aws

ユーザーガイド

AWS Batch

AWS Batch: ユーザーガイド

Table of Contents

AWS Batch とは?

AWS Batch を使用すると、AWS クラウド でバッチコンピューティングワークロードを実行できます。バッチコンピューティングは、開発者、科学者、エンジニアが大量のコンピューティングリソースにアクセスするための一般的な方法です。AWS Batch は、従来のバッチコンピューティングソフトウェアと同様に、必要なインフラストラクチャの設定および管理に伴う、差別化につながらない力仕事を排除します。このサービスでは、送信されたジョブに応じてリソースを効率的にプロビジョニングし、キャパシティー制限の排除、コンピューティングコストの削減、および結果の迅速な提供を行うことができます。

AWS Batch はフルマネージドサービスであり、あらゆるスケールのバッチコンピューティングワークロードを実行できます。AWS Batch は、コンピューティングリソースを自動的にプロビジョニングし、ワークロードの量と規模に基づいてワークロードのディストリビューションを最適化します。AWS Batch を使うと、バッチコンピューティングソフトウェアのインストールや管理が不要になり、結果の分析と問題の解決に集中できます。

トピック

AWS Batch のコンポーネント

AWS Batch は、リージョン内の複数のアベイラビリティーゾーン間で実行中のバッチジョブを簡略化します。新規または既存の VPC 内に AWS Batch コンピューティング環境を作成できます。コンピューティング環境が稼働し、ジョブキューに関連付けられた後で、ジョブを実行する Docker コンテナイメージを指定するジョブ定義を指定できます。コンテナのイメージは、コンテナレジストリに保存され引き出されます。これは AWS インフラストラクチャの内にある場合も外にある場合もあります。

ジョブ

AWS Batch に送信する作業単位 (シェルスクリプト、Linux 実行可能ファイル、Docker コンテナイメージなど)。ジョブには名前を付けます。ジョブは、ジョブ定義で指定したパラメーターを使用して、コンピューティング環境でコンテナ化されたアプリケーションとして AWS Fargate または

Amazon EC2 リソースで実行されます。ジョブは、他のジョブを名前または ID で参照できます。また、他のジョブの正常な完了に依存する場合があります。詳細については、ジョブを参照してください。

ジョブ定義

ジョブ定義は、ジョブの実行方法を指定します。ジョブ定義は、ジョブのリソースのブループリントであると考えることができます。他の AWS リソースへのアクセスを提供するために、ジョブに IAM ロールを指定することができます。また、メモリ要件と CPU 要件の両方を指定できます。また、ジョブ定義では、永続的ストレージのコンテナのプロパティ、環境変数、マウントポイントを制御できます。ジョブ定義の多くの仕様は、個別のジョブを送信するときに新しい値を指定してオーバーライドできます。詳細については、ジョブ定義を参照してください。

ジョブキュー

AWS Batch ジョブを送信するときは、コンピューティング環境にスケジュールされるまでそのジョブが存在する、特定のジョブキューに送信します。1 つのジョブキューには、1 つ以上のコンピューティング環境を関連付けることができます。これらのコンピューティング環境には優先度の値を割り当てることができ、複数のジョブキュー自体にまたがって割り当てることもできます。たとえば、時間が重要なジョブを送信する高優先度キューや、コンピューティングリソースが安価であるときにいつでも実行できるジョブ用の低優先度キューを持つことができます。

コンピューティング環境

コンピューティング環境は、ジョブを実行するために使用されるマネージドまたはアンマネージドコンピューティングリソースのセットです。マネージド型のコンピューティング環境では、複数の詳細レベルで目的のコンピューティングタイプ (Fargate または EC2) を指定できます。コンピューティング環境は、特定の EC2 インスタンスタイプや特定のモデル (`c5.2xlarge` や `m5.10xlarge` など) を使用するように設定できます。または、最新のインスタンスタイプを使用するように指定することのみを選択できます。また、環境の vCPU の最小数、希望数、最大数を、スポットインスタンスに対して支払う金額と共に、オンデマンドインスタンスの価格と一連のターゲット VPC サブネットの割合として指定できます。AWS Batch によって必要に応じてコンピューティングタイプの起動、管理、終了が効率的に行われます。お客様独自のコンピューティング環境を管理することもできます。その場合は、ユーザー自身が AWS Batch によって作成される Amazon ECS クラスターでインスタンスの設定とスケーリングを行う必要があります。詳細については、コンピューティング環境を参照してください。

ご利用開始にあたって

AWS Batch を開始するには、AWS Batch コンソールでジョブ定義、コンピューティング環境、およびジョブキューを作成します。

AWS Batch 初回実行ウィザードには、コンピューティング環境とジョブキューを作成してサンプルのHello Worldジョブを送信するオプションがあります。AWS Batch で起動する Docker イメージが既にある場合は、そのイメージでジョブ定義を作成してキューに送信することもできます。詳細については、の開始方法 AWS Batchを参照してください。

ダッシュボード

AWS Batch ダッシュボードから、最近のジョブ、ジョブキュー、コンピューティング環境をモニタリングできます。デフォルトでは、以下のダッシュボードウィジェットが表示されます。

- ジョブの概要 – AWS Batch の詳細については、ジョブを参照してください。
- ジョブキューの概要 – AWS Batch ジョブキューの詳細については、ジョブキューを参照してください。
- コンピューティング環境の概要 — AWS Batch コンピューティング環境の詳細については、コンピューティング環境を参照してください。

ダッシュボードページに表示されるウィジェットはカスタマイズできます。以下のセクションでは、インストールできるその他のウィジェットについて説明します。

シングルジョブのキュー

このウィジェットは、シングルジョブキューに関する詳細情報を表示します。

このウィジェットを追加するには、次の手順に従います。

1. AWS Batch コンソールを開きます。
2. ナビゲーションバーから、ご希望の AWS リージョン を選択します。
3. ナビゲーションペインで、ダッシュボード を選択します。
4. ウィジェットの追加 を選択します。
5. シングルジョブキュー では、ウィジェットを追加 を選択します。
6. ジョブキュー では、表示するジョブキューを選択します。

7. ジョブステータス では、表示するジョブステータスを選択します。
8. (オプション) 接続されたコンピューティング環境を表示 をオフにすると、コンピューティング環境のプロパティを非表示にできます。
9. コンピューティング環境のプロパティ で、必要なプロパティを選択します。
10追加] を選択します。

CloudWatch Container Insights

このウィジェットには、AWS Batch コンピューティング環境とジョブの集計メトリックが表示されます。コンテナインサイトの詳細については、CloudWatch Container Insightsを参照してください。

このウィジェットを追加するには、次の手順に従います。

1. AWS Batch コンソールを開きます。
2. ナビゲーションバーから、ご希望の AWS リージョン を選択します。
3. ナビゲーションペインで、ダッシュボード を選択します。
4. ウィジェットの追加 を選択します。
5. Container insights では、ウィジェットを追加 を選択します。
6. コンピューティング環境 では、表示するコンピューティング環境を選択します。
7. 追加 を選択します。

ジョブの ログ

このウィジットは、ジョブからのさまざまなログを一箇所の便利な場所に表示します。ジョブのログの詳細については、the section called “ ジョブのログ”を参照してください。

このウィジェットを追加するには、次の手順に従います。

1. AWS Batch コンソールを開きます。
2. ナビゲーションバーから、ご希望の AWS リージョン を選択します。
3. ナビゲーションペインで、ダッシュボード を選択します。
4. ウィジェットの追加 を選択します。
5. ジョブログ では、ウィジェットを追加 を選択します。
6. ジョブ ID で、必要なジョブのジョブ ID を入力します。

7. 追加 を選択します。

で をセットアップする AWS Batch

すでに Amazon Web Services (AWS) にサインアップしていて、Amazon Elastic Compute Cloud (Amazon EC2)または Amazon Elastic Container Service (Amazon ECS) を使用している場合は、まもなく AWS Batchを使用できるようになります。これらのサービスのセットアッププロセスは似ています。これは、 がコンピューティング環境で Amazon ECS コンテナインスタンス AWS Batch を使用するためです。 AWS CLI で を使用するには AWS Batch 、最新の AWS Batch 機能 AWS CLI をサポートする のバージョンを使用する必要があります。 AWS Batch の機能のサポートが表示されない場合は AWS CLI、最新バージョンにアップグレードします。詳細については、http://aws.amazon.com/cli/ を参照してください。

> **Note**
>
> AWS Batch は Amazon EC2 のコンポーネントを使用するため、これらのステップの多くには Amazon EC2 コンソールを使用します。

のセットアップを行うには、次のタスクを実行します AWS Batch。完了済みのステップがあればスキップして、直接 AWS CLIのインストールに進むことができます。

トピック

- にサインアップする AWS アカウント
- 管理アクセスを持つユーザーを作成する
- コンピューティング環境およびコンテナインスタンスの IAM ロールの作成
- キーペアを作成する
- 「VPC を作成する」
- セキュリティグループの作成
- のインストール AWS CLI

にサインアップする AWS アカウント

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

1. https://portal.aws.amazon.com/billing/signup を開きます。

2. オンラインの手順に従います。

   サインアップ手順の一環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザーが作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

AWS サインアッププロセスが完了すると、 から確認メールが送信されます。https://aws.amazon.com/ の [アカウント] をクリックして、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理することができます。

管理アクセスを持つユーザーを作成する

にサインアップしたら AWS アカウント、 を保護し AWS アカウントのルートユーザー、 を有効にして AWS IAM Identity Center、日常的なタスクにルートユーザーを使用しないように管理ユーザーを作成します。

のセキュリティ保護 AWS アカウントのルートユーザー

1. ルートユーザーを選択し、 AWS アカウント E メールアドレスを入力して、アカウント所有者AWS Management Consoleとして にサインインします。次のページでパスワードを入力します。

   ルートユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドの「ルートユーザーとしてサインインする」を参照してください。

2. ルートユーザーの多要素認証 (MFA) を有効にします。

   手順については、「IAM ユーザーガイド」の AWS アカウント 「ルートユーザーの仮想 MFA デバイスを有効にする (コンソール)」を参照してください。

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Centerの有効化」を参照してください。

2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

   を ID ソース IAM アイデンティティセンターディレクトリ として使用する方法のチュートリアルについては、「 ユーザーガイド」の「デフォルト でユーザーアクセス IAM アイデンティティセンターディレクトリを設定するAWS IAM Identity Center 」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

  IAM Identity Center ユーザーを使用してサインインする方法については、「 AWS サインイン ユーザーガイド」の AWS 「 アクセスポータルにサインインする」を参照してください。

追加のユーザーにアクセス権を割り当てる

1. IAM アイデンティティセンターで、最小特権のアクセス許可を適用するというベストプラクティスに従ったアクセス許可セットを作成します。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」を参照してください。

2. グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「グループの参加」を参照してください。

コンピューティング環境およびコンテナインスタンスの IAM ロールの作成

AWS Batch コンピューティング環境とコンテナインスタンスには、ユーザーに代わって他の AWS APIsを呼び出すための AWS アカウント 認証情報が必要です。これらの認証情報をコンピューティング環境およびコンテナインスタンスに提供するための IAM ロールを作成し、そのロールをコンピューティング環境に関連付けます。

> **Note**
>
> AWS Batch コンピューティング環境とコンテナインスタンスのロールは、コンソールの初回実行時に自動的に作成されます。したがって、 AWS Batch コンソールを使用する場合は、次のセクションに進むことができます。 AWS CLI 代わりに を使用する予定がある場合は、最初のコンピューティング環境を作成するAmazon ECS インスタンスロール前に、のサービスにリンクされたロールの使用 AWS Batch「」と「」の手順を完了してください。

キーペアを作成する

AWS は、パブリックキー暗号化を使用してインスタンスのログイン情報を保護します。 AWS Batch コンピューティング環境コンテナインスタンスなどの Linux インスタンスには、SSH アクセスに使用するパスワードがありません。キーペアを使用してインスタンスに安全にログインします。コンピューティング環境の作成時にキーペアの名前を指定し、SSH を使ってログインするときにプライベートキーを指定します。

キーペアを既に作成していない場合は、Amazon EC2 コンソールを使用して作成できます。複数のでインスタンスを起動する場合は AWS リージョン、各リージョンにキーペアを作成することに注意してください。リージョンの詳細については、Amazon EC2 ユーザーガイド」の「リージョンとアベイラビリティーゾーン」を参照してください。

キーペアを作成するには

1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. ナビゲーションバーから、キーペア AWS リージョン の を選択します。使用可能なリージョンは場所に関係なく選択できますが、キーペアはリージョンに固有のものです。たとえば、米国西部 (オレゴン) リージョンでインスタンスを起動する予定の場合は、その同じリージョン内のインスタンス用にキーペアを作成します。
3. ナビゲーションペインで キーペア] を選択し、キーペアの作成] を選択します。
4. キーペア作成 ダイアログボックスの キーペア名 フィールドで、新しいキーペアの名前を入力し、作成 を選択します。自分のユーザー名の後に、`-key-pair` とリージョン名を続けたものなど、覚えやすいものにしましょう。たとえば、me-key-pair-uswest2 などです。
5. ブラウザによって秘密キーファイルが自動的にダウンロードされます。ベースファイル名はキーペア名として指定した名前であり、ファイル名の拡張子は `.pem` です。ダウンロードしたプライベートキーのファイルを安全な場所に保存します。

> **Important**
>
> プライベートキーのファイルを保存できるのは、このタイミングだけです。インスタンスの起動時にキーペアの名前を入力し、インスタンスへの毎回の接続時に対応するプライベートキーを入力する必要があります。

6. Mac または Linux コンピュータの SSH クライアントを使用して Linux インスタンスに接続する場合は、次のコマンドを使用してプライベートキーファイルの権限を設定します。これにより、自分以外のユーザーはファイルを読み取ることができません。

```
$ chmod 400 your_user_name-key-pair-region_name.pem
```

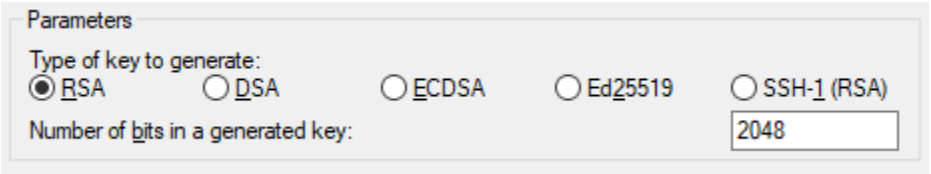
詳細については、Amazon EC2 ユーザーガイド」の「Amazon EC2 キーペア」を参照してください。 Amazon EC2

キーペアを使用してインスタンスに接続するには

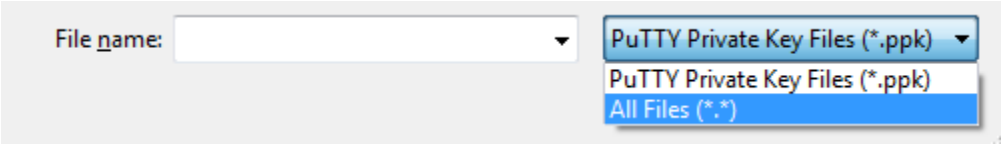
Mac または Linux を実行しているコンピュータから Linux インスタンスに接続するには、-i オプションとプライベートキーへのパスを指定して、SSH クライアントに対する .pem ファイルを指定します。Windows を実行しているコンピュータから Linux インスタンスに接続するには、MindTerm または PuTTY のいずれかを使用します。PuTTY を使用する場合は、PuTTY をインストールしてから、次の手順に従って .pem ファイルを .ppk ファイルに変換します。

(オプション) PuTTY を使用して Windows から Linux インスタンスに接続するには

1. http://www.chiark.greenend.org.uk/~sgtatham/putty/ から PuTTY をダウンロードしてインストールします。必ずスイート全体をインストールします。
2. PuTTYgen を起動します (例: Start] (スタート) メニューで All Programs] (すべてのプログラム)、PuTTY]、PuTTYgen] の順に選択します)。
3. Type of key to generate] (生成するキーのタイプ) で、RSA] を選択します。以前のバージョンの PuTTYgen を使用している場合は、SSH-2 RSA] を選択します。



4. ロード] を選択します。PuTTYgen では、デフォルトでは .ppk 拡張子を持つファイルだけが表示されます。.pem ファイルの場所を特定するには、すべてのタイプのファイルを表示するオプションを選択します。



5. 前の手順で作成したプライベートキーファイルを選択してから、開く を選択します。OK] を選択して、確認ダイアログボックスを閉じます。
6. Save private key] (プライベートキーの保存) を選択します。PuTTYgen に、パスフレーズなしでキーを保存することに関する警告が表示されます。Yes] を選択します。
7. キーペアに使用した名前と同じ名前をキーに指定します。PuTTY によって、.ppk ファイルに拡張子が自動的に追加されます。

「VPC を作成する」

Amazon Virtual Private Cloud (Amazon VPC) を使用すると、定義した仮想ネットワークに AWS リソースを起動できます。コンテナインスタンスは、VPC で起動することを強くお勧めします。

デフォルトの VPC がある場合は、このセクションもスキップして、次のタスクセキュリティグループの作成に移動できます。デフォルト VPC があるかどうかを確認するには、Amazon EC2 ユーザーガイド」の「Amazon EC2 コンソールでサポートされているプラットフォーム」を参照してください。 Amazon EC2

Amazon VPC の作成方法については、Amazon VPC ユーザーガイドのVPC のみを作成するを参照してください。次の表を参考にして、選択するオプションを決定します。

オプション	値
作成するためのリソース	VPC 専用
名前	オプションで、VPC の名前を指定します。
IPv4 CIDR ブロック	IPv4 CIDR 手動入力

オプション	値
	CIDR ブロックサイズは /16 から /28 の間である必要があります。
IPv6 CIDR ブロック	IPv6 CIDR ブロックなし
テナンシー	デフォルト値

Amazon VPC の詳細については、Amazon VPC ユーザーガイドのAmazon VPC とはを参照してください。

セキュリティグループの作成

セキュリティグループは、関連付けられたコンピューティング環境コンテナインスタンスのファイアウォールとして動作し、インバウンドトラフィックとアウトバウンドトラフィックの両方をコンテナインスタンスレベルで制御します。セキュリティグループは、それが対象としている VPC 内でのみ使用が可能です。

SSH を使用して IP アドレスからコンテナインスタンスに接続するためのルールをセキュリティグループに追加できます。さらに、任意の場所からのインバウンドおよびアウトバウンドの HTTP アクセスおよび HTTPS アクセスを可能にするルールを追加できます。タスクで使用するポートを開くためのルールを追加します。

複数のリージョンでコンテナインスタンスを起動する予定がある場合は、各リージョンでセキュリティグループを作成する必要があります。詳細については、Amazon EC2 ユーザーガイドの「リージョンとアベイラビリティゾーン」を参照してください。

> **Note**
>
> ローカルコンピュータのパブリック IP アドレスが必要になります。このアドレスはサービスを使って取得できます。例えば、次のサービスが提供されています。http://checkip.amazonaws.com/ または https://checkip.amazonaws.com/。IP アドレスを提供する別のサービスを検索するには、検索フレーズwhat is my IP addressを使用します。インターネットサービスプロバイダー (ISP) 経由で、またはファイアウォールの背後から静的 IP アドレスなしで接続している場合は、クライアントコンピュータで使用されている IP アドレスの範囲を調べる必要があります。

コンソールを使用してセキュリティグループを作成するには

1. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
2. ナビゲーションペインで、セキュリティグループ] を選択します。
3. セキュリティグループの作成 を選択します。
4. セキュリティグループの名前と説明を入力します。セキュリティグループの作成後に名前と説明を変更することはできません。
5. VPC で、VPC を選択します。
6. (オプション) デフォルトでは、新しいセキュリティグループにはすべてのトラフィックがリソースを離れることを許可するアウトバウンドルールのみが設定されています。任意のインバウンドトラフィックを許可するには、またはアウトバウンドトラフィックを制限するには、ルールを追加する必要があります。

   AWS Batch コンテナインスタンスでは、インバウンドポートを開く必要はありません。ただし、SSH ルールを追加することもできます。これにより、コンテナインスタンスにログインして Docker コマンドでジョブのコンテナを確認できるようになります。また、ウェブサーバーを実行するジョブをコンテナインスタンスでホストする場合は、HTTP ルールを追加できます。これらのオプションのセキュリティグループルールを追加するには、以下のステップを実行します。

   インバウンド] タブで以下のルールを作成し、作成] を選択します。

   - ルールの追加] を選択します。タイプ] で HTTP] を選択します。ソース] では、任意の場所] (`0.0.0.0/0`) を選択します。
   - ルールの追加] を選択します。タイプ] で SSH] を選択します。ソース で、カスタム IP を選択し、コンピュータまたはネットワークのパブリック IP アドレスを Classless Inter-Domain Routing (CIDR) 表記で指定します。会社が特定の範囲からアドレスを割り当てている場合、`203.0.113.0/24` などの範囲全体を指定します。CIDR 表記で個々の IP アドレスを指定するには、My IP を選択します。これにより、パブリック IP アドレスに `/32` ルーティングプレフィックスが追加されます。

   > **Note**
   >
   > セキュリティ上の理由で、すべての IP アドレス (`0.0.0.0/0`) からインスタンスへの SSH アクセスを許可することはお勧めしません。ただし、それがテスト目的で短期間の場合は例外です。

7. タグはここで追加することも、後で追加することもできます。タグを追加するには、新しいタグを追加 をクリックし、タグのキーと値を入力します。
8. セキュリティグループの作成 を選択します。

コマンドラインを使用してセキュリティグループを作成するには、create-security-group (AWS CLI) を参照してください。

セキュリティグループの詳細については、セキュリティグループの操作を参照してください。

のインストール AWS CLI

AWS CLI で を使用するには AWS Batch、 AWS CLI 最新バージョンをインストールします。のインストール AWS CLI または最新バージョンへのアップグレードについては、 AWS Command Line Interface ユーザーガイドの AWS 「 コマンドラインインターフェイスのインストール」を参照してください。

の開始方法 AWS Batch

AWS Batch 初回実行ウィザードを使用すると、 の使用をすばやく開始できます AWS Batch。前提条件を確認したら、初回実行ウィザードを使用してコンピューティング環境、ジョブ定義、およびジョブキューを作成できます。

AWS Batch 初回実行ウィザードを使用して、サンプルの「Hello World」ジョブを送信し、設定をテストすることもできます。で起動する Docker イメージが既にある場合は AWS Batch、そのイメージを使用してジョブ定義を作成できます。

前提条件

AWS Batch 初回実行ウィザードを開始する前に、必ず以下を実行してください。

- で をセットアップする AWS Batchで説明されているステップを完了する。
- に必要なアクセス許可 AWS アカウント があることを確認します。

Amazon EC2 のご利用開始にあたって

Amazon Elastic Compute Cloud (Amazon EC2) は、 AWS クラウドでスケーラブルなコンピューティング容量を提供します。Amazon EC2 の使用により、ハードウェアに事前投資する必要がなくなり、アプリケーションをより速く開発およびデプロイできます。

Amazon EC2 を使用すると、必要な数 (またはそれ以下) の仮想サーバーの起動、セキュリティおよびネットワーキングの構成、ストレージの管理ができます。Amazon EC2 は、要件変更や需要増に応じてスケールアップまたはスケールダウンできるため、トラフィック予測の必要性を軽減できます。

コンピューティング環境の作成

Amazon EC2 オーケストレーション用のコンピューティング環境を作成するには、以下の操作を実行します。

1. AWS Batch コンソールの初回実行ウィザード を開きます。
2. オーケストレーションタイプを選択 で、Amazon Elastic Compute Cloud(Amazon EC2) を選択します。

3. 次へ を選択します。
4. コンピューティング環境設定 の 名前 で、コンピューティング環境の一意な名前を指定します。名前の最大長は 128 文字です。大文字、小文字、数字、ハイフン (-)、アンダースコア (_) を含めることができます。
5. インスタンスロール では、必要な IAM アクセス許可がアタッチされた既存のインスタンスプロファイルを使用することを選択します。このインスタンスプロファイルにより、コンピューティング環境の Amazon ECS コンテナインスタンスは必要な AWS API オペレーションを呼び出すことができます。詳細については、「Amazon ECS インスタンスロール」を参照してください。
6. (オプション) タグは、リソースに割り当てられるラベルです。タグまたは Amazon EC2 タグを追加するには、タグ を展開し、タグを追加 を選択します。キーと値のペアを入力し、もう一度タグを追加 を選択します。

> **Important**
>
> タグを追加 を選択した場合は、キーと値のペアを入力して タグを追加 をもう一度選択するか、タグを解除 を選択する必要があります。

7. (オプション) Amazon EC2 スポットインスタンスの使用を有効にする の インスタンス設定 セクションで、スポットインスタンスを使用する をオンにします。
8. (スポットのみ) オンデマンド価格の最大割合 (%)には、スポットリソースに対して支払うオンデマンド価格の最大パーセンテージを入力します。
9. (オプション) (スポットのみ) スポットフリートロール では、既存の Amazon EC2 スポットフリートの IAM ロールを使用することを選択して、スポットコンピューティング環境に適用します。既存の Amazon EC2 スポットフリート IAM ロールが存在しない場合には、まずこのロールを作成する必要があります。詳細については、「Amazon EC2 スポットフリートロール」を参照してください。

> **Important**
>
> 作成時にスポットインスタンスにタグを付けるには、Amazon EC2 スポットフリート IAM ロールが新しい AmazonEC2SpotFleetTaggingRole 管理ポリシーを使用する必要があります。AmazonEC2SpotFleetRole 管理ポリシーには、スポットインスタンスにタグを付けるために必要なアクセス許可がありません。詳細については、作成時にタグが付けられていないスポットインスタンスおよびthe section called “ リソースのタグ付け”を参照してください。

10. 最小 vCPU では、ジョブキューの需要にかかわらず、コンピューティング環境で維持する EC2 vCPU の最小数を選択します。
11. 必要な vCPU では、コンピューティング環境の起動に必要な EC2 vCPU の数を選択します。AWS Batch は、ジョブキューの需要が増えるにつれて、必要な vCPU の数を増やし、EC2 インスタンスを追加します。vCPU の数は vCPU の最大数まで増やすことができます。需要が減少すると、 は必要な vCPUs の数 AWS Batch を減らし、インスタンスを削除します。この数は vCPU の最小数まで減らすことができます。
12. 最大 vCPU] では、ジョブキューの需要にかかわらず、コンピューティング環境でスケールアウトできる EC2 vCPU の最大数を選択します。
13. 許可されたインスタンスタイプ] では、起動できる Amazon EC2 インスタンスタイプを選択します。インスタンスファミリーを指定してそのファミリー内のいずれかのインスタンスタイプ (c5、c5n、p3など) を起動できます。または、ファミリー内の特定のサイズを指定することもできます (c5.8xlarge)。メタルインスタンスタイプはインスタンスファミリーに含まれていません。たとえば、c5 は c5.metal を含んでいません。また、(C4、M4、および R4 インスタンスファミリーから) optimal を選択して、ジョブキューの需要に見合ったインスタンスタイプを使用することもできます。

> **Note**
>
> コンピューティング環境を作成する際、そのコンピューティング環境で選択するインスタンスタイプで同じアーキテクチャを使用する必要があります。例えば、x86 と ARM インスタンスを同じコンピューティング環境で使用することはできません。

> **Note**
>
> AWS Batch はGPUs をスケーリングします。GPU スケジューリングを使用するには、コンピューティング環境に p2、p3、p4、p5、g3、g3s、g4、または g5 ファミリーのインスタンスタイプを含める必要があります。

> **Note**
>
> 現在、optimal は C4、M4、および R4 インスタンスファミリーのインスタンスタイプを使用しています。これらのインスタンスファミリーのインスタンスタイプ AWS リー

ジョン を持たない では、C5、、M5および インスタンスファミリーのR5インスタンスタイプが使用されます。

14. 追加設定] を展開します。
15. (オプション) プレイスメントグループ では、プレイスメントグループ名を入力して、コンピューティング環境内のリソースをグループ化します。
16. (オプション) EC2 キーペア で、インスタンス接続時のセキュリティ認証情報としてパブリックキーとプライベートキーのペアを選択します。Amazon EC2 キーペアの詳細については、Amazon EC2 キーペアおよび Linux インスタンスを参照してください。
17. 配分戦略] では、許可されるインスタンスタイプのリストからインスタンスタイプを選択するときに使用する配分戦略を選択します。EC2 のオンデマンドコンピューティング環境では BEST_FIT_PROGRESSIVE を、EC2 のスポットコンピューティング環境では SPOT_CAPACITY_OPTIMIZED を選択するのが一般的です。詳細については、the section called “配分戦略”を参照してください。
18. (オプション) EC2 設定 で、EC2 設定を追加 を選択します。イメージタイプとイメージ ID の上書き値を選択して、 がコンピューティング環境のインスタンスに Amazon マシンイメージ (AMIsを選択 AWS Batch するための情報を提供します。イメージ ID の上書きがイメージタイプ ごとに指定されていない場合、 は最新の Amazon ECS 最適化 AMI AWS Batch を選択します。イメージタイプを指定しない場合、デフォルトは GPU 以外の AWS Graviton インスタンス用の Amazon Linux 2 です。

> **Important**
>
> カスタム AMI を使用するには、イメージタイプを選択し、イメージ ID のオーバーライド ボックスにカスタム AMI ID を入力します。

Amazon Linux 2

すべての AWS Graviton ベースのインスタンスファミリー (、、C6gM6gR6g、 などT4g) のデフォルト。GPU 以外のすべてのインスタンスタイプに使用できます。

Amazon Linux 2 (GPU)

すべての GPU インスタンスファミリー (P4や などG4) のデフォルト。非 AWS Graviton ベースのすべてのインスタンスタイプに使用できます。

Amazon Linux

GPU 以外の AWS Graviton インスタンスファミリーに使用できます。Amazon Linux AMI の標準サポートは終了しました。詳細については、Amazon Linux AMI を参照してください。

> **Note**
>
> コンピューティング環境用に選択する AMI は、そのコンピューティング環境で使用するインスタンスタイプのアーキテクチャと一致している必要があります。たとえば、コンピューティング環境で A1 インスタンスタイプを使用する場合、選択するコンピューティングリソース AMI で Arm インスタンスをサポートしている必要があります。Amazon ECS は、Amazon ECS に最適化された Amazon Linux 2 AMI の、x86 と Arm の両バージョンを提供しています。詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS に最適化された Amazon Linux 2 AMIを参照してください。

19. (オプション)起動テンプレート] では、既存の Amazon EC2 起動テンプレートを選択して、コンピューティングリソースを設定します。テンプレートのデフォルトバージョンは自動的に設定されます。詳細については、起動テンプレートのサポートを参照してください。

> **Note**
>
> 起動テンプレートで、作成したカスタム AMI を指定できます。

20. (オプション) 起動テンプレートのバージョン] では、`$Default` または `$Latest` を使用するか、あるいは起用するバージョン番号を指定します。

> **Important**
>
> コンピューティング環境の作成後、起動テンプレートの `$Default` または `$Latest` バージョンが更新されていても、使用する起動テンプレートのバージョンは変更されません。新しい起動テンプレートのバージョンを使用するには、まず新しいコンピューティング環境を作成し、新しいコンピューティング環境を既存のジョブキューに追加します。次に、古いコンピューティング環境をジョブキューから削除し、古いコンピューティング環境を削除します。

21. ネットワーク設定 セクションで:

a. 仮想プライベートクラウド (VPC) ID では、Amazon VPC を選択します。

b. サブネット には、 AWS アカウント のサブネットが一覧表示されます。サブネットのカスタムセットを作成する場合は、サブネットをクリア を選択してから、必要なサブネットを選択します。

> **Important**
>
> コンピューティングリソースは、VPC エンドポイントまたは複数のパブリック IP アドレスを介して Amazon ECS VPC エンドポイントと通信する必要があります。詳細については、Amazon ECR インターフェイス VPC エンドポイント (AWS PrivateLink)をご参照ください。インスタンスに VPC エンドポイントが設定されていない場合やパブリック IP アドレスがない場合は、ネットワークアドレス変換 (NAT) を使用できます。NAT の詳細については、NAT ゲートウェイおよび 仮想プライベートクラウド (VPC) の作成 を参照してください

c. セキュリティグループ で、インスタンスに関連付ける Amazon EC2 セキュリティグループを選択します。セキュリティグループのカスタムセットを作成する場合は、セキュリティグループをクリア を選択します。関連付けるセキュリティグループを選択します。

22. 次へ を選択します。

ジョブキューの作成

ジョブキューは、スケ AWS Batch ジューラがコンピューティング環境のリソースでジョブを実行するまで、送信されたジョブを保存します。詳細については、ジョブキューを参照してください。

Amazon EC2 オーケストレーション用のジョブキューを作成するには、以下の操作を実行します。

1. ジョブキュー設定 セクションの 名前 で、コンピューティング環境の一意の名前を指定します。名前の最大長は 128 文字です。大文字、小文字、数字、ハイフン (-)、アンダースコア (_) を含めることができます。
2. 優先度 では、ジョブキューの 0 から 100 までの整数を入力します。

> **Important**
>
> 整数値が高いほど、 AWS Batch スケジューラーにより高い優先度が割り当てられます。

3. 次へ を選択します。

ジョブ定義の作成

AWS Batch ジョブ定義では、ジョブの実行方法を指定します。各ジョブはジョブ定義を参照する必要がありますが、ジョブ定義に指定されているパラメータの多くはランタイムに上書きできます。

ジョブ定義を作成するには

1. 一般設定 セクションで:

   a. 一般設定 セクションの 名前 で、コンピューティング環境に固有の名前を指定します。名前の最大長は 128 文字です。名前には、大文字および小文字の ASCII 文字、数字、ハイフン (-)、アンダースコア (_) を含めることができます。

   b. (オプション) 実行タイムアウト で、未完了のジョブが終了するまでの時間 (秒単位) を入力します。

   > **Important**
   > 最小タイムアウトは 60 秒です。

   c. (オプション) タグは、リソースに割り当てられるラベルです。タグを追加するには、タグを展開し、タグを追加 を選択します。キーと値のペアを入力し、もう一度 タグを追加 を選択します。

   > **Important**
   > タグを追加 を選択した場合は、キーと値のペアを入力して タグを追加 をもう一度選択するか、タグを解除 を選択する必要があります。

   d. (オプション) タグを伝播 をオンにして、タグをAmazon Elastic Container Service タスクに伝達することができます。

2. コンテナ設定 セクションで次の操作を行います。

   a. イメージ では、コンテナの起動に使用されるイメージの名前を入力します。デフォルトでは、Docker Hub レジストリのイメージをすべて使用できます。他のリポジトリを repository-url/image: tag の形式で指定することもできます。パラメータの最大長は 255 文字です。アルファベット (大文字、小文字)、数字、ハイフン (-)、アンダースコア (_)、コロ

ン (:)、ピリオド (.)、スラッシュ (/)、およびシャープ (#) を含めることができます。このパラメータは、Docker Remote API の コンテナの作成 セクションにある Image にマッピングされ、IMAGE パラメータは docker run にマッピングされます。

> **Note**
>
> Docker イメージのアーキテクチャは、スケジュールされているコンピューティングリソースのプロセッサアーキテクチャと一致している必要があります。例えば、Arm ベースの Docker イメージは、Arm ベースのコンピューティングリソースでのみ実行することができます。

- Amazon ECR Public リポジトリ内のイメージには、完全な `registry/repository[:tag]` または `registry/repository[@digest]` 命名規則が使用されます (例えば、`public.ecr.aws/`*`registry_alias`*`/`*`my-web-app`*`:`*`latest`*)。
- Amazon ECR リポジトリ内のイメージには、完全な `registry/repository:tag` 命名規則が使用されます (例えば、*`aws_account_id`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`my-web-app`*`:`*`latest`*)。
- Docker ハブの公式リポジトリのイメージでは、1 つの名前 (例: ubuntu、mongo) を使用します。
- Docker ハブの他のリポジトリのイメージは、組織名で修飾されます (例: `amazon/amazon-ecs-agent`)。
- 他のオンラインリポジトリのイメージは、さらにドメイン名で修飾されます (例: `quay.io/assemblyline/ubuntu`)。

b. コマンド] では、コンテナに渡すコマンドを指定します。このパラメータは、Docker Remote API の コンテナの作成 セクションにある Cmd にマッピングされ、COMMAND パラメータは docker run にマッピングされます。Docker CMD パラメータの詳細については、https://docs.docker.com/engine/reference/builder/#cmd を参照してください。

> **Note**
>
> コマンドには、パラメータ置換のデフォルト値とプレースホルダーを使用できます。詳細については、パラメータを参照してください。

c. (オプション) 実行ロール で、ユーザーに代わって AWS API コールを実行するためのアクセス許可を Amazon ECS コンテナエージェントに付与する IAM ロールを指定できます。この

機能では、タスク用の Amazon ECS IAM ロールを使用します。詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS タスク実行 IAM ロールを参照してください。

d. (オプション) ジョブロール設定 で、 AWS APIs へのアクセス許可を持つ IAM ロールを選択します。この機能では、タスク用の Amazon ECS IAM ロールを使用します。詳細については、Amazon Elastic Container Service デベロッパーガイドのタスク用の IAM ロールを参照してください。

> **Note**
>
> ここには、Amazon Elastic Container Service Task Role] 信頼関係があるロールのみが表示されます。 AWS Batch ジョブ用の IAM ロールの作成の詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「タスク用の IAM ロールとポリシーの作成」を参照してください。

e. (オプション) パラメータをキーと値のペアとしてジョブ定義に追加して、ジョブ定義のデフォルトを上書きできます。パラメータを追加するには:

- パラメータ で、パラメータを追加 を選択します。キーと値のペアを入力し、もう一度 パラメータを追加 を選択します。

> **Important**
>
> パラメータを追加 を選択した場合は、少なくとも 1 つのパラメータを設定するか、パラメータの削除 を選択する必要があります。

f. 環境設定 セクションの vCPU で、コンテナ用に予約する vCPU の数を指定します。このパラメータは、Docker Remote API の コンテナの作成 セクションにある `CpuShares` にマッピングされ、`--cpu-shares` オプションは docker run にマッピングされます。各 vCPU は 1,024 個の CPU 配分に相当します。

g. メモリ では、ジョブのコンテナに適用されるメモリのハード制限 (MiB 単位) を指定します。コンテナは、ここで指定したメモリを超えようとすると、停止されます。このパラメータは、Docker Remote API の コンテナの作成 セクションにある `Memory` にマッピングされ、`--memory` オプションは docker run にマッピングされます。

h. GPU の数 では、コンテナ用に予約する GPU の数を指定します。

i. (オプション) 環境変数設定 では、環境変数を追加 を選択し、コンテナに渡す環境変数を追加します。このパラメータは、Docker Remote API の コンテナの作成 セクションにある Env にマッピングされ、--env オプションは docker run にマッピングされます。

j. (オプション) シークレット で、シークレットを追加 を選択して、シークレットを名前と値のペアとして追加します。これらのシークレットはコンテナに公開されます。詳細については、のジョブ定義パラメータ ContainerProperties のsecretOptionsを参照してください。

k. (オプション) Linux 設定 セクションで:

   i. ユーザー] には、コンテナ内で使用するユーザー名を入力します。このパラメータは、Docker Remote API の コンテナの作成 セクションにある User にマッピングされ、--user オプションは docker run にマッピングされます。

   ii. 特権付与 スライダーを右にドラッグすることで、ホストインスタンスに対する昇格されたアクセス許可 (root ユーザーと同様) をジョブコンテナに付与することができます。このパラメータは、Docker Remote API の コンテナの作成 セクションにある Privileged にマッピングされ、--privileged オプションは docker run にマッピングされます。

   iii. init プロセスを有効にする をオンにすると、コンテナ内で init プロセスを実行できます。このプロセスは信号を転送し、プロセスを利用します。

l. (オプション) ファイルシステム設定 セクションで :

   i. 読み取り専用ファイルシステムを有効にする をオンにして、ボリュームへの書き込みアクセスを削除します。

   ii. 共有メモリサイズ では、/dev/shm ボリュームのサイズ (MiB) を入力します。

   iii. 最大スワップサイズ では、コンテナが使用できるスワップメモリの合計容量 (MiB 単位) を入力します。

   iv. スワップ動作 では、コンテナのスワップ動作を示す 0 ~ 100 の値を入力します。値を指定せず、スワップが有効になっている場合、この値はデフォルトで 60 に設定されます。詳細については、のジョブ定義パラメータ ContainerProperties のスワップ動作を参照してください。

   v. (オプション) 追加設定] を展開します。

   vi. Tmpfs では、tmpfs を追加 を選択して tmpfs マウントを追加します。

   vii. デバイス で、デバイスを追加 を選択してデバイスを追加します。

A. コンテナパス] では、コンテナインスタンスでのデバイスのパスを指定します。このパスは、ホストインスタンスにマッピングされたデバイスを公開するために使用されます。空白のままにすると、ホストパスがコンテナで使用されます。

B. ホストパス] では、ホストインスタンスでのデバイスのパスを指定します。

C. アクセス許可 では、デバイスに適用する 1 つ以上のアクセス許可を選択します。使用できる権限は、読み取り、書き込み、MKNOD です。

viii. (オプション) Ulimit 設定 では、ulimit を追加 を選択して、コンテナの `ulimits` 値を追加します。名前、ソフトリミット、ハードリミット の値を入力し、ulimit を追加 を選択します。

3. 次へ を選択します。

ジョブの作成

ジョブを作成するには、以下の手順を実行します。

1. ジョブの設定 セクションの 名前 で、ジョブの一意の名前を指定します。名前の最大長は 128 文字です。大文字、小文字、数字、ハイフン (-)、アンダースコア (_) を含めることができます。
2. 次へ を選択します。

確認と作成

レビューと作成では、設定手順を確認してください。変更する必要がある場合は、編集 を選択します。完了したら、リソースを作成 を選択します。

Fargate のご利用開始にあたって

AWS Fargate は、コンテナに指定したリソース要件に厳密に一致するようにコンピューティングを起動してスケーリングします。Fargate を使用すると、追加のサーバーに対してオーバープロビジョニングまたは料金を支払う必要はありません。詳細については、Fargateを参照してください。

コンピューティング環境の作成

Fargate オーケストレーション用のコンピューティング環境を作成するには、以下の操作を実行します。

1. AWS Batch コンソールの初回実行ウィザード を開きます。

2. オーケストレーションタイプの選択 で Fargate を選択します。
3. 次へ を選択します。
4. コンピューティング環境設定 の 名前 で、コンピューティング環境の一意な名前を指定します。名前の最大長は 128 文字です。大文字、小文字、数字、ハイフン (-)、アンダースコア (_) を含めることができます。
5. (オプション) タグは、リソースに割り当てられるラベルです。タグを追加するには、タグ を展開し、タグを追加 を選択します。キーと値のペアを入力し、もう一度 タグを追加 を選択します。

   > **Important**
   >
   > タグを追加 を選択した場合は、キーと値のペアを入力して タグを追加 をもう一度選択するか、タグを解除 を選択する必要があります。

6. (オプション) インスタンス設定 セクションの Fargate スポット容量を使用する で、スポットインスタンスの使用を有効にする をオンにします。
7. 最大 vCPU では、インスタンスが使用できるvCPU の最大数を入力します。
8. ネットワーク設定 セクションで:

   a. 仮想プライベートクラウド (VPC) ID では、Amazon VPC を選択します。
   b. サブネット には、 AWS アカウント のサブネットが一覧表示されます。サブネットのカスタムセットを作成する場合は、サブネットをクリア を選択してから、必要なサブネットを選択します。

