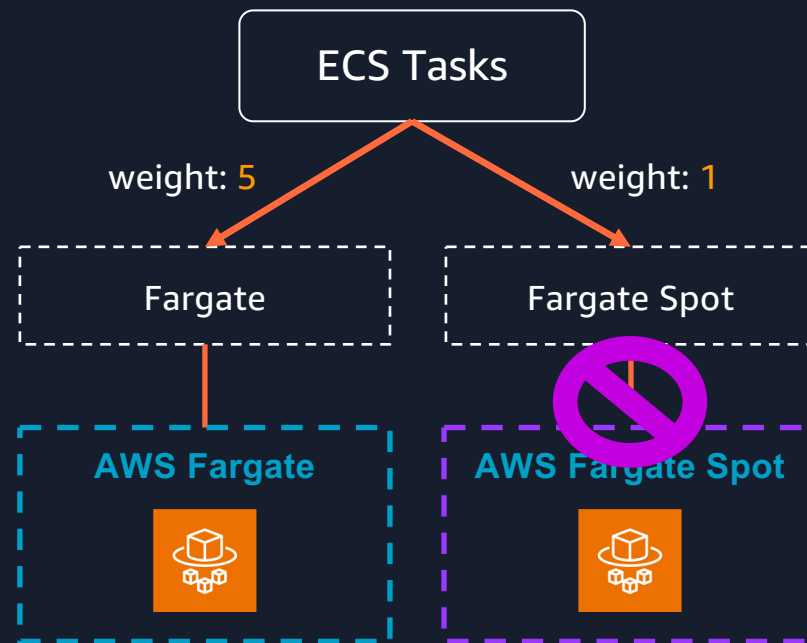


Capacity Provider における注意事項

Fargate と EC2 の Capacity Provider を
混在させた Strategy を設定できない



特定の Capacity Provider でタスクが実行できない場合、
別の Capacity Provider にフォールバックできない



フォールバックできない点の対策案（サービス並列稼働）



AWS Fargate Spot



Service 1



Task



Task



Task



AWS Auto Scaling の条件例

ターゲット追跡 スケーリング (Spot 用)

- ECSServiceAverageCPUUtilization (ECS サービスの平均 CPU 使用率) を **35%** に保つ
- ECSServiceAverageMemoryUtilization (ECS サービスの平均メモリ使用率) を **35%** に保つ
- ALBRequestCountPerTarget (ターゲットあたりの ALB からのリクエスト数) を **500** に保つ



AWS Fargate On-demand



Service 2



Task



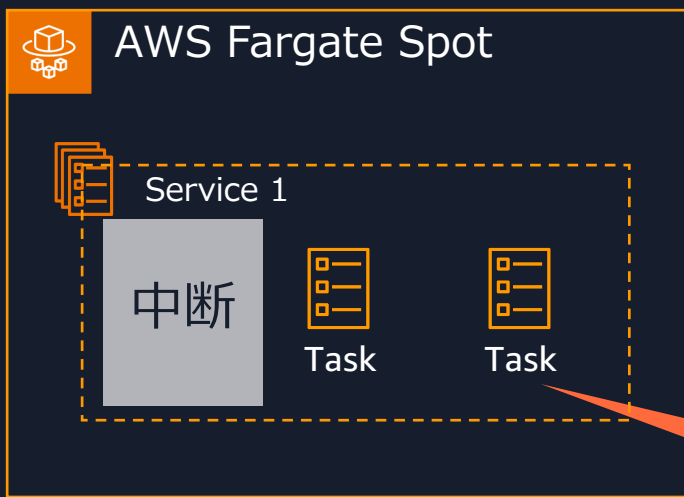
Task

負荷が高くなると、こっちが増える

ターゲット追跡 スケーリング (On-Demand 用)

- ECSServiceAverageCPUUtilization (ECS サービスの平均 CPU 使用率) を **70%** に保つ
- ECSServiceAverageMemoryUtilization (ECS サービスの平均メモリ使用率) を **70%** に保つ
- ALBRequestCountPerTarget (ターゲットあたりの ALB からのリクエスト数) を **1000** に保つ