      > **Important**
      >
      > コンピューティングリソースは、VPC エンドポイントまたは複数のパブリック IP アドレスを介して Amazon ECS VPC エンドポイントと通信する必要があります。詳細については、Amazon ECR インターフェイス VPC エンドポイント (AWS PrivateLink)をご参照ください。インスタンスに VPC エンドポイントが設定されていない場合やパブリック IP アドレスがない場合は、ネットワークアドレス変換 (NAT) を使用できます。NAT の詳細については、NAT ゲートウェイおよび 仮想プライベートクラウド (VPC) の作成 を参照してください

   c. セキュリティグループ で、インスタンスに関連付ける Amazon EC2 セキュリティグループを選択します。セキュリティグループのカスタムセットを作成する場合は、セキュリティグループをクリア を選択します。関連付けるセキュリティグループを選択します。

9. 次へ を選択します。

ジョブキューの作成

ジョブキューは、スケ AWS Batch ジューラがコンピューティング環境のリソースでジョブを実行するまで、送信されたジョブを保存します。ジョブキューを作成するには:

Fargate オーケストレーションのジョブキューを作成するには、以下の操作を実行します。

1. ジョブキュー設定 セクションの 名前 で、コンピューティング環境の一意の名前を指定します。名前の最大長は 128 文字です。大文字、小文字、数字、ハイフン (-)、アンダースコア (_) を含めることができます。
2. 優先度 では、ジョブキューの 0 から 100 までの整数を入力します。

   > **Important**
   >
   > 整数値が高いほど、 AWS Batch スケジューラーにより高い優先度が割り当てられます。

3. 次へ を選択します。

ジョブ定義の作成

ジョブ定義を作成するには

1. 一般設定 セクションで:

   a. 名前 では、カスタムジョブ定義名を入力します。

      一般設定 セクションの 名前 で、コンピューティング環境に固有の名前を指定します。名前の最大長は 128 文字です。大文字、小文字、数字、ハイフン (-)、アンダースコア (_) を含めることができます。

   b. (オプション) 実行タイムアウト で、未完了のジョブが終了するまでの時間 (秒単位) を入力します。

> **Important**
>
> 最小タイムアウトは 60 秒です。

c. (オプション) タグは、リソースに割り当てられるラベルです。タグを追加するには、タグを展開し、タグを追加 を選択します。キーと値のペアを入力し、もう一度 タグを追加 を選択します。

> **Important**
>
> タグを追加 を選択した場合は、キーと値のペアを入力して タグを追加 をもう一度選択するか、タグを解除 を選択する必要があります。

d. (オプション) タグを伝播 をオンにして、タグをAmazon Elastic Container Service タスクに伝達することができます。

2. Fargate プラットフォーム設定 セクションで:

a. (オプション) Fargate プラットフォームバージョン では、必要な特定のランタイム環境を入力します。

b. ランタイムプラットフォーム では、LINUX または Windows を選択します。

c. (Windows のみ) オペレーティングシステムファミリー では、オペレーティングシステムを選択します。

d. CPU アーキテクチャ では、必要な CPU アーキテクチャを選択します。

e. (オプション) パブリック IP アドレスの割り当て をオンにして、パブリック IP アドレスを割り当てます。

f. エフェメラルストレージ で、必要なエフェメラルストレージの容量を入力します。

> **Note**
>
> デフォルトでは、20 GiB のエフェメラルストレージが使用されます。追加のエフェメラルストレージを使用するには、21 GiB から 100 GiB までの値を入力します。

g. 実行ロール で、Amazon Elastic Container Service (Amazon ECS) エージェントがユーザーに代わって を AWS 呼び出すことを許可するタスク実行ロールを選択します。例えば、ecsTaskExecutionロール を選択できます。

3. コンテナ設定 セクションで次の操作を行います。

a. イメージ では、コンテナの起動に使用されるイメージの名前を入力します。デフォルトでは、Docker Hub レジストリのイメージをすべて使用できます。他のリポジトリを repository-url/image: tag の形式で指定することもできます。パラメータの最大長は 255 文字です。大文字、小文字、数字、ハイフン (-)、アンダースコア (_)、コロン (:)、ピリオド (.)、スラッシュ (/)、および数字 (#) を含めることができます。このパラメータは、Docker Remote API の コンテナの作成 セクションにある `Image` にマッピングされ、`IMAGE` パラメータは docker run にマッピングされます。

> **Note**
>
> Docker イメージのアーキテクチャは、スケジュールされているコンピューティングリソースのプロセッサアーキテクチャと一致している必要があります。例えば、Arm ベースの Docker イメージは、Arm ベースのコンピューティングリソースでのみ実行することができます。

- Amazon ECR Public リポジトリ内のイメージには、完全な `registry/repository[:tag]` または `registry/repository[@digest]` 命名規則が使用されます (例えば、`public.ecr.aws/`*`registry_alias`*`/`*`my-web-app`*`:`*`latest`*)。
- Amazon ECR リポジトリ内のイメージには、完全な `registry/repository:tag` 命名規則が使用されます (例えば、*`aws_account_id`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`my-web-app`*`:`*`latest`*)。
- Docker ハブの公式リポジトリのイメージでは、1 つの名前 (例: ubuntu、mongo) を使用します。
- Docker ハブの他のリポジトリのイメージは、組織名で修飾されます (例: `amazon/amazon-ecs-agent`)。
- 他のオンラインリポジトリのイメージは、さらにドメイン名で修飾されます (例: `quay.io/assemblyline/ubuntu`)。

b. コマンド] では、コンテナに渡すコマンドを指定します。このパラメータは、Docker Remote API の コンテナの作成 セクションにある `Cmd` にマッピングされ、`COMMAND` パラメータは docker run にマッピングされます。Docker `CMD` パラメータの詳細については、https://docs.docker.com/engine/reference/builder/#cmd を参照してください。

> **Note**
>
> コマンドには、パラメータ置換のデフォルト値とプレースホルダーを使用できます。詳細については、パラメータを参照してください。

> **Tip**
>
> 情報 を選択して、Bash と JSON のコード例を確認できます。

c. (オプション) パラメータをキーと値のペアとしてジョブ定義に追加して、ジョブ定義のデフォルトを上書きできます。パラメータを追加するには:

- パラメータ で、パラメータを追加 を選択します。キーと値のペアを入力し、もう一度パラメータを追加 を選択します。

> **Important**
>
> パラメータを追加 を選択した場合は、少なくとも 1 つのパラメータを設定するか、パラメータの削除 を選択する必要があります。

d. (オプション) ジョブロール設定 の環境設定 セクションで、 AWS APIs を使用するアクセス許可を提供する IAM ロールを選択します。

e. 環境設定 セクションの vCPU で、コンテナ用に予約する vCPU の数を指定します。このパラメータは、Docker Remote API の コンテナの作成 セクションにある `CpuShares` にマッピングされ、`--cpu-shares` オプションは docker run にマッピングされます。各 vCPU は 1,024 個の CPU 配分に相当します。

f. メモリ では、ジョブのコンテナに適用されるメモリのハード制限 (MiB 単位) を指定します。コンテナは、ここで指定したメモリを超えようとすると、停止されます。このパラメータは、Docker Remote API の コンテナの作成 セクションにある `Memory` にマッピングされ、`--memory` オプションは docker run にマッピングされます。

g. (オプション) 環境変数 では、環境変数の追加 を選択し、コンテナに渡す環境変数を追加します。このパラメータは、Docker Remote API の コンテナの作成 セクションにある `Env` にマッピングされ、`--env` オプションは docker run にマッピングされます。

4. 次へ を選択します。

ジョブの作成

Fargateジョブを作成するには、以下の手順を実行します。

1. ジョブの設定 セクションの 名前 で、ジョブの一意の名前を指定します。名前の最大長は 128 文字です。大文字、小文字、数字、ハイフン (-)、アンダースコア (_) を含めることができます。
2. 次へ を選択します。

確認と作成

レビューと作成では、設定手順を確認してください。変更する必要がある場合は、編集 を選択します。完了したら、リソースを作成 を選択します。

Amazon EKS AWS Batch での の開始方法

AWS Batch on Amazon EKS は、バッチワークロードを既存の Amazon EKS クラスターにスケジューリングおよびスケーリングするためのマネージドサービスです。 AWS Batch は、ユーザーに代わって Amazon EKS クラスターのライフサイクルオペレーションを作成、管理、実行しません。 AWS Batch orchestration は、 が管理するノードをスケールアップおよびスケールダウン AWS Batch し、それらのノードでポッドを実行します。

AWS Batch は、Amazon EKS クラスター内の AWS Batch コンピューティング環境に関連付けられていないノード、Auto Scaling ノードグループ、またはポッドのライフサイクルには影響しません。が効果的に動作 AWS Batch するためには、そのサービスにリンクされたロールに、既存の Amazon EKS クラスターでKubernetesロールベースのアクセスコントロール (RBAC) アクセス許可が必要です。詳細については、Kubernetes ドキュメントのRBAC 認証の使用を参照してください。

AWS Batch には、ポッドを AWS Batch ジョブとしてスコープできるKubernetes名前空間が必要です。 AWS Batch ポッドを他のクラスターワークロードから分離するには、専用の名前空間を使用することをお勧めします。

AWS Batch に RBAC アクセスが付与され、名前空間が確立されたら、 CreateComputeEnvironment API オペレーションを使用して、その Amazon EKS クラスターを AWS Batch コンピューティング環境に関連付けることができます。ジョブキューは、この新しい Amazon EKS コンピューティング環境に関連付けることができます。 AWS Batch ジョブは、 SubmitJob API オペレーションを使用して Amazon EKS ジョブ定義に基づいてジョブキューに送信されます。 AWS Batch 次に、 は AWS Batch マネージドノードを起動し、ジョブキューからポッドとしてジョブを AWS Batch コンピューティング環境に関連付けられた EKS Kubernetes クラスターに配置します。

以下のセクションでは、Amazon EKS AWS Batch で のセットアップ方法について説明します。

目次

前提条件

このチュートリアルを開始する前に、 と Amazon EKS リソースの両方を作成および管理するために必要な以下のツールとリソースをインストール AWS Batch して設定する必要があります。

- AWS CLI – Amazon EKS など AWS のサービスを操作するためのコマンドラインツールです。このガイドでは、バージョン 2.8.6 以降または 1.26.0 以降の使用を想定しています。詳細については、「AWS Command Line Interface ユーザーガイド」の「AWS CLIのインストール、更新、およびアンインストール」を参照してください。をインストールしたら AWS CLI、設定もお勧めします。詳細については、AWS Command Line Interface ユーザーガイドの `aws configure` を使用したクイック設定を参照してください。
- **`kubectl`** - Kubernetes クラスターを操作するためのコマンドラインツール。このガイドでは、バージョン `1.23` 以降の使用を想定しています。詳細については、Amazon EKS ユーザーガイドの`kubectl` のインストールまたは更新を参照してください。

- **`eksctl`** — 多くの個々のタスクを自動化する Amazon EKS クラスターを操作するためのコマンドラインツール。このガイドでは、バージョン `0.115.0` 以降の使用を想定しています。詳細については、Amazon EKS ユーザーガイドの`eksctl` のインストールまたは更新を参照してください。
- 必要な IAM アクセス許可 – 使用する IAM セキュリティプリンシパルには、Amazon EKS IAM ロールとサービスにリンクされたロール AWS CloudFormation、VPC および関連リソースを操作するためのアクセス許可が必要です。詳細については、IAM ユーザーガイドの「Amazon Elastic Kubernetes Service のアクション、リソース、および条件キー」および「サービスにリンクされたロールの使用」を参照してください。このガイドのすべての手順は、1 つのユーザーとして実行する必要があります。
- Amazon EKS クラスターの作成 – 詳細については、「Amazon EKS ユーザーガイド」の`eksctl`「Amazon EKS の開始方法」を参照してください。

> **Note**
>
> AWS Batch は、パブリックインターネットにアクセスできるパブリックアクセス権を持つ API サーバーエンドポイントを持つ Amazon EKS クラスターのみをサポートします。デフォルトでは、Amazon EKS クラスター API サーバーエンドポイントにはパブリックアクセスがあります。詳細については、「Amazon EKS ユーザーガイド」の 「Amazon EKS クラスターエンドポイントアクセスコントロール」を参照してください。

> **Note**
>
> AWS Batch は、CoreDNS やその他のデプロイポッドのマネージドノードオーケストレーションを提供しません。CoreDNS が必要な場合は、Amazon EKS ユーザーガイドのCoreDNS Amazon EKS アドオンの追加を参照してください。または、`eksctl create cluster create` を使用してクラスターを作成すると、クラスターにはデフォルトで CoreDNS が含まれています。

- アクセス許可 – Amazon EKS リソースを使用するコンピューティング環境を作成するために CreateComputeEnvironment API オペレーションを呼び出すユーザーには、`eks:DescribeCluster` API オペレーションに対するアクセス許可が必要です。を使用して Amazon EKS リソースを使用してコンピューティングリソース AWS Management Console を作成するには、 `eks:DescribeCluster`と の両方に対するアクセス許可が必要です`eks:ListClusters`。

ステップ 1: 用の Amazon EKS クラスターを準備する AWS Batch

すべての手順を実行する必要があります。

1. AWS Batch ジョブ専用の名前空間を作成する

   kubectl を使用して新しい名前空間を作成します。

```
$ namespace=my-aws-batch-namespace
$ cat - <<EOF | kubectl create -f -
{
  "apiVersion": "v1",
  "kind": "Namespace",
  "metadata": {
    "name": "${namespace}",
    "labels": {
      "name": "${namespace}"
    }
  }
}
EOF
```

   出力:

```
namespace/my-aws-batch-namespace created
```

2. ロールベースアクセス制御 (RBAC) を有効にします。

   kubectl を使用してクラスターのKubernetesロールを作成し、 AWS Batch がノードとポッドを監視できるようにし、ロールをバインドできるようにします。これを EKS クラスターごとに 1 回実行する必要があります。

> **Note**
>
> RBAC 認証の使用の詳細については、「 ユーザーガイド」の「RBAC 認証の使用」を参照してください。 Kubernetes

```
$ cat - <<EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: ClusterRole
metadata:
  name: aws-batch-cluster-role
rules:
  - apiGroups: [""]
    resources: ["namespaces"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["configmaps"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["apps"]
    resources: ["daemonsets", "deployments", "statefulsets", "replicasets"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["clusterroles", "clusterrolebindings"]
    verbs: ["get", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: aws-batch-cluster-role-binding
subjects:
- kind: User
  name: aws-batch
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: aws-batch-cluster-role
  apiGroup: rbac.authorization.k8s.io
EOF
```

出力:

```
clusterrole.rbac.authorization.k8s.io/aws-batch-cluster-role created
clusterrolebinding.rbac.authorization.k8s.io/aws-batch-cluster-role-binding created
```

の名前空間スコープのKubernetesロールを作成して AWS Batch 、ポッドを管理およびライフサイクルし、バインドします。これは固有の名前空間ごとに 1 回行う必要があります。

```
$ namespace=my-aws-batch-namespace
$ cat - <<EOF | kubectl apply -f - --namespace "${namespace}"
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: aws-batch-compute-environment-role
  namespace: ${namespace}
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["create", "get", "list", "watch", "delete", "patch"]
  - apiGroups: [""]
    resources: ["serviceaccounts"]
    verbs: ["get", "list"]
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["roles", "rolebindings"]
    verbs: ["get", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: aws-batch-compute-environment-role-binding
  namespace: ${namespace}
subjects:
- kind: User
  name: aws-batch
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: aws-batch-compute-environment-role
  apiGroup: rbac.authorization.k8s.io
EOF
```

出力:

```
role.rbac.authorization.k8s.io/aws-batch-compute-environment-role created
rolebinding.rbac.authorization.k8s.io/aws-batch-compute-environment-role-binding
 created
```

Kubernetes aws-auth 設定マップを更新して、前述の RBAC アクセス許可を AWS Batch サービスにリンクされたロールにマッピングします。

```
$ eksctl create iamidentitymapping \
    --cluster my-cluster-name \
    --arn "arn:aws:iam::<your-account>:role/AWSServiceRoleForBatch" \
    --username aws-batch
```

出力:

```
2022-10-25 20:19:57 [#]  adding identity "arn:aws:iam::<your-account>:role/
AWSServiceRoleForBatch" to auth ConfigMap
```

> **Note**
>
> パス aws-service-role/batch.amazonaws.com/が、サービスにリンクされたロールの ARN から削除されました。これは aws-auth 設定マップに問題があるためです。詳細については、「」の「パスを含むロールは、パスが ARN に含まれていると機能しませんaws-authconfigmap」を参照してください。

ステップ 2: Amazon EKS コンピューティング環境を作成する

AWS Batch コンピューティング環境は、バッチワークロードのニーズに合わせてコンピューティングリソースパラメータを定義します。マネージド型のコンピューティング環境 AWS Batch では、Amazon EKS クラスター内のコンピューティングリソース (Kubernetes ノード) の容量とインスタンスタイプを管理できます。これは、コンピューティング環境の作成時に定義するコンピューティングリソースの仕様に基づいています。EC2 オンデマンドインスタンスまたは EC2 スポットインスタンスを選択できます。

AWSServiceRoleForBatch サービスにリンクされたロールが Amazon EKS クラスターにアクセスできるようになったので、 AWS Batch リソースを作成できます。まず、Amazon EKS クラスターを指すコンピューティング環境を作成します。

```
$ cat <<EOF > ./batch-eks-compute-environment.json
{
  "computeEnvironmentName": "My-Eks-CE1",
  "type": "MANAGED",
```

```
  "state": "ENABLED",
  "eksConfiguration": {
    "eksClusterArn": "arn:aws:eks:<region>:123456789012:cluster/<cluster-name>",
    "kubernetesNamespace": "my-aws-batch-namespace"
  },
  "computeResources": {
    "type": "EC2",
    "allocationStrategy": "BEST_FIT_PROGRESSIVE",
    "minvCpus": 0,
    "maxvCpus": 128,
    "instanceTypes": [
        "m5"
    ],
    "subnets": [
        "<eks-cluster-subnets-with-access-to-internet-for-image-pull>"
    ],
    "securityGroupIds": [
        "<eks-cluster-sg>"
    ],
    "instanceRole": "<eks-instance-profile>"
  }
}
EOF
$ aws batch create-compute-environment --cli-input-json file://./batch-eks-compute-environment.json
```

メモ

- serviceRole パラメータを指定しないでください。サービス AWS Batch にリンクされたロールが使用されます。Amazon EKS AWS Batch では、 AWS Batch サービスにリンクされたロールのみがサポートされます。
- Amazon EKS コンピューティング環境で はSPOT_CAPACITY_OPTIMIZED、、BEST_FIT_PROGRESSIVE、および SPOT_PRICE_CAPACITY_OPTIMIZEDの割り当て戦略のみがサポートされています。

> **Note**
>
> ほとんどの場合、SPOT_CAPACITY_OPTIMIZED ではなく SPOT_PRICE_CAPACITY_OPTIMIZED を使用することをお勧めします。

- `instanceRole` については、Amazon EKS ユーザーガイドのAmazon EKS ノード IAM ロールの作成とクラスターへの IAM プリンシパルアクセスを有効にするを参照してください。ポッドネットワークを使用している場合は、Amazon EKS ユーザーガイドのサービスアカウントに IAM ロールを使用するための Kubernetes Amazon VPC CNI プラグインの設定を参照してください。
- `subnets` パラメータのサブネットを動作させる方法の 1 つとして、Amazon EKS クラスターの作成時に `eksctl` が作成した Amazon EKS マネージドノードグループのパブリックサブネットを使用することができます。それ以外の場合は、イメージをプルできるネットワークパスを持つサブネットを使用します。
- `securityGroupIds` パラメータには、Amazon EKS クラスターと同じセキュリティグループを使用できます。このコマンドは、クラスターのセキュリティグループ ID を取得します。

```
$ eks describe-cluster \
    --name <cluster-name> \
    --query cluster.resourcesVpcConfig.clusterSecurityGroupId
```

- Amazon EKS コンピューティング環境のメンテナンスは責任共有です。詳細については、「Kubernetes ノードの責任分担」を参照してください。

> **Important**
>
> 処理を進める前に、コンピューティング環境が正常であることを確認することが重要です。DescribeComputeEnvironments API オペレーションを使用してこれを行うことができます。
>
> ```
> $ aws batch describe-compute-environments --compute-environments My-Eks-CE1
> ```
>
> `status` パラメータが `INVALID` ではないことを確認してください。そうであれば、`statusReason` パラメータを調べて原因を調べてください。詳細については、トラブルシューティング AWS Batchを参照してください。

ステップ 3: ジョブキューを作成してコンピューティング環境をアタッチする

```
$ aws batch describe-compute-environments --compute-environments My-Eks-CE1
```

この新しいジョブキューに送信されたジョブは、コンピューティング環境に関連付けられた Amazon EKS クラスターに参加した AWS Batch マネージドノードでポッドとして実行されます。

```
$ cat <<EOF > ./batch-eks-job-queue.json
 {
    "jobQueueName": "My-Eks-JQ1",
    "priority": 10,
    "computeEnvironmentOrder": [
      {
        "order": 1,
        "computeEnvironment": "My-Eks-CE1"
      }
    ]
  }
EOF
$ aws batch create-job-queue --cli-input-json file://./batch-eks-job-queue.json
```

ステップ 4: ジョブ定義を作成する

```
$ cat <<EOF > ./batch-eks-job-definition.json
{
  "jobDefinitionName": "MyJobOnEks_Sleep",
  "type": "container",
  "eksProperties": {
    "podProperties": {
      "hostNetwork": true,
      "containers": [
        {
          "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
          "command": [
            "sleep",
            "60"
          ],
          "resources": {
            "limits": {
              "cpu": "1",
              "memory": "1024Mi"
            }
          }
        }
      ],
      "metadata": {
        "labels": {
```

```
          "environment": "test"
        }
      }
    }
  }
}
EOF
$ aws batch register-job-definition --cli-input-json file://./batch-eks-job-
definition.json
```

メモ

- 単一のコンテナジョブのみがサポートされます。
- cpu および memory パラメータには考慮事項があります。詳細については、Amazon EKSAWS Batchのメモリとvcpuに関する考慮事項を参照してください。

ステップ 5: ジョブを送信する

```
$ aws batch submit-job --job-queue My-Eks-JQ1 \
    --job-definition MyJobOnEks_Sleep --job-name My-Eks-Job1
$ aws batch describe-jobs --job <jobId-from-submit-response>
```

メモ

- 単一のコンテナジョブのみがサポートされます。
- cpu および memory パラメータに関連するすべての考慮事項をよく理解しておいてください。詳細については、「Amazon EKSAWS Batchのメモリとvcpuに関する考慮事項」を参照してください。
- Amazon EKS リソースでのジョブの実行の詳細については、「」を参照してくださいAmazon EKS ジョブ。

(オプション) オーバーライドを含むジョブを送信する

このジョブは、コンテナに渡されたコマンドをオーバーライドします。

```
$ cat <<EOF > ./submit-job-override.json
{
  "jobName": "EksWithOverrides",
```

```
  "jobQueue": "My-Eks-JQ1",
  "jobDefinition": "MyJobOnEks_Sleep",
  "eksPropertiesOverride": {
    "podProperties": {
      "containers": [
        {
          "command": [
            "/bin/sh"
          ],
          "args": [
            "-c",
            "echo hello world"
          ]
        }
      ]
    }
  }
}
EOF
$ aws batch submit-job --cli-input-json file://./submit-job-override.json
```

メモ

- AWS Batch は、ジョブの完了後にポッドを積極的にクリーンアップして、 への負荷を軽減しますKubernetes。ジョブの詳細を確認するには、ログ記録を設定する必要があります。詳細については、CloudWatch Logs を使用して Amazon EKS ジョブにおける AWS Batch をモニタリングするを参照してください。
- 操作の詳細を把握しやすくするには、Amazon EKS コントロールプレーンのログ記録を有効にします。詳細については、Amazon EKS ユーザーガイドのAmazon EKS クラスターコントロールプレーンのログ記録を参照してください。
- Daemonsets と kubelets オーバーヘッドは、使用可能な vCPU とメモリのリソース、特にスケーリングとジョブの配置に影響します。詳細については、Amazon EKSAWS BatchのメモリとvCPUに関する考慮事項を参照してください。

Amazon EKS プライベートクラスター AWS Batch での の開始方法

AWS Batch は、Amazon Elastic Kubernetes Service (Amazon EKS) クラスター内のバッチワークロードをオーケストレートするマネージドサービスです。これには、キューイング、依存関係の追跡、マネージドジョブの再試行と優先順位、ポッド管理、ノードスケーリングが含まれます。この機能は、既存のプライベート Amazon EKS クラスターを に接続 AWS Batch して、大規模なジョブ

を実行します。eksctl (Amazon EKS のコマンドラインインターフェイス) 、 AWS コンソール、または AWS Command Line Interface を使用して、他のすべての必要なリソースを含むプライベート Amazon EKS クラスターを作成できます。でのプライベート Amazon EKS クラスターのサポート AWS Batch は、 AWS リージョン が利用可能な商用 AWS Batch で一般的に利用できます。

Amazon EKS プライベート専用クラスターには、インバウンド/アウトバウンドのインターネットアクセスがなく、プライベートサブネットのみがあります。Amazon VPC エンドポイントは、他の AWS のサービスへのプライベートアクセスを有効にするために使用されます。 は、既存の Amazon VPC とサブネットを使用したフルプライベートクラスターの作成eksctlをサポートしています。eksctlまた、 は、提供された Amazon VPC に Amazon VPC エンドポイントを作成し、指定されたサブネットのルートテーブルを変更します。

eksctl はメインルートテーブルを変更しないため、各サブネットには明示的なルートテーブルが関連付けられている必要があります。クラスターは、Amazon VPC 内のコンテナレジストリからイメージをプルする必要があります。また、Amazon VPC に Amazon Elastic Container Registry を作成し、コンテナイメージをそのレジストリにコピーして、ノードがプルできるようにすることもできます。詳細については、「あるリポジトリから別のリポジトリにコンテナイメージをコピーする」を参照してください。Amazon ECR プライベートリポジトリの使用を開始するには、「Amazon ECR プライベートリポジトリ」を参照してください。

オプションで、Amazon ECR を使用してプルスルーキャッシュルールを作成できます。外部パブリックレジストリのプルスルーキャッシュルールが作成されたら、Amazon ECR プライベートレジストリ URIform リソース識別子 (URI) を使用して、その外部パブリックレジストリからイメージをプルできます。次に、Amazon ECR はリポジトリを作成し、イメージをキャッシュします。Amazon ECR プライベートレジストリ URI を使用してキャッシュされたイメージをプルすると、Amazon ECR はリモートレジストリをチェックしてイメージの新しいバージョンがあるかどうかを確認し、24 時間ごとに 1 回までプライベートレジストリを更新します。

目次

- トラブルシューティング

前提条件

このチュートリアルを開始する前に、 と Amazon EKS リソースの両方を作成および管理するために必要な以下のツール AWS Batch とリソースをインストールして設定する必要があります。また、VPC、サブネット、ルートテーブル、VPC エンドポイント、Amazon EKS クラスターなど、必要なすべてのリソースを作成する必要があります。を使用する必要があります AWS CLI。

- AWS CLI – Amazon EKS などの AWS のサービスで使用するコマンドラインツール。このガイドでは、バージョン 2.8.6 以降または 1.26.0 以降の使用を想定しています。詳細については、「AWS Command Line Interface ユーザーガイド」の「AWS CLIのインストール、更新、およびアンインストール」を参照してください。

  をインストールしたら AWS CLI、設定することをお勧めします。詳細については、AWS Command Line Interface ユーザーガイドの aws configure を使用したクイック設定を参照してください。
- **kubectl** – Kubernetesクラスターを操作するコマンドラインツール。このガイドでは、バージョン 1.23 以降の使用を想定しています。詳細については、Amazon EKS ユーザーガイドのkubectl のインストールまたは更新を参照してください。
- **eksctl** – 多くの個々のタスクを自動化する Amazon EKS クラスターと連携するコマンドラインツール。このガイドでは、バージョン 0.115.0 以降の使用を想定しています。詳細については、Amazon EKS ユーザーガイドのeksctl のインストールまたは更新を参照してください。
- 必須 AWS Identity and Access Management (IAM) アクセス許可 – 使用する IAM セキュリティプリンシパルには、Amazon EKS IAM ロールとサービスにリンクされたロール AWS CloudFormation、および VPC と関連リソースを操作するためのアクセス許可が必要です。詳細については、IAM ユーザーガイドの「Amazon Elastic Kubernetes Service のアクション、リソース、および条件キー」および「サービスにリンクされたロールの使用」を参照してください。このガイドのすべての手順は、1 つのユーザーとして実行する必要があります。
- Amazon EKS クラスターの作成 – 詳細については、「Amazon EKS ユーザーガイド」のeksctl「Amazon EKS の開始方法」を参照してください。

> **Note**
>
> AWS Batch は、CoreDNS やその他のデプロイポッドのマネージドノードオーケストレーションを提供しません。CoreDNS が必要な場合は、Amazon EKS ユーザーガイドのCoreDNS Amazon EKS アドオンの追加を参照してください。または、eksctl

`create cluster create` を使用してクラスターを作成すると、クラスターにはデフォルトで CoreDNS が含まれています。

- アクセス許可 – Amazon EKS リソースを使用するコンピューティング環境を作成するために CreateComputeEnvironment API オペレーションを呼び出すユーザーには、`eks:DescribeCluster` API オペレーションに対するアクセス許可が必要です。を使用して Amazon EKS リソースを使用してコンピューティングリソース AWS Management Console を作成するには、 `eks:DescribeCluster`と の両方に対するアクセス許可が必要ですeks:ListClusters。
- サンプル設定ファイルを使用して、us-east-1 リージョンにプライベート EKS `eksctl` クラスターを作成します。

```
kind: ClusterConfig
apiVersion: eksctl.io/v1alpha5
availabilityZones:
    - us-east-1a
    - us-east-1b
    - us-east-1d
managedNodeGroups:
    privateNetworking: true
privateCluster:
    enabled: true
    skipEndpointCreation: false
```

コマンドを使用してリソースを作成します。 `eksctl create cluster -f clusterConfig.yaml`

- バッチマネージドノードは、必要な VPC インターフェイスエンドポイントを持つサブネットにデプロイする必要があります。詳細については、「プライベートクラスターの要件」を参照してください。

ステップ 1: 用の EKS クラスターを準備する AWS Batch

すべての手順を実行する必要があります。

1. AWS Batch ジョブ専用の名前空間を作成する

   `kubectl` を使用して新しい名前空間を作成します。

```
$ namespace=my-aws-batch-namespace
```

```
$ cat - <<EOF | kubectl create -f -
{
  "apiVersion": "v1",
  "kind": "Namespace",
  "metadata": {
    "name": "${namespace}",
    "labels": {
      "name": "${namespace}"
    }
  }
}
EOF
```

出力:

```
namespace/my-aws-batch-namespace created
```

2. ロールベースアクセス制御 (RBAC) を有効にします。

kubectl を使用して、クラスターの Kubernetes ロールを作成すると、 AWS Batch はノードとポッドを監視したり、ロールをバインドしたりできるようになります。これは、Amazon EKS クラスターごとに 1 回行う必要があります。

> **Note**
>
> RBAC 認証の詳細については、Kubernetes ドキュメントのRBAC 認証の使用を参照してください。

```
$ cat - <<EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: aws-batch-cluster-role
rules:
  - apiGroups: [""]
    resources: ["namespaces"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "list", "watch"]
```

```
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["configmaps"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["apps"]
    resources: ["daemonsets", "deployments", "statefulsets", "replicasets"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["clusterroles", "clusterrolebindings"]
    verbs: ["get", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: aws-batch-cluster-role-binding
subjects:
- kind: User
  name: aws-batch
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: aws-batch-cluster-role
  apiGroup: rbac.authorization.k8s.io
EOF
```

出力:

```
clusterrole.rbac.authorization.k8s.io/aws-batch-cluster-role created
clusterrolebinding.rbac.authorization.k8s.io/aws-batch-cluster-role-binding created
```

の名前空間スコープのKubernetesロールを作成して AWS Batch 、ポッドを管理およびライフサイクルし、バインドします。これは固有の名前空間ごとに 1 回行う必要があります。

```
$ namespace=my-aws-batch-namespace
$ cat - <<EOF | kubectl apply -f - --namespace "${namespace}"
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: aws-batch-compute-environment-role
  namespace: ${namespace}
```

```
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["create", "get", "list", "watch", "delete", "patch"]
  - apiGroups: [""]
    resources: ["serviceaccounts"]
    verbs: ["get", "list"]
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["roles", "rolebindings"]
    verbs: ["get", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: aws-batch-compute-environment-role-binding
  namespace: ${namespace}
subjects:
- kind: User
  name: aws-batch
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: aws-batch-compute-environment-role
  apiGroup: rbac.authorization.k8s.io
EOF
```

出力:

```
role.rbac.authorization.k8s.io/aws-batch-compute-environment-role created
rolebinding.rbac.authorization.k8s.io/aws-batch-compute-environment-role-binding
 created
```

Kubernetes aws-auth 設定マップを更新して、前述の RBAC アクセス許可を AWS Batch サービスにリンクされたロールにマッピングします。

```
$ eksctl create iamidentitymapping \
    --cluster my-cluster-name \
    --arn "arn:aws:iam::<your-account>:role/AWSServiceRoleForBatch" \
    --username aws-batch
```

出力:

```
2022-10-25 20:19:57 [#]  adding identity "arn:aws:iam::<your-account>:role/
AWSServiceRoleForBatch" to auth ConfigMap
```

> **Note**
>
> パス aws-service-role/batch.amazonaws.com/が、サービスにリンクされたロールの ARN から削除されました。これは aws-auth 設定マップに問題があるためです。詳細については、「」の「パスが ARN に含まれている場合、パスを持つロールは機能しませんaws-authconfigmap」を参照してください。

ステップ 2: Amazon EKS コンピューティング環境を作成する

AWS Batch コンピューティング環境は、バッチワークロードのニーズに合わせてコンピューティングリソースパラメータを定義します。マネージド型のコンピューティング環境 AWS Batch では、Amazon EKS クラスター内のコンピューティングリソース (Kubernetes ノード) の容量とインスタンスタイプを管理できます。これは、コンピューティング環境の作成時に定義するコンピューティングリソースの仕様に基づいています。EC2 オンデマンドインスタンスまたは EC2 スポットインスタンスを選択できます。

AWSServiceRoleForBatch サービスにリンクされたロールが Amazon EKS クラスターにアクセスできるようになったので、 AWS Batch リソースを作成できます。まず、Amazon EKS クラスターを指すコンピューティング環境を作成します。

```
$ cat <<EOF > ./batch-eks-compute-environment.json
{
  "computeEnvironmentName": "My-Eks-CE1",
  "type": "MANAGED",
  "state": "ENABLED",
  "eksConfiguration": {
    "eksClusterArn": "arn:aws:eks:<region>:123456789012:cluster/<cluster-name>",
    "kubernetesNamespace": "my-aws-batch-namespace"
  },
  "computeResources": {
    "type": "EC2",
    "allocationStrategy": "BEST_FIT_PROGRESSIVE",
    "minvCpus": 0,
    "maxvCpus": 128,
    "instanceTypes": [
```

```
        "m5"
    ],
    "subnets": [
        "<eks-cluster-subnets-with-access-to-the-image-for-image-pull>"
    ],
    "securityGroupIds": [
        "<eks-cluster-sg>"
    ],
    "instanceRole": "<eks-instance-profile>"
  }
}
EOF
$ aws batch create-compute-environment --cli-input-json file://./batch-eks-compute-environment.json
```

メモ

- `serviceRole` パラメータを指定しないでください。サービス AWS Batch にリンクされたロールが使用されます。Amazon EKS AWS Batch では、 AWS Batch サービスにリンクされたロールのみがサポートされます。
- Amazon EKS コンピューティング環境で はSPOT_CAPACITY_OPTIMIZED、、BEST_FIT_PROGRESSIVE、および SPOT_PRICE_CAPACITY_OPTIMIZEDの割り当て戦略のみがサポートされています。

  > **Note**
  >
  > ほとんどの場合、SPOT_CAPACITY_OPTIMIZED ではなく SPOT_PRICE_CAPACITY_OPTIMIZED を使用することをおすすめします。

- `instanceRole` については、Amazon EKS ユーザーガイドのAmazon EKS ノード IAM ロールの作成とクラスターへの IAM プリンシパルアクセスを有効にするを参照してください。ポッドネットワークを使用している場合は、Amazon EKS ユーザーガイドのサービスアカウントに IAM ロールを使用するための Kubernetes Amazon VPC CNI プラグインの設定を参照してください。
- `subnets` パラメータのサブネットを動作させる方法の 1 つとして、Amazon EKS クラスターの作成時に `eksctl` が作成した Amazon EKS マネージドノードグループのパブリックサブネットを使用することができます。それ以外の場合は、イメージをプルできるネットワークパスを持つサブネットを使用します。
- `securityGroupIds` パラメータには、Amazon EKS クラスターと同じセキュリティグループを使用できます。このコマンドは、クラスターのセキュリティグループ ID を取得します。

```
$ eks describe-cluster \
    --name <cluster-name> \
    --query cluster.resourcesVpcConfig.clusterSecurityGroupId
```

- Amazon EKS コンピューティング環境のメンテナンスは責任共有です。詳細については、「Amazon EKS のセキュリティ」を参照してください。

> **Important**
>
> 処理を進める前に、コンピューティング環境が正常であることを確認することが重要です。DescribeComputeEnvironments API オペレーションを使用してこれを行うことができます。
>
> ```
> $ aws batch describe-compute-environments --compute-environments My-Eks-CE1
> ```
>
> status パラメータが INVALID ではないことを確認してください。そうであれば、statusReason パラメータを調べて原因を調べてください。詳細については、トラブルシューティング AWS Batchを参照してください。

ステップ 3: ジョブキューを作成してコンピューティング環境をアタッチする

```
$ aws batch describe-compute-environments --compute-environments My-Eks-CE1
```

この新しいジョブキューに送信されたジョブは、コンピューティング環境に関連付けられた Amazon EKS クラスターに参加した AWS Batch マネージドノードでポッドとして実行されます。

```
$ cat <<EOF > ./batch-eks-job-queue.json
 {
    "jobQueueName": "My-Eks-JQ1",
    "priority": 10,
    "computeEnvironmentOrder": [
      {
        "order": 1,
        "computeEnvironment": "My-Eks-CE1"
      }
    ]
  }
```

```
EOF
$ aws batch create-job-queue --cli-input-json file://./batch-eks-job-queue.json
```

ステップ 4: ジョブ定義を作成する

ジョブ定義の イメージフィールドで、パブリック ECR リポジトリ内のイメージへのリンクを提供する代わりに、プライベート ECR リポジトリに保存されているイメージへのリンクを指定します。次のサンプルジョブ定義を参照してください。

```
$ cat <<EOF > ./batch-eks-job-definition.json
{
  "jobDefinitionName": "MyJobOnEks_Sleep",
  "type": "container",
  "eksProperties": {
    "podProperties": {
      "hostNetwork": true,
      "containers": [
        {
          "image": "account-id.dkr.ecr.region.amazonaws.com/amazonlinux:2",
          "command": [
            "sleep",
            "60"
          ],
          "resources": {
            "limits": {
              "cpu": "1",
              "memory": "1024Mi"
            }
          }
        }
      ],
      "metadata": {
        "labels": {
          "environment": "test"
        }
      }
    }
  }
}
EOF
$ aws batch register-job-definition --cli-input-json file://./batch-eks-job-
definition.json
```

kubectl コマンドを実行するには、Amazon EKS クラスターへのプライベートアクセスが必要です。つまり、クラスター API サーバーへのすべてのトラフィックは、クラスターの VPC または接続されたネットワーク内から送信する必要があります。

ステップ 5: ジョブを送信する

```
$ aws batch submit-job - -job-queue My-Eks-JQ1 \
    - -job-definition MyJobOnEks_Sleep - -job-name My-Eks-Job1
$ aws batch describe-jobs - -job <jobId-from-submit-response>
```

メモ

- 単一のコンテナジョブのみがサポートされます。
- cpu および memory パラメータに関連するすべての考慮事項をよく理解しておいてください。詳細については、「Amazon EKSAWS Batchのメモリとvcpuに関する考慮事項」を参照してください。
- Amazon EKS リソースでのジョブの実行の詳細については、「」を参照してくださいAmazon EKS ジョブ。

(オプション) オーバーライドを含むジョブを送信する

このジョブは、コンテナに渡されたコマンドをオーバーライドします。

```
$ cat <<EOF > ./submit-job-override.json
{
  "jobName": "EksWithOverrides",
  "jobQueue": "My-Eks-JQ1",
  "jobDefinition": "MyJobOnEks_Sleep",
  "eksPropertiesOverride": {
    "podProperties": {
      "containers": [
        {
          "command": [
            "/bin/sh"
          ],
          "args": [
            "-c",
            "echo hello world"
          ]
        }
```

```
      ]
    }
  }
}
EOF
$ aws batch submit-job - -cli-input-json file://./submit-job-override.json
```

メモ

- AWS Batch は、ジョブの完了後にポッドを積極的にクリーンアップして、 への負荷を軽減しますKubernetes。ジョブの詳細を確認するには、ログ記録を設定する必要があります。詳細については、CloudWatch Logs を使用して Amazon EKS ジョブにおける AWS Batch をモニタリングするを参照してください。
- 操作の詳細を把握しやすくするには、Amazon EKS コントロールプレーンのログ記録を有効にします。詳細については、Amazon EKS ユーザーガイドのAmazon EKS クラスターコントロールプレーンのログ記録を参照してください。
- Daemonsets と kubelets オーバーヘッドは、使用可能な vCPU とメモリのリソース、特にスケーリングとジョブの配置に影響します。詳細については、「Amazon EKSAWS Batchのメモリと vCPUに関する考慮事項」を参照してください。

トラブルシューティング

によって起動されたノードが、イメージを保存する Amazon ECR リポジトリ (または他のリポジトリ) にアクセス AWS Batch できない場合、ジョブは STARTING 状態のままになる可能性があります。これは、ポッドがイメージをダウンロードして AWS Batch ジョブを実行できないためです。によって起動されたポッド名をクリックすると AWS Batch 、エラーメッセージが表示され、問題を確認できます。エラーメッセージは次のようになります。

```
Failed to pull image "public.ecr.aws/amazonlinux/amazonlinux:2": rpc error: code =
Unknown desc = failed to pull and unpack image
"public.ecr.aws/amazonlinux/amazonlinux:2": failed to resolve reference
"public.ecr.aws/amazonlinux/amazonlinux:2": failed to do request: Head
"https://public.ecr.aws/v2/amazonlinux/amazonlinux/manifests/2": dial tcp: i/o timeout
```

その他の一般的なトラブルシューティングシナリオについては、「 のトラブルシューティング AWS Batch」を参照してください。pod-status に基づくトラブルシューティングについては、「Amazon EKS のポッドステータスのトラブルシューティング方法を教えてください」を参照してください。

ジョブ

ジョブは、 によって開始される作業の単位です AWS Batch。ジョブは、ECSクラスタ内のAmazon ECSコンテナインスタンス上で実行されるコンテナ化されたアプリケーションとして呼び出すことができます。

コンテナ化されたジョブは、コンテナイメージ、コマンド、およびパラメータを参照できます。詳細については、のジョブ定義パラメータ ContainerPropertiesを参照してください。

多数の独立したシンプルなジョブを送信できます。

トピック

ジョブの送信

ジョブ定義を登録したら、ジョブとして ジョブ AWS Batch キューに送信できます。ジョブ定義で指定されたパラメーターの多くは、実行時にオーバーライドできます。

ジョブを送信するには

1. https://console.aws.amazon.com/batch/ で AWS Batch コンソールを開きます。

2. ナビゲーションバーから、 AWS リージョン 使用する を選択します。
3. ナビゲーションペインで ジョブを選択します。
4. ジョブの送信を選択します。
5. 名前 に、一意のジョブ定義名を入力します。名前の最大長は 128 文字です。大文字、小文字、数字、ハイフン (-)、アンダースコア (_) を含めることができます。
6. ジョブ定義で、作成済みのジョブ定義を選択します。詳細については、シングルノードのジョブ定義を作成する を参照してください。
7. ジョブキューで、既存のジョブキューを選択します。詳細については、ジョブキューの作成を参照してください。
8. ジョブ依存関係を追加 で を選択します。
   - ジョブ IDには、すべての依存関係のジョブ ID を入力します。次に ジョブの依存関係を追加 を選択します。ジョブは最大 20 個の依存関係を持つことができます。詳細については、ジョブの依存関係を参照してください。
9. (配列ジョブのみ) Array size] (配列サイズ) で、配列サイズを 2 から 10,000 の間で指定します。
10. (オプション)タグを展開し、タグを追加 を選択してリソースにタグを追加します。キーとオプショナル値を入力し、新しいタグを追加を選択します。
11. 次のページ を選択します。
12. ジョブオーバーライドセクションで:
    a. (オプショナル) スケジュールの優先度 には、0 から 100 までのスケジューリング優先度の値を入力します。値が大きいほど優先度が高くなります。
    b. (オプショナル) Job 試行回数 には、 AWS Batch ジョブをステータスに移行しようとするRUNNABLE 最大回数を入力します。1から10までの数字を入力できます。詳細については、ジョブの再試行の自動化を参照してください。
    c. (オプション) 実行タイムアウト で、タイムアウト値 (秒単位) を入力します。実行タイムアウトは、未完了のジョブが終了するまでの時間です。試行がタイムアウト時間を超えた場合、試行は中止され、FAILEDのステータスに移行します。詳細については、ジョブのタイムアウトを参照してください。最小値は 60 秒です。

> **Important**
>
> Fargateリソースで実行されるジョブが14日以上実行されることを当てにしないでください。14日後、ファーゲートのリソースは使用できなくなり、仕事が打ち切られる可能性があります。

d. (オプション) タグを伝播 をオンにして、タグをジョブとジョブ定義から Amazon ECS タスクに伝達することができます。

13. Additional configuration] (追加設定) を展開します。
14. (オプション) 再試行戦略の条件 では、終了時に評価を追加 を選択します。少なくとも 1 つのパラメータ値を入力し、アクション を選択します。条件セットごとに、アクション を 再試行 または 終了 に設定する必要があります。これらのアクションは、以下のことを意味します。

    - 再試行 — 指定したジョブ試行回数に達するまで AWS Batch 再試行します。
    - 終了 — ジョブの再試行を AWS Batch 停止します。

> **Important**
>
> 終了時に評価を追加 を選択した場合は、少なくとも 1 つのパラメータを設定してアクション を選択するか、終了時に評価を削除 を選択します。

15. パラメーター で パラメーターの追加 を選択し、パラメーター置換プレースホルダーを追加します。キーを入力し、オプションで 値を入力します。
16. コンテナオーバーライドセクションで:

    a. Command] (コマンド) で、コンテナに渡すコマンドを指定します。単純なコマンドの場合は、コマンドプロンプトと同じようにコマンドを入力します。より複雑なコマンド (特殊文字など) には、JSON 構文を使用してください。

> **Note**
>
> このパラメータには空の文字列を含めることはできません。

    b. vCPU で、コンテナ用に予約する vCPU の数を指定します。このパラメータは、Docker Remote API のコンテナの作成セクションの `CpuShares` にマップされ、また`--cpu-`

shares オプションは docker run にマップされます。各 vCPU は 1,024 個の CPU 配分に相当します。少なくとも 1 つの vCPU を指定する必要があります。

c. メモリ で、コンテナで使用できるメモリ制限を入力します。コンテナは、ここで指定したメモリを超えようとすると、停止されます。このパラメータは、Docker Remote API のコンテナの作成セクションの Memory にマップされ、また--memory オプションは docker run にマップされます。ジョブに対して少なくとも 4 MiB のメモリを指定する必要があります。

> **Note**
>
> 特定のインスタンスタイプのジョブにメモリの優先順位を付けることで、リソース使用率を最大化できます。詳細については、コンピューティングリソースメモリ管理を参照してください。

d. (オプショナル) GPU の数 には、コンテナ用に予約する GPU の数を選択します。

e. (オプション) 環境変数 で 環境変数を追加 を選択し、環境変数を名前と値のペアとして追加します。これらの変数は、コンテナに渡されます。

f. 次のページ を選択します。

g. ジョブレビューについては、設定手順を確認してください。変更する必要がある場合は、Edit] (編集) を選択します。完了したら、ジョブ定義の作成 を選択します。

ジョブの状態

ジョブキューに AWS Batch ジョブを送信すると、ジョブは SUBMITTED状態になります。その後、以下の状態を経由して完了 (コード 0 で終了) または失敗 (0 以外のコードで終了) します。 AWS Batch ジョブの各状態は以下のとおりです。

SUBMITTED

キューに投入され、まだスケジューラによって評価されていないジョブです。スケジューラは、ジョブを評価し、他のジョブの正常な完了に依存するかどうか (未処理の依存関係があるかどうか) を判断します。依存関係がある場合、ジョブの状態は PENDING に移行します。依存関係がない場合、ジョブの状態は RUNNABLE に移行します。

PENDING

ジョブはキュー内にあり、別のジョブやリソースへの依存関係があるため、まだ実行できません。依存関係が満たされると、ジョブの状態は RUNNABLE に移行します。

RUNNABLE

ジョブはキュー内にあり、未処理の依存関係はありません。したがって、ホストにスケジューリングされる準備ができています。この状態のジョブは、ジョブのキューにマッピングされたコンピューティング環境のいずれかで十分なリソースが使用可能になり次第、開始されます。ただし、十分なリソースが使用不可の場合、この状態が無限に続くことがあります。

> **Note**
>
> ジョブが STARTING に進行しない場合は、トラブルーシューティングセクションの「RUNNABLE 状態でジョブが止まる」を参照してください。

STARTING

これらのジョブはホストにスケジュールされており、関連するコンテナ初期化操作が進行中です。コンテナイメージがプルされてコンテナが稼働状態になると、ジョブの状態は RUNNING に移行します。

イメージのプル期間、Amazon EKS initContainer の完了期間、Amazon ECS containerDependency 解決期間は、STARTING 状態で発生します。ジョブのイメージをプルするのにかかる時間は、ジョブが STARTING 状態になるまでの時間と同等です。

例えば、ジョブのイメージをプルするのに 3 分かかる場合、ジョブは 3 分間 STARTING 状態になります。initContainers の完了に合計 10 分かかる場合、Amazon EKS ジョブは 10 分間開始されます。Amazon ECS ジョブに Amazon ECS containerDependencies セットがある場合、すべてのコンテナ依存関係 (ランタイム) が解決されるまで、ジョブは STARTING になります。STARTING はタイムアウトに含まれず、期間は RUNNING から始まります。詳細については、「ジョブの状態」を参照してください。

RUNNING

ジョブは、コンピューティング環境内の Amazon ECS コンテナインスタンスでコンテナジョブとして実行中です。ジョブのコンテナが終了すると、プロセス終了コードでジョブの正否が判定されます。終了コードの 0 は成功を示し、ゼロ以外の終了コードは失敗を示します。試行が失敗したジョブの再試行戦略に設定されている再試行回数が残っている場合は、ジョブの状態が RUNNABLE に移行します。詳細については、「ジョブの再試行の自動化」を参照してください。

> **Note**
>
> RUNNING ジョブのログは CloudWatch ログで確認できます。ロググループは /aws/batch/job、ログストリーム名形式は*first200CharsOfJobDefinitionName*/default/*ecs_task_id*です。この形式は将来変わる可能性があります。
> ジョブが RUNNINGステータスになったら、 DescribeJobs API オペレーションを使用してログストリーム名をプログラムで取得できます。詳細については、「Amazon Logs ユーザーガイド」の CloudWatch 「ログに送信されたログデータの表示」を参照してください。 CloudWatch デフォルトでは、これらのログには有効期限はありません。ただし、保存期間を変更することは可能です。詳細については、「Amazon Logs ユーザーガイド」の CloudWatch 「ログのログデータ保持の変更」を参照してください。 CloudWatch

SUCCEEDED

ジョブは、終了コード 0 で正常に完了しました。ジョブのSUCCEEDEDジョブ状態は、 AWS Batch 少なくとも 7 日間 に保持されます。

> **Note**
>
> SUCCEEDED ジョブのログは CloudWatch ログで確認できます。ロググループは /aws/batch/job、ログストリーム名形式は*first200CharsOfJobDefinitionName*/default/*ecs_task_id*です。この形式は将来に変更される可能性があります。
> ジョブが RUNNINGステータスに達したら、 DescribeJobs API オペレーションを使用してログストリーム名をプログラムで取得できます。詳細については、「Amazon Logs ユーザーガイド」の CloudWatch 「ログに送信されたログデータの表示」を参照してください。 CloudWatch デフォルトでは、これらのログには有効期限はありません。ただし、保存期間を変更することは可能です。詳細については、「Amazon Logs ユーザーガイド」の CloudWatch 「ログのログデータ保持の変更」を参照してください。 CloudWatch

FAILED

ジョブのすべての使用可能な試行が失敗しました。FAILED のジョブの状態は、少なくとも7日間 AWS Batch に保存されます。

> **Note**
>
> FAILED ジョブのログは CloudWatch ログで確認できます。ロググループは /aws/batch/job、ログストリーム名形式は*first200CharsOfJobDefinitionName*/default/*ecs_task_id*です。この形式は将来に変更される可能性があります。ジョブが RUNNINGステータスに達したら、 DescribeJobs API オペレーションを使用してログストリームをプログラムで取得できます。詳細については、「Amazon Logs ユーザーガイド」の CloudWatch 「ログに送信されたログデータの表示」を参照してください。 CloudWatch デフォルトでは、これらのログには有効期限はありません。ただし、保存期間を変更することは可能です。詳細については、「Amazon Logs ユーザーガイド」の CloudWatch 「ログのログデータ保持の変更」を参照してください。 CloudWatch

AWS Batch ジョブ環境変数

AWS Batch は、コンテナジョブに特定の環境変数を設定します。これらの環境変数は、ジョブ内のコンテナのイントロスペクションを行います。これらの変数の値は、アプリケーションのロジックで使用できます。が AWS Batch 設定するすべての変数はAWS_BATCH_、 プレフィックスで始まります。これは、保護された環境変数プレフィックスです。このプレフィックスは、ジョブ定義やオーバーライド内の独自の変数には使用できません。

ジョブコンテナでは、次の環境変数を使用できます。

AWS_BATCH_CE_NAME

この変数は、ジョブが配置されるコンピューティング環境の名前に設定されます。

AWS_BATCH_JOB_ARRAY_INDEX

この変数は、子配列ジョブでのみ設定されます。配列ジョブのインデックスは 0 から始まり、各子ジョブは一意のインデックス番号を受け取ります。たとえば、10 の子を持つ配列ジョブのインデックス値は 0 ～ 9 です。このインデックス値を使用して、配列ジョブの子がどのように区別されるかを制御できます。詳細については、チュートリアル: 配列ジョブインデックスを使用したジョブの差別化の制御を参照してください。

AWS_BATCH_JOB_ARRAY_SIZE

この変数は、親配列ジョブのサイズに設定されます。親配列ジョブのサイズは、この変数の子配列ジョブに渡されます。

AWS_BATCH_JOB_ATTEMPT

この変数は、ジョブ試行番号に設定されます。最初の試行番号は 1 となります。詳細については、「ジョブの再試行の自動化」を参照してください。

AWS_BATCH_JOB_ID

この変数は AWS Batch ジョブ ID に設定されます。

AWS_BATCH_JOB_KUBERNETES_NODE_UID

この変数は、ポッドが実行される Kubernetes クラスターにあるノードオブジェクトの Kubernetes UID として設定されます。この変数は、Amazon EKS リソースで実行されるジョブにのみ設定されます。 の詳細については、Kubernetes ドキュメンテーションのを参照してください。

AWS_BATCH_JOB_MAIN_NODE_INDEX

この変数は複数ノード並列ジョブの子ノードでのみ設定されます。この変数は、ジョブの主要なノードのインデックス番号に設定されます。アプリケーションコードは、AWS_BATCH_JOB_MAIN_NODE_INDEX と AWS_BATCH_JOB_NODE_INDEX を単一ノードで比較して、これが主要なノードであるかを確認できます。

AWS_BATCH_JOB_MAIN_NODE_PRIVATE_IPV4_ADDRESS

この変数は複数ノード並列ジョブの子ノードでのみ設定されます。この変数はメインノードには存在しないが、ジョブのメインノードのプライベートIPv4アドレスに設定されます。子ノードのアプリケーションコードは、このアドレスを使用して主要なノードと通信できます。

AWS_BATCH_JOB_NODE_INDEX

この変数は複数ノード並列ジョブの子ノードでのみ設定されます。この変数は、ノードのノードインデックス番号に設定されます。ノードインデックスは 0 で始まり、各ノードは一意のインデックス番号を受け取ります。たとえば、10 の子を持つマルチノードの並列ジョブのインデックス値は 0 ～ 9 です。

AWS_BATCH_JOB_NUM_NODES

この変数は複数ノード並列ジョブの子ノードでのみ設定されます。この変数は、マルチノードの並列ジョブにリクエストしたノードの数に設定されます。

AWS_BATCH_JQ_NAME

この変数は、ジョブが送信されたジョブキューの名前に設定されます。

ジョブの再試行の自動化

ジョブおよびジョブ定義に再試行戦略を適用し、失敗したジョブを自動的に再試行できます。考えられる故障のシナリオは以下の通りです：

- コンテナジョブのゼロ以外の終了コード
- Amazon EC2 インスタンスの障害または終了
- 内部 AWS サービスエラーまたは停止

ジョブがジョブキューに送信されて RUNNING 状態になると、1 回の試行とみなされます。デフォルトでは、各ジョブは 1 回の試行で SUCCEEDED または FAILED のいずれかの状態に移行します。ただし、ジョブ定義とジョブ送信の両方のワークフローで、1〜10 回の再試行戦略を指定できます。評価OnExitが指定されている場合、最大 5 つの再試行戦略を含めることができます。evaluateOnExit が指定されていても、どの再試行戦略も一致しない場合、ジョブは再試行されます。終了と一致しないジョブについては、何らかの理由で終了する最後のエントリを追加します。たとえば、evaluateOnExit このオブジェクトには、アクションがの 2 RETRY つのエントリと、EXIT アクションがの最後のエントリがあります。

```
"evaluateOnExit": [
    {
        "action": "RETRY",
        "onReason": "AGENT"
    },
    {
        "action": "RETRY",
        "onStatusReason": "Task failed to start"
    },
    {
        "action": "EXIT",
        "onReason": "*"
    }
]
```

ランタイムに、AWS_BATCH_JOB_ATTEMPT 環境変数がコンテナの対応するジョブ試行番号に設定されます。最初の試行は 1 番となり、後続の試行は昇順の番号 (2、3、4 など) になります。

たとえば、何らかの理由でジョブの試行が失敗し、再試行設定で指定されている試行回数がその AWS_BATCH_JOB_ATTEMPT 回数よりも多いとします。その後、ジョブは元の RUNNABLE 状態に戻ります。詳細については、ジョブの状態を参照してください。

> **Note**
>
> ジョブをキャンセルまたは終了した場合、ジョブは再試行されません。無効なジョブ定義のためにジョブが失敗した場合も、ジョブは再試行されません。

詳細については、再試行戦略、シングルノードのジョブ定義を作成する 、ジョブの送信 および停止したタスクのエラーコード を参照してください。

ジョブの依存関係

AWS Batch ジョブを送信するときに、ジョブが依存するジョブ IDs を指定できます。この場合、 AWS Batch スケジューラにより、ジョブは指定された依存関係が正常に完了した後にのみ実行されます。成功すると、依存するジョブが PENDING から RUNNABLE に移行し、その後 STARTING、RUNNING と移行します。いずれかのジョブ依存関係が失敗すると、依存するジョブは自動的に PENDING から FAILED に移行します。

たとえば、ジョブ A は実行前に成功する必要がある他のジョブとの依存関係を最大 20 個記述できます。その後、ジョブ A および他の 19 個のジョブに依存する追加ジョブを送信できます。

配列ジョブの場合、ジョブ ID を指定せずに SEQUENTIAL タイプの依存関係を指定できます。こうすることで各子配列ジョブがインデックス 0 から開始して連続的に完了します。また、ジョブ ID を使用して N_TO_N タイプの依存関係を指定することもできます。この場合、このジョブの各インデックスの子は各依存関係の対応するインデックスの子が完了するまで待機してから開始されます。詳細については、「配列ジョブ」を参照してください。

依存関係を持つ AWS Batch ジョブを送信するには、「」を参照してくださいジョブの送信。

ジョブのタイムアウト

この期間を超えてジョブが実行されると AWS Batch でジョブが終了するように、ジョブのタイムアウト期間を設定できます。たとえば、15 分で完了することがわかっているジョブがあるとします。アプリケーションがループ状態に止まり、永続的に実行される場合に、止まったジョブを終了するためにタイムアウトを 30 分に設定できます。

> **Important**
>
> デフォルトでは、ジョブのタイムアウト AWS Batch はありません。ジョブタイムアウトを定義しない場合、ジョブはコンテナが終了するまで実行されます。

`attemptDurationSeconds` パラメータを指定します。ジョブ定義の際、またはジョブ送信時に行い、60 秒以上にする必要があります。ジョブ試行の`startedAt`タイムスタンプの後にこの秒数が経過すると、 はジョブを AWS Batch 終了します。コンピューティングリソースで、ジョブのコンテナは `SIGTERM` シグナルを受け取り、アプリケーションが適切にシャットダウンできるようにします。30 秒後にコンテナがまだ実行されている場合、`SIGKILL` シグナルが送信されてコンテナを強制的にシャットダウンします。

タイムアウトの終了はベストエフォートベースで処理されます。ジョブ試行がタイムアウトするタイミングでタイムアウトが終了するとは限りません (数秒長くかかることがあります)。アプリケーションで正確にタイムアウトを実行する必要がある場合は、アプリケーション内にこのロジックを実装します。多数のジョブを同時にタイムアウトする場合、タイムアウトの終了は先入れ先出しキューとして行われ、ジョブはバッチで終了します。

> **Note**
>
> AWS Batch ジョブの最大タイムアウト値はありません。

タイムアウト期間の超過で終了したジョブは再試行されません。ジョブ自体が原因でジョブ試行に失敗した場合、再試行が有効であれば再試行され、新しいジョブ試行のタイムアウトカウントダウンが開始します。

> **Important**
>
> Fargate リソースで実行されるジョブは、14 日を超えて実行することはできません。タイムアウト期間が 14 日を超えると、Fargate リソースが使用できなくなり、ジョブが終了します。

配列ジョブの場合、子ジョブは、親ジョブと同様にタイムアウト設定されています。

タイムアウト設定で AWS Batch ジョブを送信する方法については、「」を参照してくださいジョブの送信。

Amazon EKS ジョブ

ジョブは、 における作業の最小単位です AWS Batch。Amazon EKS の AWS Batch ジョブには、Kubernetesポッドへの one-to-one マッピングがあります。 AWS Batch ジョブ定義は、 AWS Batch ジョブのテンプレートです。 AWS Batch ジョブを送信するときは、ジョブ定義を参照し、ジョブキューをターゲットにして、ジョブの名前を指定します。Amazon EKS での AWS Batch ジョブのジョブ定義では、eksProperties パラメータは、Amazon EKS AWS Batch ジョブで がサポートするパラメータのセットを定義します。SubmitJob リクエストでは、eksPropertiesOverride パラメータはいくつかの一般的なパラメータの上書きを許可します。これにより、複数のジョブにジョブ定義のテンプレートを使用できます。ジョブが Amazon EKS クラスターにディスパッチされると、はジョブ AWS Batch を podspec () に変換しますKind： Pod。は、いくつかの追加 AWS Batch パラメータpodspecを使用して、ジョブが正しくスケーリングおよびスケジュールされるようにします。 AWS Batch はラベルとテイントを組み合わせて、ジョブが AWS Batch マネージドノードでのみ実行され、他のポッドがそれらのノードで実行されないようにします。

> **Important**
>
> - Amazon EKS ジョブ定義で hostNetworkパラメータが明示的に設定されていない場合、のポッドネットワークモードは AWS Batch デフォルトでホストモードになります。具体的には、hostNetwork=true と dnsPolicy=ClusterFirstWithHostNet という設定が適用されます。
> - AWS Batch は、ポッドがジョブを完了するとすぐにジョブポッドをクリーンアップします。ポッドアプリケーションログを表示するには、クラスターのロギングサービスを設定します。詳細については、CloudWatch Logs を使用して Amazon EKS ジョブにおける AWS Batch をモニタリングするを参照してください。

実行中のジョブをポッドとノードにマップします。

podProperties 実行中のジョブには podName、nodeName 現在のジョブ試行用のパラメータが設定されています。DescribeJobs API オペレーションを使用して、これらのパラメータを表示します。

以下は出力例です。

```
$ aws batch describe-jobs --job 2d044787-c663-4ce6-a6fe-f2baf7e51b04
{
 "jobs": [
```

```
  {
   "status": "RUNNING",
   "jobArn": "arn:aws:batch:us-east-1:123456789012:job/2d044787-c663-4ce6-a6fe-
f2baf7e51b04",
   "jobDefinition": "arn:aws:batch:us-east-1:123456789012:job-definition/
MyJobOnEks_SleepWithRequestsOnly:1",
   "jobQueue": "arn:aws:batch:us-east-1:123456789012:job-queue/My-Eks-JQ1",
   "jobId": "2d044787-c663-4ce6-a6fe-f2baf7e51b04",
   "eksProperties": {
    "podProperties": {
     "nodeName": "ip-192-168-55-175.ec2.internal",
     "containers": [
      {
       "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
       "resources": {
        "requests": {
         "cpu": "1",
         "memory": "1024Mi"
        }
       }
      }
     ],
     "podName": "aws-batch.b0aca953-ba8f-3791-83e2-ed13af39428c"
    }
   }
  }
 ]
}
```

再試行が有効になっているジョブの場合、完了したすべての試行nodeNameの podNameと は DescribeJobs API オペレーションのリストeksAttemptsパラメータにあります。現在実行中の podName、nodeName の試行の終了は podProperties オブジェクト内にあります。

実行中のポッドをそのジョブにマップし直す方法

ポッドには、それが属するコンピューティング環境uuidの jobIdと を示すラベルがあります。AWS Batch は、ジョブのランタイムがジョブ情報を参照できるように環境変数を挿入します。詳細については、「AWS Batch ジョブ環境変数」を参照してください。以下のコマンドを実行すれば、この情報を見ることができる。出力は次のとおりです。

```
$ kubectl describe pod aws-batch.14638eb9-d218-372d-ba5c-1c9ab9c7f2a1 -n my-aws-batch-
namespace
```

```
Name:         aws-batch.14638eb9-d218-372d-ba5c-1c9ab9c7f2a1
Namespace:    my-aws-batch-namespace
Priority:     0
Node:         ip-192-168-45-88.ec2.internal/192.168.45.88
Start Time:   Wed, 26 Oct 2022 00:30:48 +0000
Labels:       batch.amazonaws.com/compute-environment-uuid=5c19160b-
d450-31c9-8454-86cf5b30548f
              batch.amazonaws.com/job-id=f980f2cf-6309-4c77-a2b2-d83fbba0e9f0
              batch.amazonaws.com/node-uid=a4be5c1d-9881-4524-b967-587789094647
...
Status:       Running
IP:           192.168.45.88
IPs:
  IP:  192.168.45.88
Containers:
  default:
    Image:         public.ecr.aws/amazonlinux/amazonlinux:2
    ...
    Environment:
      AWS_BATCH_JOB_KUBERNETES_NODE_UID:  a4be5c1d-9881-4524-b967-587789094647
      AWS_BATCH_JOB_ID:                   f980f2cf-6309-4c77-a2b2-d83fbba0e9f0
      AWS_BATCH_JQ_NAME:                  My-Eks-JQ1
      AWS_BATCH_JOB_ATTEMPT:              1
      AWS_BATCH_CE_NAME:                  My-Eks-CE1

...
```

AWS Batch Amazon EKS ジョブがサポートする機能

これらは、Amazon EKS で実行されるKubernetesジョブにも共通する AWS Batch 特定の機能です。

- ジョブの依存関係
- 配列ジョブ
- ジョブのタイムアウト
- ジョブの再試行の自動化
- 公平配分のスケジューリング

KubernetesSecrets および ServiceAccounts

AWS Batch は、 KubernetesSecretsおよび の参照をサポートしていますServiceAccounts。サービスアカウントの Amazon EKS IAM ロールを使用するようにポッドを設定できます。詳細につ

いては、Amazon EKSユーザーガイド のKubernetes サービスアカウントを使用するポッドの設定 を参照してください。

関連ドキュメント

- Amazon EKSAWS Batchのメモリとvcpuに関する考慮事項
- Amazon EKS リソースで GPU ベースのジョブを作成するには
- RUNNABLE 状態でジョブが止まる

配列ジョブ

配列ジョブは、ジョブ定義、vCPU、メモリなどの共通パラメータを共有するジョブです。これは、関連しているが個別の基本ジョブのコレクションとして実行されます。複数のホストに分散されたり、同時に実行される場合もあります。配列ジョブは、モンテカルロシミュレーションジョブ、パラメータスイープジョブ、大規模なレンダリングジョブなど、大量の並列ジョブを実行するもっとも効率的な方法です。

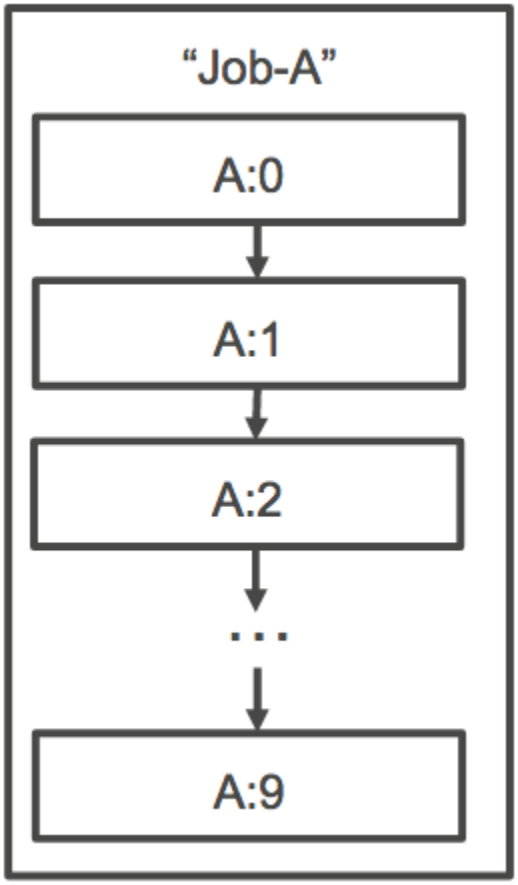
AWS Batch 配列ジョブは、通常のジョブと同様に送信されます。ただし、配列内で実行する子ジョブの数を定義する配列サイズ (2 ~ 10,000) を指定します。配列サイズが 1,000 以内のジョブを送信する場合は、単一ジョブが実行され 1,000 個の子ジョブが生成されます。配列ジョブは、すべての子ジョブを管理するリファレンスまたはポインタです。これにより、1つのクエリで大量のワークロードを送信することができます。`attemptDurationSeconds`パラメータで指定されたタイムアウトは、それぞれの子ジョブに適用されます。親アレイジョブには、タイムアウトはありません。

配列ジョブを送信すると、親配列ジョブは通常の AWS Batch ジョブ ID を取得します。子ジョブのベース ID は、それぞれ同じです。各子ジョブは同じベース ID を持ちますが、子ジョブの配列インデックスが親 ID の末尾に付加されます。たとえば、配列の最初の子ジョブは*example_job_ID*`:0`です。

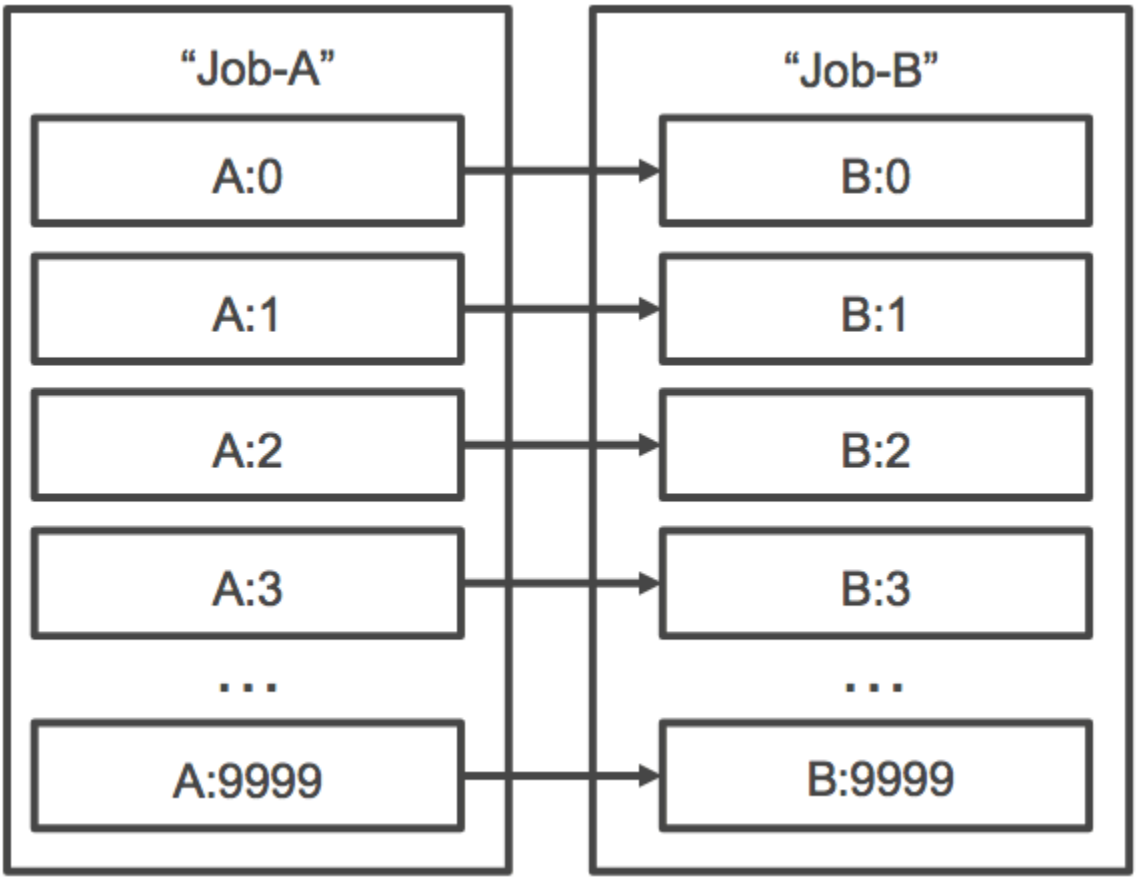
親アレイジョブは、 `SUBMITTED`、 `PENDING`、`FAILED`、または `SUCCEEDED` ステータスを入力できます。アレイの親ジョブは、`PENDING` 子ジョブが `RUNNABLE` に更新されるとに更新されます。これらの依存関係の詳細については、ジョブの依存関係を参照してください。

実行時、`AWS_BATCH_JOB_ARRAY_INDEX` 環境変数がコンテナの対応するジョブ配列インデックス番号に設定されます。最初の配列ジョブインデックスは `0` 番となり、後続の試行は昇順の番号 (1、2、3 など) になります。このインデックス値を使用して、配列ジョブの子がどのように区別されるかを制御できます。詳細については、チュートリアル: 配列ジョブインデックスを使用したジョブの差別化の制御を参照してください。

配列ジョブの依存関係では、依存関係のタイプを指定できます (SEQUENTIAL または N_TO_N など)。SEQUENTIAL タイプの依存関係 (ジョブ ID を指定しない) を指定できます。こうすることで各子配列ジョブがインデックス 0 から開始して連続的に完了します。たとえば、配列サイズが 100 の配列ジョブを送信する場合、依存関係を SEQUENTIAL タイプに指定すると、100 個の子ジョブが連続押して生成され、最初の子ジョブが成功してから次の子ジョブが開始されます。以下の図で示すジョブ A は、配列サイズが 10 である配列ジョブです。ジョブ A の子インデックスの各ジョブは、前の子ジョブに依存します。ジョブ A:1 はジョブ A:0 が完了するまで開始できません。



また、アレイジョブのジョブ ID を使用して N_TO_N タイプの依存関係を指定することもできます。この場合、このジョブの各インデックスの子は各依存関係の対応するインデックスの子が完了するまで待機してから開始されます。以下の図で示すジョブ A およびジョブ B は、配列サイズがそれぞれ 10,000 である 2 つの配列ジョブです。ジョブ B の子インデックスの各ジョブは、ジョブ A の対応するインデックスに依存します。ジョブ B:1 はジョブ A:1 が完了するまで開始できません。



親配列ジョブをキャンセルまたは終了した場合、子ジョブもすべてキャンセルまたは終了します。個々の子ジョブを、他の子ジョブに影響を与えずにキャンセルまたは終了できます (FAILED ステータスに移動させる)。ただし、子配列ジョブが失敗した場合 (それ自身の失敗または手動でキャンセル/終了)、親ジョブも失敗します。

配列ジョブのワークフロー例

AWS Batch お客様にとって一般的なワークフローは、前提条件のセットアップジョブを実行し、多数の入力タスクに対して一連のコマンドを実行し、結果を集約して概要データを Amazon S3、DynamoDB、Amazon Redshift、または Aurora に書き込むジョブで終了することです。

例:

- JobA: 配列ではない標準的なジョブです。Amazon S3 バケット BucketA 内のオブジェクトの高速リスト化およびメタデータ検証を実行します。JSON SubmitJob 構文は次のとおりです。

```
{
    "jobName": "JobA",
    "jobQueue": "ProdQueue",
```

```
    "jobDefinition": "JobA-list-and-validate:1"
}
```

- JobB : JobA に依存する 10,000 個のコピーを持つ配列ジョブです。CPU 負荷の高いコマンドを BucketA の各オブジェクトに対して実行し、結果を BucketB にアップロードします。JSON SubmitJob 構文は次のとおりです。

```
{
    "jobName": "JobB",
    "jobQueue": "ProdQueue",
    "jobDefinition": "JobB-CPU-Intensive-Processing:1",
    "containerOverrides": {
        "resourceRequirements": [
            {
                "type": "MEMORY",
                "value": "4096"
            },
            {
                "type": "VCPU",
                "value": "32"
            }
        ]
   }
    "arrayProperties": {
        "size": 10000
    },
    "dependsOn": [
        {
            "jobId": "JobA_job_ID"
  }
    ]
}
```

- JobC: JobB に N_TO_N 依存関係モデルで依存する 10,000 個のコピーを持つ別の配列ジョブです。メモリ負荷の高いコマンドを BucketB の各項目に対して実行し、メタデータを DynamoDB に書き込んで、結果の出力を BucketC にアップロードします。JSON SubmitJob 構文は次のとおりです。

```
{
    "jobName": "JobC",
    "jobQueue": "ProdQueue",
    "jobDefinition": "JobC-Memory-Intensive-Processing:1",
```

```
    "containerOverrides": {
        "resourceRequirements": [
            {
                "type": "MEMORY",
                "value": "32768"
            },
            {
                "type": "VCPU",
                "value": "1"
            }
        ]
   }
    "arrayProperties": {
        "size": 10000
    },
    "dependsOn": [
        {
            "jobId": "JobB_job_ID",
            "type": "N_TO_N"
        }
    ]
}
```

- JobD: 10 個の検証ステップを実行する配列ジョブです。検証ステップはそれぞれ DynamoDB にクエリする必要があり、上記の Amazon S3 バケットのいずれかとやり取りする可能性があります。の各ステップは同じコマンドを JobD 実行します。しかし、ジョブのコンテナ内の環境変数AWS_BATCH_JOB_ARRAY_INDEXの値によって動作は異な異なります。これらの検証ステップは順番に実行されます (たとえば、JobD:0 の次に JobD:1)。JSON SubmitJob 構文は次のとおりです。

```
{
    "jobName": "JobD",
    "jobQueue": "ProdQueue",
    "jobDefinition": "JobD-Sequential-Validation:1",
    "containerOverrides": {
        "resourceRequirements": [
            {
                "type": "MEMORY",
                "value": "32768"
            },
            {
                "type": "VCPU",
                "value": "1"
```

```
            }
        ]
    }
     "arrayProperties": {
         "size": 10
     },
     "dependsOn": [
         {
             "jobId": "JobC_job_ID"
         },
         {
             "type": "SEQUENTIAL"
         },

     ]
}
```

- JobE: 最終的な配列ではないジョブです。シンプルなクリーンアップオペレーションをいくつか実行し、パイプラインが完了したことおよび出力 URL へのリンクを記載したメッセージを含む Amazon SNS 通知を送信します。JSON SubmitJob 構文は次のとおりです。

```
{
    "jobName": "JobE",
    "jobQueue": "ProdQueue",
    "jobDefinition": "JobE-Cleanup-and-Notification:1",
    "parameters": {
        "SourceBucket": "s3://JobD-Output-Bucket",
        "Recipient": "pipeline-notifications@mycompany.com"
    },
    "dependsOn": [
        {
            "jobId": "JobD_job_ID"
        }
    ]
}
```

チュートリアル: 配列ジョブインデックスを使用したジョブの差別化の制御

このチュートリアルでは、AWS_BATCH_JOB_ARRAY_INDEX 環境変数を使用して子ジョブを区別する方法を説明します。各子ジョブは、この変数に割り当てられます。この例では、子ジョブのインデックス番号を使用して、ファイル内の特定の行を読み込みます。次に、その行番号に関連付けられ

たパラメータを、ジョブのコンテナ内のコマンドで置き換えます。その結果、同じ Docker イメージとコマンド引数を実行する複数の AWS Batch ジョブを持つことができます。ただし、配列ジョブインデックスが修飾子として使用されるため、結果が異なります。

このチュートリアルでは、虹のすべての色を持つテキストファイルを作成します。次に、インデックスをカラーファイルの行番号に使用できる値に変換する Dockerコンテナ 用のエントリポイントスクリプトを作成します。インデックスはゼロから始まりますが、行番号は 1 から始まります。カラーファイルとインデックスファイルをコンテナイメージにコピーし、イメージの `ENTRYPOINT` をエントリポイントスクリプトに設定する Dockerfile を作成します。Dockerfile とリソースは Amazon ECR にプッシュされる Docker イメージに組み込まれています。次に、新しいコンテナイメージを使用するジョブ定義を登録し、そのジョブ定義で AWS Batch 配列ジョブを送信して、結果を表示します。

前提条件

このチュートリアルには、次のような前提条件があります。

- AWS Batch コンピューティング環境。詳細については、「コンピューティング環境の作成」を参照してください。
- AWS Batch ジョブキューおよび関連するコンピューティング環境。詳細については、「ジョブキューの作成」を参照してください。
- ローカルシステムに AWS CLI インストールされている 。詳細については、AWS Command Line Interface ユーザーガイドのAWS Command Line Interfaceのインストールを参照してください。
- Docker は、ローカルシステムにインストールされます。詳細については、Docker ドキュメントのDocker CE についてを参照してください。

ステップ 1: コンテナイメージの構築

コマンドパラメータのジョブ定義を`AWS_BATCH_JOB_ARRAY_INDEX`で使用できます。 ただし、エントリポイントスクリプトで変数を使用するコンテナイメージを作成することをお勧めします。このセクションでは、そのようなコンテナイメージを作成する方法について説明します。

Docker コンテナイメージを構築するには

1. Docker イメージワークスペースとして使用する新しいディレクトリを作成し、そのディレクトリに移動します。
2. WorkSpace ディレクトリで、`colors.txt`という名前のファイルを作成し、以下を貼り付けます。

```
red
orange
yellow
green
blue
indigo
violet
```

3. WorkSpace ディレクトリで、print-color.shという名前のファイルを作成し、以下を貼り付けます。

> **Note**
>
> 配列インデックスは 0 から始まり、行番号は 1 から始まるため、LINE 変数は AWS_BATCH_JOB_ARRAY_INDEX +1 に設定されます。COLOR 変数は、行番号に関連付けられているcolors.txtの色に設定されます。

```
#!/bin/sh
LINE=$((AWS_BATCH_JOB_ARRAY_INDEX + 1))
COLOR=$(sed -n ${LINE}p /tmp/colors.txt)
echo My favorite color of the rainbow is $COLOR.
```

4. WorkSpace ディレクトリで、Dockerfileという名前のファイルを作成し、以下を貼り付けます。この Dockerfile は、以前のファイルをコンテナにコピーし、コンテナの起動時に実行するようにエントリポイントスクリプトを設定します。

```
FROM busybox
COPY print-color.sh /tmp/print-color.sh
COPY colors.txt /tmp/colors.txt
RUN chmod +x /tmp/print-color.sh
ENTRYPOINT /tmp/print-color.sh
```

5. Docker イメージをビルドします。

```
$ docker build -t print-color .
```

6. 次のスクリプトを使用してコンテナをテストします。このスクリプトは、変数をローカルで 0 に設定し、それをインクリメントして 7 つの子 が を行う配列ジョブが何をするのかをシミュレートします。

```
$ AWS_BATCH_JOB_ARRAY_INDEX=0
while [ $AWS_BATCH_JOB_ARRAY_INDEX -le 6 ]
do
    docker run -e AWS_BATCH_JOB_ARRAY_INDEX=$AWS_BATCH_JOB_ARRAY_INDEX print-color
    AWS_BATCH_JOB_ARRAY_INDEX=$((AWS_BATCH_JOB_ARRAY_INDEX + 1))
done
```

出力を次に示します。

```
My favorite color of the rainbow is red.
My favorite color of the rainbow is orange.
My favorite color of the rainbow is yellow.
My favorite color of the rainbow is green.
My favorite color of the rainbow is blue.
My favorite color of the rainbow is indigo.
My favorite color of the rainbow is violet.
```

ステップ 2: イメージを Amazon ECR にプッシュする

Docker コンテナを構築してテストしたので、それをイメージリポジトリにプッシュする必要があります。この例では Amazon ECR を使用していますが、 などの別のレジストリを使用できます DockerHub。

1. コンテナイメージを保存する Amazon ECR イメージを作成します。この例では のみを使用していますが AWS CLI、 を使用することもできます AWS Management Console。詳細については、Amazon Elastic Container Registry ユーザーガイドのリポジトリの作成を参照してください。

```
$ aws ecr create-repository --repository-name print-color
```

2. 前のステップから返されたAmazon ECR リポジトリ URI を使用して、`print-color` イメージにタグを付けます。

```
$ docker tag print-color aws_account_id.dkr.ecr.region.amazonaws.com/print-color
```

3. Amazon ECR レジストリにログインします。詳細については、Amazon Elastic Container Registry ユーザーガイドのレジストリの認証を参照してください。

```
$ aws ecr get-login-password \
    --region region | docker login \
    --username AWS \
    --password-stdin aws_account_id.dkr.ecr.region.amazonaws.com
```

4. Amazon ECR にイメージをプッシュします。

```
$ docker push aws_account_id.dkr.ecr.region.amazonaws.com/print-color
```

ステップ 3: ジョブ定義の作成と登録

Docker イメージがイメージレジストリにあるので、 AWS Batch ジョブ定義で指定できます。次に、配列ジョブを実行するために後でそれを使用できます。この例では、 AWS CLIを使用します。ただし、 AWS Management Consoleを使用することもできます。詳細については、「シングルノードのジョブ定義を作成する 」を参照してください。

ジョブ定義を作成するには

1. WorkSpace ディレクトリで、`print-color-job-def.json` という名前のファイルを作成し、以下を貼り付けます。イメージリポジトリの URI を自分のイメージの URI に置き換えます。

```
{
  "jobDefinitionName": "print-color",
  "type": "container",
  "containerProperties": {
    "image": "aws_account_id.dkr.ecr.region.amazonaws.com/print-color",
    "resourceRequirements": [
        {
            "type": "MEMORY",
            "value": "250"
        },
        {
            "type": "VCPU",
            "value": "1"
        }
    ]
```

```
  }
}
```