フォールバックできない点の対策案（サービス並列稼働）

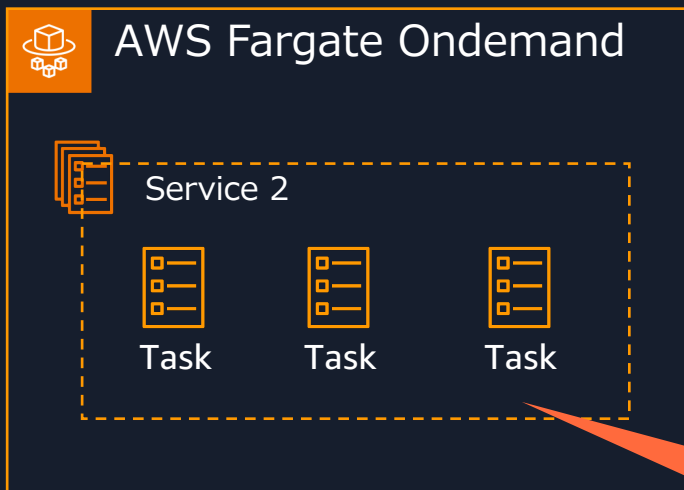


AWS Auto Scaling の条件例

ターゲット追跡 スケーリング (Spot 用)

- ECSServiceAverageCPUUtilization (ECS サービスの平均 CPU 使用率) を **35%** に保つ
- ECSServiceAverageMemoryUtilization (ECS サービスの平均メモリ使用率) を **35%** に保つ
- ALBRequestCountPerTarget (ターゲットあたりの ALB からのリクエスト数) を **500** に保つ

タスクが配置できなくなり、稼働中のタスクの負荷が高くなる

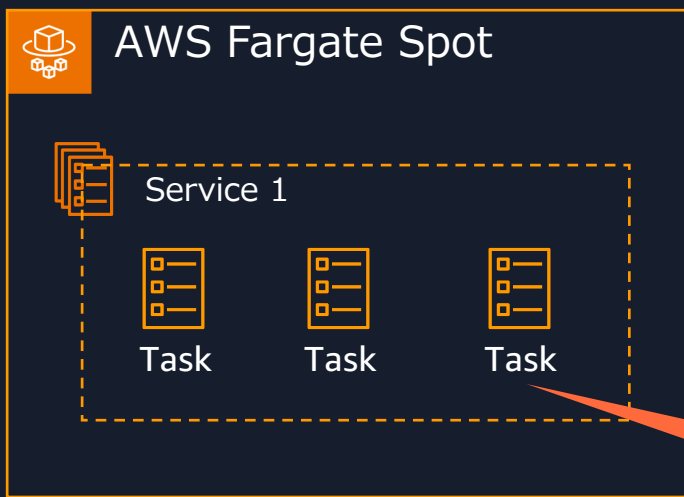


ターゲット追跡 スケーリング (On-Demand 用)

- ECSServiceAverageCPUUtilization (ECS サービスの平均 CPU 使用率) を **70%** に保つ
- ECSServiceAverageMemoryUtilization (ECS サービスの平均メモリ使用率) を **70%** に保つ
- ALBRequestCountPerTarget (ターゲットあたりの ALB からのリクエスト数) を **1000** に保つ

同様に、稼働中のタスクの負荷が高くなり、スケールアウトし始める

フォールバックできない点の対策案（サービス並列稼働）

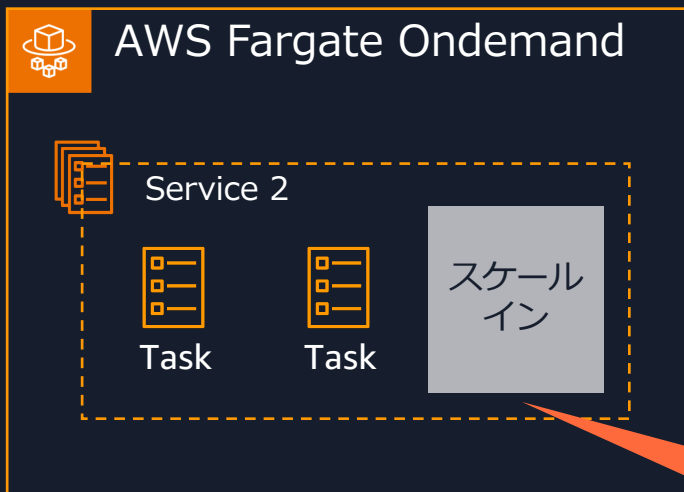


AWS Auto Scaling の条件例

ターゲット追跡 スケーリング (Spot 用)