2. ジョブ定義を に登録します AWS Batch。

```
$ aws batch register-job-definition --cli-input-json file://print-color-job-
def.json
```

ステップ 4: AWS Batch 配列ジョブを送信する

ジョブ定義を登録したら、新しいコンテナイメージを使用する AWS Batch 配列ジョブを送信できます。

AWS Batch 配列ジョブを送信するには

1. WorkSpace ディレクトリで、`print-color-job.json`という名前のファイルを作成し、以下を貼り付けます。

> **Note**
>
> この例では、the section called “前提条件” セクションで説明したジョブキューを使用しています。

```
{
  "jobName": "print-color",
  "jobQueue": "existing-job-queue",
  "arrayProperties": {
    "size": 7
  },
  "jobDefinition": "print-color"
}
```

2. ジョブを AWS Batch ジョブキューに送信します。出力で返されるジョブ ID を書き留めておいてください。

```
$ aws batch submit-job --cli-input-json file://print-color-job.json
```

3. ジョブのステータスを記述し、ジョブが SUCCEEDED に移動するのを待ちます。

ステップ 5: 配列ジョブログの表示

ジョブが SUCCEEDEDステータスになったら、ジョブのコンテナから CloudWatch ログを表示できます。

Logs でジョブの CloudWatch ログを表示するには

1. https://console.aws.amazon.com/batch/ で AWS Batch コンソールを開きます。
2. 左のナビゲーションペインで ジョブ を選択します。
3. Job queue] (ジョブキュー) で、キューを選択します。
4. Status] (ステータス) セクションで、succeeded] (成功) を選択します。
5. 配列ジョブのすべての子ジョブを表示するには、前のセクションで返されたジョブ ID を選択します。
6. ジョブのコンテナからログを表示するには、子ジョブのいずれかを選択し、View logs] (ログの表示) を選択します。

7. 他の子ジョブのログを表示します。各ジョブは、虹の別の色を返します。

マルチノードの並列ジョブ

マルチノードの並列ジョブでは、複数の Amazon EC2 インスタンスにまたがる単一のジョブを実行できます。 AWS Batch のマルチノードの並列ジョブでは、Amazon EC2 リソースを直接起動、設定、管理する必要なく、ラージスケールで密結合された高パフォーマンスのコンピューティングアプリケーションと分散された GPU モデルトレーニングを実行できます。 AWS Batch マルチノードの並列ジョブは、IP ベースのノード間通信をサポートする任意のフレームワークと互換性があります。例としては、Apache MXNet 、 TensorFlow、Caffe2、Message Passing Interface (MPI) などがあります。

マルチノードの並行ジョブは、単一のジョブとして送信されます。ただし、ジョブ定義 (あるいは、ジョブ送信ノードの上書き) は、ジョブに作成するノードの数および作成するノードグループを指定します。各マルチノードの並列ジョブには主要なノードが含まれ、まずこれが起動されます。主要なノードが確立したら、子ノードが起動されて開始します。ジョブは、メインノードが終了した場合にのみ終了します。その後、すべての子ノードが停止します。詳細については、ノードグループを参照してください。

マルチノードの並列ジョブは、シングルテナントです。つまり、各 Amazon EC2 インスタンスごとに、単一のジョブコンテナのみが実行されます。

最終的なジョブステータス (SUCCEEDED あるいは FAILED) は、主要なノードの最終的なジョブステータスによって決定されます。マルチノードの並列ジョブのステータスを取得するには、ジョブの送信時に返されるジョブ ID を使用して、ジョブを記述できます。子ノードの詳細が必要な場合には、各子ノードごとに個別に記述する必要があります。ノードは *#N* 表記を使用して対処されます (0 から開始)。例えば、ジョブの 2 番目のノードの詳細にアクセスするには、 API オペレーションを使用して *aws_batch_job_id* #1 を記述します AWS Batch DescribeJobs。マルチノードの並列ジョブの `started`、`stoppedAt`、`statusReason`、`exit` 情報は、主要なノードから入力されます。

ジョブの再試行を指定した場合、メインノードに障害が発生すると、別の試行が行われます。子ノードに障害が発生しても、再試行回数は発生しません。マルチノードの並列ジョブの新しい試行ごとに、関連付けられた子ノードに対応する試行が更新されます。

でマルチノードの並列ジョブを実行するには AWS Batch、アプリケーションコードに分散通信に必要なフレームワークとライブラリが含まれている必要があります。

環境変数

実行時に、各ノードには、すべての AWS Batch ジョブが受け取る標準環境変数が設定されます。さらに、ノードは、マルチノードの並列ジョブに固有の次の環境変数で構成されます。

`AWS_BATCH_JOB_MAIN_NODE_INDEX`

この変数は、ジョブの主要なノードのインデックス番号に設定されます。アプリケーションコードは、`AWS_BATCH_JOB_MAIN_NODE_INDEX` と `AWS_BATCH_JOB_NODE_INDEX` を単一ノードで比較して、これが主要なノードであるかを確認できます。

`AWS_BATCH_JOB_MAIN_NODE_PRIVATE_IPV4_ADDRESS`

この変数は複数ノード並列ジョブの子ノードでのみ設定されます。この変数は、メインノードには存在しません。この変数はメインノードには存在しないが、ジョブのメインノードのプライ

ベートIPv4アドレスに設定されます。子ノードのアプリケーションコードは、このアドレスを使用して主要なノードと通信できます。

`AWS_BATCH_JOB_NODE_INDEX`

この変数は、ノードのノードインデックス番号に設定されます。ノードインデックスは 0 で始まり、各ノードは一意のインデックス番号を受け取ります。たとえば、10 の子を持つマルチノードの並列ジョブのインデックス値は 0 ～ 9 です。

`AWS_BATCH_JOB_NUM_NODES`

この変数は、マルチノードの並列ジョブにリクエストしたノードの数に設定されます。

ノードグループ

ノードグループとは、同じコンテナプロパティを共有するジョブノードの同一グループです。 では、ジョブごとに 5 つまでの個別のノードグループを指定できます。を使用して AWS Batch 、ジョブごとに最大 5 つの異なるノードグループを指定できます。

各グループでは、独自のコンテナイメージ、コマンド、環境変数などを持つことができます。たとえば、`c5.xlarge` メインノードに 1 つのインスタンス、5 `c5.xlarge` つのインスタンスの子ノードに必要なジョブを送信できます。これらのノードグループはそれぞれ、ジョブごとに異なるコンテナイメージやコマンドを実行するように指定できます。

あるいは、ジョブ内のすべてのノードで 1つのノードグループを使用することもできます。さらに、アプリケーションコードではメインノードや子ノードなどのノードロールを区別できます。そのためには、`AWS_BATCH_JOB_MAIN_NODE_INDEX` 環境変数を `AWS_BATCH_JOB_NODE_INDEX` の独自の値と比較します。単一のジョブでは最大で 1000 までのノードを使用できます。これは、Amazon ECS クラスターのインスタンスのデフォルト制限です。この制限はリクエストに応じて増やすことができます。この制限の引き上げをリクエストすることができます。

> **Note**
>
> 現在のところ、マルチノードの並列ジョブのすべてのノードグループでは、同じインスタンスタイプを使用する必要があります。

ジョブのライフサイクル

マルチノードの並列ジョブを送信すると、ジョブは SUBMITTED ステータスになります。その後、ジョブは、ジョブの依存関係がすべて終了するのを待ちます。ジョブも RUNNABLE ステータスに移行します。最後に、 はジョブの実行に必要なインスタンス容量を AWS Batch プロビジョニングし、これらのインスタンスを起動します。

各マルチノードの並列ジョブには主要なノードが含まれます。メインノードは、 が AWS Batch モニタリングして、送信されたマルチノードジョブの結果を決定する単一のサブタスクです。主要なノードと最初に起動され、STARTING ステータスに移行します。attemptDurationSeconds パラメータで指定されたタイムアウト値は、ジョブ全体に適用され、ノードには適用されません。

主要なノードが RUNNING ステータスに到達すると (ノードのコンテナが実行されてから)、子ノードが起動され、これもまた STARTING ステータスに移行します。子ノードはランダムな順序で始まります。子ノードの起動のタイミングや順序は保証できません。ジョブのすべてのノードが RUNNING ステータスにあること (ノードのコンテナが実行されたあと) を確認するには、アプリケーションコードで AWS Batch API をクエリして主要なノードおよび子ノードの情報を取得するか、あるいはアプリケーションコード内で整合して、分散された処理タスクを開始する前にすべてのノードがオンラインになるまで待機することができます。あるいは、アプリケーションコードは、すべてのノードがオンラインになるまで待ってから、分散処理タスクを開始することもできます。主要なノードのプライベート IP アドレスは、子ノードごとの AWS_BATCH_JOB_MAIN_NODE_PRIVATE_IPV4_ADDRESS 環境変数で利用可能です。アプリケーションコードでは、この情報を各タスク間の整合と通信に使用できます。

個別のノードが存在する場合、これらは終了コードに応じて SUCCEEDED あるいは FAILED に移行します。主要なノードが終了すると、ジョブは完了したと見なされ、すべての子ノードは停止します。子ノードが死亡した場合、ジョブ内の他のノードに対して何も実行 AWS Batch しません。ノード数を減らしてジョブを続行しない場合、これをアプリケーションコードに組み来み、ジョブを終了あるいはキャンセルする必要があります。これを行うと、ジョブは終了またはキャンセルされます。

コンピューティング環境に関する考慮事項

AWS Batchでマルチノードの並列ジョブを実行するためのコンピューティング環境を設定するときに、いくつかの考慮事項があります。

- マルチノードの並列ジョブは、スポットインスタンスを使用する UNMANAGED コンピューティング環境ではサポートされていません。
- マルチノードの並列ジョブをコンピューティング環境に送信する場合、単一のアベイラビリティゾーンでクラスタープレースメントグループを作成して、これをコンピューティングリソース

に関連付けることを検討します。これにより、論理的グループ化されたインスタンス上のマルチノードの並列ジョブが、潜在的に高度なネットワークフローにより近くなります。詳細については、Amazon EC2 ユーザーガイドのプレイスメントグループを参照してください。

- マルチノードの並列ジョブは、スポットインスタンスを使用するコンピューティング環境ではサポートされていません。
- AWS Batch マルチノードの並列ジョブは Amazon ECS `awsvpc`ネットワークモードを使用します。これにより、マルチノードの並列ジョブコンテナに Amazon EC2 インスタンスと同じネットワークプロパティが提供されます。各マルチノードの並列ジョブコンテナは、独自の Elastic Network Interface、プライマリプライベート IP アドレス、および内部の DNS ホスト名を取得します。ネットワークインターフェイスは、ホストコンピューティングリソースと同じ VPC サブネットで作成されます。コンピューティングリソースに適用されるすべてのセキュリティグループも同じく適用されます。詳細については、Amazon Elastic Container Service デベロッパーガイドのawsvpc ネットワークモードを使用したタスクネットワーキングを参照してください。
- コンピューティング環境には、最大で 5 つまでのセキュリティグループが関連付けられている場合があります。
- `awsvpc` ネットワークモードでは、マルチノードの並列ジョブ用にパブリック IP アドレスを使用する Elastic Network Interface を用意していません。インターネットにアクセスするには、NAT ゲートウェイを使用するよう設定されたプライベートサブネットでコンピューティングリソースを起動する必要があります。詳細については、Amazon VPC ユーザーガイド の NAT ゲートウェイを参照してください。ノード間の通信には、プライベート IP アドレス、あるいはノードの DNS ホスト名を使用する必要があります。パブリックサブネット内のコンピューティングリソースで実行されるマルチノードの並列ジョブには、アウトバウンドのネットワークアクセスがありません。プライベートサブネットの VPC および NAT ゲートウェイを作成するには、仮想プライベートクラウド (VPC) の作成 を参照してください。
- 作成されてコンピューティングリソースにアタッチされた Elastic Network Interface は、手動でデタッチしたり、ユーザーのアカウントを使用して変更することはできません。これは、実行中のジョブに関連付けられている Elastic Network Interface が誤って削除されることを回避するためです。Elastic Network Interface を解放するには、ジョブを終了します。
- コンピューティング環境には、マルチノードの並列ジョブをサポートするために十分な最大数の vCPU があることが必要です。
- Amazon EC2 インスタンスクォータには、ジョブを実行するために必要なインスタンスの数が含まれます。たとえば、30 個のインスタンスを必要とするジョブで、リージョンではアカウントが 20 個のインスタンスのみを実行できる場合、このジョブは ステータスで停止します。そうなると、`RUNNABLE` ジョブは、ステータスのままになります。

- マルチノードの並列ジョブでノードグループにインスタンスタイプを指定する場合、コンピューティング環境がそのインスタンスタイプを起動できることが必要です。

GPU ジョブ

GPU ジョブを使用して、インスタンスの GPU を使用するジョブを実行できます。

以下の Amazon EC2 GPU ベースのインスタンスタイプがサポートされています。詳細については、Amazon EC2 P2 インスタンス、Amazon EC2 P3 インスタンス、Amazon EC2 P4d インスタンス、Amazon EC2 P5 インスタンス、Amazon EC2 G3 インスタンス、Amazon EC2 G4 インスタンス、Amazon EC2 G5 インスタンスを参照してください。

インスタンスタイプ	GPU	GPU メモリ	vCPUs	「メモリ」	ネットワーク帯域幅
g3s.xlarge	1	8 GiB	4	30.5 GiB	10 Gbps
g3.4xlarge	1	8 GiB	16	122 GiB	最大 10 Gbps
g3.8xlarge	2	16 GiB	32	244 GiB	10 Gbps
g3.16xlarge	4	32 GiB	64	488 GiB	25 Gbps
g4dn.xlarge	1	16 GiB	4	16 GiB	最大 25 Gbps
g4dn.2xlarge	1	16 GiB	8	32 GiB	最大 25 Gbps
g4dn.4xlarge	1	16 GiB	16	64 GiB	最大 25 Gbps
g4dn.8xlarge	1	16 GiB	32	128 GiB	50 Gbps
g4dn.12xlarge	4	64 GiB	48	192 GiB	50 Gbps
g4dn.16xlarge	1	16 GiB	64	256 GiB	50 Gbps
g5.xlarge	1	24 GiB	4	16 GiB	最大 10 Gbps
g5.2xlarge	1	24 GiB	8	32 GiB	最大 10 Gbps
g5.4xlarge	1	24 GiB	16	64 GiB	最大 25 Gbps

インスタンスタイプ	GPU	GPU メモリ	vCPUs	「メモリ」	ネットワーク帯域幅
g5.8xlarge	1	24 GiB	32	128 GiB	25 Gbps
g5.16xlarge	1	24 GiB	64	256 GiB	25 Gbps
g5.12xlarge	4	96 GiB	48	192 GiB	40 Gbps
g5.24xlarge	4	96 GiB	96	384 GiB	50 Gbps
g5.48xlarge	8	192 GiB	192	768 GiB	100 Gbps
p2.xlarge	1	12 GiB	4	61 GiB	高い
p2.8xlarge	8	96 GiB	32	488 GiB	10 Gbps
p2.16xlarge	16	192 GiB	64	732 GiB	20 Gbps
p3.2xlarge	1	16 GiB	8	61 GiB	最大 10 Gbps
p3.8xlarge	4	64 GiB	32	244 GiB	10 Gbps
p3.16xlarge	8	128 GiB	64	488 GiB	25 Gbps
p3dn.24xlarge	8	256 GiB	96	768 GiB	100 Gbps
p4d.24xlarge	8	320 GiB	96	1152 GiB	4x100 Gbps
p5.48xlarge	8	640 GiB	192	2 TiB	2x100 Gbps

Note

GPU ジョブ AWS Batchでは、NVIDIA GPU をサポートし、x86_64 アーキテクチャを使用するインスタンスタイプのみがサポートされます。たとえば、G4ad および G5g のインスタンスファミリーはサポートされていません。

ジョブ定義の resourceRequirements] パラメータは、コンテナに固定される GPU の数を指定します。この GPUの数は、そのジョブの期間中にインスタンスで実行される他のジョブでは使用

できません。GPU ジョブを実行するコンピューティング環境のすべてのインスタンスタイプは p2、p3、p4、p5、g3、g3s, g4, g5 インスタンスファミリーのいずれかにする必要があります。これを行わないと、GPU ジョブが RUNNABLE 状態で固まる可能性があります。

GPU を使用しないジョブは GPU インスタンスで実行できます。ただし、類似の GPU 以外のインスタンスで実行するよりも、GPU インスタンスで実行する方がコストがかかる場合があります。特定の vCPU、メモリ、および所要時間によっては、このような GPU を使用しないジョブによって GPU ジョブの実行がブロックされる場合があります。

Amazon EKS リソースで GPU ベースのジョブを作成するには

このセクションでは、 AWS BatchのAmazon EKS GPU ワークロードをで実行する方法について説明します。

目次

Amazon EKS で GPU ベースのKubernetes クラスターを作成するには

Amazon EKS で GPU ベースのKubernetes クラスターを作成する前に、Amazon EKS AWS Batch での の開始方法 のステップを完了しておく必要があります。また、次の操作を行います。

- AWS Batch は NVIDIA GPUsを使用したインスタンスタイプをサポートします。
- デフォルトでは、 は Amazon EKS クラスターコントロールプレーンKubernetesのバージョンと一致するバージョンで Amazon EKS 高速 AMI AWS Batch を選択します。

```
$ cat <<EOF > ./batch-eks-gpu-ce.json
{
  "computeEnvironmentName": "My-Eks-GPU-CE1",
  "type": "MANAGED",
  "state": "ENABLED",
  "eksConfiguration": {
    "eksClusterArn": "arn:aws:eks:<region>:<account>:cluster/<cluster-name>",
    "kubernetesNamespace": "my-aws-batch-namespace"
  },
  "computeResources": {
```

```
    "type": "EC2",
    "allocationStrategy": "BEST_FIT_PROGRESSIVE",
    "minvCpus": 0,
    "maxvCpus": 1024,
    "instanceTypes": [
      "p3dn.24xlarge",
      "p4d.24xlarge"
    ],
    "subnets": [
        "<eks-cluster-subnets-with-access-to-internet-for-image-pull>"
    ],
    "securityGroupIds": [
        "<eks-cluster-sg>"
    ],
    "instanceRole": "<eks-instance-profile>"
  }
}
EOF

$ aws batch create-compute-environment --cli-input-json file://./batch-eks-gpu-ce.json
```

AWS Batch はユーザーに代わって NVIDIA GPU デバイスプラグインを管理しません。このプラグインを Amazon EKS クラスターにインストールし、 AWS Batch ノードをターゲットにできるようにする必要があります。詳細については、「 での GPU サポートの有効化Kubernetes」を参照してください GitHub。

AWS Batch ノードをターゲットにするようにNVIDIAデバイスプラグイン (DaemonSet) を設定するには、次のコマンドを実行します。

```
# pull nvidia daemonset spec
$ curl -O https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.12.2/nvidia-device-plugin.yml
# using your favorite editor, add Batch node toleration
# this will allow the DaemonSet to run on Batch nodes
- key: "batch.amazonaws.com/batch-node"
  operator: "Exists"

$ kubectl apply -f nvidia-device-plugin.yml
```

コンピューティングベース (CPU とメモリ) のワークロードと GPU ベースのワークロードを、コンピューティング環境とジョブキューを同じ組み合わせで混在させることはお勧めしません。これは、コンピューティングジョブが GPU の容量を使い果たす可能性があるためです。

ジョブキューをアタッチするには、以下のコマンドを実行します。

```
$ cat <<EOF > ./batch-eks-gpu-jq.json
 {
    "jobQueueName": "My-Eks-GPU-JQ1",
    "priority": 10,
    "computeEnvironmentOrder": [
      {
        "order": 1,
        "computeEnvironment": "My-Eks-GPU-CE1"
      }
    ]
  }
EOF

$ aws batch create-job-queue --cli-input-json file://./batch-eks-gpu-jq.json
```

Amazon EKS GPU ジョブ定義を作成するには

現時点では nvidia.com/gpu のみサポートされており、設定するリソース値は整数でなければなりません。GPU の一部を使用することはできません。詳細については、 ドキュメントのSchedule GPUsKubernetes を参照してください。

Amazon EKS に GPU ジョブ定義を登録するには、以下のコマンドを実行します。

```
$ cat <<EOF > ./batch-eks-gpu-jd.json
{
    "jobDefinitionName": "MyGPUJobOnEks_Smi",
    "type": "container",
    "eksProperties": {
        "podProperties": {
            "hostNetwork": true,
            "containers": [
                {
                    "image": "nvcr.io/nvidia/cuda:10.2-runtime-centos7",
                    "command": ["nvidia-smi"],
                    "resources": {
                        "limits": {
                            "cpu": "1",
                            "memory": "1024Mi",
                            "nvidia.com/gpu": "1"
```

```
                        }
                    }
                }
            ]
        }
    }
}
EOF

$ aws batch register-job-definition --cli-input-json file://./batch-eks-gpu-jd.json
```

Amazon EKS クラスターで GPU ジョブを実行するには

GPU リソースは非圧縮です。 は GPU ジョブのポッド仕様 AWS Batch を作成します。リクエストの値は制限 の値と等しくなります。Kubernetes は必須です。

ジョブを再起動するには、以下のコマンドを実行します。

```
$ aws batch submit-job --job-queue My-Eks-GPU-JQ1 --job-definition MyGPUJobOnEks_Smi --
job-name My-Eks-GPU-Job

# locate information that can help debug or find logs (if using Amazon CloudWatch Logs
 with Fluent Bit)
$ aws batch describe-jobs --job <job-id> | jq '.jobs[].eksProperties.podProperties |
 {podName, nodeName}'
{
  "podName": "aws-batch.f3d697c4-3bb5-3955-aa6c-977fcf1cb0ca",
  "nodeName": "ip-192-168-59-101.ec2.internal"
}
```

AWS Batch ジョブの検索とフィルタリング

AWS Batch コンソールを使用して、ジョブキュー内のジョブを一覧表示できます。ただし、ジョブキューに多数のジョブがあると、特定のジョブを検出のが難しい場合があります。

検索とフィルターを使用して、指定した検索条件に一致するジョブを一覧表示できます。

1. AWS Batch コンソールを開きます。
2. ジョブを選択します。
3. 検索とフィルタリング を有効にします。

> **Note**
>
> ジョブが複数ある場合、このプロセスには数分かかることがあります。

4. ジョブキューを選択してください ボックスで、ジョブキューを選択します。
5. プロパティまたは値でリソースを絞り込むボックスで、一覧表示されているプロパティのいずれかを選択します。
6. 使用するスペースを選択します。この例では、ステータスを選択します。

> **Tip**
>
> 別のプロパティまたは演算子を使用するには、現在の条件を閉じてください。次に、目的のプロパティと演算子を選択します。

7. プロパティ値を入力または選択します。たとえば、ジョブ名の全部または一部を入力するか、Status = RUNNABLE を選択します。
8. フィルターされたリストで目的のジョブを選択します。

> **Tip**
>
> 目的のジョブが表示されない場合は、フィルタリングされたリストをスクロールします。

ジョブのログ

ログ情報を CloudWatch Logs に送信するように AWS Batch ジョブを設定できます。これにより、1 つの便利な場所でジョブからのさまざまなログを表示できます。詳細については、「での CloudWatch ログの使用 AWS Batch」を参照してください。

AWS Batch コンソールで ジョブログを使用して、 AWS Batch ジョブをモニタリングまたはトラブルシューティングすることもできます。

1. AWS Batch コンソールを開きます。
2. ジョブ を選択します。
3. Job キュー で、目的のジョブキューを選択します。

> **Tip**
>
> ジョブキューに複数のジョブがある場合は、検索とフィルタリングをオンにすると、ジョブをすばやく検出できます。詳細については、AWS Batch ジョブの検索とフィルタリングを参照してください。

4. ステータス では、目的のジョブステータスを選択します。
5. 必要な色を選択します。
6. 詳細ページで、ジョブのログまでスクロールします。
7. ログを取得 を選択します。
8. 認証に必須の には、 と入力し**OK**、Amazon の CloudWatch 料金を受け入れるための認証を選択します。

> **Note**
>
> CloudWatch 料金の承認を取り消すには：
>
> 1. 左側のナビゲーションペインで、を選択します。
> 2. ジョブ・ログ の場合は、編集 を選択します。
> 3. バッチの使用を許可する CloudWatchチェックボックスをオフにします。
> 4. [変更の保存] をクリックします。

9. AWS Batch ジョブのログデータを確認します。

> **Tip**
>
> キーワード、最大結果数、ソート に基づいてログをフィルタリングできます。デフォルトの時間間隔のいずれかを選択するか、カスタム間隔を作成して結果をカスタマイズすることもできます。

ジョブ情報

ステータス、 AWS Batch ジョブ定義、コンテナ情報などのジョブ情報を確認できます。

1. AWS Batch コンソールを開きます。

2. ジョブ を選択します。
3. Job キュー で、目的のジョブキューを選択します。

> **Tip**
>
> ジョブキューに複数のジョブがある場合は、検索とフィルタリングをオンにすると、ジョブをすばやく検出できます。詳細については、AWS Batch ジョブの検索とフィルタリングを参照してください。

4. 必要な色を選択します。

> **Note**
>
> AWS Command Line Interface（ AWS CLI) を使用して、 AWS Batch ジョブの詳細を表示することもできます。詳細については、AWS CLI コマンドリファレンス のdescribe-domain を参照してください。

ジョブ定義

AWS Batch ジョブ定義では、ジョブの実行方法を指定します。各ジョブはジョブ定義を参照する必要がありますが、ジョブ定義に指定されているパラメータの多くはランタイムに上書きできます。

コンテンツ

ジョブ定義には以下のような属性が指定されています。

- ジョブのコンテナで使用する Docker イメージ
- コンテナで使用する vCPU の数とメモリの量
- コンテナの開始時に実行するコマンド
- コンテナの開始時に渡す環境変数 (ある場合)
- コンテナで使用するデータボリューム
- ジョブが AWS アクセス許可に使用する IAM ロール (存在する場合)

ジョブ定義で使用できるパラメータの詳細については、「のジョブ定義パラメータ ContainerProperties」を参照してください。

シングルノードのジョブ定義を作成する

AWS Batch でジョブを実行する前に、ジョブ定義を作成する必要があります。このプロセスは、シングルノードとマルチノードの並列ジョブでは若干異なります。このトピックでは、特に、マルチノードの並列ジョブ以外の AWS Batch ジョブのジョブ定義の作成方法を取り上げます。

Amazon Elastic コンテナサービスのリソースで、マルチノードの並列ジョブ定義を作成できます。詳細については、the section called “マルチノードの並列ジョブ定義を作成する”を参照してください。

トピック

- Amazon EKS リソースにシングルノードのジョブ定義を作成する
- AWS Fargate リソースに単一ノードのジョブ定義を作成する
- Amazon EKS リソースでのシングルノードジョブ定義の作成

Amazon EKS リソースにシングルノードのジョブ定義を作成する

Amazon EC2 リソースに新しいジョブ定義を作成するには：

1. https://console.aws.amazon.com/batch/ で AWS Batch コンソールを開きます。
2. ナビゲーションバーから、使用する AWS リージョン を選択します。
3. 左側のナビゲーションペインで ジョブ定義 を選択します。
4. 作成] を選択します。
5. オーケストレーションタイプ には、Amazon Elastic Compute Cloud (Amazon EC2) を選択します。
6. EC2 プラットフォームの設定 で、マルチノード並列を有効にする をオフにします。
7. 名前] には、一意のジョブ定義名を入力します。名前の最大長は 128 文字です。大文字、小文字、数字、ハイフン (-)、アンダースコア (_) を含めることができます。
8. (オプション) 実行タイムアウト で、タイムアウト値 (秒単位) を入力します。実行タイムアウトは、未完了のジョブが終了するまでの時間です。試行時間がこの秒数を超過すると、試行は停止し、FAILED ステータスに移行します。詳細については、ジョブのタイムアウトを参照してください。最小値は 60 秒です。
9. (オプション) スケジューリング優先度 をオンにします。0 ~ 100 の値でスケジューリング優先度を入力します。値が大きいほど優先度が高くなります。
10. (オプション) ジョブの試行回数 には、AWS Batch ジョブを RUNNABLE ステータスに移行する最大回数を入力します。1 ~ 10 の整数を入力します。
11. (オプション) 再試行戦略の条件 では、終了時に評価を追加 を選択します。少なくとも 1 つのパラメータ値を入力し、アクション を選択します。条件セットごとに、アクション を 再試行 または 終了 に設定する必要があります。これらのアクションは、以下のことを意味します。

    - 再試行 — AWS Batch は、指定したジョブ試行回数に達するまで再試行します。

- 終了 — AWS Batch は、ジョブの再試行を停止します。

> **Important**
>
> 終了時に評価を追加 を選択した場合は、少なくとも 1 つのパラメータを設定して アクション を選択するか、終了時に評価を削除 を選択します。

12. (オプション) タグ を展開し、タグを追加 を選択してリソースにタグを追加します。キーとオプショナル値を入力し、新しいタグを追加を選択します。
13. (オプション) タグを伝播 をオンにして、タグをジョブとジョブ定義から Amazon ECS タスクに伝達することができます。
14. 次のページ を選択します。
15. コンテナ設定 セクションで次の操作を行います。

    a. イメージ で、ジョブに使用する Docker イメージを選択します。デフォルトでは、Docker Hub レジストリのイメージを使用できます。*repository-url*/*image*:*tag* で他のリポジトリを指定することもできます。名前の最大長は 225 文字です。大文字、小文字、数字、ハイフン (-)、アンダースコア (_)、コロン (:)、スラッシュ (/)、およびシャープ (#) を含めることができます。このパラメータは、Docker Remote API の コンテナの作成 セクションにある `Image` および docker run の `IMAGE` パラメータにマッピングされます。

    > **Note**
    >
    > Docker イメージのアーキテクチャは、スケジュールされているコンピューティングリソースのプロセッサアーキテクチャと一致している必要があります。例えば、Arm ベースの Docker イメージは、Arm ベースのコンピューティングリソースでのみ実行することができます。

    - Amazon ECR Public リポジトリ内のイメージには、完全な `registry/repository[:tag]` または `registry/repository[@digest]` 命名規則が使用されます (例えば、`public.ecr.aws/`*registry_alias*`/`*my-web-app*`:`*latest*)。
    - Amazon ECR リポジトリ内のイメージには、完全な `registry/repository[:tag]` 命名規則が使用されます (例えば、*aws_account_id*`.dkr.ecr.`*region*`.amazonaws.com /`*my-web-app*`:`*latest*)。

- Docker Hub の公式リポジトリのイメージでは、1 つの名前 (例: `ubuntu` または `mongo`) を使用します。
- Docker Hub の他のリポジトリのイメージは、組織名で修飾されます (例: `amazon/amazon-ecs-agent`)。
- 他のオンラインリポジトリのイメージは、さらにドメイン名で修飾されます (例: `quay.io/assemblyline/ubuntu`)。

b. コマンド構文 には、Bash または JSON を選択します。

c. コマンド] では、コンテナに渡すコマンドを指定します。より単純なコマンドの場合は、コマンドプロンプトの場合と同じようにコマンドを入力します。次に、JSON 結果が正しく、Docker daemon に渡されることを確認します。より複雑なコマンド (特殊文字など) には、JSON 構文を使用してください。

> **Tip**
> 情報 を選択して、Bash と JSON のサンプルコードを参照することができます。

このパラメータは Docker Remote API の コンテナの作成 セクションにある `Cmd` にマッピングされ、`COMMAND` パラメータは docker run にマッピングされます。Docker`CMD` パラメータの詳細については、https://docs.docker.com/engine/reference/builder/#cmdを参照してください。

> **Note**
> コマンドには、パラメータ置換のデフォルト値とプレースホルダーを使用できます。詳細については、パラメータを参照してください。

d. (オプション) 実行ロール で、ユーザーに代わって AWS API コールを実行するためのアクセス許可を Amazon ECS コンテナエージェントに付与する IAM ロールを指定できます。この機能では、タスク用の Amazon ECS IAM ロールを使用します。詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS タスク実行 IAM ロールを参照してください。

e. ジョブロール設定 で、AWS API へのアクセス権限を持つ IAM ロールを選択します。この機能では、タスク用の Amazon ECS IAM ロールを使用します。詳細については、Amazon Elastic Container Service デベロッパーガイドのタスク用の IAM ロールを参照してください。

> **Note**
>
> ここには、Amazon Elastic Container Service Task Role] 信頼関係があるロールのみが表示されます。AWS Batch ジョブの IAM ロールの作成に関する詳細については、Amazon Elastic Container Service デベロッパーガイドのタスク用の IAM ロールとポリシーの作成を参照してください。

16. パラメータ で パラメータを追加 を選択し、パラメータ代替プレースホルダーをキーとオプションの値のペアとして追加します。
17. 環境設定 セクションで:
    a. vCPU では、コンテナ用に予約する vCPU の数を指定します。このパラメータは、Docker Remote API の コンテナの作成 セクションにある `CpuShares` にマッピングされ、`--cpu-shares` オプションは docker run にマッピングされます。各 vCPU は 1,024 個の CPU 配分に相当します。少なくとも 1 つの vCPU を指定する必要があります。
    b. メモリ には、コンテナで使用できるメモリー制限を入力します。コンテナは、ここで指定したメモリを超えようとすると、強制終了されます。このパラメータは、Docker Remote API の コンテナの作成 セクションにある `Memory` にマッピングされ、`--memory` オプションは docker run にマッピングされます。ジョブに対して少なくとも 4 MiB のメモリを指定する必要があります。

    > **Note**
    >
    > 特定のインスタンスタイプのジョブにメモリの優先順位を付けることで、リソース使用率を最大化できます。詳細については、コンピューティングリソースメモリ管理を参照してください。

    c. GPU の数 では、コンテナ用に予約する GPU の数を指定します。
    d. (オプション) 環境変数 で 環境変数を追加 を選択し、環境変数を名前と値のペアとして追加します。これらの変数は、コンテナに渡されます。
    e. (オプション) シークレット で、シークレットを追加 を選択して、シークレットを名前と値のペアとして追加します。これらのシークレットはコンテナに公開されます。詳細については、のジョブ定義パラメータ ContainerProperties のsecretOptionsを参照してください。
18. 次のページ を選択します。
19. Linux 設定 セクションで :

a. ユーザー] には、コンテナ内で使用するユーザー名を入力します。このパラメータは、Docker Remote API の コンテナの作成 セクションにある `User` にマッピングされ、`--user` オプションは docker run にマッピングされます。

b. (オプション) 特権付与 スライダーを右にドラッグすることで、ホストインスタンスに対する昇格されたアクセス許可 (`root` ユーザーと同様) をジョブコンテナに付与することができます。このパラメータは、Docker Remote API の コンテナの作成 セクションにある `Privileged` にマッピングされ、`--privileged` オプションは docker run にマッピングされます。

c. (オプション)init プロセスを有効にする をオンにすると、コンテナ内で `init` プロセスを実行できます。このプロセスは信号を転送し、プロセスを利用します。

20. (オプション) ファイルシステム設定 セクションで :

a. 読み取り専用ファイルシステムを有効にする をオンにして、ボリュームへの書き込みアクセスを削除します。

b. 共有メモリサイズ では、`/dev/shm` ボリュームのサイズ (MiB) を入力します。

c. 最大スワップサイズ では、コンテナが使用できるスワップメモリの合計容量 (MiB 単位) を入力します。

d. スワップ動作 では、コンテナのスワップ動作を示す 0 ~ 100 の値を入力します。値を指定せず、スワップが有効になっている場合、この値はデフォルトで 60 に設定されます。詳細については、のジョブ定義パラメータ ContainerProperties のスワップ動作を参照してください。

e. (オプション) 追加設定] を展開します。

f. (オプション) Tmpfs では、tmpfs を追加 を選択して `tmpfs` マウントを追加します。

g. (オプション) デバイス では、デバイスを追加 を選択してデバイスを追加します。

i. コンテナパス] では、コンテナインスタンスでのデバイスのパスを指定します。このパスは、ホストインスタンスにマッピングされたデバイスを公開するために使用されます。空白のままにすると、ホストパスがコンテナで使用されます。

ii. ホストパス] では、ホストインスタンスでのデバイスのパスを指定します。

iii. アクセス許可 では、デバイスに適用する 1 つ以上のアクセス許可を選択します。使用できる権限は、読み取り、書き込み、MKNOD です。

h. (オプション) ボリューム設定 で、ボリュームを追加 を選択して、コンテナに渡すボリュームのリストを作成します。ボリュームの 名前 と ソースパス を入力し、ボリュームを追加 を選択します。EFS を有効にする オプションを選択することもできます。

i. (オプション) マウントポイント で、マウントポイント構成を追加 を選択し、データボリュームにマウントポイントを追加します。ソースボリュームとコンテナパスを指定する必要があります。これらのマウントポイントはコンテナインスタンス上の Docker daemon に渡されます。ボリュームを 読み取り専用 にすることもできます。

j. (オプション) Ulimit 設定 では、ulimit を追加 を選択して、コンテナの `ulimits` 値を追加します。名前、ソフトリミット、ハードリミット の値を入力し、ulimit を追加 を選択します。

21. (オプション) ロギング設定 セクションで :

a. ログドライバー で、使用するログドライバーを選択します。使用できるログドライバーの詳細については、のジョブ定義パラメータ ContainerProperties のlogDriverを参照してください。

> **Note**
>
> デフォルトでは、`awslogs` ログドライバーが使用されます。

b. オプション では、オプションを追加 を選択してオプションを追加します。名前と値のペアを入力し、オプションを追加 を選択します。

c. シークレット で、シークレットを追加 を選択します。名前と値のペアを入力し、シークレットを追加 を選択してシークレットを追加します。

> **Tip**
>
> 詳細については、のジョブ定義パラメータ ContainerProperties のsecretOptionsを参照してください。

22. 次のページ を選択します。

23. ジョブ定義のレビューについては、設定手順を確認してください。変更する必要がある場合は、編集] を選択します。完了したら、ジョブ定義の作成 を選択します。

AWS Fargate リソースに単一ノードのジョブ定義を作成する

AWS Fargate リソースに新しいジョブ定義を作成するには:

1. AWS Batchコンソールをhttps://console.aws.amazon.com/batch/ で 開きます。
2. 上部のナビゲーションバーから、使用するAWS リージョンを選択します。

3. 左側のナビゲーションパネルで、タスク定義 を選択します。
4. 作成] を選択します。
5. オーケストレーションタイプ] には Fargate を選択します。詳細については、AWS Fargate 上の AWS Batchを参照してください。
6. 名前] には、一意のジョブ定義名を入力します。名前の最大長は 128 文字です。大文字と小文字の英文字、数字、ハイフン(-)、アンダースコア(_)を含めることができます。
7. (オプション) 実行タイムアウト で、タイムアウト値 (秒単位) を入力します。実行タイムアウトは、未完了のジョブが終了するまでの時間です。試行時間がこの秒数を超過すると、試行は停止し、FAILED ステータスに移行します。詳細については、ジョブのタイムアウトを参照してください。最小値は 60 秒です。
8. (オプション) スケジューリング優先順位 をオンにします。0 ~ 100 の値でスケジューリング優先度を入力します。値が大きいほど、低い値よりも優先されます。
9. (オプション) タグ を展開し、タグを追加 を選択してリソースにタグを追加します。タグを伝達 をオンにして、タグをジョブとジョブ定義から伝達することができます。
10. Fargate プラットフォーム設定 セクションで:
    a. ランタイムプラットフォーム では、コンピューティング環境アーキテクチャを選択します。
    b. オペレーティングシステムファミリ では、コンピューティング環境のオペレーティングシステムを選択します。
    c. CPU アーキテクチャ で、vCPU アーキテクチャを選択します。
    d. Fargate プラットフォームバージョン では、LATEST または特定のランタイム環境バージョンを入力します。
    e. (オプション) パブリック IP アドレスの割り当て をオンにして、Fargate ジョブネットワークネットワークインターフェイスにパブリック IP アドレスを割り当てます。プライベートサブネットで動作しているジョブがインターネットにアウトバウンドトラフィックを送信するためには、プライベートサブネットに NAT ゲートウェイをインターネットへのルートリクエストに接続する必要があります。コンテナイメージをプルできるように、この操作を行う必要がある場合があります。詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS のタスクネットワーキングを参照してください。
    f. (オプション) エフェメラルストレージ で、タスクに割り当てるエフェメラルストレージの容量を入力します。エフェメラルストレージの容量は 21 GiB から 200 GiB の間に設定する必要があります。値を指定しない場合には、デフォルトで 20 GiB の一時ストレージが割り当てられます。

> **Note**
>
> エフェメラルストレージには、Fargate プラットフォームバージョン 1.4 以降が必要です。

g. 実行ロール では、Amazon ECS コンテナおよび Fargate エージェントがユーザーに代わって AWS API コールを呼び出せるようにする IAM ロールを選択します。この機能では、タスク用の Amazon ECS IAM ロールを使用します。設定の前提条件を含む詳細については、Amazon Elastic Container Service 開発者ガイドのAmazon ECS タスク実行 IAM ロールを参照してください。

h. ジョブの試行回数 には、AWS Batch ジョブを `RUNNABLE` ステータスに移行する最大回数を入力し 1 ~ 10 の整数を入力します。

i. (オプション) 再試行戦略の条件 では、終了時に評価を追加 を選択します。少なくとも 1 つのパラメータ値を入力し、アクション を選択します。条件セットごとに、アクション を 再試行 または 終了 に設定する必要があります。これらのアクションは、以下のことを意味します。

- 再試行 — AWS Batch は、指定したジョブ試行回数に達するまで再試行します。
- 終了 — AWS Batch は、ジョブの再試行を停止します。

> **Important**
>
> 終了時に評価を追加 を選択した場合は、少なくとも 1 つのパラメータを設定してアクションを選択するか、終了時に評価を削除 を選択する必要があります。

11. 次のページ を選択します。
12. コンテナ設定 セクションで次の操作を行います。

a. イメージ] で、ジョブに使用する Docker イメージを選択します。デフォルトでは、Docker Hubレジストリのイメージを使用できます。*`repository-url/image:tag`* で他のリポジトリを指定することもできます。名前の最大長は 225 文字です。大文字と小文字の英文字、数字、ハイフン (-)、アンダースコア (_)、コロン (:)、ピリオド (.)、スラッシュ (/)、および数字 (#)を含めることができます。このパラメータは、Docker Remote API の コンテナの作成 セクションにある `Image` および docker run の `IMAGE` パラメータにマッピングされます。

> **Note**
>
> Docker イメージのアーキテクチャは、スケジュールされているコンピューティングリソースのプロセッサアーキテクチャと一致している必要があります。例えば、Arm ベースの Docker イメージは、Arm ベースのコンピューティングリソースでのみ実行することができます。

- Amazon ECR Public リポジトリ内のイメージには、完全な `registry/repository[:tag]` または `registry/repository[@digest]` 命名規則が使用されます (例えば、`public.ecr.aws/`*`registry_alias`*`/`*`my-web-app`*`:`*`latest`*)。
- Amazon ECR リポジトリ内のイメージには、完全な `registry/repository[:tag]` 命名規則が使用されます (例えば、*`aws_account_id`*`.dkr.ecr.`*`region`*`.amazonaws.com /`*`my-web-app`*`:`*`latest`*)。
- Docker Hub の公式リポジトリのイメージでは、1 つの名前 (例: `ubuntu` または `mongo`) を使用します。
- Docker Hub の他のリポジトリのイメージは、組織名で修飾されます (例: `amazon/amazon-ecs-agent`)。
- 他のオンラインリポジトリのイメージは、さらにドメイン名で修飾されます (例: `quay.io/assemblyline/ubuntu`)。

b. コマンド構文 には、Bash または JSON を選択します。

c. コマンド] では、コンテナに渡すコマンドを指定します。単純なコマンドの場合は、コマンドプロンプトと同じようにコマンドを入力し、JSON の結果が正しいことを確認します。コマンドは Docker デーモンに渡されます。より複雑なコマンド (特殊文字など) には、JSON 構文を使用してください。