- ECSServiceAverageCPUUtilization (ECS サービスの平均 CPU 使用率) を **35%** に保つ
- ECSServiceAverageMemoryUtilization (ECS サービスの平均メモリ使用率) を **35%** に保つ
- ALBRequestCountPerTarget (ターゲットあたりの ALB からのリクエスト数) を **500** に保つ

キャパシティ回復に伴いタスク数が増え、稼働中のタスクの負荷が低くなる



ターゲット追跡 スケーリング (On-Demand 用)

- ECSServiceAverageCPUUtilization (ECS サービスの平均 CPU 使用率) を **70%** に保つ
- ECSServiceAverageMemoryUtilization (ECS サービスの平均メモリ使用率) を **70%** に保つ
- ALBRequestCountPerTarget (ターゲットあたりの ALB からのリクエスト数) を **1000** に保つ

同様に、稼働中のタスクの負荷が低くなり、スケールインし始める

各コスト最適化のポイント

1. スケーリングポリシーの最適化
2. イメージサイズの最適化
3. ロギング戦略の見直し
4. Capacity Provider の見直し (Spot インスタンスの活用)
5. 継続的なタスクサイズの見直し

継続的なタスクサイズの見直し

Compute Optimizer や Containers Insights をベースにタスクのリソースを定期的に見直し

Fargate での ECS サービスの推奨事項 (7) 情報

コストとパフォーマンスを向上させるための Fargate での現在の ECS サービスの推奨事項。

1 つ以上のプロパティでフィルタリング

リージョン = アジアパシフィック (東京) フィルターをクリア

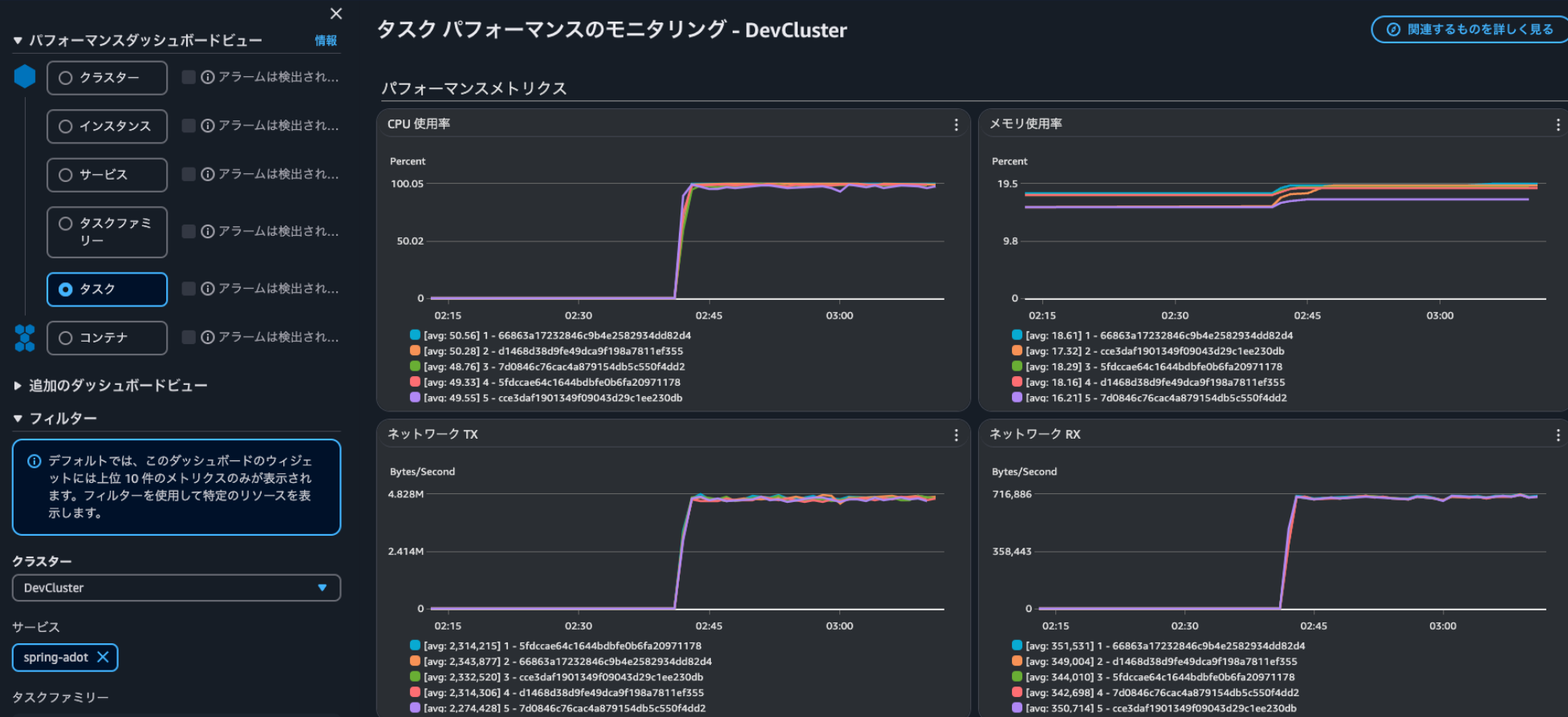
タグキー タグ値 追加

	サービス名	タスク定義名	結果 情報	検出結果の理由 情報	推定月間節約額 (割引後) 情報	推 情報
☐	socat	alpine-socat:9	過剰なプロビジョ...	CPU の過剰なプロビジョニ...	\$19.5750	
☐	ecs-exec	alpine-socat:12	過剰なプロビジョ...	CPU の過剰なプロビジョニ...	\$9.7880	
☐	bluegreen-bac...	bluegreen-bac...	過剰なプロビジョ...	CPU の過剰なプロビジョニ...	\$9.8180	