> **Tip**
>
> 情報 を選択して、Bash と JSON のサンプルコードを参照することができます。

このパラメータは、Docker Remote API の コンテナの作成 セクションにある `Cmd` にマッピングされ、`COMMAND` パラメータは docker run にマッピングされます。Docker`CMD` パラ

メータの詳細については、https://docs.docker.com/engine/reference/builder/#cmdを参照してください。

> **Note**
>
> コマンドには、パラメータ置換のデフォルト値とプレースホルダーを使用できます。詳細については、パラメータを参照してください。

d. (オプション) パラメータを名前と値のペアとしてジョブ定義に追加することで、デフォルトのジョブ定義を上書きすることができます。パラメータを追加するには:

- パラメータ で パラメータの追加 を選択し、名前と値のペアを入力して パラメータの追加 を選択します。

> **Important**
>
> パラメータを追加 を選択した場合は、少なくとも1つのパラメータを設定するか、パラメータの削除 を選択する必要があります。

e. 環境設定 セクションで:

i. ジョブロール設定 で、AWS API へのアクセス権限を持つ IAM ロールを選択します。この機能では、タスク用の Amazon ECS IAM ロールを使用します。詳細については、Amazon Elastic Container Service デベロッパーガイドのタスク用の IAM ロールを参照してください。

> **Note**
>
> ここには、Amazon Elastic Container Service Task Role] 信頼関係があるロールのみが表示されます。AWS Batch ジョブの IAM ロールの作成方法に関する詳細については、Amazon Elastic Container Service デベロッパーガイドのタスク用の IAM ロールとポリシーの作成を参照してください。

ii. vCPU では、コンテナ用に予約する vCPU の数を指定します。このパラメータは、Docker Remote API のコンテナの作成セクションの `CpuShares` にマップされ、また`--cpu-shares` オプションは docker run にマップされます。各 vCPU は 1,024 個の CPU 配分に相当します。少なくとも 1 つの vCPU を指定する必要があります。

iii. メモリ で、コンテナで使用できるメモリ制限を入力します。コンテナは、ここで指定したメモリを超えようとすると、停止されます。このパラメータは、Docker Remote API のコンテナの作成セクションの `Memory` にマップされ、また`--memory` オプションは docker run にマップされます。ジョブに対して少なくとも 4 MiB のメモリを指定する必要があります。

GuardDuty Runtime Monitoring を使用すると、 GuardDuty セキュリティエージェントのメモリオーバーヘッドがわずかに発生します。したがって、メモリ制限にはセキュリティエージェントのサイズ GuardDutyを含める必要があります。 GuardDuty セキュリティエージェントのメモリ制限については、「 ユーザーガイド」の「CPU とメモリの制限」を参照してください。 GuardDuty ベストプラクティスの詳細については、「Amazon ECS デベロッパーガイド」の「Runtime Monitoring を有効にした後で Fargate タスクのメモリ不足エラーを修正する方法」を参照してください。

> **Note**
>
> 特定のインスタンスタイプのジョブにメモリの優先順位を付けることで、リソース使用率を最大化できます。詳細については、コンピューティングリソースメモリ管理を参照してください。

f. (オプション) 環境変数 で 環境変数を追加 を選択し、環境変数を名前と値のペアとして追加します。これらの変数は、コンテナに渡されます。

g. (オプション) シークレット で、シークレットを追加 を選択して、シークレットを名前と値のペアとして追加します。これらのシークレットはコンテナに公開されます。詳細については、のジョブ定義パラメータ ContainerProperties のsecretOptionsを参照してください。

h. 次のページ を選択します。

13. (オプション) Linux 設定 セクションで:

a. ユーザー では、コンテナ内で使用するユーザー名を入力します。

b. init プロセスを有効にする をオンにすると、コンテナ内で init プロセスを実行できます。このプロセスは信号を転送し、プロセスを利用します。

c. 読み取り専用ファイルシステムを有効にする をオンにして、ボリュームへの書き込みアクセスを削除します。

d. (オプション) 追加設定] を展開します。

e. マウントポイント設定 では、マウントポイント設定の追加 を選択し、データボリュームにマウントポイントを追加します。ソースボリュームとコンテナパスを指定する必要があり

ます。これらのマウントポイントはコンテナインスタンス上の Docker daemon に渡されます。

f. ボリューム設定 で、ボリュームを追加 を選択して、コンテナに渡すボリュームのリストを作成します。ボリュームの 名前 と ソースパス を入力し、ボリュームを追加 を選択します。

g. ロギング設定 セクションで:

i. (オプション) ログドライバー で、使用するログドライバーを選択します。使用できるログドライバーの詳細については、のジョブ定義パラメータ ContainerPropertiesのlogDriverを参照してください。

> **Note**
>
> デフォルトでは、awslogs ログドライバーが使用されます。

ii. オプション では、オプションを追加 を選択してオプションを追加します。名前と値のペアを入力し、オプションを追加 を選択します。

iii. (オプション) シークレット で、シークレットを追加 を選択してシークレットを追加します。名前と値のペアを入力し、シークレットを追加 を選択します。

> **Tip**
>
> 詳細については、のジョブ定義パラメータ ContainerPropertiesのsecretOptionsを参照してください。

14. 次のページ を選択します。

15. ジョブ定義のレビューについては、設定手順を確認してください。変更する必要がある場合は、Edit] (編集) を選択します。完了したら、ジョブ定義の作成 を選択します。

Amazon EKS リソースでのシングルノードジョブ定義の作成

Amazon Elastic Kubernetes Serviceリソースに新しいジョブ定義を作成するには:

1. AWS Batchコンソールをhttps://console.aws.amazon.com/batch/ で 開きます。
2. 上部のナビゲーションバーから、使用するAWS リージョンを選択します。
3. 左側のナビゲーションパネルで、タスク定義 を選択します。

4. 作成] を選択します。
5. オーケストレーションタイプ には、Elastic Kubernetes Service (EKS) を選択します。
6. 名前] に、一意のジョブ定義名を入力します。名前の最大長は 128 文字です。大文字と小文字の英文字、数字、ハイフン(-)、アンダースコア(_)を含めることができます。
7. (オプション) 実行タイムアウト で、タイムアウト値 (秒単位) を入力します。実行タイムアウトは、未完了のジョブが終了するまでの時間です。試行時間がこの秒数を超過すると、試行は停止し、FAILED ステータスに移行します。詳細については、ジョブのタイムアウトを参照してください。最小値は 60 秒です。
8. (オプション) スケジューリング優先順位 をオンにします。0 ~ 100 の値でスケジューリング優先度を入力します。値が大きいほど、低い値よりも優先されます。
9. (オプション) タグ を展開し、次にタグを追加] を選択してリソースにタグを追加します。
10. 次のページ を選択します。
11. EKS pod プロパティ セクションで:

    a. サービスアカウント名には、podで実行されるプロセスのIDを提供するアカウントを入力します。

    b. ホストネットワーク を有効にしてKubernetespod ネットワークモデルを使用し、受信接続用のリスニングポートを開きます。この設定は送信通信でのみオフにしてください。

    c. (オプション) DNSポリシー で、次のいずれかを選択します :

    - 値なし (null化) — pod は 、Kubernetes 環境からの DNS 設定を無視します。
    - デフォルト — podは、実行されているノードからDNS名前解決を継承します。

    > **Note**
    >
    > DNSポリシーが指定されていない場合、Default はデフォルトDNSポリシーではありません。代わりに、ClusterFirst が使用されます。

    - ClusterFirst — 設定されたクラスタドメインサフィックスと一致しないDNS クエリは、ノードから継承されたアップストリームのネームサーバーに転送されます。
    - ClusterFirstWithHostNet — ホストネットワーク] がオンになっている場合に使用します。

    d. (オプション) ポッドラベル には、ポッドラベルを追加] を選択し、名前と値のペアを入力します。

> **Important**
>
> ポッドラベルのプレフィックスにはkubernetes.io/、k8s.io/、または batch.amazonaws.com/ を含めることはできません。

e. 次のページ を選択します。

f. コンテナの設定　セクションで次の操作を行います。

i. 名前 に、コンテナの一意の名前を入力します。名前は文字または数字で始まる必要があり、最長63文字まで入力できます。小文字と大文字の英文字、数字、およびハイフン (-) を含めることができます。

ii. Image (イメージ) で、ジョブに使用するDocker イメージを選択します。デフォルトでは、Docker Hubレジストリのイメージを使用できます。*repository-url*/*image*:*tag* で他のリポジトリを指定することもできます。名前の長さは最大255文字です。大文字と小文字の英文字、数字、ハイフン (-)、アンダースコア (_)、コロン (:)、ピリオド (.)、スラッシュ (/)、および数字 (#)を含めることができます。このパラメータは、Docker Remote API の コンテナの作成 セクションにある Image にマッピングされ、IMAGE オプションは docker run にマッピングされます。

> **Note**
>
> Docker イメージのアーキテクチャは、スケジュールされているコンピューティングリソースのプロセッサアーキテクチャと一致している必要があります。例えば、Arm ベースの Docker イメージは、Arm ベースのコンピューティングリソースでのみ実行することができます。

- Amazon ECR Public リポジトリ内のイメージには、完全な registry/repository[:tag] または registry/repository[@digest] 命名規則が使用されます (例えば、public.ecr.aws/*registry_alias*/*my-web-app*:*latest*)。
- Amazon ECR リポジトリ内のイメージには、完全な registry/repository[:tag] 命名規則が使用されます (例えば、*aws_account_id*.dkr.ecr.*region*.amazonaws.com /*my-web-app*:*latest*)。

- Docker Hub の公式リポジトリのイメージでは、1 つの名前 (例: ubuntu または mongo) を使用します。
- Docker Hub の他のリポジトリのイメージは、組織名で修飾されます (例:amazon/amazon-ecs-agent)。
- 他のオンラインリポジトリのイメージは、さらにドメイン名で修飾されます (例: quay.io/assemblyline/ubuntu)。

iii. (オプション) イメージプルポリシー では、イメージをプルするタイミングを選択します。

iv. (オプション) コマンド には、コンテナに渡す Bash 又は JSON コマンドを入力します。

v. (オプション) 引数、コンテナに渡す引数を入力します。引数が指定されていない場合は、コンテナイメージコマンドが使用されます。

g. (オプション) 名前と値のマッピングとしてジョブ定義にパラメータを追加して、ジョブ定義のデフォルトを上書きできます。パラメータを追加するには :

- パラメータ] には、名前と値のペアを入力し、次にパラメータの追加 を選択します。

> **Important**
> パラメータを追加 を選択した場合は、少なくとも 1つのパラメータを設定するか、パラメータの削除 を選択する必要があります。

h. 環境設定 セクションで:

i. vCPU で、コンテナ用に予約する vCPU の数を指定します。このパラメータは、Docker Remote API のコンテナの作成セクションの CpuShares にマップされ、また--cpu-shares オプションは docker run にマップされます。各 vCPU は 1,024 個の CPU 配分に相当します。少なくとも 1 つの vCPU を指定する必要があります。

ii. メモリ には、コンテナで使用できるメモリー制限を入力します。コンテナは、ここで指定したメモリを超えようとすると、停止されます。このパラメータは、Docker Remote API のコンテナの作成セクションの Memory にマップされ、また--memory オプションは docker run にマップされます。ジョブに対して少なくとも 4 MiB のメモリを指定する必要があります。

> **Note**
>
> 特定のインスタンスタイプのジョブにメモリを優先順位付けすることで、リソース使用率を最大化できます。詳細については、コンピューティングリソースメモリ管理を参照してください。

i. (オプション) 環境変数 で 環境変数を追加 を選択し、環境変数を名前と値のペアとして追加します。これらの変数は、コンテナに渡されます。

j. (オプション) ボリュームマウント では:

   i. ボリュームマウントを追加 を選択します。

   ii. 名前 を入力し、そして次にボリュームがマウントされているコンテナにマウントパス] を入力します。

   iii. 読み取り専用 を選択すると、ボリュームへの書き込み権限が削除されます。

   iv. ボリュームマウントを追加 を選択します。

k. (オプション)ユーザーとして実行 には、ユーザー ID を入力してコンテナプロセスを実行します。

> **Note**
>
> コンテナを実行するには、ユーザー ID がイメージに存在している必要があります。

l. (オプション)グループとして実行 には、コンテナプロセスのランタイムを実行するグループ ID を入力します。

> **Note**
>
> コンテナを実行するには、グループ ID がイメージに存在している必要があります。

m. (オプション) ホストインスタンスに対する昇格されたアクセス権限 (`root` ユーザーと同様に) をジョブのコンテナに付与するには、特権付与 を選択します。このパラメータは、Docker Remote API のコンテナの作成セクションの `Privileged` にマップされ、また`--privileged` オプションは docker run にマップされます。

n. (オプション) 読み取り専用ルートファイルシステム を有効にして、ルートファイルシステムへの書き込みアクセスを削除します。

o. (オプション) 非ルートユーザーとして実行するpod をオンにして、 非ルートユーザーとしてのコンテナを実行します。

> **Note**
>
> 非ルートユーザーとして実行する がオンになっている場合、kubelet は実行時にイメージを検証して、イメージが UID 0 として実行されていないことを確認します。

p. 次のページ を選択します。

12. ジョブ定義のレビューについては、設定手順を確認してください。変更する必要がある場合は、Edit] (編集) を選択します。完了したら、ジョブ定義の作成 を選択します。

マルチノードの並列ジョブ定義を作成する

AWS Batch でジョブを実行する前に、ジョブ定義を作成する必要があります。このプロセスは、シングルノードとマルチノードの並列ジョブでは若干異なります。このトピックでは、特に AWS Batch のマルチノード並列ジョブのジョブ定義を作成する方法について説明します。詳細については、マルチノードの並列ジョブを参照してください。

> **Note**
>
> AWS Fargate は、マルチノードの並列ジョブをサポートしていません。

Amazon EC2 リソースにマルチノードの並列ジョブ定義を作成する

単一ノードのジョブ定義を作成するには、シングルノードのジョブ定義を作成する を参照してください。

Amazon Elastic Compute クラウドリソースにマルチノードの並列ジョブ定義を作成するには：

1. https://console.aws.amazon.com/batch/ で AWS Batch コンソールを開きます。
2. ナビゲーションバーから、使用する AWS リージョン を選択します。
3. ナビゲーションペインで、ジョブ定義 を選択します。

4. 作成] を選択します。
5. オーケストレーションタイプ には、Amazon Elastic Compute Cloud (Amazon EC2) を選択します。
6. マルチノード並列を有効にする では、マルチノード並列をオンにします。
7. 名前 には、一意のジョブ定義名を入力します。名前には、アルファベット (大文字、小文字)、数字、ハイフン (-)、アンダースコア (_) を含めることができ、最大 128 文字まで使用可能です。
8. (オプション) 実行のタイムアウト で、ジョブの試行で実行する最大秒数を指定します。試行時間がこの秒数を超過すると、試行は停止し、FAILED ステータスに移行します。詳細については、ジョブのタイムアウトを参照してください。
9. (オプション) スケジューリング優先度] をオンにします。0 ~ 100 の値でスケジューリング優先度を入力します。値が大きいほど、低い値よりも優先されます。
10. (オプション) ジョブの試行回数 には、AWS Batch ジョブを RUNNABLE ステータスに移行する最大回数を入力します。1 ~ 10 の整数を入力します。
11. (オプション) 再試行戦略の条件 では、終了時に評価を追加 を選択します。少なくとも 1 つのパラメータ値を入力し、アクション を選択します。条件セットごとに、アクション を 再試行 または 終了 に設定する必要があります。これらのアクションは、以下のことを意味します。

    - 再試行 — AWS Batch は、指定したジョブ試行回数に達するまで再試行します。
    - 終了 — AWS Batch は、ジョブの再試行を停止します。

    > **Important**
    >
    > 終了時に評価を追加 を選択した場合は、少なくとも 1 つのパラメータを設定して アクション] を選択するか、終了時に評価を削除] を選択します。

12. (オプション) タグ を展開し、タグを追加 を選択してリソースにタグを追加します。タグの追加 を選択し、キーとオプションの値を入力します。タグを伝播 をオンにして、タグをジョブとジョブ定義から Amazon ECS タスクに伝達することもできます。
13. 次のページ を選択します。
14. ノード数] にジョブで使用するノードの合計数を入力します。
15. 主要なノード] で、主要なノードに使用するノードインデックスを入力します。デフォルトの主要なノードインデックスは、0 です。
16. インスタンスタイプ でインスタンスのタイプを選択します。

> **Note**
>
> 選択したインスタンスタイプはすべてのノードに適用されます。

17. パラメータ で パラメータを追加 を選択し、パラメータ代替プレースホルダーをキーとオプションの値のペアとして追加します。
18. ノード範囲 セクションで：

    a. ノード範囲の追加] を選択します。これにより、ノード範囲 セクションが作成されます。

    b. ターゲットノード] で、*range_start*:*range_end* 表記を使用してノードグループの範囲を指定します。

    ジョブに指定したノードに対して 5 つまでのノード範囲を作成できます。ノード範囲はノードに対してインデックス値を使用し、ノードインデックスは 0 から開始します。最終ノードグループの範囲終了インデックス値が、指定したノード数より 1 つ少ないことを確認してください。たとえば、10 個のノードを指定し、1 つのノードグループを使用するとします。その場合、終了範囲は 9 にする必要があります。

    c. イメージ で、ジョブに使用する Docker イメージを選択します。デフォルトでは、Docker Hub レジストリのイメージを使用できます。*repository-url*/*image*:*tag* で他のリポジトリを指定することもできます。名前の長さは最大 225 文字です。大文字、小文字、数字、ハイフン (-)、アンダースコア (_)、コロン (:)、スラッシュ (/)、およびシャープ (#) を含めることができます。このパラメータは、Docker Remote API の コンテナの作成 セクションにある `Image` および docker run の `IMAGE` パラメータにマッピングされます。

    > **Note**
    >
    > Docker イメージのアーキテクチャは、スケジュールされているコンピューティングリソースのプロセッサアーキテクチャと一致している必要があります。例えば、Arm ベースの Docker イメージは、Arm ベースのコンピューティングリソースでのみ実行することができます。

    - Amazon ECR Public リポジトリ内のイメージには、完全な `registry/repository[:tag]` または `registry/repository[@digest]` 命名規則が使用されます (例えば、`public.ecr.aws/`*registry_alias*`/`*my-web-app*`:`*latest*)。

- Amazon ECR リポジトリ内のイメージには、完全な `registry/repository[:tag]` 命名規則が使用されます。例えば、*aws_account_id*.dkr.ecr.*region*.amazonaws.com/*my-web-app*:*latest*
- Docker Hub の公式リポジトリのイメージでは、1 つの名前 (例: ubuntu または mongo) を使用します。
- Docker Hub の他のリポジトリのイメージは、組織名で修飾されます (例: amazon/amazon-ecs-agent)。
- 他のオンラインリポジトリのイメージは、さらにドメイン名で修飾されます (例: quay.io/assemblyline/ubuntu)。

d. コマンド構文 には、Bash または JSON を選択します。

e. コマンド] では、コンテナに渡すコマンドを指定します。シンプルなコマンドの場合は、コマンドプロンプトに入力するように スペース区切り タブにコマンドを入力します。次に、JSON 結果が正しいことを確認します。JSON の結果が Docker daemon に渡されます。より複雑なコマンド (特殊文字を含むものなど) の場合は、JSON] タブに切り替えて同等の文字列配列を入力できます。

このパラメータは Docker Remote API の コンテナの作成 セクションの `Cmd` にマッピングされ、`COMMAND` パラメータは docker run にマッピングされます。Docker`CMD` パラメータの詳細については、https://docs.docker.com/engine/reference/builder/#cmdを参照してください。

> **Note**
>
> コマンドには、パラメータ置換のデフォルト値とプレースホルダーを使用できます。詳細については、パラメータを参照してください。

f. vCPU] で、コンテナ用に予約する vCPU の数を指定します。このパラメータは、Docker Remote API の コンテナの作成 セクションにある `CpuShares` にマッピングされ、`--cpu-shares` オプションは docker run にマッピングされます。各 vCPU は 1,024 個の CPU 配分に相当します。少なくとも 1 つの vCPU を指定する必要があります。

g. メモリ] で、ジョブのコンテナに適用されるメモリのハード制限 (MiB 単位) を指定します。コンテナは、ここで指定したメモリを超えようとすると、停止されます。このパラメータは、Docker Remote API の コンテナの作成 セクションにある `Memory` にマッピングされ、`--memory` オプションは docker run にマッピングされます。ジョブに対して少なくとも 4 MiB のメモリを指定する必要があります。

> **Note**
>
> リソースの使用率を最大限に高めるには、特定のインスタンスタイプに対してできるだけ多くのメモリを提供します。詳細については、コンピューティングリソースメモリ管理を参照してください。

h. (オプション) GPU 数 に、ジョブで使用される GPU の数を指定します。ジョブは、指定された数の GPU が固定されているコンテナで実行されます。

i. (オプション) ジョブロール] では、AWS API を使用する権限をジョブのコンテナに付与する IAM ロールを指定できます。この機能では、タスク用の Amazon ECS IAM ロールを使用します。設定の前提条件を含む詳細については、Amazon Elastic Container Service デベロッパーガイド　のタスク用の IAM ロールを参照してください。

> **Note**
>
> Fargate リソースで実行されているジョブには、ジョブロールが必要です。

> **Note**
>
> ここには、Amazon Elastic Container Service Task Role] 信頼関係があるロールのみが表示されます。AWS Batch ジョブの IAM ロールの作成に関する詳細については、Amazon Elastic Container Service デベロッパーガイドのタスク用の IAM ロールとポリシーの作成を参照してください。

j. (オプション) 実行ロール で、ユーザーに代わって AWS API コールを実行するためのアクセス許可を Amazon ECS コンテナエージェントに付与する IAM ロールを指定できます。この機能では、タスク用の Amazon ECS IAM ロールを使用します。詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS タスク実行 IAM ロールを参照してください。

19. (オプション) 追加設定 を展開します。

a. 環境変数 では、環境変数を追加 を選択し、環境変数を名前と値のペアとして追加します。これらの変数は、コンテナに渡されます。

b. (オプション) ジョブロール設定 で、AWS API を使用する権限をジョブのコンテナに付与する IAM ロールを指定できます。この機能では、タスク用の Amazon ECS IAM ロールを使

用します。設定の前提条件を含む詳細については、Amazon Elastic Container Service デベロッパーガイド　のタスク用の IAM ロールを参照してください。

> **Note**
>
> Fargate リソースで実行されているジョブには、ジョブロールが必要です。

> **Note**
>
> ここには、Amazon Elastic Container Service Task Role] 信頼関係があるロールのみが表示されます。AWS Batch ジョブの IAM ロールの作成方法に関する詳細については、Amazon Elastic Container Service デベロッパーガイドのタスク用の IAM ロールとポリシーの作成を参照してください。

c. 実行ロール で、ユーザーに代わって AWS API コールを実行するためのアクセス許可を Amazon ECS コンテナエージェントに付与する IAM ロールを指定できます。この機能では、タスク用の Amazon ECS IAM ロールを使用します。詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS タスク実行 IAM ロールを参照してください。

20. セキュリティ設定 セクションで：

    a. (オプション) 特権 を有効にすると、ホストインスタンスに対する昇格されたアクセス権限 (`root` ユーザーと同様) をジョブのコンテナに付与することができます。このパラメータは、Docker Remote API の コンテナの作成 セクションにある `Privileged` にマッピングされ、`--privileged` オプションは docker run にマッピングされます。

    b. (オプション) ユーザー には、コンテナ内で使用するユーザー名を入力します。このパラメータは、Docker Remote API の コンテナの作成 セクションにある `User` にマッピングされ、`--user` オプションは docker run にマッピングされます。

    c. (オプション) シークレット で、シークレットを追加 を選択して、シークレットを名前と値のペアとして追加します。これらのシークレットはコンテナに公開されます。詳細については、のジョブ定義パラメータ ContainerProperties のsecretOptionsを参照してください。

21. Linux 設定 セクションで：

    a. 読み取り専用ファイルシステムを有効にする をオンにして、ボリュームへの書き込みアクセスを削除します。

b. (オプション) init プロセスを有効にする をオンにして、コンテナ内で `init` プロセスを実行します。このプロセスは信号を転送し、プロセスを利用します。

c. 共有メモリサイズ では、`/dev/shm` ボリュームのサイズ (MiB) を入力します。

d. 最大スワップサイズ では、コンテナが使用できるスワップメモリの合計容量 (MiB 単位) を入力します。

e. スワップ動作 では、コンテナのスワップ動作を示す 0 ~ 100 の値を入力します。値を指定せず、スワップが有効になっている場合、デフォルト値は 60 です。詳細については、のジョブ定義パラメータ ContainerProperties のスワップ動作を参照してください。

f. (オプション) デバイス では、デバイスを追加 を選択してデバイスを追加します。

   i. コンテナパス] では、コンテナインスタンスでのデバイスのパスを指定します。このパスは、ホストインスタンスにマッピングされたデバイスを公開するために使用されます。空白のままにすると、ホストパスがコンテナで使用されます。

   ii. ホストパス] では、ホストインスタンスでのデバイスのパスを指定します。

   iii. アクセス許可] では、デバイスに適用する 1 つ以上のアクセス許可を選択します。使用できる権限は、読み取り、書き込み、MKNOD です。

22. (オプション) マウントポイント で、マウントポイント構成を追加 を選択し、データボリュームにマウントポイントを追加します。ソースボリュームとコンテナパスを指定する必要があります。これらのマウントポイントは、コンテナインスタンスの Docker デーモンに渡されます。ボリュームを 読み取り専用 にすることもできます。

23. (オプション) Ulimit 設定 では、ulimit を追加 を選択して、コンテナの `ulimits` 値を追加します。名前、ソフトリミット、ハードリミット の値を入力し、ulimit を追加 を選択します。

24. (オプション) ボリューム設定 で、ボリュームを追加 を選択して、コンテナに渡すボリュームのリストを作成します。ボリュームの 名前 と ソースパス を入力し、ボリュームを追加 を選択します。EFS を有効にする オプションを選択することもできます。

25. (オプション) Tmpfs では、tmpfs を追加 を選択して `tmpfs` マウントを追加します。

26. (オプション) ロギング設定 セクションで：

    a. ログドライバー で、使用するログドライバーを選択します。使用できるログドライバーの詳細については、のジョブ定義パラメータ ContainerProperties のlogDriverを参照してください。

> **Note**
>
> デフォルトでは、awslogs ログドライバーが使用されます。

b. オプション では、オプションを追加 を選択してオプションを追加します。名前と値のペアを入力し、オプションを追加 を選択します。

c. シークレット で、シークレットを追加 を選択します。名前と値のペアを入力し、シークレットを追加 を選択してシークレットを追加します。

> **Tip**
>
> 詳細については、のジョブ定義パラメータ ContainerProperties のsecretOptionsを参照してください。

27. 次のページ を選択します。
28. ジョブ定義のレビューについては、設定手順を確認してください。変更する必要がある場合は、編集] を選択します。完了したら、ジョブ定義の作成 を選択します。

を使用したジョブ定義の作成 ContainerProperties

以下は、1 つのコンテナを含む空のジョブ定義テンプレートです。このテンプレートを使用してジョブ定義を作成し、それをファイルに保存して オプションで AWS CLI `--cli-input-json`使用できます。これらのパラメータの詳細については、「のジョブ定義パラメータ ContainerProperties」を参照してください。

```
{
    "jobDefinitionName": "",
    "type": "container",
    "parameters": {
        "KeyName": ""
    },
    "schedulingPriority": 0,
    "containerProperties": {
        "image": "",
        "vcpus": 0,
        "memory": 0,
        "command": [
            ""
```

```
        ],
        "jobRoleArn": "",
        "executionRoleArn": "",
        "volumes": [
            {
                "host": {
                    "sourcePath": ""
                },
                "name": "",
                "efsVolumeConfiguration": {
                    "fileSystemId": "",
                    "rootDirectory": "",
                    "transitEncryption": "ENABLED",
                    "transitEncryptionPort": 0,
                    "authorizationConfig": {
                        "accessPointId": "",
                        "iam": "DISABLED"
                    }
                }
            }
        ],
        "environment": [
            {
                "name": "",
                "value": ""
            }
        ],
        "mountPoints": [
            {
                "containerPath": "",
                "readOnly": true,
                "sourceVolume": ""
            }
        ],
        "readonlyRootFilesystem": true,
        "privileged": true,
        "ulimits": [
            {
                "hardLimit": 0,
                "name": "",
                "softLimit": 0
            }
        ],
        "user": "",
```

```
        "instanceType": "",
        "resourceRequirements": [
            {
                "value": "",
                "type": "MEMORY"
            }
        ],
        "linuxParameters": {
            "devices": [
                {
                    "hostPath": "",
                    "containerPath": "",
                    "permissions": [
                        "WRITE"
                    ]
                }
            ],
            "initProcessEnabled": true,
            "sharedMemorySize": 0,
            "tmpfs": [
                {
                    "containerPath": "",
                    "size": 0,
                    "mountOptions": [
                        ""
                    ]
                }
            ],
            "maxSwap": 0,
            "swappiness": 0
        },
        "logConfiguration": {
            "logDriver": "syslog",
            "options": {
                "KeyName": ""
            },
            "secretOptions": [
                {
                    "name": "",
                    "valueFrom": ""
                }
            ]
        },
        "secrets": [
```

```
            {
                "name": "",
                "valueFrom": ""
            }
        ],
        "networkConfiguration": {
            "assignPublicIp": "DISABLED"
        },
        "fargatePlatformConfiguration": {
            "platformVersion": ""
        }
    },
    "nodeProperties": {
        "numNodes": 0,
        "mainNode": 0,
        "nodeRangeProperties": [
            {
                "targetNodes": "",
                "container": {
                    "image": "",
                    "vcpus": 0,
                    "memory": 0,
                    "command": [
                        ""
                    ],
                    "jobRoleArn": "",
                    "executionRoleArn": "",
                    "volumes": [
                        {
                            "host": {
                                "sourcePath": ""
                            },
                            "name": "",
                            "efsVolumeConfiguration": {
                                "fileSystemId": "",
                                "rootDirectory": "",
                                "transitEncryption": "DISABLED",
                                "transitEncryptionPort": 0,
                                "authorizationConfig": {
                                    "accessPointId": "",
                                    "iam": "ENABLED"
                                }
                            }
                        }
```

```
                    ],
                    "environment": [
                        {
                            "name": "",
                            "value": ""
                        }
                    ],
                    "mountPoints": [
                        {
                            "containerPath": "",
                            "readOnly": true,
                            "sourceVolume": ""
                        }
                    ],
                    "readonlyRootFilesystem": true,
                    "privileged": true,
                    "ulimits": [
                        {
                            "hardLimit": 0,
                            "name": "",
                            "softLimit": 0
                        }
                    ],
                    "user": "",
                    "instanceType": "",
                    "resourceRequirements": [
                        {
                            "value": "",
                            "type": "MEMORY"
                        }
                    ],
                    "linuxParameters": {
                        "devices": [
                            {
                                "hostPath": "",
                                "containerPath": "",
                                "permissions": [
                                    "WRITE"
                                ]
                            }
                        ],
                        "initProcessEnabled": true,
                        "sharedMemorySize": 0,
                        "tmpfs": [
```

```
                        {
                            "containerPath": "",
                            "size": 0,
                            "mountOptions": [
                                ""
                            ]
                        }
                    ],
                    "maxSwap": 0,
                    "swappiness": 0
                },
                "logConfiguration": {
                    "logDriver": "awslogs",
                    "options": {
                        "KeyName": ""
                    },
                    "secretOptions": [
                        {
                            "name": "",
                            "valueFrom": ""
                        }
                    ]
                },
                "secrets": [
                    {
                        "name": "",
                        "valueFrom": ""
                    }
                ],
                "networkConfiguration": {
                    "assignPublicIp": "DISABLED"
                },
                "fargatePlatformConfiguration": {
                    "platformVersion": ""
                }
            }
        }
    ]
},
"retryStrategy": {
    "attempts": 0,
    "evaluateOnExit": [
        {
            "onStatusReason": "",
```

```
                "onReason": "",
                "onExitCode": "",
                "action": "RETRY"
            }
        ]
    },
    "propagateTags": true,
    "timeout": {
        "attemptDurationSeconds": 0
    },
    "tags": {
        "KeyName": ""
    },
    "platformCapabilities": [
        "EC2"
    ],
    "eksProperties": {
        "podProperties": {
            "serviceAccountName": "",
            "hostNetwork": true,
            "dnsPolicy": "",
            "containers": [
                {
                    "name": "",
                    "image": "",
                    "imagePullPolicy": "",
                    "command": [
                        ""
                    ],
                    "args": [
                        ""
                    ],
                    "env": [
                        {
                            "name": "",
                            "value": ""
                        }
                    ],
                    "resources": {
                        "limits": {
                            "KeyName": ""
                        },
                        "requests": {
                            "KeyName": ""
```

```
                        }
                    },
                    "volumeMounts": [
                        {
                            "name": "",
                            "mountPath": "",
                            "readOnly": true
                        }
                    ],
                    "securityContext": {
                        "runAsUser": 0,
                        "runAsGroup": 0,
                        "privileged": true,
                        "readOnlyRootFilesystem": true,
                        "runAsNonRoot": true
                    }
                }
            ],
            "volumes": [
                {
                    "name": "",
                    "hostPath": {
                        "path": ""
                    },
                    "emptyDir": {
                        "medium": "",
                        "sizeLimit": ""
                    },
                    "secret": {
                        "secretName": "",
                        "optional": true
                    }
                }
            ]
        }
    }
}
```

> **Note**
>
> 次の AWS CLI コマンドを使用して、シングルコンテナジョブ定義テンプレートを生成できます。

```
$ aws batch register-job-definition --generate-cli-skeleton
```

のジョブ定義パラメータ ContainerProperties

を使用するジョブ定義ContainerPropertiesは、いくつかのパートに分かれています。

- ジョブ定義名
- ジョブ定義のタイプ
- パラメータ置換プレースホルダーのデフォルト
- テーブルプロパティのコンテナ。
- Amazon EKS リソースで実行されるジョブに必要なジョブ定義の Amazon EKS プロパティ
- マルチノード並列ジョブに必要なノードプロパティ
- Fargate リソースで実行されるジョブに必要なプラットフォーム機能
- ジョブ定義のデフォルトタグ伝達の詳細
- ジョブ定義のデフォルト再試行戦略
- ジョブ定義のデフォルトのスケジューリング優先度
- ジョブ定義のデフォルトタグ
- ジョブ定義のデフォルトタグ

目次

- スケジューリング優先順位
- タグ
- タイムアウト

ジョブ定義名

`jobDefinitionName`

ジョブ定義の登録時に名前を指定します。名前の最大長は 128 文字です。大文字および小文字の ASCII 文字、数字、ハイフン(-)、アンダースコア(_)を含めることができます。その名前で最初に登録するジョブ定義のリビジョン番号は 1 です。その名前で登録する後続のジョブ定義には、増分のリビジョン番号が付けられます。

型: 文字列

必須: はい

タイプ

`type`

ジョブ定義の登録時にジョブのタイプを指定します。ジョブが Fargate リソースで実行されている場合、`multinode` はサポートされません。マルチノードの並列ジョブの詳細については、the section called “マルチノードの並列ジョブ定義を作成する”を参照してください。

タイプ: 文字列

有効な値: `container` | `multinode`

必須: はい

パラメータ

`parameters`

ジョブの送信時に、プレースホルダーを置き換えるパラメータやジョブ定義のデフォルトパラメータを上書きするパラメータを指定できます。ジョブ送信リクエストのパラメータは、ジョブ定義のデフォルトよりも優先されます。これにより、同じ形式を使用する複数のジョブに同じジョブ定義を使用できます。送信時にコマンドの値をプログラムで変更することもできます。

型: 文字列間のマッピング

必須: いいえ

ジョブ定義の登録時に、ジョブのコンテナプロパティの command フィールドでパラメータ置換プレースホルダーを使用できます。構文は次のとおりです。

```
"command": [
    "ffmpeg",
    "-i",
    "Ref::inputfile",
    "-c",
    "Ref::codec",
    "-o",
    "Ref::outputfile"
]
```

上の例では、パラメータ置換プレースホルダーとして *Ref::inputfile*、*Ref::codec*、*Ref::outputfile* がコマンドで使用されています。ジョブ定義の parameters オブジェクトで、これらのプレースホルダーのデフォルト値を設定できます。たとえば、*Ref::codec* プレースホルダーのデフォルトを設定するには、ジョブ定義で次のように指定します。

```
"parameters" : {"codec" : "mp4"}
```

このジョブ定義を送信すると、実行時にコンテナのコマンドの *Ref::codec* 引数がデフォルト値の mp4 に置き換えられます。

コンテナプロパティ

ジョブを配置するときにコンテナインスタンスの Docker デーモンに渡すコンテナプロパティのリストを、ジョブ定義の登録時に指定する必要があります。ジョブ定義では、以下のコンテナプロパティを使用できます。単一のノードジョブでは、上記のプロパティはジョブ定義レベルに設定されます。複数ノードの並列ジョブでは、コンテナプロパティは各ノードグループごとに ノードプロパティ レベルで設定されます。

command

コンテナに渡されるコマンド。このパラメータは Docker Remote API の コンテナの作成 セクションの Cmd にマッピングされ、COMMAND パラメータは docker run にマッピングされます。Docker

CMD パラメータの詳細については、https://docs.docker.com/engine/reference/builder/#cmd を参照してください。

```
"command": ["string", ...]
```

タイプ: 文字列配列

必須: いいえ

`environment`

コンテナに渡す環境変数。このパラメータは、Docker Remote API のコンテナの作成セクションにEnvにマップされ、--envオプションはdocker runにマップされます。

> **Important**
>
> 認証情報データなどの機密情報にプレーンテキストの環境変数を使用することはお勧めしません。

> **Note**
>
> 環境変数は AWS_BATCH で始まることはできません。この命名規則は、 AWS Batch サービスによって設定された変数用に予約されています。

型: キーと値のペアの配列

必須: いいえ

`name`

環境変数の名前。

タイプ: 文字列

必須: はい (`environment` を使用する場合)。

`value`

環境変数の値。

タイプ: 文字列

必須: はい (environment を使用する場合)。

```
"environment" : [
    { "name" : "envName1", "value" : "envValue1" },
    { "name" : "envName2", "value" : "envValue2" }
]
```

`executionRoleArn`

ジョブ定義の登録時に IAM ロールを指定できます。Amazon ECS コンテナエージェントは、このロールから付与されるアクセス権限を使用して、関連するポリシーに指定されている API アクションをユーザーに代わって呼び出します。Fargate リソースで実行されるジョブの場合、実行ロールを指定する必要があります。詳細については、AWS Batch 実行 IAM ロールを参照してください。

型: 文字列

必須: いいえ

`fargatePlatformConfiguration`

Fargate リソースで実行されているジョブのプラットフォーム構成。EC2 リソースで実行されているジョブでは、このパラメータを指定しないでください。

タイプ: FargatePlatformConfiguration オブジェクト

必須: いいえ

`platformVersion`

AWS Fargate プラットフォームバージョンは、ジョブ、または Fargate AWS プラットフォームの承認された最新バージョンLATESTを使用するために使用されます。

タイプ: 文字列

デフォルト: LATEST

必須: いいえ

`image`

ジョブの開始に使用するイメージ。この文字列は Docker デーモンに直接渡されます。Docker Hub レジストリのイメージはデフォルトで使用できます。*repository-url*/*image*:*tag* で他

のリポジトリを指定することもできます。最大 255 文字の英字 (大文字と小文字)、数字、ハイフン、アンダースコア、コロン、ピリオド、スラッシュ、シャープ記号を使用できます。このパラメータは、Docker Remote API の コンテナの作成 セクションにある Image および docker run の IMAGE パラメータにマッピングされます。

> **Note**
>
> Docker イメージのアーキテクチャは、スケジュールされているコンピューティングリソースのプロセッサアーキテクチャと一致している必要があります。例えば、Arm ベースの Docker イメージは、Arm ベースのコンピューティングリソースでのみ実行することができます。

- Amazon ECR Public リポジトリ内のイメージには、完全な `registry/repository[:tag]` または `registry/repository[@digest]` 命名規則が使用されます (例えば、`public.ecr.aws/`*`registry_alias`*`/`*`my-web-app`*`:`*`latest`*)。
- Amazon ECR リポジトリ内のイメージには、完全な `registry/repository:[tag]` 命名規則が使用されます。例えば *`aws_account_id`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`my-web-app`*`:`*`latest`* です。
- Docker ハブの公式リポジトリのイメージでは、1 つの名前 (例: ubuntu、mongo) を使用します。
- Docker ハブの他のリポジトリのイメージは、組織名で修飾されます (例: `amazon/amazon-ecs-agent`)。
- 他のオンラインリポジトリのイメージは、さらにドメイン名で修飾されます (例: `quay.io/assemblyline/ubuntu`)。

型: 文字列

必須: はい

`instanceType`

マルチノード並列ジョブに使用するインスタンスタイプ。マルチノードの並列ジョブのすべてのノードグループは、同じインスタンスタイプを使用する必要があります。このパラメータは、単一ノードのコンテナジョブ、または Fargate リソースで実行されるジョブでは無効です。

タイプ: 文字列

必須: いいえ

`jobRoleArn`

ジョブ定義の登録時に IAM ロールを指定できます。ジョブコンテナは、このロールから付与されるアクセス権限を使用して、関連するポリシーに指定されている API アクションをユーザーに代わって呼び出します。詳細については、Amazon Elastic Container Service デベロッパーガイドのタスク用の IAM ロールを参照してください。

タイプ: 文字列

必須: いいえ

`linuxParameters`

コンテナに適用される Linux 固有の変更 (デバイスマッピングの詳細など)。

```
"linuxParameters": {
    "devices": [
        {
            "hostPath": "string",
            "containerPath": "string",
            "permissions": [
                "READ", "WRITE", "MKNOD"
            ]
        }
    ],
    "initProcessEnabled": true|false,
    "sharedMemorySize": 0,
    "tmpfs": [
        {
            "containerPath": "string",
            "size": integer,
            "mountOptions": [
                "string"
            ]
        }
    ],
    "maxSwap": integer,
    "swappiness": integer
}
```

タイプ : LinuxParameters オブジェクト

必須: いいえ

`devices`

コンテナにマッピングされたデバイスのリスト。このパラメータは、Docker Remote API のコンテナの作成セクションの `Devices` にマップされ、`--device` オプションは docker run] にマップされます。

> **Note**
> このパラメータは、Fargate リソースで実行されているジョブには適用されません。

型: デバイスオブジェクト配列

必須: いいえ

`hostPath`

ホストコンテナインスタンスで使用可能なデバイスがあるパス。

型: 文字列

必須: はい

`containerPath`

コンテナでデバイスが公開されるパス。指定しない場合、デバイスはホストパスと同じパスで公開されます。

タイプ: 文字列

必須: いいえ

`permissions`

コンテナでのデバイスのアクセス許可。指定しない場合、アクセス許可は `READ`、`WRITE`、`MKNOD` に設定されます。

タイプ : 文字列の配列

必須: いいえ

有効な値: `READ` | `WRITE` | `MKNOD`

`initProcessEnabled`

true の場合、信号を転送し、プロセスを利用するコンテナ内で `init` プロセスを実行します。このパラメータは、docker run] の `--init` オプションにマッピングされます。このパラメータは、コンテナインスタンスで Docker Remote API バージョン 1.25 以上を使用する必要があります。コンテナインスタンスの Docker Remote API のバージョンを確認するには、コンテナインスタンスにログインし、`sudo docker version | grep "Server API version"` コマンドを実行します。