☐	bluegreen-fro...	bluegreen-fro...	過剰なプロビジョ...	CPU の過剰なプロビジョニ...	\$3.2180	
☐	rails-bg	rails-app:13	最適化済み	-	\$0.0000	
☐	enhanced-surv...	EnhancedSurv...	最適化済み	-	\$0.0000	
☐	enhanced-surv...	EnhancedSurv...	最適化済み	-	\$0.0000	

Container Insights のアップデート

2024

- サービス全体の平均値だけでなく、タスク、コンテナの詳細なメトリクスを確認できるように
- サイジングの参考情報として活用可能



Container Insights のアップデート

2024

- タスク・コンテナ（例えば他に比べ CPU 使用率が高いもの）をフィルターしてログの検索やトレースの表示が可能。ダウンサイジングに向けた詳細な検討が可能



Container Insights のアップデート

2024

- すでに Containers Insights を利用している場合でも利用にあたって設定の更新が必要
- クラスター単位、アカウント単位両方で設定可能

▼ モニタリング 情報

CloudWatch Container Insights は、コンテナ化されたアプリケーションとマイクロサービスのモニタリングやトラブルシューティングを行うソリューションです。

Container Insights を使用して達成したいオブザーバビリティのレベルを選択

オブザーバビリティが強化された Container Insights は、アカウント設定でデフォルトでオンになっています。ここでクラスターレベルで上書きできます。

- ☒ **オブザーバビリティが強化された Container Insights** 新規 推奨
クラスターレベルとサービスレベルでの集計メトリクスに加えて、タスクレベルとコンテナレベルでの詳細な正常性およびパフォーマンスメトリクスを提供します。掘り下げが容易になり、問題の切り分けとトラブルシューティングを迅速に行えます。
- ☐ **Container Insights**
クラスターレベルとサービスレベルで集計されたメトリクスを提供します。Logs Insights 分析を使用すると、詳細な分析を実行できます。
- ☐ **オフになっています**
デフォルトの CloudWatch メトリクスのみを提供します。