型: ブール値

必須: いいえ

`maxSwap`

ジョブで使用可能なスワップメモリの合計容量 (単位: MiB) を指定します。このパラメータは、docker run の `--memory-swap` オプションに変換されます。値はコンテナメモリの合計に `maxSwap` 値を加えた値です。詳細については、Docker ドキュメントの`--memory-swap` 詳細を参照してください。

`0` の `maxSwap` 値を指定した場合、コンテナはスワップを使用しません。許容値は、`0` または任意の正の整数です。`maxSwap` パラメータを省略すると、コンテナは実行中のコンテナインスタンスのスワップ設定を使用します。`swappiness` パラメータを使用するには、`maxSwap` 値を設定する必要があります。

> **Note**
> このパラメータは、Fargate リソースで実行されているジョブには適用されません。

タイプ: 整数

必須: いいえ

`sharedMemorySize`

`/dev/shm` ボリュームのサイズ値 (MiB) です。このパラメータは、docker run の `--shm-size` オプションにマッピングされます。

> **Note**
> このパラメータは、Fargate リソースで実行されているジョブには適用されません。

タイプ: 整数

必須: いいえ

`swappiness`

これにより、コンテナのメモリスワップ動作を調整できます。`swappiness` 値が `0` の場合、絶対に必要な場合を除きスワップは行われません。`swappiness` の値が `100` の場合は、ページが積極的にスワップされます。使用できる値は、`0` と `100` の間の整数です。`swappiness` パラメータを指定しない場合、デフォルト値の `60` が使用されます。`maxSwap` の値が指定されていない場合、このパラメータは無視されます。`maxSwap` が 0 に設定されている場合、コンテナはスワップを使用しません。このパラメータは、docker run] の `--memory-swappiness` オプションにマッピングされます。

コンテナごとのスワップ構成を使用する場合は、次の点を考慮してください。

- スワップ領域を有効にし、コンテナが使用するコンテナインスタンスで割り当てる必要があります。

  > **Note**
  >
  > Amazon ECS 最適化 AMI では、スワップはデフォルトで有効になっていません。この機能を使用するには、インスタンスでスワップを有効にする必要があります。詳細については、Amazon EC2 ユーザーガイド」の「インスタンスストアスワップボリューム」または「スワップファイルを使用して Amazon EC2 インスタンスのスワップスペースとして動作するメモリを割り当てる方法」を参照してください。

- スワップ領域のパラメータは、EC2 リソースを使用するジョブ定義でのみサポートされます。
- `maxSwap` および `swappiness` パラメータがジョブ定義から省略されている場合、各コンテナの `swappiness` には、デフォルト値の 60 が設定されます。加えて、スワップの合計使用量は、コンテナが予約したメモリ量の 2 倍に制限されます。

> **Note**
>
> このパラメータは、Fargate リソースで実行されているジョブには適用されません。

タイプ: 整数

必須: いいえ

`tmpfs`

tmpfs マウントのコンテナパス、マウントオプション、およびサイズ。

型: Tmpfs オブジェクト配列

> **Note**
> このパラメータは、Fargate リソースで実行されているジョブには適用されません。

必須: いいえ

`containerPath`

tmpfs ボリュームをマウントするコンテナ内のファイルの絶対パス。

型: 文字列

必須: はい

`mountOptions`

tmpfs ボリュームのマウントオプションのリストです。

有効な値:`defaults|ro|rw|suid|nosuid|dev|nodev|exec|noexec|sync|async|dirsync|remount|ma`

タイプ：文字列の配列

必須: いいえ

`size`

tmpfs ボリュームのサイズ (MiB) です。

タイプ: 整数

必須: はい

`logConfiguration`

ジョブの口グ設定の仕様。

このパラメータは、Docker Remote API のコンテナサービスを作成 セクションの `LogConfig` にマップされ、`--log-driver` オプションは docker run にマップされます。デフォルトでは、コ

ンテナは Docker デーモンで使用されるのと同じロギングドライバーを使用します。ただし、コンテナで Docker デーモンとは異なるログドライバーを使用するには、コンテナの定義内でこのパラメータを使用してログドライバーを指定します。コンテナに異なるロギングドライバーを使用するには、コンテナインスタンス またはリモートログ記録オプションの別のログサーバーでログシステムを適切に設定する必要があります。サポートされているさまざまなログドライバーのオプションの詳細については、Docker ドキュメントのログドライバーの設定を参照してください。

> **Note**
>
> AWS Batch は現在、Docker デーモンで使用できるログドライバーのサブセットをサポートしています (LogConfiguration データ型に表示) 。

このパラメータは、コンテナインスタンスで Docker Remote API バージョン 1.18 以上を使用する必要があります。コンテナインスタンスの Docker Remote API のバージョンを確認するには、コンテナインスタンスにログインし、`sudo docker version | grep "Server API version"` コマンドを実行します。

```
"logConfiguration": {
    "devices": [
        {
            "logDriver": "string",
            "options": {
                "optionName1" : "optionValue1",
                "optionName2" : "optionValue2"
            }
            "secretOptions": [
              {
                  "name" : "secretOptionName1",
                  "valueFrom" : "secretOptionArn1"
              },
              {
                  "name" : "secretOptionName2",
                  "valueFrom" : "secretOptionArn2"
              }
            ]
        }
    ]
}
```

タイプ：LogConfiguration オブジェクト

必須: いいえ

`logDriver`

ジョブに使用するログドライバー。デフォルトでは、 は`awslogs`ログドライバー AWS Batch を有効にします。このパラメータの有効な値は、Amazon ECS コンテナエージェントがデフォルトで通信できるログドライバです。

このパラメータは、Docker Remote API の コンテナサービスを作成セクションの `LogConfig` にマップされ、`--log-driver` オプションは docker run にマップされます。デフォルトでは、ジョブは Docker デーモンで使用されるのと同じロギングドライバーを使用します。ただし、ジョブで Docker デーモンとは異なるログドライバーを使用するには、コンテナの定義内でこのパラメータを使用してログドライバーを指定します。ジョブに別のログドライバを指定する場合は、コンピューティング環境のコンテナインスタンスでログシステムを設定する必要があります。または、別のログサーバーでリモートログオプションを提供するように設定する必要があります。サポートされているさまざまなログドライバーのオプションの詳細については、Docker ドキュメントのログドライバーの設定を参照してください。

> **Note**
>
> AWS Batch は現在、Docker デーモンで使用できるログドライバーのサブセットをサポートしています。Amazon ECS コンテナエージェントの今後のリリースで他のログドライバーが追加される可能性があります。

サポートされているログドライバーは `awslogs`、`fluentd`、`gelf`、`json-file`、`journald`、`logentries`、`syslog`、`splunk` です。

> **Note**
>
> Fargate リソースで実行されているジョブでは、`awslogs` および `splunk` ログドライバーに制限されます。

このパラメータは、コンテナインスタンスで Docker Remote API バージョン 1.18 以上を使用する必要があります。コンテナインスタンスの Docker Remote API のバージョンを確認するには、コンテナインスタンスにログインし、`sudo docker version | grep "Server API version"` コマンドを実行します。

> **Note**
>
> コンテナインスタンスで実行される Amazon ECS コンテナエージェントは、そのインスタンスで利用可能なロギングドライバを `ECS_AVAILABLE_LOGGING_DRIVERS` 環境変数に登録する必要があります。そうしないと、そのインスタンスに配置されたコンテナはこれらのログ設定オプションを使用できません。詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS コンテナエージェントの構成を参照してください。

`awslogs`

Amazon CloudWatch Logs ログ記録ドライバーを指定します。詳細については、Docker ドキュメントのawslogs ログドライバーを使用する「」および「Amazon Logs ログドライバー」を参照してください。 CloudWatch

`fluentd`

Fluentd ログ記録ドライバーを指定します。使用方法やオプションなどの詳細については、Docker ドキュメントのFluentd logging driver を参照してください。

`gelf`

Graylog 拡張形式 (GELF) ログ記録ドライバーを指定します。使用方法やオプションなどの詳細については、Docker ドキュメントのGraylog Extended Format logging driver を参照してください。

`journald`

Journald ログ記録ドライバーを指定します。使用方法やオプションなどの詳細については、Docker ドキュメントのJournald logging driver(Journald logging driver) を参照してください。

`json-file`

JSON ファイルログ記録ドライバーを指定します。使用方法やオプションなどの詳細については、Docker ドキュメントのJSON File logging driver(JSON ファイルlogging driver) を参照してください。

`splunk`

Splunk ログ記録ドライバーを指定します。使用方法やオプションなどの詳細については、Docker ドキュメントのSplunk logging driver を参照してください。

`syslog`

syslog ログ記録ドライバーを指定します。使用方法やオプションなどの詳細については、Docker ドキュメントのSyslog logging driver(Syslog logging driver) を参照してください。

型: 文字列

必須: はい

有効な値: `awslogs` | `fluentd` | `gelf` | `journald` | `json-file` | `splunk` | `syslog`

> **Note**
>
> Amazon ECS コンテナエージェントで作業するカスタムドライバーが以前にリストされていない場合は、 で利用可能な GitHub Amazon ECS コンテナエージェントプロジェクトをフォークし、そのドライバーで作業するようにカスタマイズできます。含めたい変更については、プルリクエストを送信することをお勧めします。ただし、現在、Amazon Web Services では、このソフトウェアの変更されたコピーの実行をサポートしていません。

`options`

ジョブのログドライバーに送信するログ設定オプション。

このパラメータは、コンテナインスタンスで Docker Remote API バージョン 1.19 以上を使用する必要があります。

型: 文字列間のマッピング

必須: いいえ

`secretOptions`

ログ設定に渡すシークレットを示すオブジェクト。詳細については、機密データの指定を参照してください。

タイプ: オブジェクト配列

必須: いいえ

`name`

ジョブで設定するログドライバーオプションの名前。

型: 文字列

必須: はい

`valueFrom`

コンテナのログ設定に公開するシークレットのAmazon Resource Name (ARN)。サポートされている値は、Secrets Manager シークレットの完全な ARN または SSN パラメータストア内のパラメータの完全な ARN のいずれかです。

> **Note**
>
> 起動するタスク AWS リージョン と同じ に SSM パラメータストアパラメータが存在する場合は、パラメータの完全な ARN または名前を使用できます。パラメータが別のリージョンに存在する場合は、完全な ARN を指定する必要があります。

型: 文字列

必須: はい

`memory`

このパラメータは廃止されました。代わりに *resourceRequirements* を使用してください。

ジョブ用に予約されたメモリの MiB 数。

resourceRequirements の使い方の一例として、ジョブ定義に次のような行が含まれている場合は、次のようになります。

```
"containerProperties": {
  "memory": 512
}
```

resourceRequirementsを使用した同等のラインは以下の通りです。

```
"containerProperties": {
  "resourceRequirements": [
```

```
    {
      "type": "MEMORY",
      "value": "512"
    }
  ]
}
```

タイプ: 整数

必須: はい

mountPoints

コンテナでのデータボリュームのマウントポイント。このパラメータは、Docker Remote APIのコンテナの作成セクションにVolumesにマップされ、--volumeオプションはdocker runにマップされます。

```
"mountPoints": [
                {
                  "sourceVolume": "string",
                  "containerPath": "string",
                  "readOnly": true|false
                }
              ]
```

タイプ: オブジェクト配列

必須: いいえ

sourceVolume

マウントするボリュームの名前。

型: 文字列

必須: はい (mountPoints を使用する場合)。

containerPath

ホストボリュームをマウントするコンテナ上のパス。

型: 文字列

必須: はい (mountPoints を使用する場合)。

`readOnly`

この値が `true` の場合、コンテナはボリュームへの読み取り専用アクセスを許可されます。この値が `false` の場合、コンテナはボリュームに書き込むことができます。

型: ブール値

必須: いいえ

デフォルト: False

`networkConfiguration`

Fargate リソースで実行されているジョブのネットワーク構成。EC2 リソースで実行されているジョブでは、このパラメータを指定しないでください。

```
"networkConfiguration": {
   "assignPublicIp": "string"
}
```

タイプ: オブジェクト配列

必須: いいえ

`assignPublicIp`

ジョブにパブリック IP アドレスがあるかどうかを示します。これは、ジョブがアウトバウンドネットワークアクセスが必要な場合に必要です。

タイプ: 文字列

有効な値: `ENABLED` | `DISABLED`

必須: いいえ

デフォルト: `DISABLED`

`privileged`

このパラメータが true のとき、コンテナには、ホストコンテナインスタンスに対する昇格されたアクセス許可 (`root` ユーザーと同様) が付与されます。このパラメータは、Docker Remote API の コンテナの作成セクションの `Privileged` にマップされ、`--privileged` オプションは docker run にマップされます。このパラメータは、Fargate リソースで実行されているジョブには適用されません。これを指定したり、false と指定したりしないでください。

```
"privileged": true|false
```

型: ブール値

必須: いいえ

`readonlyRootFilesystem`

このパラメータが true のとき、コンテナはそのルートファイルシステムへの読み取り専用アクセスを許可されます。このパラメータは、Docker Remote API の コンテナの作成セクションの `ReadonlyRootfs` にマップされ、`--read-only` オプションは docker run にマップされます。

```
"readonlyRootFilesystem": true|false
```

型: ブール値

必須: いいえ

`resourceRequirements`

コンテナに割り当てるリソースのタイプと量。サポートされているリソースには `GPU`、`MEMORY` および `VCPU` などがあります。

```
"resourceRequirements" : [
  {
    "type": "GPU",
    "value": "number"
  }
]
```

タイプ: オブジェクト配列

必須: いいえ

`type`

コンテナに割り当てるリソースのタイプ。サポートされているリソースには `GPU`、`MEMORY` および `VCPU` などがあります。

型: 文字列

必須: はい (`resourceRequirements` を使用する場合)。

`value`

コンテナ用に予約する指定されたリソースの量。値は、指定された `type` によって異なります。

タイプ =GPU

コンテナ用に予約する物理 GPU の数。ジョブ内のすべてのコンテナ用に予約されているGPUの数は、ジョブが起動されたコンピューティングリソースで使用できる GPU の数以下である必要があります。

タイプ =MEMORY

コンテナに適用されるメモリのハード制限 (MiB 単位)。コンテナは、ここで指定したメモリを超えようとすると、強制終了されます。このパラメータは、Docker Remote API のコンテナサービスを作成 セクションの `Memory` にマップされ、`--memory` オプションは docker run にマップされます。ジョブに対して少なくとも 4 MiB のメモリを指定する必要があります。これは必須ですが、マルチノードパラレル (MNP) ジョブでは複数の場所で指定できます。各ノードに少なくとも 1 回指定する必要があります。このパラメータは、Docker Remote API のコンテナサービスを作成 セクションの `Memory` にマップされ、`--memory` オプションは docker run にマップされます。

> **Note**
>
> 特定のインスタンスタイプに対してできるだけ多くのメモリを提供してリソースの使用率を最大限に高めるには、コンピューティングリソースメモリ管理を参照してください。

Fargate リソースで実行されているジョブの場合、`value` はサポートされている値の 1 つと一致する必要があります。さらに、`VCPU` の値は、そのメモリ値でサポートされている値の 1 つである必要があります。

VCPU	MEMORY
0.25 vCPU	512、1024、2048 MiB
0.5 vCPU	1024-4096 MiB、1024 MiB 単位
1 vCPU	2048-8192 MiB、1024 MiB 単位

VCPU	MEMORY
2 vCPU	4096-16384 MiB、1024 MiB 単位
4 vCPU	8192-30720 MiB、1024 MiB 単位
8 vCPU	16384-61440 MiB、4096 MiB 単位
16 vCPU	32768-122880 MiB、8192 MiB 単位

タイプ = vCPU

ジョブ用に予約された vCPU の数。このパラメータは、Docker Remote API のコンテナサービスを作成 セクションの CpuShares にマップされ、--cpu-shares オプションは docker run にマップされます。各 vCPU は 1,024 個の CPU 配分に相当します。EC2 リソースを実行しているジョブの場合、少なくとも 1 つの vCPU を指定する必要があります。これは必須ですが、複数の場所で指定できます。各ノードに少なくとも 1 回指定する必要があります。

Fargate リソースで実行されているジョブの場合、value はサポートされている値のいずれかに一致し、MEMORY 値は VCPU 値でサポートされている値のいずれかである必要があります。サポートされている値は 0.25、0.5、1、2、4、8、および 16 です。

デフォルトでは、Fargate オンデマンド vCPU リソース数のクォータは 6 vCPUs です。Fargate Quotas の詳細については、Amazon Web Services 全般のリファレンスのAWS Fargate quotasを参照してください。

型: 文字列

必須: はい (resourceRequirements を使用する場合)。

secrets

環境変数として公開されるジョブのシークレット。詳細については、機密データの指定を参照してください。

```
"secrets": [
    {
      "name": "secretName1",
      "valueFrom": "secretArn1"
    },
```

```
    {
      "name": "secretName2",
      "valueFrom": "secretArn2"
    }
    ...
]
```

タイプ: オブジェクト配列

必須: いいえ

`name`

シークレットを含む環境変数の名前。

型: 文字列

必須: はい (`secrets` を使用する場合)。

`valueFrom`

コンテナに公開するシークレット。サポートされている値は、 シークレットの完全な Amazon リソースネーム (ARN)、または パラメーターストア内のパラメータの完全な ARN のいずれかです。

> **Note**
>
> 起動するジョブ AWS リージョン と同じ に SSM パラメータストアパラメータが存在する場合は、パラメータの完全な ARN または名前を使用できます。パラメータが別のリージョンに存在する場合は、完全な ARN を指定する必要があります。

型: 文字列

必須: はい (`secrets` を使用する場合)。

`ulimits`

コンテナで設定する `ulimits` 値のリスト。このパラメータは、Docker Remote API の コンテナの作成セクションの `Ulimits` にマップされ、`--ulimit` オプションは docker run にマップされます。

```
"ulimits": [
  {
```

```
    "name": string,
    "softLimit": integer,
    "hardLimit": integer
  }
  ...
]
```

タイプ: オブジェクト配列

必須: いいえ

name

ulimit の type。

型: 文字列

必須: はい (ulimits を使用する場合)。

hardLimit

ulimit タイプのハード制限。

タイプ: 整数

必須: はい (ulimits を使用する場合)。

softLimit

ulimit タイプのソフト制限。

タイプ: 整数

必須: はい (ulimits を使用する場合)。

user

コンテナ内で使用するユーザー名。このパラメータは、Docker Remote API の コンテナの作成セクションの User にマップされ、--user オプションは docker run にマップされます。

```
"user": "string"
```

タイプ: 文字列

必須: いいえ

vcpus

このパラメータは廃止されました。代わりに *resourceRequirements* を使用してください。

コンテナ用に予約された vCPU の数。

resourceRequirements の使い方の一例として、ジョブ定義に次のような行が含まれている場合は、次のようになります。

```
"containerProperties": {
  "vcpus": 2
}
```

resourceRequirements を使用した同等のラインは以下の通りです。

```
"containerProperties": {
  "resourceRequirements": [
    {
      "type": "VCPU",
      "value": "2"
    }
  ]
}
```

タイプ: 整数

必須: はい

volumes

ジョブ定義の登録時に、コンテナインスタンスの Docker デーモンに渡すボリュームのリストを指定できます。コンテナプロパティでは、以下のパラメータを使用できます。

```
"volumes": [
  {
    "name": "string",
    "host": {
      "sourcePath": "string"
    },
    "efsVolumeConfiguration": {
      "authorizationConfig": {
        "accessPointId": "string",
        "iam": "string"
      },
```

```
      "fileSystemId": "string",
      "rootDirectory": "string",
      "transitEncryption": "string",
      "transitEncryptionPort": number
    }
  }
]
```

name

ボリュームの名前。最大 255 文字の英字 (大文字と小文字)、数字、ハイフン、アンダースコアを使用できます。この名前は、コンテナ定義 sourceVolume の mountPoints パラメータで参照されます。

タイプ: 文字列

必須: いいえ

host

host パラメータの内容により、データボリュームがホストコンテナインスタンスで保持されるかどうか、保持される場合はその場所が決まります。host パラメータが空の場合は、Docker デーモンによってデータボリュームのホストパスが割り当てられます。ただし、関連付けられたコンテナが実行を停止した後も、データが保持されるという保証はありません。

> **Note**
>
> このパラメータは、Fargate リソースで実行されているジョブには適用されません。

タイプ: オブジェクト

必須: いいえ

sourcePath

コンテナに渡されるホストコンテナインスタンス上のパス。このパラメータが空の場合は、Docker デーモンによってホストパスが割り当てられます。

host パラメータに sourcePath の場所が含まれている場合、データボリュームは手動で削除するまでホストコンテナインスタンスの指定された場所に保持されます。sourcePath の値がホストコンテナインスタンスに存在しない場は、Docker デーモ

ンによって作成されます。その場所が存在する場合は、ソースパスフォルダの内容がエクスポートされます。

タイプ: 文字列

必須: いいえ

`efsVolumeConfiguration`

このパラメータは、タスクストレージに Amazon Elastic File System を使用している場合に指定します。詳細については、Amazon EFS ボリュームを参照してください。

タイプ: オブジェクト

必須: いいえ

`authorizationConfig`

Amazon EFS ファイルシステムに対する認可構成の詳細。

型: 文字列

必須: いいえ

`accessPointId`

使用する Amazon EFS アクセスポイントの ID。アクセスポイントを指定した場合、`EFSVolumeConfiguration` で指定されているルートディレクトリの値を省略するか、`/` に設定する必要があります。これにより、EFS アクセスポイントに設定されたパスが強制されます。アクセスポイントを使用する場合は、`EFSVolumeConfiguration` で転送中の暗号化を有効にする必要があります。詳細については、Amazon Elastic ファイルシステムユーザーガイドのAmazon EFS アクセスポイントの使用を参照してください。

タイプ: 文字列

必須: いいえ

`iam`

Amazon EFS ファイルシステムをマウントするときに、 AWS Batch ジョブ定義で定義されたジョブ IAM ロールを使用するかどうかを決定します。 EFS 使用する場合は、`EFSVolumeConfiguration` で転送中の暗号化を有効にする必要があります。このパラメータを省略すると、`DISABLED` のデフォルト値が使用されます。詳細については、Amazon EFS アクセスポイントを使用するを参照してください。

タイプ: 文字列

有効な値: ENABLED | DISABLED

必須: いいえ

`fileSystemId`

使用する Amazon EFS ファイルシステムの ID。

型: 文字列

必須: いいえ

`rootDirectory`

ホスト内にルートディレクトリとしてマウントする Amazon EFS ファイルシステム内のディレクトリ。このパラメータを省略すると、Amazon EFS ボリュームのルートが使用されます。/ を指定すると、このパラメータを省略した場合と同じ結果になります。最大長は 4,096 文字です。

> **Important**
>
> `authorizationConfig` に EFS アクセスポイントを指定する場合は、ルートディレクトリパラメータを省略するか、または / に設定する必要があります。これにより、Amazon EFS アクセスポイントに設定されているパスが強制されます。

タイプ: 文字列

必須: いいえ

`transitEncryption`

Amazon ECS ホストと Amazon EFS サーバー間で Amazon EFS データの転送中の暗号化を有効にするかどうかを指定します。Amazon EFS IAM 認証を使用する場合は、転送中の暗号化を有効にする必要があります。このパラメータを省略すると、DISABLED のデフォルト値が使用されます。詳細については、Amazon Elastic File System ユーザーガイドのEncrypting data in transitを参照してください。

タイプ: 文字列

有効な値: ENABLED | DISABLED

必須: いいえ

`transitEncryptionPort`

Amazon ECS ホストと Amazon EFS サーバーとの間で、暗号化されたデータを送信するときに使用するポート。転送中の暗号化ポートを指定しないと、Amazon EFS マウントヘルパーが使用するポート選択方式が使用されます。この値は 0 ~ 65,535 の範囲の値にする必要があります。詳細については、Amazon Elastic File System User Guide] (Amazon Elastic File System ユーザーガイド) のEFS Mount Helper] (EFS マウントヘルパー) を参照してください。

タイプ: 整数

必須: いいえ

Amazon EKS プロパティー

Amazon EKS ベースのジョブに固有のさまざまなプロパティが用意されたオブジェクト。Amazon ECS ベースのジョブ定義には、これを指定できません。

`podProperties`

ジョブの Kubernetes ポッドリソースのプロパティ。

タイプ: EksPodProperties オブジェクト

必須: いいえ

`containers`

Amazon EKS ポッドで使用されるコンテナのプロパティ。

タイプ : EksContainer オブジェクト

必須: いいえ

`args`

エントリポイントへの引数の配列。これを指定しない場合、コンテナイメージの `CMD` が使用されます。これは Kubernetes のポッドのエントリポイント部分の メンバーに対応します。環境変数の参照は、コンテナの環境を使用して展開されます。

参照先の環境変数が存在しない場合、コマンド内の参照は変更されません。例えば、参照先が`$(NAME1)`で、`NAME1` 環境変数が存在しない場合、コマンド文字列は`$(NAME1)`の

ままになります。$$ が $ に置き換えられ、結果の文字列は展開されません。例えば、$$(VAR_NAME) は VAR_NAME 環境変数が存在するかどうかに関係なく、$(VAR_NAME) として渡されます。詳細については、Dockerfile リファレンス のCMDと Kubernetesドキュメンテーションのポッドのコマンドと引数を定義するを参照してください。

タイプ：文字列の配列

必須: いいえ

command

コンテナのエントリポイント。これはシェル内では実行されません。これを指定しない場合、コンテナイメージの ENTRYPOINT が使用されます。環境変数の参照は、コンテナの環境を使用して展開されます。

参照先の環境変数が存在しない場合、コマンド内の参照は変更されません。例えば、参照先が$(NAME1)で、NAME1 環境変数が存在しない場合、コマンド文字列は$(NAME1)のままになります。$$ が $ に置き換えられ、結果の文字列は展開されません。例えば、$$(VAR_NAME) は VAR_NAME 環境変数が存在するかどうかに関係なく、$(VAR_NAME) として渡されます。エントリポイントは更新できません。詳細については、Dockerfileリファレンスとコンテナのコマンドと引数を定義する とエントリーポイントをKubernetesドキュメントのエントリーポイントを参照してください。

タイプ：文字列の配列

必須: いいえ

env

コンテナに渡す環境変数。

> **Note**
>
> 環境変数はAWS_BATCHで始まることはできません。この命名規則は、 が AWS Batch 設定する変数用に予約されています。

型: EksContainerEnvironmentVariable オブジェクトの配列

必須: いいえ

name

環境変数の名前。

タイプ: 文字列

必須: はい

`value`

環境変数の値。

タイプ: 文字列

必須: いいえ

`image`

コンテナの開始に使用する Docker イメージ。

型: 文字列

必須: はい

`imagePullPolicy`

コンテナのイメージプルポリシー。サポートされている値は `Always`、`IfNotPresent`、`Never` です。このパラメータのデフォルトは `IfNotPresent` です。ただし、`:latest` タグが指定されている場合、デフォルトは `Always` です。詳細については、Kubernetesドキュメンテーションのイメージの更新を参照してください。

タイプ: 文字列

必須: いいえ

`name`

コンテナの名前。名前が指定されなかった場合、デフォルト名`Default`が使用されます。ポッド内の各コンテナには一意の名前が必要です。

タイプ: 文字列

必須: いいえ

`resources`

コンテナに割り当てるリソースのタイプと量。サポートされているリソースには `memory`、`cpu` および `nvidia.com/gpu` などがあります。詳細については、ドキュメントのとコンテナのリソース管理を参照してください。

タイプ: EksContainerResourceRequirements オブジェクト

必須: いいえ

`limits`

コンテナ用に予約する指定されたリソースのタイプと量。値は、指定された `name` によって異なります。リソースは、`limits` または `requests` オブジェクトを使用してリクエストできます。

メモリ

コンテナ用のメモリのハード制限 (MiB)。整数とMiサフィックスを使用します。コンテナが指定されたメモリを超えようとすると、強制終了されます。ジョブに対して少なくとも 4 MiB のメモリを指定する必要があります。`limits`、`requests`、または両方で `memory` を指定できます。`memory` が両方の場所で指定されている場合、`limits` で指定される値は、`requests` で指定されている値と等しくなければなりません。

> **Note**
>
> リソースの使用率を最大限に高めるには、使用している特定のインスタンスタイプに対して可能な限り多くのメモリをジョブに割り当てます。この方法の詳細は、コンピューティングリソースメモリ管理を参照してください。

cpu

コンテナ用に予約された CPU の数。値は `0.25` の倍数の乗数でなければなりません。`limits`、`requests`、または両方で `cpu` を指定できます。`cpu` が両方の場所で指定されている場合、`limits` で指定される値は、`requests` で指定されている値以上でなければなりません。

nvidia.com/gpu

コンテナ用に予約された GPU の数。値は整数でなければなりません。`memory` は `limits`、`requests`、または両方で指定できます。`memory` が両方の場所で指定されている場合、`limits` で指定される値は、`requests` で指定されている値と等しくなければなりません。

型: 文字列間のマッピング

値の長さの制限: 最小長さは 1 です。最大長は 256 です。

必須: いいえ

`requests`

コンテナのリクエストに対するリソースのタイプと量。値は、指定された `name` によって異なります。リソースは、`limits` または `requests` オブジェクトを使用してリクエストできます。

メモリ

コンテナ用のメモリのハード制限 (MiB)。整数とMiサフィックスを使用します。コンテナが指定されたメモリを超えようとすると、強制終了されます。ジョブに対して少なくとも 4 MiB のメモリを指定する必要があります。`limits`、`requests`、または両方で `memory` を指定できます。`memory` が両方で指定されている場合、`requests` で指定される値は、`limits` で指定されている値と等しくなければなりません。

> **Note**
>
> 特定のインスタンスタイプに対してできるだけ多くのメモリを提供してリソースの使用率を最大限に高めるには、コンピューティングリソースメモリ管理を参照してください。

cpu

コンテナ用に予約された CPU の数。値は `0.25` の倍数の乗数でなければなりません。`limits`、`requests`、または両方で `cpu` を指定できます。`cpu` が両方で指定されている場合、`limits` で指定される値は、`requests` で指定されている値以上でなければなりません。

nvidia.com/gpu

コンテナ用に予約された GPU の数。値は整数でなければなりません。`nvidia.com/gpu` は `limits`、`requests`、または両方で指定できます。`nvidia.com/gpu` が両方で指定されている場合、`requests` で指定される値は、`limits` で指定されている値と等しくなければなりません。

型: 文字列間のマッピング

値の長さの制限: 最小長さは 1 です。最大長は 256 です。

必須: いいえ

`securityContext`

ジョブのセキュリティコンテキスト。詳細については、Kubernetes ドキュメントのポッドまたはコンテナのセキュリティコンテキストを設定する (ポッドまたはコンテナにセキュリティーコンテキストを設定する) を参照してください。

タイプ: EksContainerSecurityContext オブジェクト

必須: いいえ

`privileged`

このパラメータが `true` のとき、コンテナには、ホストコンテナインスタンスに対する昇格されたアクセス許可が付与されます。権限のレベルは `root` ユーザー権限と同様です。デフォルト値は、`false`です。このパラメータは、Kubernetesドキュメンテーションの特権ポッドのセキュリティポリシーの `privileged` ポリシーに対応しています。

型: ブール値

必須: いいえ

`readOnlyRootFilesystem`

このパラメータが `true` のとき、コンテナはそのルートファイルシステムへの読み取り専用アクセスを許可されます。デフォルト値は、`false`です。このパラメータは、ドキュメンテーションのボリュームとファイルシステムのポッドセキュリティポリシーの `ReadOnlyRootFilesystem` ポリシーに対応しています。

型: ブール値

必須: いいえ

`runAsGroup`

このパラメータを指定すると、コンテナは指定されたグループ ID (gid) で実行されます。このパラメータを指定しない場合、デフォルトはイメージメタデータに指定されているグループです。このパラメータは、ドキュメンテーションのユーザーとグループのポッドセキュリティポリシーの `RunAsGroup` および `MustRunAs` ポリシーと対応しています。

型: Long

必須: いいえ

runAsNonRoot

このパラメータを指定すると、コンテナは 0 以外の uid のユーザーとして実行されます。このパラメータを指定しない場合、そのようなルールが適用されます。このパラメータは、ドキュメンテーションのユーザーとグループのポッドセキュリティポリシーの RunAsUser および MustRunAsNonRoot ポリシーと対応しています。

型: Long

必須: いいえ

runAsUser

このパラメータを指定すると、コンテナは指定されたユーザー ID (uid) で実行されます。このパラメータを指定しない場合、デフォルトはイメージメタデータに指定されているユーザーです。このパラメータは、ドキュメンテーションのユーザーとグループのポッドセキュリティポリシーの RunAsUser および MustRanAs ポリシーと対応しています。

型: Long

必須: いいえ

volumeMounts

Amazon EKS ジョブのコンテナのボリュームマウント。Kubernetes のボリュームとボリュームマウントの詳細については、ドキュメンテーションのボリュームを参照してください。

型: EksContainerVolumeMount オブジェクトの配列

必須: いいえ

mountPath

ボリュームをマウントするコンテナ上のパス。

タイプ: 文字列

必須: いいえ

name

ボリュームマウントの名前。これは、ポッド内のいずれかのボリュームの名前と一致する必要があります。

タイプ: 文字列

必須: いいえ

`readOnly`

この値が `true` の場合、コンテナはボリュームへの読み取り専用アクセスを許可されます。それ以外の場合、コンテナはボリュームに書き込むことができます。デフォルト値は `false` です。

タイプ: ブール

必須: いいえ

`dnsPolicy`

ポッドの DNS ポリシー。デフォルト値は、`ClusterFirst`です。`hostNetwork` パラメータが指定されていない場合、デフォルトは `ClusterFirstWithHostNet` です。`ClusterFirst` は、設定されたクラスタドメインサフィックスと一致しない DNS クエリが、ノードから継承されたアップストリームのネームサーバーに転送されることを示します。RegisterJob定義 API オペレーション`dnsPolicy`で に値が指定されていない場合、定義または DescribeJobs API DescribeJobオペレーションのいずれか`dnsPolicy`によって の値が返されません。ポッド仕様設定には、`hostNetwork` パラメータの値に応じて、`ClusterFirst` または `ClusterFirstWithHostNet` のいずれかが含まれます。詳細については、Kubernetesドキュメンテーションの[Pod's DNS policy](ポッドの DNS ポリシー) を参照してください。

有効な値: `Default` | `ClusterFirst` | `ClusterFirstWithHostNet`

タイプ: 文字列

必須: いいえ

`hostNetwork`

ポッドがホストのネットワーク IP アドレスを使用するかどうかを示します。デフォルト値は、`true`です。これを に設定すると、Kubernetes ポッドネットワークモデルが有効になります。ほとんどの AWS Batch ワークロードは出力専用であり、受信接続の各ポッドに IP 割り当てのオーバーヘッドは必要ありません。詳細については、Kubernetes ドキュメンテーションのホストの名前空間とPポッドネットワーキングを参照してください。

型: ブール値

必須: いいえ

`serviceAccountName`

ポッドを実行するために使用されるサービスアカウントの名前。詳細については、Amazon EKS ユーザーガイドのKubernetes サービスアカウントとIAM ロールを引き継ぐように Kubernetes サービスアカウントを設定する、Kubernetes ドキュメンテーションのポッドでのKubernetesサービスアカウントのドキュメンテーションKubernetesを参照してください。

タイプ: 文字列

必須: いいえ

`volumes`

Amazon EKS リソースを使用するジョブ定義のボリュームを指定します。

型: EksVolume オブジェクトの配列

必須: いいえ

EmptyDir

Kubernetes `emptyDir`ボリュームの設定を指定します。`emptyDir` ボリュームは、ポッドがノードに割り当てられるときに最初に作成されます。そのポッドがそのノードで実行されている限り存在します。`emptyDir` ボリュームは最初は空です。ポッド内のすべてのコンテナは、`emptyDir` ボリューム内のファイルを読み書きできます。ただし、`emptyDir` ボリュームは各コンテナの同じパスまたは異なるパスにマウントできます。何らかの理由でポッドがノードから削除されると、`emptyDir` のデータは完全に削除されます。詳細については、ドキュメントのemptyDirを参照してください。

タイプ: EksEmptyDir オブジェクト

必須: いいえ

medium

ボリュームを保存するメディア。デフォルト値は、ノードのストレージを使用する空の文字列です。

""

(デフォルト) ノードのディスクストレージを使用します。

メモリ

ノードの RAM にバックアップされている tmpfs ボリュームを使用します。ノードが再起動するとボリュームの内容は失われ、ボリューム上のストレージはコンテナのメモリ制限に対して計算されます。

タイプ: 文字列

必須: いいえ

サイズ制限

ボリュームの最大サイズ。デフォルトでは、最大サイズは定義されていません。

型: 文字列

長さの制限：最小長は 1 です。最大長は 256 です。

必須: いいえ

HostPath

Kubernetes hostPathボリュームの設定を指定します。hostPath ボリュームは、ホストノードのファイルシステムから既存のファイルまたはディレクトリをポッドにマウントします。詳細については、ドキュメンテーションのhostPathを参照してください。

タイプ: EksHostパスオブジェクト

必須: いいえ

パス

ポッドのコンテナにマウントするホスト上のファイルまたはディレクトリのパス。

タイプ: 文字列

必須: いいえ

name

ボリュームの名前。名前は DNS サブドメイン名として許可されている必要があります。詳細については、ドキュメンテーションのDNS サブドメイン名を参照してください。

型: 文字列

必須: はい

シークレット

Kubernetes secretボリュームの設定を指定します。詳細については、 ドキュメントのサービスを参照してください。

タイプ : EksSecret オブジェクト

必須: いいえ

オプション

シークレットとシークレットのキーのどちらを定義する必要があるかを指定します。

型: ブール値

必須: いいえ

SecretName

シークレットの名前。名前は DNS サブドメイン名として許可されている必要があります。詳細については、ドキュメンテーションのDNS サブドメイン名を参照してください。

型: 文字列

必須: はい

プラットフォーム機能

`platformCapabilities`

ジョブ定義に必要なプラットフォーム機能。値が指定されていない場合、デフォルトは EC2 になります。Fargate リソースで実行されるジョブの場合、FARGATEが指定されている。

> **Note**
> ジョブを Amazon EKS リソースで実行する場合は、platformCapabilities を指定してはならず、

タイプ: 文字列

有効な値: EC2 | FARGATE

必須: いいえ

タグの伝播

`propagateTags`

タグをジョブまたはジョブ定義から対応する Amazon ECS タスクに伝播するかどうかを指定します。値を指定しない場合、タグは伝播されません。タグは、タスクの作成時にのみタスクに伝播できます。同じ名前のタグの場合、ジョブタグはジョブ定義タグよりも優先されます。ジョブとジョブ定義から合わせたタグの合計数が 50 を超える場合、ジョブは `FAILED` 状態に移行します。

> **Note**
>
> ジョブを Amazon EKS リソースで実行する場合は、`propagateTags` を指定してはならず、

型: ブール値

必須: いいえ

ノードプロパティ

`nodeProperties`

マルチノードの並列ジョブ定義を登録する場合、ノードプロパティの一覧を指定する必要があります。これらのノードプロパティは、ジョブで使用するノード数、主要なノードのインデックスおよび使用する別のノード範囲を定義する必要があります。ジョブを Fargate リソースで実行する場合は、`nodeProperties` を指定することはできません。代わりに `containerProperties` を使用してください。ジョブ定義では、以下のノードプロパティを使用できます。詳細については、マルチノードの並列ジョブを参照してください。

> **Note**
>
> ジョブを Amazon EKS リソースで実行する場合は、`nodeProperties` を指定してはならず、

タイプ : NodeProperties オブジェクト

必須: いいえ

`mainNode`

複数ノードの並列ジョブの主要なノードにノードインデックスを指定します。このノードインデックス値は、ノード数未満である必要があります。

タイプ: 整数

必須: はい

`numNodes`

マルチノードの並列ジョブに関連付けられたノードの数。

タイプ: 整数

必須: はい

`nodeRangeProperties`

マルチノードの並列ジョブに関連付けられたノード範囲とそのプロパティの一覧。

> **Note**
> ノードグループとは、同じコンテナプロパティを共有するジョブノードの同一グループです。 では、ジョブごとに 5 つまでの個別のノードグループを指定できます。を使用して AWS Batch 、ジョブごとに最大 5 つの異なるノードグループを指定できます。

タイプ: NodeRangeProperty オブジェクトの配列

必須: はい

`targetNodes`

ノードのインデックス値を使用したノードの範囲。`0:3` の範囲は、インデックス値が `0` から `3` のノードを示しています。開始範囲値が省略されている場合 ()、範囲の開始値に が使用されます。終了範囲値が省略されている場合 (`n:`)、範囲の終了値にはできるだけ高いノードインデックスが使用されます。累積ノード範囲は、すべてのノード (`0:n`) を考慮す

る必要があります。たとえば、ノード範囲をネストできます。例：0:10 および 4:5。この場合は、4:5 の範囲プロパティは、0:10 プロパティを上書きします。

タイプ: 文字列

必須: いいえ

container

ノード範囲のコンテナの詳細。詳細については、「コンテナプロパティ」を参照してください。

タイプ：ContainerProperties オブジェクト

必須: いいえ

再試行戦略

retryStrategy

ジョブ定義の登録時に、オプションとして、このジョブ定義で送信したジョブが失敗したときの再試行戦略を指定できます。SubmitJob オペレーション中に指定された再試行戦略は、ここで定義された再試行戦略よりも優先されます。デフォルトでは、各ジョブは 1 回試行されます。複数の試行を指定すると、ジョブが失敗した場合、ジョブが再試行されます。失敗の例としては、ジョブがゼロ以外の終了コードが返した場合やコンテナインスタンスが終了した場合が含まれます。詳細については、「ジョブの再試行の自動化」を参照してください。

タイプ：RetryStrategy オブジェクト

必須: いいえ

attempts

ジョブを RUNNABLE ステータスに移行する回数。1〜10 回の試行を指定できます。attempts の回数が 1 より大きい場合、ジョブは RUNNABLE に移行するまでにその回数内で再試行されます。

```
"attempts": integer
```

タイプ: 整数

必須: いいえ

evaluateOnExit

ジョブの再試行または失敗の条件を指定する、最大 5 つのオブジェクトの配列。このパラメータを指定する場合は、attempts パラメータも指定する必要があります。evaluateOnExitを指定しても一致するエントリがない場合、ジョブは再試行されます。

```
"evaluateOnExit": [
   {
      "action": "string",
      "onExitCode": "string",
      "onReason": "string",
      "onStatusReason": "string"
   }
]
```

タイプ: EvaluateOnExit オブジェクトの配列

必須: いいえ

action

指定された条件 (onStatusReason、onReason および onExitCode)がすべて満たされた場合に実行するアクションを指定します。値は大文字と小文字が区別されません。

型: 文字列

必須: はい

有効な値: RETRY | EXIT

onExitCode

ジョブに対して返された ExitCode の10進表現と照合する glob パターンが含まれています。パターンの最大長は 512 文字です。数字のみを含めることができます。文字や特殊文字を含めることはできません。必要に応じて末尾をアスタリスク (*) にでき、文字列の先頭だけが完全に一致する必要があります。

タイプ: 文字列

必須: いいえ

onReason

ジョブに対して返された Reason と照合する glob パターンが含まれます。パターンの最大長は 512 文字です。文字、数字、ピリオド (.)、コロン (:)、および空白 (スペースまたはタブを含む) を含めることができます。必要に応じて末尾をアスタリスク (*) にでき、文字列の先頭だけが完全に一致する必要があります。

タイプ: 文字列

必須: いいえ

onStatusReason

ジョブに対して返された StatusReason と照合する glob パターンが含まれます。パターンの最大長は 512 文字です。文字、数字、ピリオド (.)、コロン (:)、および空白 (スペースまたはタブを含む) を含めることができます。必要に応じて末尾をアスタリスク (*) にでき、文字列の先頭だけが完全に一致する必要があります。