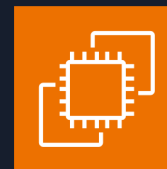
参考) ECS Managed Instance

これまでの Amazon ECS データプレーン



AWS Fargate

サーバー管理における
運用オーバーヘッドの削減
設計によるワークロードの分離



Amazon EC2

ワークロードの要件を満たす
インスタンスタイプを選択

キャパシティ予約、リザーブドインス
タンス、スポットインスタンスを含む
すべての EC2 料金モデルを活用

※厳密には ECS Anywhere なども

Amazon ECS Managed Instance

2025

- Fargate と EC2 のそれぞれのメリットを両方享受することが可能な選択肢
- 元々東京リージョンのみ利用できていたが、10/27に大阪リージョンも利用可能に！



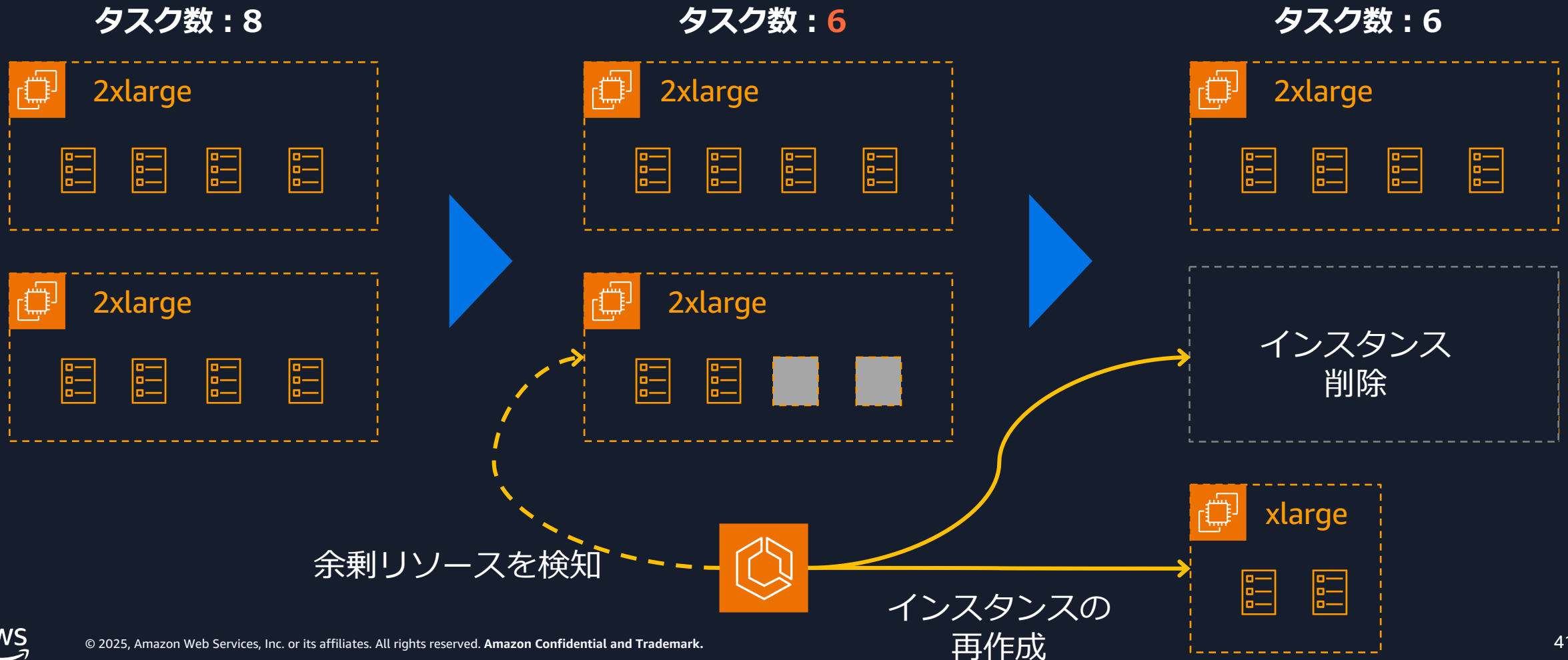
ECS Managed Instance の 起動

- インスタンスの起動には Auto Scaling Group を利用せずに EC2 CreateFleet API を利用
- タスクのサイズに応じて 適切なインスタンスタイプをECS側で選定
- インスタンスタイプに優先度をつけ、Fleet API を用いて 優先度順に起動可能か確認し、インスタンスを起動

```
"CreateFleetRequest": {  
  "LaunchTemplateConfigs": {  
    "Overrides": [  
      {  
        "Priority": "0.0",  
        "InstanceType": "m6g.medium",  
        "AvailabilityZone": "ap-northeast-1a",  
        // ... その他のフィールド  
      },  
      {  
        "Priority": "1.0",  
        "InstanceType": "m7g.medium",  
        "AvailabilityZone": "ap-northeast-1a",  
        // ...  
      },  
      {  
        "Priority": "2.0",  
        "InstanceType": "m6gd.medium",  
        "AvailabilityZone": "ap-northeast-1a",  
        // ...  
      },  
    ]  
  }  
}
```