タイプ: 文字列

必須: いいえ

スケジューリング優先順位

schedulingPriority

このジョブ定義で送信されるジョブのタイムアウト時間。これは、公平配分ポリシーが適用されたジョブキュー内のジョブにのみ影響します。スケジューリング優先度が高いジョブは、スケジューリング優先度の低いジョブの前にスケジュールされます。

サポートされている最小値は 0 で、サポートされている最大値は 9999 です。

タイプ: 整数

必須: いいえ

タグ

tags

ジョブ定義に関連付けるキーバリューペアのタグ。詳細については、AWS Batch リソースのタグ付けを参照してください。

型: 文字列間のマッピング

必須: いいえ

タイムアウト

`timeout`

ジョブのタイムアウト期間を設定して、ジョブの実行時間がそれよりも長い場合、 がジョブ AWS Batch を終了するようにすることができます。詳細については、「ジョブのタイムアウト」を参照してください。タイムアウトが原因でジョブが終了した場合は、再試行されません。SubmitJob オペレーション中に指定されたタイムアウト設定は、ここで定義されているタイムアウト設定よりも優先されます。詳細については、「ジョブのタイムアウト」を参照してください。

タイプ : JobTimeout オブジェクト

必須: いいえ

`attemptDurationSeconds`

`startedAt` によって未終了のジョブが終了されるまでの時間 (秒) (ジョブ試行の AWS Batch タイムスタンプから計測)。タイムアウトの最小値は 60 秒です。

配列ジョブの場合、タイムアウトは親配列ジョブではなく子ジョブに適用されます。

マルチノード並列 (MNP) ジョブの場合、タイムアウトは、ジョブ全体に適用され、個々のノードには適用されません。

タイプ: 整数

必須: いいえ

を使用してジョブ定義を作成する EcsProperties

AWS Batch ジョブ定義を使用すると`EcsProperties`、ハードウェア、センサー、3D 環境、その他のシミュレーションを別々のコンテナでモデル化できます。この機能を使用すると、ワークロードコンポーネントを論理的に整理し、メインアプリケーションから切り離すことができます。この機能は、Amazon Elastic Container Service (Amazon ECS)、Amazon Elastic Kubernetes サービス (Amazon EKS)、およびで使用できます。 AWS Batch AWS Fargate

ContainerPropertiesEcsPropertiesジョブ定義との比較

ユースケースに応じて、ContainerPropertiesEcsPropertiesまたはジョブ定義を使用することを選択できます。大まかに言うと、 AWS Batch EcsPropertiesでジョブを実行することは、を使用してジョブを実行するのと似ています。ContainerProperties

ContainerPropertiesを使用する従来のジョブ定義構造は引き続きサポートされます。現在この構造を使用しているワークフローがある場合は、引き続き実行できます。

主な違いは、EcsPropertiesベース定義に対応するためにジョブ定義に新しいオブジェクトが追加されていることです。

たとえば、Amazon ECS と Fargate ContainerProperties で使用するジョブ定義は次のような構造になっています。

```
{
   "containerProperties": {
     ...
     "image": "my_ecr_image1",
     ...
  },
...
}
```

Amazon ECS と Fargate EcsProperties で使用するジョブ定義の構造は次のとおりです。

```
{
  "ecsProperties": {
    "taskProperties": [{
      "containers": [
        {
          ...
          "image": "my_ecr_image1",
          ...
        },
        {
          ...
          "image": "my_ecr_image2",
          ...
        },
```

API の一般的な変更点 AWS Batch

以下に、EcsPropertiesおよび EcsProperties API データ型を使用する場合の主な相違点についてさらに概説します。

- ContainerPropertiesTaskContainerProperties内で使用されるパラメータの多くは内にあります。例としては、command、image、privileged、secrets、などがありますusers。それらはすべて内部にありますTaskContainerProperties。
- TaskContainerPropertiesパラメータの中には、レガシー構造と同等の機能を持たないものもあります。例としては、、、dependsOn、essentialnameipcMode、などがあります。pidMode詳細については、「」EcsTaskDetailsと「」を参照してくださいTaskContainerProperties。

  また、ContainerPropertiesパラメータの中には、構造内に同等のものや用途がないものもあります。EcsPropertiesではtaskProperties、containercontainersがに置き換えられ、新しいオブジェクトは最大 10 個の要素を受け付けることができるようになりました。詳細については、「コンテナプロパティ」と「: container」を参照してくださいRegisterJobDefinition。EcsTaskProperties
- taskRoleArnjobRoleArn機能的にはと同等です。詳細については、「」taskRoleArn と「EcsTaskProperties:」を参照してくださいContainerProperties。jobRoleArn
- EcsProperties構造には 1 ~ 10 個のコンテナを含めることができます。詳細については、「コンテナ」を参照してください。EcsTaskProperties
- taskPropertiesと instanceTypes オブジェクトは配列ですが、現在受け入れられる要素は 1 つだけです。例:: タスクプロパティと:EcsPropertiesinstanceTypes。NodeRangeProperty

Amazon ECS 用のマルチコンテナジョブ定義

Amazon ECS のマルチコンテナ構造に対応するため、一部の API データ型は異なります。例えば、などです

- ecsPropertiescontainerProperties単一コンテナ定義と同じレベルです。詳細については、『AWS Batch API リファレンスガイド』EcsPropertiesのを参照してください。
- taskPropertiesAmazon ECS タスクに定義されているプロパティが含まれています。詳細については、『AWS Batch API リファレンスガイド』EcsPropertiesのを参照してください。

- containerscontainerProperties単一コンテナ定義と同様の情報が含まれています。主な違いは、最大 10 containers 個のコンテナを定義できることです。詳細については、API TaskPropertiesAWS Batch リファレンスガイドの ECS: Containers を参照してください。
- essentialパラメータは、コンテナがジョブにどのように影響するかを示します。ジョブを進めるには、すべての必須コンテナが正常に完了 (0 として終了) する必要があります。essentail とマークされたコンテナに障害が発生した (0 以外として終了する) と、ジョブは失敗します。

  trueデフォルト値はで、少なくとも 1 つのコンテナをとしてマークする必要があります。essential詳細については、essential API リファレンスガイドの「AWS Batch 」を参照してください。
- dependsOnパラメータを使用すると、コンテナの依存関係のリストを定義できます。詳細については、dependsOn API リファレンスガイドの「AWS Batch 」を参照してください。

> **Note**
>
> dependsOnリストの複雑さと関連するコンテナランタイムは、ジョブの開始時間に影響する可能性があります。依存関係の実行に時間がかかる場合、STARTINGジョブは完了するまでその状態のままになります。

ecsPropertiesと構造の詳細については、「ECSProperties RegisterJobDefinitionのリクエスト構文」を参照してください。

Amazon EKS 用のマルチコンテナジョブ定義

Amazon EKS のマルチコンテナ構造に対応するため、一部の API データ型は異なります。例えば、などです

- nameはコンテナの一意の識別子です。このオブジェクトは 1 つのコンテナには必要ありませんが、Pod で複数のコンテナを定義する場合は必須です。nameが単一のコンテナに対して定義されていない場合は、デフォルトの名前であるが適用されます。default
- initContainerseksPodPropertiesデータ型内で定義されます。これらはアプリケーションコンテナの前に実行され、常に最後まで実行され、次のコンテナが起動する前に正常に完了する必要があります。

  これらのコンテナは Amazon EKS コネクタエージェントに登録され、登録情報を Amazon Elastic Kubernetes Service バックエンドデータストアに保持します。initContainersオブジェクト

は最大 10 個の要素を受け入れることができます。詳細については、Kubernetesドキュメントの「InitContainers」を参照してください。

> **Note**
>
> initContainersオブジェクトはジョブの開始時間に影響を与える可能性があります。initContainers実行に時間がかかる場合、STARTINGジョブは完了するまでその状態のままになります。

- shareProcessNamespacePod 内のコンテナが同じプロセス名前空間を共有できるかどうかを示します。デフォルト値は false です。trueこれをに設定すると、同じ Pod にある他のコンテナ内のプロセスをコンテナが見たり通知したりできるようになります。
- どのコンテナにも重要性があります。ジョブが成功するには、すべてのコンテナが正常に完了 (0 として終了) する必要があります。1 つのコンテナに障害が発生すると (0 以外で終了)、ジョブは失敗します。

eksPropertiesand 構造の詳細については、「EKSProperties RegisterJobDefinitionのリクエスト構文」を参照してください。

AWS Batch を使用した ジョブシナリオ EcsProperties

このトピックでは、 を使用する AWS Batch ジョブ定義をニーズに基づいてどのように構造化EcsPropertiesできるかを説明しますRegisterJobDefinition。これらの例をファイルにコピーし、必要に応じてカスタマイズしてから、 AWS Command Line Interface （ AWS CLI) を使用して を呼び出すことができますRegisterJobDefinition。

AWS Batch Amazon Elastic Compute Cloud 上の Amazon Elastic Container Service のジョブ

```
{
    "jobDefinitionName": "multicontainer-ecs-ec2",
    "type": "container",
    "ecsProperties": {
        "taskProperties": [
          {
            "containers": [
              {
                "name": "c1",
                "essential": false,
```

```
                "command": [
                  "echo",
                  "hello world"
                ],
                "image": "public.ecr.aws/amazonlinux/amazonlinux:latest",
                "resourceRequirements": [
                  {
                    "type": "VCPU",
                    "value": "2"
                  },
                  {
                    "type": "MEMORY",
                    "value": "4096"
                  }
                ]
              },
              {
                "name": "c2",
                "essential": true,
                "command": [
                  "echo",
                  "hello world"
                ],
                "image": "public.ecr.aws/amazonlinux/amazonlinux:latest",
                "resourceRequirements": [
                  {
                    "type": "VCPU",
                    "value": "6"
                  },
                  {
                    "type": "MEMORY",
                    "value": "12288"
                  }
                ]
              }
            ]
          }
        ]
  }
}
```

AWS Batch での Amazon ECS の ジョブ AWS Fargate

```
{
    "jobDefinitionName": "multicontainer-ecs-fargate",
    "type": "container",
    "platformCapabilities": [
        "FARGATE"
    ],
    "ecsProperties": {
        "taskProperties": [
          {
            "containers": [
              {
                "name": "c1",
                "command": [
                  "echo",
                  "hello world"
                ],
                "image": "public.ecr.aws/amazonlinux/amazonlinux:latest",
                "resourceRequirements": [
                  {
                    "type": "VCPU",
                    "value": "2"
                  },
                  {
                    "type": "MEMORY",
                    "value": "4096"
                  }
                ]
              },
              {
                "name": "c2",
                "essential": true,
                "command": [
                  "echo",
                  "hello world"
                ],
                "image": "public.ecr.aws/amazonlinux/amazonlinux:latest",
                "resourceRequirements": [
                  {
                    "type": "VCPU",
                    "value": "6"
                  },
```

```
                {
                  "type": "MEMORY",
                  "value": "12288"
                }
              ]
            }
          ],
          "executionRoleArn": "arn:aws:iam::1112223333:role/ecsTaskExecutionRole"
        }
      ]
  }
}
```

AWS Batch Amazon Elastic Kubernetes Service の ジョブ

```
{
  "jobDefinitionName": "multicontainer-eks",
  "type": "container",
  "eksProperties": {
    "podProperties": {
      "shareProcessNamespace": true,
      "initContainers": [
        {
          "name": "init-container",
          "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
          "command": [
            "echo"
          ],
          "args": [
            "hello world"
          ],
          "resources": {
            "requests": {
              "cpu": "1",
              "memory": "512Mi"
            }
          }
        },
        {
          "name": "init-container-2",
          "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
          "command": [
            "echo",
```

```
            "my second init container"
          ],
          "resources": {
            "requests": {
              "cpu": "1",
              "memory": "512Mi"
            }
          }
        }
      ],
      "containers": [
        {
          "name": "c1",
          "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
          "command": [
            "echo world"
         ],
          "resources": {
            "requests": {
              "cpu": "1",
              "memory": "512Mi"
            }
          }
        },
        {
          "name": "sleep-container",
          "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
          "command": [
            "sleep",
            "20"
          ],
          "resources": {
            "requests": {
              "cpu": "1",
              "memory": "512Mi"
            }
          }
        }
      ]
    }
  }
}
```

ノードごとに複数のコンテナを持つマルチノード並列 (MNP) AWS Batch ジョブ

```
{
  "jobDefinitionName": "multicontainer-mnp",
  "type": "multinode",
  "nodeProperties": {
    "numNodes": 6,
    "mainNode": 0,
    "nodeRangeProperties": [
      {
        "targetNodes": "0:5",
        "ecsProperties": {
          "taskProperties": [
            {
              "containers": [
                {
                  "name": "range05-c1",
                  "command": [
                    "echo",
                    "hello world"
                  ],
                  "image": "public.ecr.aws/amazonlinux/amazonlinux:latest",
                  "resourceRequirements": [
                    {
                      "type": "VCPU",
                      "value": "2"
                    },
                    {
                      "type": "MEMORY",
                      "value": "4096"
                    }
                  ]
                },
                {
                  "name": "range05-c2",
                  "command": [
                    "echo",
                    "hello world"
                  ],
                  "image": "public.ecr.aws/amazonlinux/amazonlinux:latest",
                  "resourceRequirements": [
                    {
                      "type": "VCPU",
```

```
                    "value": "2"
                  },
                  {
                    "type": "MEMORY",
                    "value": "4096"
                  }
                ]
              }
            ]
          }
        ]
      }
    ]
  }
}
```

awslogs ログドライバーを使用する

デフォルトでは、AWS Batch は `awslogs` ログドライバが CloudWatch Logs にログ情報を送信することを可能にします。この機能により、コンテナからの異なるログを 1 か所で便利に表示できます。また、コンテナログがコンテナインスタンスのディスク容量を占めることも防止できます。このトピックでは、ジョブ定義で `awslogs` ログドライバーを設定する方法について説明します。

> **Note**
>
> AWS Batch コンソールでは、ジョブ定義を作成するときに ロギング設定 セクションで `awslogs` ログドライバーを設定できます。

> **Note**
>
> ジョブのコンテナによってログ記録される情報のタイプは、`ENTRYPOINT` コマンドによって大きく異なります。デフォルトでは、キャプチャされるログは、コンテナをローカルに実行した場合にインタラクティブターミナルに表示されるコマンド出力 (`STDOUT` および `STDERR` I/O ストリーム) を示します。`awslogs` ログドライバーは、これらのログを Docker から CloudWatch Logs に渡します。Docker ログの処理方法 (ファイルデータやストリームをキャプチャする別の方法) の詳細については、Docker ドキュメントのコンテナまたはサービスのログを表示するを参照してください。

コンテナインスタンスから CloudWatch Logs にシステムログを送信するには、での CloudWatch ログの使用 AWS Batchを参照してください。CloudWatch Logs の詳細については、Amazon CloudWatch Logs ユーザーガイドのログファイルのモニタリングおよびCloudWatch Logs クォータを参照してください。

使用できる awslogs ログドライバーのオプション

awslogs ログドライバーは、AWS Batch ジョブ定義で以下のオプションをサポートします。詳細については、Docker ドキュメントのCloudWatch Logs ロギングドライバーの設定を参照してください。

awslogs-region

必須: いいえ

awslogs ログドライバーが Docker ログを送信するリージョンを指定します。デフォルトでは、使用されるリージョンはジョブのリージョンと同じです。CloudWatch Logs では、異なるリージョンのジョブからすべてのログを 1 つのリージョンに送信するように選択できます。これにより、それらを 1 つの場所からすべて表示できます。または、より詳細なアプローチのために、リージョンごとにそれらを分離することもできます。ただし、このオプションを選択する場合は、指定したロググループが、指定したリージョンに存在することを確認してください。

awslogs-group

必須：オプション

この awslogs-group オプションを選択すると、awslogs ログドライバーがログストリームを送信するロググループを指定できます。これを指定しない場合、aws/batch/job が使用されます。

awslogs-stream-prefix

必須：オプション

awslogs-stream-prefix オプションを使用して、指定したプレフィックス、コンテナ名、コンテナの所属先における AWS Batch ジョブの Amazon ECS タスクの ID に、ログストリーミングを関連付けることができます。このオプションでプレフィックスを指定した場合、ログストリームの形式は以下のようになります。

```
prefix-name/default/ecs-task-id
```

`awslogs-datetime-format`

必須: いいえ

このオプションは、Python `strftime` 形式で複数行起動パターンを定義します。ログメッセージは、パターンに一致する 1 行と、それに続くパターンに一致しない行で構成されます。したがって、一致した行はログメッセージ間の区切り文字です。

この形式を使用する場合のユースケースの例としては、スタックダンプなどの解析された出力があり、これを使用しなければ、複数のエントリに記録されることになります。適切なパターンにより、単一のエントリにキャプチャさせます。

詳細については、awslogs-datetime-formatを参照してください。

`awslogs-datetime-format` と `awslogs-multiline-pattern` の両方が設定されている場合、このオプションは常に優先されます。

> **Note**
>
> 複数行のログ記録は、すべてのログメッセージの正規表現の解析とマッチングを実行します。これによりログ記録のパフォーマンスに悪影響が及ぶ可能性があります。

`awslogs-multiline-pattern`

必須: いいえ

このオプションでは、正規表現を使用して複数行起動パターンを定義します。ログメッセージは、パターンに一致する 1 行と、それに続くパターンに一致しない行で構成されます。したがって、一致した行はログメッセージ間の区切り文字です。

詳細については、Docker ドキュメントのawslogs-multiline-patternを参照してください。

`awslogs-datetime-format` も設定されている場合は、このオプションは無視されます。

> **Note**
>
> 複数行のログ記録は、すべてのログメッセージの正規表現の解析とマッチングを実行します。これによりログ記録のパフォーマンスに悪影響が及ぶ可能性があります。

`awslogs-create-group`

必須: いいえ

自動的に作成されたロググループが必要かどうかを指定します。このオプションを指定しない場合、デフォルトは `false` です。

> **Warning**
>
> このオプションは推奨されません。各ジョブがロググループの作成を試みるため、ジョブが失敗する可能性が高くなるため、CloudWatch Logs CreateLogGroup API アクションを使用して、事前にロググループを作成することをお勧めします。

> **Note**
>
> `logs:CreateLogGroup` を使用しようとする前に、実行ロールの IAM ポリシーには `awslogs-create-group` アクセス権限が含まれている必要があります。

ジョブ定義でログ設定を指定する

デフォルトでは、AWS Batch が `awslogs` ログドライバーを有効にします。ここでは、ジョブの `awslogs` ログ設定をカスタマイズする方法について説明します。詳細については、シングルノードのジョブ定義を作成する を参照してください。

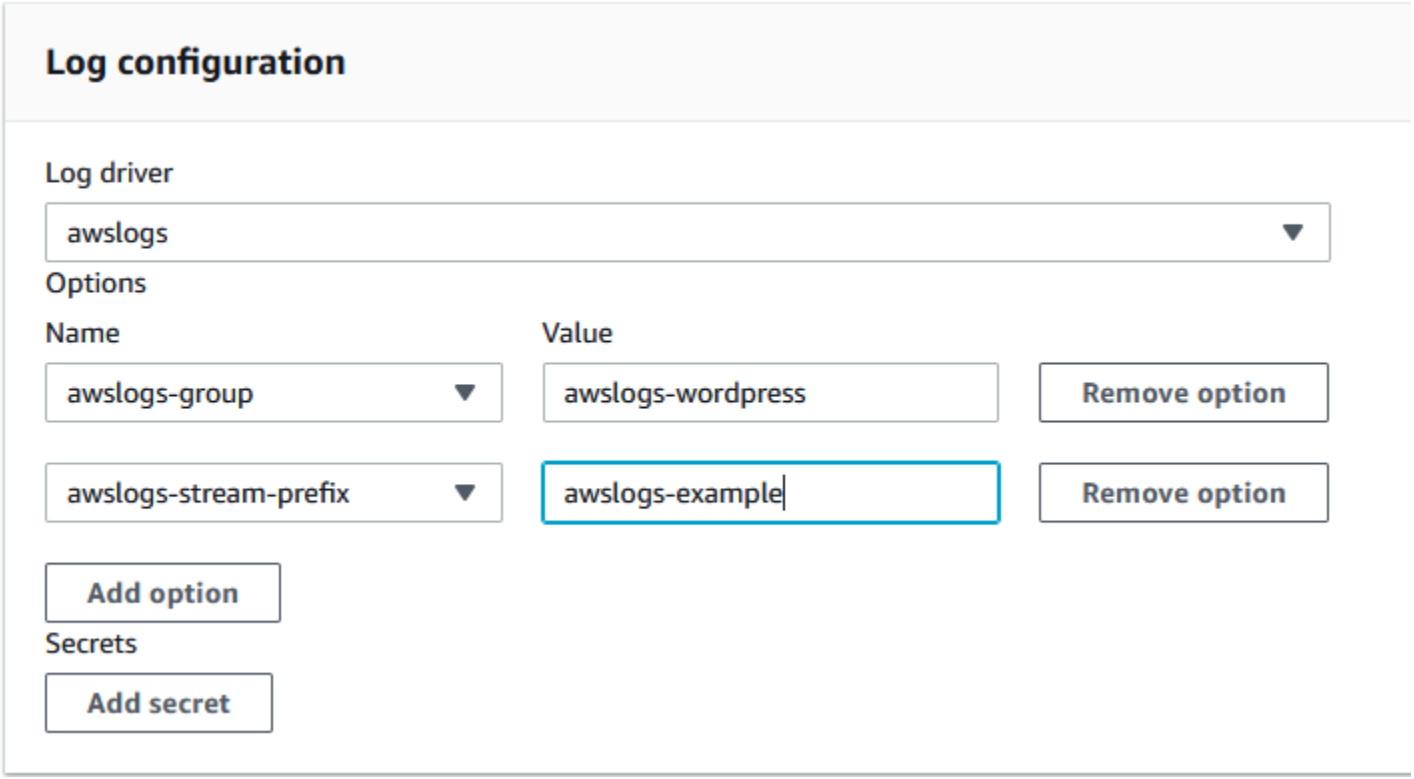
次のログ設定 JSON スニペットは、ジョブごとに `logConfiguration` オブジェクトが指定されています。1 つは `awslogs-wordpress` というロググループにログを送る WordPress のジョブで、もう 1 つは `awslogs-mysql` というロググループにログを送る MySQL コンテナのものです。どちらのコンテナも `awslogs-example` ログストリームプレフィックスを使用します。

```
"logConfiguration": {
    "logDriver": "awslogs",
    "options": {
        "awslogs-group": "awslogs-wordpress",
        "awslogs-stream-prefix": "awslogs-example"
    }
}
```

```
"logConfiguration": {
```

```
    "logDriver": "awslogs",
    "options": {
        "awslogs-group": "awslogs-mysql",
        "awslogs-stream-prefix": "awslogs-example"
    }
}
```

AWS Batch コンソールで、wordpress ジョブ定義のログ設定は次の図のように指定されています。



awslogs ログドライバーを使用するタスク定義をジョブ定義ログ設定に登録すると、ジョブ定義を使用してジョブを送信し、CloudWatch Logsへのログの送信をスタートできます。詳細については、ジョブの送信を参照してください。

機密データの指定

AWS Batchを使用すると、AWS Secrets Manager シークレットまたは AWS Systems Manager パラメータストアのパラメータに機密データを保存してジョブの定義でそれを参照することによって、ジョブに機密データを挿入できます。

以下の方法でシークレットをジョブに公開できます。

- 機密データを環境変数としてコンテナに挿入するには、secrets ジョブ定義パラメータを使用します。
- ジョブのログ設定内の機密情報を参照するには、secretOptions ジョブ定義パラメータを使用します。

トピック

Secrets Manager を使用した機密データの指定

を使用すると AWS Batch、機密データをシー AWS Secrets Manager クレットに保存し、ジョブ定義で参照することで、ジョブに機密データを挿入できます。Secrets Manager シークレットに保存された機密データは、環境変数として、またはログ設定の一部としてジョブに公開できます。

シークレットを環境変数として挿入する場合は、挿入するシークレットの JSON キーまたはバージョンを指定できます。このプロセスは、ジョブに公開される機密データの制御に役立ちます。シークレットのバージョニングの詳細については、AWS Secrets Manager ユーザーガイドの「AWS Secrets Managerの主な用語と概念」を参照してください。

Secrets Manager を使用した機密データの指定に関する考慮事項

Secrets Manager を使用してジョブの機密データを指定する場合は、以下を考慮する必要があります。

- シークレットの特定の JSON キーやバージョンを使用してシークレットを挿入するには、コンピューティング環境内のコンテナインスタンスに、 Amazon ECS コンテナエージェントのバージョン 1.37.0 以降が必要です。ただし、最新のコンテナエージェントのバージョンを使用することをお勧めします。エージェントのバージョンの確認と最新バージョンへの更新の詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS コンテナエージェントの更新を参照してください。

  シークレットの内容全体を環境変数として挿入したり、シークレットをログ設定に挿入したりするには、コンテナインスタンスにコンテナエージェントのバージョン 1.23.0 以降が必要です。
- CreateSecret API の SecretStringパラメータで作成されたシークレットであるテキストデータを格納するシークレットのみがサポートされます。CreateSecret API の SecretBinaryパラメー

タで作成されたシークレットであるバイナリデータを格納するシークレットはサポートされていません。

- Secrets Manager シークレットを参照するジョブ定義を使用してジョブの機密データを取得する場合、インターフェイス VPC エンドポイントも使用している場合は、Secrets Manager のインターフェイス VPC エンドポイントを作成する必要があります 。詳細については、AWS Secrets Manager ユーザーガイドの「VPC EndpointでSecrets Managerを使用する」を参照してください。
- 重要なデータは、ジョブが最初に開始されたときにジョブに挿入されます。シークレットを後で更新またはローテーションすると、ジョブには更新された値が自動的に送信されなくなります。更新されたシークレット値で新しいジョブを起動するようサービスに強制するには、新しいジョブの起動が必要です。

AWS Batch シークレットに必要な IAM アクセス許可

この機能を使用するには、実行ロールを持っていて、ジョブ定義でそのロールを参照する必要があります。これにより、コンテナエージェントは必要な Secrets Manager リソースをプルすることを許可されます。詳細については、「AWS Batch 実行 IAM ロール」を参照してください。

作成した Secrets Manager シークレットへのアクセスを許可するには、以下のアクセス許可をインラインポリシーとして実行ロールに手動で追加します。詳細については、IAM ユーザーガイドの「IAM ポリシーの追加と削除 」を参照してください。

- `secretsmanager:GetSecretValue` – Secrets Manager シークレットを参照する場合に必須です。
- `kms:Decrypt` – シークレットでカスタムの KMS キーを使用し、デフォルトのキーを使用しない場合にのみ必須です。そのカスタムキーの ARN はリソースとして追加されている必要があります。

次の例のインラインポリシーでは必須の アクセス許可を追加しています。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue",
        "kms:Decrypt"
```

```
      ],
      "Resource": [
        "arn:aws:secretsmanager:<region>:<aws_account_id>:secret:<secret_name>",
        "arn:aws:kms:<region>:<aws_account_id>:key/<key_id>"
      ]
    }
  ]
}
```

環境変数としての機密データの挿入

ジョブの定義内では、以下の項目を指定できます。

- ジョブに設定する環境変数の名前が含まれている secrets オブジェクト
- Secrets Manager シークレットの Amazon リソースネーム (ARN)。
- ジョブに渡す機密データが含まれている追加のパラメータ

次の例は、Secrets Manager シークレットに指定する必要がある完全な構文を示しています。

```
arn:aws:secretsmanager:region:aws_account_id:secret:secret-name:json-key:version-stage:version-id
```

次のセクションでは、追加のパラメータについて説明します。これらのパラメータは省略可能です。ただし、それらを使用しない場合は、デフォルト値を使用するためにコロン : が含まれている必要があります。以下の例でより詳細なコンテキストを示します。

`json-key`

キーと値のペアのキーの名前を指定します。値は設定する環境変数の値です。JSON 形式の値のみがサポートされます。JSON キーを指定しないと、シークレットの内容全体が使用されます。

`version-stage`

使用するシークレットのバージョンのステージングラベルを指定します。バージョンのステージングラベルを指定した場合、バージョン ID は指定できません。バージョンのステージを指定しないと、デフォルトの動作として、AWSCURRENT ステージングラベルのシークレットが取得されます。

ステージングラベルは、シークレットが更新またはローテーションされたときに、シークレットのさまざまなバージョンを追跡するために使用します。シークレットの各バージョンには、1

つ以上のステージングラベルと 1 つの ID があります。詳細については、「 ユーザーガイド」の AWS 「 Secrets Manager の主要な用語と概念」を参照してください。 AWS Secrets Manager

`version-id`

使用するシークレットのバージョンの固有 ID を指定します。バージョン ID を指定した場合、バージョンのステージングラベルは指定できません。バージョン ID を指定しないと、デフォルトの動作として、AWSCURRENT ステージングラベルのシークレットが取得されます。

バージョン ID は、シークレットが更新またはローテーションされたときに、シークレットのさまざまなバージョンを追跡するために使用します。シークレットの各バージョンには ID があります。詳細については、「 ユーザーガイド」の AWS 「 Secrets Manager の主要な用語と概念」を参照してください。 AWS Secrets Manager

コンテナの定義の例

以下の例では、コンテナの定義で Secrets Manager シークレットを参照する方法を示します。

Example シークレット全体を参照する

次に示すのは、Secrets Manager シークレットのテキスト全体を参照するときの形式を示すタスク定義のス二ペットです。

```
{
  "containerProperties": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-AbCdEf"
    }]
  }]
}
```

Example シークレット内の特定のキーを参照する

以下は、シークレットの内容と、シークレットに関連付けられているバージョンのステージングラベルとバージョン ID を表示するget-secret-valueコマンドからの出力例です。

```
{
    "ARN": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf",
    "Name": "appauthexample",
    "VersionId": "871d9eca-18aa-46a9-8785-981dd39ab30c",
```

```
    "SecretString": "{\"username1\":\"password1\",\"username2\":\"password2\",
\"username3\":\"password3\"}",
    "VersionStages": [
        "AWSCURRENT"
    ],
    "CreatedDate": 1581968848.921
}
```

前のコンテナの定義の出力で特定のキーを参照するには、ARN の最後にキー名を指定します。

```
{
  "containerProperties": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-
AbCdEf:username1::"
    }]
  }]
}
```

Example 特定のシークレットバージョンを参照する

次に示すのは、シークレットの暗号化されていない内容と、シークレットのすべてのバージョンのメタデータを表示する describe-secret コマンドの出力例です。

```
{
    "ARN": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf",
    "Name": "appauthexample",
    "Description": "Example of a secret containing application authorization data.",
    "RotationEnabled": false,
    "LastChangedDate": 1581968848.926,
    "LastAccessedDate": 1581897600.0,
    "Tags": [],
    "VersionIdsToStages": {
        "871d9eca-18aa-46a9-8785-981dd39ab30c": [
            "AWSCURRENT"
        ],
        "9d4cb84b-ad69-40c0-a0ab-cead36b967e8": [
            "AWSPREVIOUS"
        ]
    }
}
```

前のコンテナの定義の出力で特定のバージョンのステージングラベルを参照するには、ARN の最後にキー名を指定します。

```
{
  "containerProperties": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf::AWSPREVIOUS:"
    }]
  }]
}
```

前のコンテナの定義の出力で特定のバージョン ID を参照するには、ARN の最後にキー名を指定します。

```
{
  "containerProperties": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf::9d4cb84b-ad69-40c0-a0ab-cead36b967e8"
    }]
  }]
}
```

Example シークレットの特定のキーおよびバージョンのステージングラベルを参照する

シークレット内の特定のキーと特定のバージョンのステージングラベルの両方を参照する方法は次のとおりです。

```
{
  "containerProperties": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf:username1:AWSPREVIOUS:"
    }]
  }]
}
```

特定のキーとバージョン ID を指定するには、次の構文を使用します。

```
{
  "containerProperties": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-
AbCdEf:username1::9d4cb84b-ad69-40c0-a0ab-cead36b967e8"
    }]
  }]
}
```

ログ設定への機密データの挿入

ジョブの定義内で logConfiguration を指定するときに、コンテナに設定するログドライバーオプションの名前と、コンテナに提示する機密データが含まれている Secrets Manager シークレットの 完全なARNを使用して secretOptions を指定できます。

以下に示すのは、Secrets Manager シークレットを参照するときの形式を示すジョブ定義のスニペットです。

```
{
  "containerProperties": [{
    "logConfiguration": [{
      "logDriver": "splunk",
      "options": {
        "splunk-url": "https://cloud.splunk.com:8080"
      },
      "secretOptions": [{
        "name": "splunk-token",
        "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-
AbCdEf"
      }]
    }]
  }]
}
```

AWS Secrets Manager シークレットの作成

Secrets Manager コンソールを使用して、機密データ用のシークレットを作成できます。詳細については、AWS Secrets Manager ユーザーガイドの 「ベーシックシークレットの作成 を参照してください。

基本的なシークレットを作成するには

機密データのシークレットを作成するには、Secrets Manager を使用します。

1. https://console.aws.amazon.com/secretsmanager/ から Secrets Manager コンソール を開きます。
2. [Store a new secret] (新しいシークレットの保存) を選択します。
3. [Select secret type] (シークレットタイプの選択) で、[Other type of secrets] (他の種類のシークレット) を選択します。
4. カスタムシークレットの詳細を [Key] (キー) と [Value] (値) のペアの形式で指定します。たとえば、`UserName` のキーを指定し、その値として適切なユーザー名を指定できます。`Password` の名前とパスワードのテキストを値として持つ 2 番目のキーを追加します。データベース名、サーバーアドレス、TCP ポートなどのエントリを追加することもできます。必要な情報を格納するのに必要な数だけペアを追加できます。

   または、[Plaintext] (プレーンテキスト) タブを選択して、任意の方法でシークレット値を入力することもできます。
5. シークレット内の保護されたテキストの AWS KMS 暗号化に使用する暗号化キーを選択します。選択しない場合、Secrets Manager は、アカウントのデフォルトキーが存在するかどうかを調べ、存在する場合はそれを使用します。デフォルトのキーが存在しない場合は、Secrets Manager が自動的に作成します。また、[Add new key(新しいキーの追加)] を選択して、このシークレット専用のカスタム KMS キーを作成します。独自の KMS キーを作成するには、アカウントに KMS キーを作成するアクセス権限が必要です。
6. [Next] (次へ) を選択します。
7. [Secret name] (シークレット名) に、オプションのパスと名前 (例: **`production/MyAwesomeAppSecret`** または **`development/TestSecret`**) を入力し、[Next] (次) を選択します。オプションで説明を追加することもできます。後でこのシークレットを思い出すのに役立ちます。

   シークレット名に使用できるのは、ASCII 文字、数字、または次の記号のみです: /_+=.@-
8. (オプション) この時点で、シークレットのローテーションを設定することができます。この手順では、[Disable automatic rotation] (自動ローテーションを無効化) は無効のままにし、[次] を選択します。

   新規または既存のシークレットでローテーションを設定する方法については、「シークレットのローテーション」を参照してください。

9. 設定を確認し、次に [Store secret] (シークレットの保存) を選択して入力した内容すべてを Secrets Manager の新しいシークレットとして保存します。

Systems Manager Parameter Store を使用して機密データを指定する

を使用すると AWS Batch、 パラメータストアパラメータに機密データを保存してコンテナの定義でそれを参照することによって、コンテナに AWS Systems Manager 機密データを挿入できます。

トピック

- Systems Manager パラメータストアを使用した機密データの指定に関する考慮事項
- AWS Batch シークレットに必要な IAM アクセス許可
- 環境変数としての機密データの挿入
- ログ設定への機密データの挿入
- AWS Systems Manager Parameter Store パラメータの作成

Systems Manager パラメータストアを使用した機密データの指定に関する考慮事項

Systems Manager パラメータストアのパラメータを使用してコンテナの機密データを指定する場合は、以下を考慮する必要があります。

- この機能を使用するには、コンテナインスタンスにコンテナエージェントのバージョン 1.23.0 以降が必要です。ただし、最新のコンテナエージェントのバージョンを使用することをお勧めします。エージェントのバージョンの確認と最新バージョンへの更新の詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS コンテナエージェントの更新を参照してください。
- 機密データは、コンテナが最初に開始されたときにジョブのコンテナに挿入されます。シークレットまたは Parameter Store パラメータが後で更新またはローテーションされたときに、コンテナは更新された値を自動的に受け取りません。更新されたシークレットを使用して新しいジョブを強制的に起動するには、新しいジョブを起動する必要があります。

AWS Batch シークレットに必要な IAM アクセス許可

この機能を使用するには、実行ロールを持っていて、ジョブ定義でそのロールを参照する必要があります。これにより、Amazon ECS コンテナエージェントは必要な AWS Systems Manager リソースを取得できます。詳細については、「AWS Batch 実行 IAM ロール」を参照してください。

作成した AWS Systems Manager Parameter Store パラメータへのアクセスを許可するには、以下のアクセス許可をインラインポリシーとして実行ロールに手動で追加します。詳細については、IAM ユーザーガイドのIAM ポリシーの追加と削除を参照してください。

- `ssm:GetParameters`— Systems Managerパラメータストアのパラメータをタスク定義で参照する場合は必須です。
- `secretsmanager:GetSecretValue` — Secrets Manager シークレットを直接参照するか、Systems Manager パラメータストアのパラメータがタスク定義で Secrets Manager シークレットを参照している場合は必須です。
- `kms:Decrypt` – シークレットでカスタムの KMS キーを使用し、デフォルトのキーを使用しない場合にのみ必須です。そのカスタムキーの ARN はリソースとして追加されている必要があります。

次の例のインラインポリシーでは必須許可を追加しています。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ssm:GetParameters",
        "secretsmanager:GetSecretValue",
        "kms:Decrypt"
      ],
      "Resource": [
        "arn:aws:ssm:<region>:<aws_account_id>:parameter/<parameter_name>",
        "arn:aws:secretsmanager:<region>:<aws_account_id>:secret:<secret_name>",
        "arn:aws:kms:<region>:<aws_account_id>:key/<key_id>"
      ]
    }
  ]
}
```

環境変数としての機密データの挿入

コンテナの定義内で、コンテナに設定する環境変数の名前と、コンテナに渡す機密データが含まれている Systems Manager パラメータストアのパラメータの ARN 全体を使用して `secrets` を指定できます。

以下に示すのは、Systems Manager パラメータストアのパラメータを参照するときの形式を示すタスク定義のスニペットです。起動するタスクと同じリージョンに、Systems Manager パラメータストアのパラメータが存在する場合は、パラメータの完全な ARN または名前のどちらも使用できます。パラメータが別のリージョンに存在する場合は、完全な ARN を指定する必要があります。

```
{
  "containerProperties": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter/parameter_name"
    }]
  }]
}
```

ログ設定への機密データの挿入

コンテナの定義内で `logConfiguration` を指定するときに、コンテナに設定するログドライバーオプションの名前と、コンテナに渡す機密データが含まれている Systems Manager パラメータストアのパラメータの ARN 全体を使用して `secretOptions` を指定できます。

> **Important**
>
> 起動するタスクと同じリージョンに、Systems Manager パラメータストアのパラメータが存在する場合は、パラメータの完全な ARN または名前のどちらも使用できます。パラメータが別のリージョンに存在する場合は、完全な ARN を指定する必要があります。

以下に示すのは、Systems Manager パラメータストアのパラメータを参照するときの形式を示すタスク定義のスニペットです。

```
{
  "containerProperties": [{
    "logConfiguration": [{
      "logDriver": "fluentd",
      "options": {
        "tag": "fluentd demo"
      },
      "secretOptions": [{
        "name": "fluentd-address",
        "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter/parameter_name"
```

```
      }]
    }]
  }]
}
```

AWS Systems Manager Parameter Store パラメータの作成

AWS Systems Manager コンソールを使用して、機密データ用の Systems Manager パラメータストアパラメータを作成できます。詳細については、AWS Systems Manager ユーザーガイドのコマンドでパラメータを作成して使用する (コンソール)を参照してください。

パラメータストア パラメータを作成するには

1. https://console.aws.amazon.com/systems-manager/ で AWS Systems Manager コンソールを開きます。
2. ナビゲーションペインで パラメータストア、パラメータの作成 の順に選択します。
3. 名前 に、階層とパラメータ名を入力します。たとえば、`test/database_password` と入力します。
4. Description] (説明) に説明を入力します (省略可能)。
5. タイプ で、文字列 、StringList、または を選択しますSecureString。

> **Note**
>
> - を選択するとSecureString、KMS キー ID フィールドが表示されます。KMS キー ID、KMS キー ARN、エイリアス名、またはエイリアス ARN が指定されていない場合、システムは `alias/aws/ssm` を使用します。これは、Systems Manager 用のデフォルトの KMS キーです。このキーを使用しない場合は、カスタムキーを選択します。詳細については、AWS Systems Manager ユーザーガイドのSecure String パラメータを使用するを参照してください。
> - コンソールで `key-id` パラメータにカスタム KMS キーエイリアス名またはエイリアス ARN のいずれかを指定して Secure String パラメータを作成する場合は、そのエイリアスの前にプレフィックス `alias/` を指定する必要があります。ARN の例を次に示します。
>
>   ```
>   arn:aws:kms:us-east-2:123456789012:alias/MyAliasName
>   ```
>
>   エイリアス名の例を次に示します。

```
alias/MyAliasName
```

6. Value] (値) に値を入力します。例えば、`MyFirstParameter` です。を選択した場合SecureString、値は入力したとおりにマスクされます。
7. パラメータの作成 を選択します。

ジョブのプライベートレジストリ認証

を使用したジョブのプライベートレジストリ認証 AWS Secrets Manager では、認証情報を安全に保存し、ジョブ定義で参照できます。これにより、ジョブ定義で認証 AWS が必要な の外部のプライベートレジストリに存在するコンテナイメージを参照できます。この機能は、Amazon EC2 インスタンスおよび Fargate でホストされているジョブでサポートされています。

> **Important**
>
> ジョブ定義で Amazon ECR に保存されているイメージを参照する場合、このトピックは適用されません。詳細については、Amazon Elastic Container Registry ユーザーガイドの「Amazon ECS で Amazon ECR イメージを使用する」を参照してください。

Amazon EC2 インスタンスでホストされているジョブの場合、この機能にはバージョン `1.19.0`以降の コンテナエージェントが必要です。ただし、最新のコンテナエージェントのバージョンを使用することをお勧めします。エージェントのバージョンを確認して最新バージョンに更新する方法については、「Amazon Elastic Container Service デベロッパーガイド」の「Amazon ECS コンテナエージェントの更新」を参照してください。

Fargate でホストされているジョブの場合、この機能にはプラットフォームバージョン `1.2.0` 以降が必要です。詳細については、「Amazon Elastic Container Service デベロッパーガイド」のAWS「Fargate Linux プラットフォームのバージョン」を参照してください。

コンテナの定義内で、作成したシークレットの詳細 で `repositoryCredentials` オブジェクトを指定します。参照するシークレットは、それを使用するジョブとは異なる AWS リージョン アカウントからでも、異なるアカウントからでもかまいません。

> **Note**
>
> AWS Batch API、 AWS CLI、または AWS SDK を使用する場合、起動するジョブ AWS リージョン と同じ にシークレットが存在する場合は、シークレットの完全な ARN または名前のどちらも使用できます。シークレットが別のアカウントに存在する場合は、シークレットの完全な ARN を指定する必要があります。を使用する場合 AWS Management Console、シークレットの完全な ARN を常に指定する必要があります。

以下は、必要なパラメータを示すジョブ定義のスニペットです。

```
"containerProperties": [
  {
    "image": "private-repo/private-image",
    "repositoryCredentials": {
      "credentialsParameter":
 "arn:aws:secretsmanager:region:123456789012:secret:secret_name"
    }
  }
]
```

プライベートレジストリの認証で必須の IAM アクセス許可

この機能を使用するには、実行ロールが必要です。このロールを使用して、コンテナエージェントでコンテナイメージをプルできます。詳細については、「AWS Batch 実行 IAM ロール」を参照してください。