コンソリデーション等によるインスタンスの最適化

リソースの利用効率が最大となるように自動でインスタンスを最適化



インスタンス最適化の実行例

① インスタンス上のタスクがスケールイン

コンテナインスタンス (5) 情報

Q プロパティまたは値でコンテナインスタンスをフィルタリング

☐	コンテナインスタンス	インスタンス...
☐	2c30baeaffa44f46a6ca4e155a09227e	m6g.large
☐	364a147eb1e5434aa1693a6486400e...	m6g.large
☐	6f51c7c8fc0b404baf75a75633ed03d2	m6g.large
☐	8de18f3dea11403692ce55450f739ea3	m6g.large
☐	b96d020846324522a0ceef1f8cb9a4ea	m6g.2xlarge

コンテナインスタンス (6) 情報

Q プロパティまたは値でコンテナインスタンスをフィルタリング

☐	コンテナインスタンス	インスタンス...
☐	034c164a507d4f1a815537c8d000fa13	m6g.large
☐	049c3844a4ff4f7cad19341f9ffe0036	m6g.large
☐	2c30baeaffa44f46a6ca4e155a09227e	m6g.large
☐	364a147eb1e5434aa1693a6486400e...	m6g.large
☐	6f51c7c8fc0b404baf75a75633ed03d2	m6g.large
☐	8de18f3dea11403692ce55450f739ea3	m6g.large

② ECS側でインスタンスの余剰リソースを検知 2xlargeのインスタンスのリソース使用が50パーセントに

コンテナインスタンス (7) 情報

最終更新日: 2025年10月31日 17:51 (UTC+9:00)

Q プロパティまたは値でコンテナインスタンスをフィルタリング

☐	コンテナインスタンス	インスタンス...	ステ...	状況の理由
☐	034c164a507d4f1a815537c8d000fa13	m6g.large	🟢 アクティブ	-
☐	049c3844a4ff4f7cad19341f9ffe0036	m6g.large	🟢 アクティブ	-
☐	2c30baeaffa44f46a6ca4e155a09227e	m6g.large	🟢 アクティブ	-
☐	364a147eb1e5434aa1693a6486400e...	m6g.large	🟢 アクティブ	-
☐	6f51c7c8fc0b404baf75a75633ed03d2	m6g.large	🟢 アクティブ	-
☐	8de18f3dea11403692ce55450f739ea3	m6g.large	🟢 アクティブ	-
☐	b96d020846324522a0ceef1f8cb9a4ea	m6g.2xlarge	⚠️ ドレイン中	INFRASTRUCTURE_OPTIMIZATION

③ タスクのドレイン、インスタンスの見直し 2xlarge 1台から large 2台に変更

ECS Managed Instance の関連資料

[公式ブログ](#) とあわせて JAWS にて発表している以下の資料も是非ご覧ください。

[Gov-JAWS#4](#)

今月登場した ECS Managed Instances は Fargate と何が違うのか

Kenichi Azuma

Amazon Web Services Japan G.K.
Solutions Architect



© 2025, Amazon Web Services, Inc. or its affiliates. All rights reserved. Amazon Confidential and Trademark.

1

<https://speakerdeck.com/kenicazu/ecs-managed-instance-overview-key-differences-from-fargate>

まとめ

まとめ

- ECS にはいくつかのコスト最適化のポイントが存在
- 本日は5点ご紹介したが、まずは各組織で取り組みやすい内容からチャレンジすることを推奨
- あらたに ECS Managed Instance という選択肢も登場したため、Fargate や EC2 の運用で抱えていた課題が解決する場合は検討する

Thank you!

Kenichi Azuma

Amazon Web Services Japan G.K.
Solutions Architect