作成したシークレットへのアクセスを提供するには、次のアクセス許可を実行ロールにインラインポリシーとして追加します。詳細については、「IAM ポリシーの追加と削除」を参照してください。

- `secretsmanager:GetSecretValue`
- `kms:Decrypt` - カスタムの KMS キーを使用するが、デフォルトのキーは使用しない場合にのみ必須。カスタムキーの Amazon リソースネーム (ARN) は、リソースとして追加する必要があります。

次の例では、インラインポリシーによりアクセス許可を追加しています。

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "secretsmanager:GetSecretValue"
      ],
      "Resource": [
        "arn:aws:secretsmanager:region:123456789012:secret:secret_name",
        "arn:aws:kms:region:123456789012:key/key_id"
      ]
    }
  ]
}
```

プライベートレジストリの認証の使用

基本的なシークレットを作成するには

AWS Secrets Manager を使用して、プライベートレジストリ認証情報のシークレットを作成します。

1. https://console.aws.amazon.com/secretsmanager/ で AWS Secrets Manager コンソールを開きます。
2. [Store a new secret] (新しいシークレットの保存) を選択します。
3. [Select secret type] (シークレットタイプの選択) で、[Other type of secrets] (他の種類のシークレット) を選択します。
4. [プレーンテキスト] を選択し、次の形式でプライベートレジストリ認証情報を入力します。

   ```
   {
     "username" : "privateRegistryUsername",
     "password" : "privateRegistryPassword"
   }
   ```

5. [次へ] を選択します。
6. [Secret name] (シークレット名) に、オプションのパスと名前 (例: **production/MyAwesomeAppSecret** または **development/TestSecret**) を入力し、[Next] (次へ) を選択します。オプションで説明を追加することもできます。後でこのシークレットを思い出すのに役立ちます。

シークレット名に使用できるのは、ASCII 文字、数字、または以下の記号のみです: /_+=.@-。

7. (オプション) この時点で、シークレットのローテーションを設定することができます。この手順では、[Disable automatic rotation] (自動ローテーションを無効化) は無効のままにし、[次] を選択します。

   新規または既存のシークレットでローテーションを設定する方法については、「シークレットのローテーションAWS Secrets Manager」を参照してください。

8. 設定を確認した上で、[Store secret] (シークレットの保存) を選択し、入力した全内容を Secrets Manager の新しいシークレットとして保存します。

ジョブ定義を登録し、プライベートレジストリ でプライベートレジストリ認証 を有効にします。その後、[Secrets Manager ARN または名前] で、シークレットの Amazon リソースネーム (ARN) を入力します。詳細については、「プライベートレジストリの認証で必須の IAM アクセス許可」を参照してください。

Amazon EFS ボリューム

Amazon Elastic File System (Amazon EFS) では、AWS Batch ジョブで使用するためのシンプルでスケーラブルなファイルストレージを提供します。Amazon EFSでは、ストレージ容量は伸縮性があります。ファイルの追加や削除時に、自動的にスケールされます。アプリケーションでは、必要なときに必要なストレージを確保できます。

Amazon EFS ファイルシステムを AWS Batch で使用して、コンテナインスタンスのフリート全体のファイルシステムデータをエクスポートできます。これにより、ジョブは、同じ永続的ストレージにアクセスできます。ただし、Docker デーモンが起動する前に、Amazon EFS ファイルシステムをマウントするように、コンテナインスタンス AMI を設定する必要があります。また、ファイルシステムを使用するには、ジョブ定義でコンテナインスタンスのボリュームマウントを参照する必要があります。以下のセクションは、AWS Batch でAmazon EFSの使用を開始するのに役立ちます。

Amazon EFS ボリュームに関する考慮事項

Amazon EFS ボリュームを使用する際には、以下の点を考慮する必要があります:

- EC2 リソースを使用するジョブの場合、Amazon ECS に最適化された AMI バージョン 20191212、コンテナエージェントバージョン 1.35.0 で、Amazon EFS ファイルシステムのサポートがパブリックプレビューとして追加されました。ただし、Amazon ECS 最適化 AMI バー

ジョン 20200319、コンテナエージェントバージョン 1.38.0 で、Amazon EFS ファイルシステムのサポート (Amazon EFS アクセスポイントと IAM 認証機能を含む) が一般提供されました。これらの機能を利用するには、Amazon ECS に最適化された AMI バージョン 20200319 以降を使用することをお勧めします。詳細については、Amazon Elastic Container Service デベロッパーガイドの「Amazon ECS に最適化された AMI バージョン」を参照してください。

> **Note**
>
> 独自の AMI を作成する場合、コンテナエージェント 1.38.0 以降、`ecs-init` バージョン 1.38.0-1 以降を使用し、Amazon EC2 インスタンスで以下のコマンドを実行する必要があります。これが Amazon ECS ボリュームプラグインを有効にするためのものです。コマンドは、ベースイメージとして Amazon Linux 2 と Amazon Linux のどちらを使用しているかによって異なります。
>
> Amazon Linux 2
>
> ```
> $ yum install amazon-efs-utils
> systemctl enable --now amazon-ecs-volume-plugin
> ```
>
> Amazon Linux
>
> ```
> $ yum install amazon-efs-utils
> sudo shutdown -r now
> ```

- Fargate リソースを使用するジョブの場合、プラットフォームバージョン 1.4.0 以降で Amazon EFS ファイルシステムのサポートが追加されました。詳細については、Amazon Elastic Container Service デベロッパーガイドの「AWS Fargate プラットフォームのバージョン」を参照してください。
- Fargate リソースで使用されるジョブに Amazon EFS ボリュームを指定する場合、Fargate は Amazon EFS ボリュームの管理を担当するスーパーバイザーコンテナを作成します。スーパーバイザーコンテナは、ジョブのメモリを少しだけ使用します。スーパーバイザーコンテナは、タスクメタデータバージョン 4 エンドポイントにクエリを実行するときに表示されます。詳細については、Amazon Elastic Container Service ユーザーガイドAWS Fargate用のタスクメタデータエンドポイントバージョン 4を参照してください。

Amazon EFS アクセスポイントを使用する

Amazon EFS アクセスポイントは、EFS ファイルシステムへのアプリケーション固有のエントリポイントです。これにより、共有データセットへのアプリケーションアクセスが管理しやすくなります。Amazon EFS アクセスポイントの詳細およびアクセス制御方法については、Amazon Elastic File System ユーザーガイドの「Amazon EFS アクセスポイントの使用」を参照してください。

アクセスポイントを使用すると、アクセスポイントを介したすべてのファイルシステム要求に対してユーザーアイデンティティ (ユーザーの POSIX グループなど) を適用できます。また、ファイルシステムに対して別のルートディレクトリを適用し、このディレクトリまたはそのサブディレクトリ内のデータに対してのみ、クライアントにアクセスを許可することもできます。

> **Note**
>
> EFS アクセスポイントを作成するときは、ルートディレクトリとして機能するファイルシステム上のパスを指定します。AWS Batch ジョブ定義でアクセスポイント ID を持つ EFS ファイルシステムを参照する場合、ルートディレクトリを省略するか、EFS アクセスポイントに設定されたパスを強制する / に設定する必要があります。

AWS Batch ジョブの IAM ロールを使用して、特定のアプリケーションで使用するアクセスポイントを限定できます。IAM ポリシーとアクセスポイントを組み合わせると、アプリケーションから特定のデータセットへのアクセスを簡単に保護できます。この機能では、タスク用の Amazon ECS IAM ロールを使用します。詳細については、Amazon Elastic Container Service デベロッパーガイドの「タスク用の IAM ロール」を参照してください。

ジョブ定義でのAmazon EFS ファイルシステムの指定

コンテナに Amazon EFS ファイルシステムボリュームを使用するには、ジョブ定義でボリュームとマウントポイントの設定を指定する必要があります。次のジョブ定義の JSON スニペットは、コンテナの [volumes] と [mountPoints] オブジェクトの構文を示します。

```
{
    "containerProperties": [
        {
            "image": "amazonlinux:2",
            "command": [
                "ls",
                "-la",
                "/mount/efs"
```

```
            ],
            "mountPoints": [
                {
                    "sourceVolume": "myEfsVolume",
                    "containerPath": "/mount/efs",
                    "readOnly": true
                }
            ],
            "volumes": [
                {
                    "name": "myEfsVolume",
                    "efsVolumeConfiguration": {
                        "fileSystemId": "fs-12345678",
                        "rootDirectory": "/path/to/my/data",
                        "transitEncryption": "ENABLED",
                        "transitEncryptionPort": integer,
                        "authorizationConfig": {
                            "accessPointId": "fsap-1234567890abcdef1",
                            "iam": "ENABLED"
                        }
                    }
                }
            ]
        }
    ]
}
```

`efsVolumeConfiguration`

型: オブジェクト

必須: いいえ

このパラメータは、Amazon EFS ボリュームを使用する場合に指定します。

`fileSystemId`

型: 文字列

必須: はい

使用する Amazon EFS ファイルシステムの ID。

`rootDirectory`

型: 文字列

必須: いいえ

ホスト内にルートディレクトリとしてマウントする Amazon EFS ファイルシステム内のディレクトリ。このパラメータを省略すると、Amazon EFS ボリュームのルートが使用されます。/ を指定すると、このパラメータを省略した場合と同じ結果になります。最大 4,096 文字を使用できます。

> **Important**
>
> [authorizationConfig] に EFS アクセスポイントを指定する場合は、ルートディレクトリパラメータを省略するか、または [/] に設定する必要があります。これにより、EFS アクセスポイントに設定されたパスが強制されます。

`transitEncryption`

型: 文字列

有効な値: `ENABLED` | `DISABLED`

必須: いいえ

AWS Batch ホストと Amazon EFS サーバー間で Amazon EFS データの転送中の暗号化を有効にするかどうかをします。Amazon EFS IAM 認証を使用する場合は、転送中の暗号化を有効にする必要があります。このパラメータを省略すると、[DISABLED] のデフォルト値が使用されます。詳細については、Amazon Elastic File System ユーザーガイドの「トランジット中のデータの暗号化」を参照してください。

`transitEncryptionPort`

タイプ: 整数

必須: いいえ

AWS Batch ホストと Amazon EFS サーバーの間で暗号化されたデータを送信するときに使用するポート。転送中の暗号化ポートを指定しないと、Amazon EFS マウントヘルパーが使用するポート選択方式が使用されます。この値は 0 ~ 65,535 の範囲の値にする必要があります。詳細については、[Amazon Elastic File System User Guide] (Amazon Elastic File System ユーザーガイド) の[EFS Mount Helper] (EFS マウントヘルパー) を参照してください。

`authorizationConfig`

タイプ: オブジェクト

必須: いいえ

Amazon EFS ファイルシステムに対する認可構成の詳細。

`accessPointId`

型: 文字列

必須: いいえ

使用するアクセスポイント ID。アクセスポイントを指定する場合は、[efsVolumeConfiguration] のルートディレクトリ値を省略するか、これを [/] に設定する必要があります。これにより、EFS アクセスポイントに設定されたパスが強制されます。アクセスポイントを使用する場合は、EFSVolumeConfiguration で転送中の暗号化を有効にする必要があります。詳細については、Amazon Elastic ファイルシステムユーザーガイドの「Amazon EFS アクセスポイントの使用」を参照してください。

`iam`

型: 文字列

有効な値: ENABLED | DISABLED

必須: いいえ

Amazon EFS ファイルシステムのマウント時にジョブ定義で定義した AWS Batch ジョブの IAM ロールを使用するかどうかを決定します。使用する場合は、[EFSVolumeConfiguration] で転送中の暗号化を有効にする必要があります。このパラメータを省略すると、[DISABLED] のデフォルト値が使用されます。実行 IAM ロールの詳細については、「AWS Batch 実行 IAM ロール」を参照してください。

ジョブ定義の例

次のジョブ定義の例では、環境変数、パラメータ置換やボリュームのマウントなどの一般的なパターンを使用する方法を示しています。

環境変数を使用します。

次のジョブ定義の例では、環境変数を使用してファイルタイプと Amazon S3 URL を指定します。この用例は、「簡単な "Fetch & Run" AWS Batch ジョブを作成する」コンピュータブログポストから引用しています。このブログポストで説明される `fetch_and_run.sh` スクリプトでは、S3から

myjob.sh スクリプトをダウンロードし、そのファイルタイプを指定するための環境変数を使用します。

この例では、コマンドと環境変数はジョブ定義でハードコード化されていますが、さらに柔軟性のあるジョブ定義を作成するためにコマンドと環境変数を指定することもできます。

```
{
    "jobDefinitionName": "fetch_and_run",
    "type": "container",
    "containerProperties": {
        "image": "123456789012.dkr.ecr.us-east-1.amazonaws.com/fetch_and_run",
        "resourceRequirements": [
            {
                "type": "MEMORY",
                "value": "2000"
            },
            {
                "type": "VCPU",
                "value": "2"
            }
        ],
        "command": [
            "myjob.sh",
            "60"
        ],
        "jobRoleArn": "arn:aws:iam::123456789012:role/AWSBatchS3ReadOnly",
        "environment": [
            {
                "name": "BATCH_FILE_S3_URL",
                "value": "s3://my-batch-scripts/myjob.sh"
            },
            {
                "name": "BATCH_FILE_TYPE",
                "value": "script"
            }
        ],
        "user": "nobody"
    }
}
```

パラメータ置換の使用

次の例では、パラメータ置換とデフォルト値を設定するためのジョブ定義を説明しています。

Ref:: セクションの command 宣言は、パラメータ置換のためにプレースホルダ–を設定するときに使用します。このジョブ定義でジョブを送信する場合、inputfileやoutputfileのような値に上書きしてパラメータを指定します。parameters セクションは codec のためのデフォルト設定ですが、必要に応じてパラメータを上書きできます。

詳細については、「パラメータ」を参照してください。

```
{
    "jobDefinitionName": "ffmpeg_parameters",
    "type": "container",
    "parameters": {"codec": "mp4"},
    "containerProperties": {
        "image": "my_repo/ffmpeg",
        "resourceRequirements": [
            {
                "type": "MEMORY",
                "value": "2000"
            },
            {
                "type": "VCPU",
                "value": "2"
            }
        ],
        "command": [
            "ffmpeg",
            "-i",
            "Ref::inputfile",
            "-c",
            "Ref::codec",
            "-o",
            "Ref::outputfile"
        ],
        "jobRoleArn": "arn:aws:iam::123456789012:role/ECSTask-S3FullAccess",
        "user": "nobody"
    }
}
```

GPU 機能のテスト

次のジョブ定義の例では、GPU ワークロードの AMI の使用 で説明されている GPU ワークロード AMI が適切に設定されているかどうかをテストします。このジョブ定義の例では、GitHub から TensorFlow ディープ MNIST 分類子の例を実行します。

```
{
    "containerProperties": {
        "image": "tensorflow/tensorflow:1.8.0-devel-gpu",
        "resourceRequirements": [
            {
                "type": "MEMORY",
                "value": "32000"
            },
            {
                "type": "VCPU",
                "value": "8"
            }
        ],
        "command": [
            "sh",
            "-c",
            "cd /tensorflow/tensorflow/examples/tutorials/mnist; python mnist_deep.py"
        ]
    },
    "type": "container",
    "jobDefinitionName": "tensorflow_mnist_deep"
}
```

上の JSON テキストで [tensorflow_mnist_deep.json] という名前のファイルを作成し、次のコマンドを使用して AWS Batch ジョブ定義を登録できます。

```
aws batch register-job-definition --cli-input-json file://tensorflow_mnist_deep.json
```

マルチノードの並列ジョブ

以下のジョブ定義の例では、マルチノードの並列ジョブを示しています。詳細については、AWS Compute ブログの「AWS Batch のマルチノード並列ジョブによる緊密に結合した分子動力学ワークフローの構築」を参照してください。

```
{
  "jobDefinitionName": "gromacs-jobdef",
  "jobDefinitionArn": "arn:aws:batch:us-east-2:123456789012:job-definition/gromacs-
jobdef:1",
  "revision": 6,
  "status": "ACTIVE",
  "type": "multinode",
```

```
  "parameters": {},
  "nodeProperties": {
    "numNodes": 2,
    "mainNode": 0,
    "nodeRangeProperties": [
      {
        "targetNodes": "0:1",
        "container": {
          "image": "123456789012.dkr.ecr.us-east-2.amazonaws.com/gromacs_mpi:latest",
          "resourceRequirements": [
              {
                  "type": "MEMORY",
                  "value": "24000"
              },
              {
                  "type": "VCPU",
                  "value": "8"
              }
          ],
          "command": [],
          "jobRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
          "ulimits": [],
          "instanceType": "p3.2xlarge"
        }
      }
    ]
  }
}
```

ジョブキュー

ジョブはジョブキューに送信され、コンピューティング環境で実行するようにスケジュールされるまで、ジョブキューに留まります。 AWS アカウントには複数のジョブキューを含めることができます。たとえば、優先度の高いジョブには Amazon EC2 オンデマンドインスタンスを使用するキューを作成し、優先度の低いジョブには Amazon EC2 スポットインスタンスを使用する別のキューを作成できます。ジョブキューには優先順位があり、この順位に基づいてスケジューラはどのキューのどのジョブを最初に実行かを判断します。

トピック

ジョブキューの作成

AWS Batch でジョブを送信する前に、ジョブキューを作成する必要があります。ジョブキューの作成する場合、キューに 1 つ以上のコンピューティング環境を関連付け、そして各コンピューティング環境に優先順位を割り当てます。

また、ジョブキューに優先順位を設定し、それによりAWS Batch スケジューラーがジョブを配置する順序を決定します。つまり、コンピューティング環境が 1 つ以上のジョブキューに関連付けられている場合、優先度の高いジョブキューから先にスケジュールされます。

Fargate ジョブキューの作成

ジョブキューを作成するには

1. https://console.aws.amazon.com/batch/ で AWS Batch コンソールを開きます。
2. ナビゲーションバーから、使用する AWS リージョン を選択します。
3. ナビゲーションペインで キューを選択します。
4. 作成] を選択します。
5. オーケストレーションタイプ] には Fargate を選択します。

6. キューの名前 に、一意のジョブキュー名を入力します。ジョブ名には、大文字、小文字の英文字、数字、ハイフン、アンダースコア(_)を含めることができ、最大 128 文字まで使用可能です。
7. 優先度 には、ジョブキューの優先度を示す整数値を入力します。同じコンピューティング環境環境に関連付けられているジョブキュー間では、優先度が高いジョブキューほど先に処理されます。優先度は降順に決定されます。たとえば、優先度の値が 10 のジョブキューは、優先度の値が 1 のジョブキューより先にスケジュールされます。
8. (オプション) スケジューリングポリシーの Amazon リソースネーム (ARN) では、既存のスケジューリングポリシーを選択します。
9. 接続されているコンピューティング環境では、リストから 1 つ以上のコンピューティング環境を選択して、ジョブキューに関連付けます。ユーザーは、ジョブキューの配置を試行させたいと考える順序で、コンピューティング環境環境を選択します。ジョブスケジューラでは、どのコンピューティング環境で特定のジョブを開始するかを、コンピューティング環境の順番で決めます。コンピューティング環境は、ジョブキューに関連付ける前に、VALIDの状態になっていることが必要です。1 つのジョブキューには、最大3つのコンピューティング環境を関連付けることができます。

   > **Note**
   >
   > ジョブキューに関連付けられているすべてのコンピューティング環境で、同じプロビジョニングモデルを共有する必要があります。AWS Batch は、単一のジョブキューでのプロビジョニングモデルの混在をサポートしていません。

10. コンピューティング環境環境の順序 では、上矢印と下矢印を選択して希望する順序を設定します。
11. 作成 を選択して終了し、ジョブキューを作成します。

Amazon EC2 ジョブキューの作成

Amazon EC2 ジョブキューを作成するには

1. https://console.aws.amazon.com/batch/ で AWS Batch コンソールを開きます。
2. ナビゲーションバーから、使用する AWS リージョン を選択します。
3. ナビゲーションペインで キュー を選択します。
4. 作成 を選択します。

5. オーケストレーションタイプ には、Amazon Elastic Compute Cloud (Amazon EC2) を選択します。
6. キューの名前 に、一意のジョブキュー名を入力します。ジョブ名には、大文字、小文字の英文字、数字、ハイフン、アンダースコア(_)を含めることができ、最大 128 文字まで使用可能です。
7. 優先度 には、ジョブキューの優先度を示す整数値を入力します。同じコンピューティング環境環境に関連付けられているジョブキュー間では、優先度が高いジョブキューほど先に処理されます。優先度は降順に決定されます。たとえば、優先度の値が 10 のジョブキューは、優先度の値が 1 のジョブキューより先にスケジュールされます。
8. (オプション) スケジューリングポリシーの Amazon リソースネーム (ARN) では、既存のスケジューリングポリシーを選択します。
9. 接続されているコンピューティング環境では、リストから 1 つ以上のコンピューティング環境を選択して、ジョブキューに関連付けます。ユーザーは、ジョブキューの配置を試行させたいと考える順序で、コンピューティング環境環境を選択します。ジョブスケジューラーでは、どのコンピューティング環境で特定のジョブを開始するかを、コンピューティング環境の順番で決めます。コンピューティング環境は、ジョブキューに関連付ける前に、VALIDの状態になっていることが必要です。1 つのジョブキューには、最大 3 つのコンピューティング環境を関連付けることができます。既存のコンピューティング環境環境がない場合は、コンピュート環境の作成 を選択します。

   > **Note**
   >
   > ジョブキューに関連付けられているすべてのコンピューティング環境で、同じプロビジョニングモデルを共有する必要があります。AWS Batch は、単一のジョブキューでのプロビジョニングモデルの混在をサポートしていません。

10. コンピューティング環境環境の順序 では、上矢印と下矢印を選択して希望する順序を設定します。
11. 作成 を選択して終了し、ジョブキューを作成します。

Amazon EKSジョブキューの作成

Amazon EKSジョブキューを作成するには

1. https://console.aws.amazon.com/batch/ で AWS Batch コンソールを開きます。
2. ナビゲーションバーから、使用する AWS リージョン を選択します。

3. ナビゲーションペインで キューを選択します。
4. 作成 を選択します。
5. オーケストレーションタイプ では、Amazon Elastic Kubernetes Service (Amazon EKS)] を選択します。
6. キューの名前 に、一意のジョブキュー名を入力します。ジョブ名には、大文字、小文字の英文字、数字、ハイフン、アンダースコア(_)を含めることができ、最大 128 文字まで使用可能です。
7. 優先度 には、ジョブキューの優先度を示す整数値を入力します。同じコンピューティング環境環境に関連付けられているジョブキュー間では、優先度が高いジョブキューほど先に処理されます。優先度は降順に決定されます。たとえば、優先度の値が 10 のジョブキューは、優先度の値が 1 のジョブキューより先にスケジュールされます。
8. (オプション) スケジューリングポリシーの Amazon リソースネーム (ARN) では、既存のスケジューリングポリシーを選択します。
9. 接続されているコンピューティング環境では、リストから 1 つ以上のコンピューティング環境を選択して、ジョブキューに関連付けます。ユーザーは、ジョブキューの配置を試行させたいと考える順序で、コンピューティング環境環境を選択します。ジョブスケジューラーでは、どのコンピューティング環境で特定のジョブを開始するかを、コンピューティング環境の順番で決めます。コンピューティング環境は、ジョブキューに関連付ける前に、VALIDの状態になっていることが必要です。1 つのジョブキューには、最大3つのコンピューティング環境を関連付けることができます。

   > **Note**
   >
   > ジョブキューに関連付けられているすべてのコンピューティング環境で、同じプロビジョニングモデルを共有する必要があります。AWS Batch は、単一のジョブキューでのプロビジョニングモデルの混在をサポートしていません。

   > **Note**
   >
   > ジョブキューに関連付けられているすべてのコンピューティング環境で、同じアーキテクチャを共有する必要があります。AWS Batchでは、単一のジョブキューでのコンピューティング環境アーキテクチャタイプの混在をサポートしていません。

10. コンピューティング環境環境の順序 では、上矢印と下矢印を選択して希望する順序を設定します。

11. 作成 を選択して終了し、ジョブキューを作成します。

ジョブキューテンプレート

以下に示すのは、空のジョブ定義テンプレートです。このテンプレートを使用して、ジョブキューを作成できます。その後、このジョブキューをファイルに保存し、 `--cli-input-json`オプションで AWS CLI 使用できます。これらのパラメータの詳細については、 AWS Batch API リファレンスのCreateJobQueue「」を参照してください。

```
{
   "computeEnvironmentOrder": [
      {
         "computeEnvironment": "",
         "order": 0
      }
   ],
   "jobQueueName": "",
   "jobStateTimeLimitActions": [
      {
         "state": "RUNNABLE",
         "action": "CANCEL",
   "maxTimeSeconds": 0,
         "reason": ""

      }
   ],
   "priority": 0,
   "schedulingPolicyArn": "",
   "state": "ENABLED",
   "tags": {
      "KeyName": ""
   }
}
```

> **Note**
>
> 次の AWS CLI コマンドを使用して、前述のジョブキューテンプレートを生成できます。
>
> ```
> $ aws batch create-job-queue --generate-cli-skeleton
> ```

ジョブキューのパラメータ

ジョブキューは、名前、状態、優先度、コンピューティング環境の順序の 4 つの基本コンポーネントに分割されます。このセクションでは、関連するコンポーネントについて説明します。

トピック

ジョブキュー名

`jobQueueName`

ジョブキューの名前。最大 128 文字の英字 (大文字と小文字)、数字、下線を使用できます。

型: 文字列

必須: はい

ジョブキューの状態制限アクション

`jobStateTimeLimitActions`

指定された時間よりも長い指定された状態のジョブキューの先頭に残っているジョブに対してが AWS Batch 実行する一連のアクション。 AWS Batch は、 `maxTimeSeconds` が渡された後に各アクションを実行します。 (注： の最小値`maxTimeSeconds`は 600 (10 分) で、最大値は 86,400 (24 時間) です。)

型: `JobStateTimeLimitActions` オブジェクトの配列

必須: いいえ

優先度

priority

ジョブキューの優先度です。同じコンピューティング環境と関連付けられているジョブキュー間では、優先度が高い (priority パラメータの整数値が高い) ジョブキューほど先に処理されます。優先度は降順で決定されます。たとえば、優先度の値が 10 のジョブキューは、優先度の値が 1 のジョブキューより先にスケジュールされます。すべてのコンピューティング環境は、Amazon EC2 (EC2 または SPOT) または Fargate (FARGATE または) である必要がありますFARGATE_SPOT。Amazon EC2 と Fargate コンピューティング環境を混在させることはできません。

タイプ: 整数

必須: はい

スケジューリングポリシー

schedulingPolicyArn

ジョブキューのスケジューリングポリシーの Amazon リソースネーム (ARN)。タスクのスケジューリングポリシーを持たないジョブキューは、先入れ先出し (FIFO) モデルでスケジュールされます。ジョブキューにスケジューリングポリシーが設定された後は、置き換えることができますが、削除することはできません。スケジューリングポリシーのないジョブキューは FIFO ジョブキューとしてスケジュールされ、スケジューリングポリシーを追加することはできません。スケジューリングポリシーを持つジョブキューには、最大 500 個のアクティブな公平配分識別子を指定できます。上限に達すると、新しい公平分担識別子を追加したジョブの送信は失敗します。

タイプ: 文字列

必須: いいえ

都道府県

state

ジョブキューの状態です。ジョブキューの状態が ENABLED (デフォルト値) である場合は、ジョブを受け入れることができます。ジョブキューの状態が DISABLED である場合は、新しいジョブをキューに追加できませんが、すでにキューにあるジョブは終了できます。

タイプ: 文字列

有効な値: ENABLED | DISABLED

必須: いいえ

コンピューティング環境の順番

computeEnvironmentOrder

ジョブキューにマッピングされているコンピューティング環境のセットおよびコンピューティング環境間の順序。ジョブスケジューラは、このパラメータを使用して、どのコンピューティング環境で特定のジョブを実行するかを決定します。コンピューティング環境は、ジョブキューに関連付ける前に、VALID 状態になっていることが必要です。1 つのジョブキューには、最大3つのコンピューティング環境を関連付けることができます。すべてのコンピューティング環境は、Amazon EC2 (EC2 または SPOT) または Fargate (FARGATE または) である必要がありますFARGATE_SPOT。Amazon EC2 と Fargate コンピューティング環境を混在させることはできません。

> **Note**
> ジョブキューに関連付けられているすべてのコンピューティング環境は、同じ architecture. AWS Batch doesn を共有する必要があります。 は、単一のジョブキューでのコンピューティング環境アーキテクチャタイプの混在をサポートしていません。

型: ComputeEnvironmentOrder オブジェクトの配列

必須: はい

computeEnvironment

コンピューティング環境の Amazon リソースネーム (ARN)

型: 文字列

必須: はい

`order`

コンピューティング環境の順番。コンピューティング環境は昇順に試行されます。たとえば、2 つのコンピューティング環境が 1 つのジョブキューに関連付けられている場合、`order` の整数値が低いほうのコンピューティング環境にジョブの配置が先に試行されます。

タグ

`tags`

ジョブキューに関連付けるキーバリューペアのタグ。詳細については、「AWS Batch リソースのタグ付け」を参照してください。

型: 文字列間のマッピング

必須: いいえ

ジョブキューのステータスの表示

ジョブキューを作成してジョブを送信すると、その進行状況をモニタリングできることが重要です。ジョブの詳細ページを使用して、ジョブキューを確認、管理、モニタリングできます。

ジョブキュー情報の表示

AWS Batch コンソールから、ナビゲーションペインでジョブキューを選択し、目的のジョブキューを選択して詳細を表示します。このページでは、ジョブキューを確認して管理し、ジョブキューのスナップショット、ジョブの状態制限、環境の順序、タグ、ジョブキューの JSON コードなど、キューのオペレーションに関する追加情報を確認できます。

ジョブキューの詳細

このセクションでは、ジョブキューの概要とメンテナンスオプションについて説明します。このセクションには Amazon リソースネーム (ARN) が記載されています。

を通じてこの情報を検索するには AWS Command Line Interface、ジョブキュー名または対応する ARN とともに `DescribeJobQueues`オペレーションを使用します。

ジョブキュースナップショット

このセクションでは、キューに入っている最初の 100 個のRUNNABLEジョブの静的リストを示します。検索フィールドを使用して、結果セクションの任意の列から情報を検索することでリストを絞り込むことができます。スナップショット結果エリアのジョブは、ジョブキューの実行戦略に従ってソートされます。 first-in-first-out (FIFO) ジョブキューの場合、ジョブの順序は送信時間に基づきます。AWS Batch 公平配分スケジューリング (FSS) ジョブキューの場合、ジョブの順序はジョブの優先度と共有使用状況に基づきます。

結果はジョブキューのスナップショットであるため、結果リストは自動的に更新されません。リストを更新するには、セクションの上部にある更新を選択します。ジョブ名ハイパーリンクを選択して、ジョブの詳細に移動し、ジョブのステータスやその他の関連情報を表示します。

を通じてこの情報を検索するには AWS CLI、ジョブキュー名または対応する ARN とともに `GetJobQueueSnapshot`オペレーションを使用します。

ジョブの状態制限

このタブを使用して、ジョブがキャンセルされるまでに RUNNABLE 状態を維持できる時間に関する設定情報を確認します。

を通じてこの情報を検索するには AWS CLI、ジョブキュー名または対応する ARN とともに `DescribeJobQueues`オペレーションを使用します。

環境の順序

ジョブキューが複数の環境で実行されている場合、このタブにはその順序と概要が表示されます。

を通じてこの情報を検索するには AWS CLI、ジョブキュー名または対応する ARN とともに `DescribeJobQueues`オペレーションを使用します。

タグ

このタブを使用して、このジョブキューに関連付けられているタグを確認および管理します。

JSON

このタブを使用して、このジョブキューに関連付けられている JSON コードをコピーします。その後、JSON を AWS CloudFormation テンプレートと AWS CLI スクリプトに再利用できます。

ジョブスケジューリング

ス AWS Batch ケジューラは、ジョブキューに送信されたジョブをいつ、どこで、どのように実行するかを評価します。ジョブキューの作成時にスケジューリングポリシーを指定しない場合、 AWS Batch ジョブスケジューラはデフォルトで先入れ先出し (FIFO) 戦略になります。FIFO 戦略では、重要なジョブが以前に送信されたジョブの後ろでスタックする可能性があります。別のスケジューリングポリシーを指定することで、特定のニーズに応じてコンピューティングリソースを割り当てることができます。

> **Note**
>
> ジョブが実行される特定の順序をスケジュールする場合は、 の dependsOnパラメータを使用してSubmitJob、各ジョブの依存関係を指定します。

スケジューリングポリシーを作成してジョブキューに適用すると、公平分配スケジューリングが有効になります。ジョブキューにスケジューリングポリシーがある場合、スケジューリングポリシーによってジョブの実行順序が決まります。詳細については、スケジューリングポリシーを参照してください。

配分識別子

配分識別子を使用してジョブにタグを付け、ユーザーとワークロードを区別できます。 AWS Batch スケジューラは、*(T * weightFactor)*式を使用して各公平配分識別子の使用状況を追跡します。ここで、 *T*は時間の経過に伴う vCPU の使用状況です。スケジューラは、配分識別子から使用率が最も低いジョブを選択します。公平配分識別子は、オーバーライドせずに使用できます。

> **Note**
>
> 配分識別子はジョブキュー内で一意であり、ジョブキュー全体で集計されることはありません。

スケジュールの優先順位を設定して、配分識別子でのジョブの実行順序を設定できます。スケジューリング優先度の高いジョブが最初にスケジュールされます。スケジューリングポリシーを指定しない場合、ジョブキューに送信されるすべてのジョブは FIFO 順にスケジュールされます。ジョブの送信時には、配分識別子やスケジューリング優先度を指定することはできません。

> **Note**
>
> アタッチされたコンピューティングリソースは、明示的にオーバーライドされない限り、すべての配分識別子に均等に割り当てられます。

公平配分のスケジューリング

公平配分スケジューリングは、ジョブのスケジュール設定に役立つ一連の制御を提供します。

> **Note**
>
> スケジューリングポリシーパラメータの詳細については、スケジューリングポリシーパラメータを参照してください。

- 共有減衰秒 – AWS Batch スケジューラが各公平配分識別子の公平配分の割合を計算するために使用する期間 (秒単位) 。値がゼロ (0) の場合、現在の使用量のみが測定されることを示します。減衰時間が長いほど、時間にさらに多くの重みが与えられます。

  > **Note**
  >
  > 減衰時間は次のように計算されます。`shareDecaySeconds + OrderMinutes` (ここで、`OrderMinutes` は分単位の時間) 。

- コンピューティング予約 — 1 つの配分識別子に含まれるジョブが、ジョブキューにアタッチされているすべてのリソースを使い切ることを防ぎます。予約比率は で す。*computeReservation/100)^ActiveFairShares*ここで、 ActiveFairShares はアクティブな公平配分識別子の数です。

  > **Note**
  >
  > 配分識別子に SUBMITTED、PENDING、RUNNABLE、STARTING、または RUNNING の状態のジョブがある場合、そのジョブはアクティブな配分識別子と見なされます。減衰時間が切れると、配分識別子は非アクティブと見なされます。

- 重み係数 – 配分識別子の重み係数。デフォルト値は 1 です。値を小さくすると、配分識別子からジョブが実行されるか、配分識別子のランタイムが長くなります。たとえば、重み係数 0.125

(1/8) の配分識別子を使用するジョブには、重み係数 1 の配分識別子を使用するジョブの 8 倍のコンピューティングリソースが割り当てられます。

> **Note**
>
> この属性は、デフォルトの重み係数である 1 を更新する必要がある場合のみ設定します。

ジョブキューがアクティブでジョブを処理している場合、RUNNABLEジョブキュースナップショットを通じて最初の 100 個のジョブのリストを確認できます。詳細については、「ジョブキューのステータスの表示」を参照してください。

コンピューティング環境

ジョブキューは、1 つ以上のコンピューティング環境にマッピングされます。コンピューティング環境には、コンテナ化されたバッチジョブを実行するための Amazon ECS コンテナインスタンスが含まれています。特定のコンピューティング環境を 1 つ以上のジョブキューにマッピングすることもできます。ジョブキュー内では、関連付けられたコンピューティング環境ごとに順番があります。スケジューラでは、この順番に従って実行準備が完了したジョブの配置先を決定します。最初のコンピュート環境のVALIDステータスが、利用可能なリソースがある場合、ジョブはそのコンピュート環境内のコンテナインスタンスにスケジューリングされる。最初のコンピュート環境のINVALIDステータスが、適切なコンピュートリソースを提供できないか、または提供できない場合、スケジューラは次のコンピュート環境でジョブを実行しようとします。

トピック

マネージド型のコンピューティング環境

マネージドコンピューティング環境を使用して、 AWS Batch 環境内のコンピュートリソースの容量とインスタンスタイプを管理できます。これは、コンピューティング環境の作成時に定義するコンピューティングリソースの仕様に基づいています。Amazon EC2 オンデマンドインスタンスと Amazon EC2 スポットインスタンスを使用するかを選択できます。または、マネージド型のコンピューティング環境で Fargate および Fargate スポット容量を使用することもできます。スポットイ

ンスタンスを使用する場合、オプションで上限価格を設定できます。こうすることで、スポット・インスタンスは、スポット・インスタンス価格がオンデマンド価格の指定されたパーセンテージを下回った場合にのみ起動する。

> **Important**
>
> Fargate Windows containers on AWS Fargateスポットインスタンスはではサポートされていません。Fargate Spot FargateWindows コンピューティング環境のみを使用するジョブキューにジョブが送信されると、ジョブキューはブロックされます。

マネージド・コンピュート環境は、指定したVPCとサブネットにAmazon EC2インスタンスを起動し、Amazon ECSクラスタに登録します。Amazon EC2インスタンスは、Amazon ECSサービスエンドポイントと通信するために外部ネットワークアクセスが必要です。一部のサブネットでは、Amazon EC2インスタンスにパブリックIPアドレスを提供していない。Amazon EC2インスタンスがパブリックIPアドレスを持っていない場合、このアクセスを得るためにネットワークアドレス変換（NAT）を使用する必要があります。詳細については、Amazon VPC ユーザーガイド のNAT ゲートウェイを参照してください。VPC の作り方の詳細については、仮想プライベートクラウド (VPC) の作成 を参照してください。

デフォルトでは、 AWS Batch マネージド型コンピューティング環境は、承認された最新バージョンの Amazon ECS 最適化 AMI をコンピュートリソースに使用します。ただし、さまざまな理由により、マネージド型のコンピューティング環境で使用する AMI を独自に作成する場合もあります。詳細については、「コンピューティングリソースの AMI」を参照してください。

> **Note**
>
> AWS Batch コンピューティング環境の AMI は、作成後に自動的にアップグレードされません。例えば、Amazon ECS最適化AMIの新しいバージョンがリリースされても、コンピュート環境のAMIは更新されません。ゲストオペレーティングシステムの管理はユーザーの責任です。これには、アップデートとセキュリティパッチが含まれます。また、コンピューティングリソースにインストールするその他のアプリケーションソフトウェアやユーティリティについても責任を負うものとします。新しい AMI AWS Batch をジョブに使用する方法は 2 つあります。オリジナルの方法は、次のステップを完了することです。
>
> 1. 新しい AMI を使用して新しいコンピューティング環境を作成します。
> 2. コンピューティング環境を既存のジョブキューに追加します。
> 3. 古いコンピューティング環境をジョブキューから削除します。

4. 以前のコンピューティング環境を削除します。

2022 年 4 月に、 AWS Batch コンピューティング環境の更新のサポートが強化されました。詳細については、「コンピューティング環境を更新します。」を参照してください。コンピューティング環境の拡張アップデートを使用して AMI を更新するには、次のルールに従います。

- service role (serviceRole) パラメーターを設定しないか、AWSServiceRoleForBatchサービスにリンクされたロールに設定してください。
- 割り当て戦略 (allocationStrategy) パラメータを、BEST_FIT_PROGRESSIVE、SPOT_CAPACITY_OPTIMIZED または SPOT_PRICE_CAPACITY_OPTIMIZEDに設定します。
- 最新のイメージバージョンへの更新 (updateToLatestImageVersion) パラメータを true に設定します。
- imageId、imageIdOverride(ec2Configuration) または起動テンプレート (launchTemplate) には AMI ID を指定しないでください。その場合は、 AWS Batch AWS Batch インフラストラクチャの更新が開始された時点でサポートされている最新の Amazon ECS 最適化 AMI を選択してください。または、imageIdまたはimageIdOverrideパラメータで AMI ID を指定するか、LaunchTemplateプロパティで識別されるローンチテンプレートを指定することもできます。これらのプロパティのいずれかを変更すると、インフラストラクチャの更新が開始されます。AMI ID が起動テンプレートで指定されている場合、imageId または imageIdOverride パラメータで AMI ID を指定して置き換えることはできません。別の起動テンプレートを指定することでのみ置き換えることができます。$Default または $Latest、起動テンプレートのバージョンがまたはに設定されている場合は、起動テンプレートの新しいデフォルトバージョンを設定するか(設定されている場合 $Default)、起動テンプレートに新しいバージョンを追加(ある場合 $Latest)します。

これらのルールに従うと、インフラストラクチャの更新をトリガーする更新により、AMI ID が再選択されます。起動テンプレートversionの (launchTemplate)設定が$Latestまたは$Defaultに設定されている場合、launchTemplateを更新していない場合でも、インフラストラクチャの更新時に起動テンプレートの最新バージョンまたはデフォルトバージョンが評価されます。

マルチノードの並列ジョブを作成する際の考慮事項

AWS Batch マルチノードparallel (MNP) ジョブと非MNPジョブを実行するための専用のコンピューティング環境を作成することを推奨します。これは、マネージドコンピュート環境におけるコンピューティングキャパシティの作り方によるものです。マネージドコンピューティング環境を新規作成する際、`minvCpu` ゼロより大きい値を指定すると、 AWS Batch MNP 以外のジョブでのみ使用するインスタンスプールが作成されます。マルチノードparallel ジョブが送信されると、マルチノードparallel AWS Batch ジョブを実行するための新しいインスタンス容量が作成されます。`minvCpus`または`maxvCpus`値が設定されている同じコンピューティング環境で、単一ノードとマルチノードの両方のparallel ジョブが実行されている場合、必要なコンピュートリソースが使用できない場合は、 AWS Batch 現在のジョブが終了するのを待ってから、新しいジョブの実行に必要なコンピュートリソースを作成します。

アンマネージド型のコンピューティング環境

アンマネージド型のコンピューティング環境では、独自のコンピューティングリソースを管理します。コンピューティングリソースに使用する AMI が Amazon ECS コンテナインスタンスの AMI 仕様に合致していることを検証する必要があります。詳細については、「コンピューティングリソースの AMI 仕様」および「コンピューティングリソース AMI の作成」を参照してください。

> **Note**
>
> AWS Fargate リソースは、管理されていないコンピューティング環境ではサポートされていません。

アンマネージド型コンピューティング環境を作成したら、DescribeComputeEnvironmentsAPI オペレーションを使用してコンピューティング環境の詳細を表示します。環境に関連付けられている Amazon ECS クラスターを見つけ、その Amazon ECS クラスター内で手動でコンテナインスタンスを起動します。

AWS CLI 次のコマンドでは、Amazon ECS クラスター ARN も提供されます。

```
$ aws batch describe-compute-environments \
    --compute-environments unmanagedCE \
    --query "computeEnvironments[].ecsClusterArn"
```

詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS コンテナインスタンスの起動を参照してください。コンピューティングリソースを起動する際は、以下の Amazon EC2 ユーザーデータに登録される Amazon ECS クラスターの ARN を指定します。*ecsClusterArn*前のコマンドで取得したクラスター ARN に置き換えます。

```
#!/bin/bash
echo "ECS_CLUSTER=ecsClusterArn" >> /etc/ecs/ecs.config
```

コンピューティングリソースの AMI

デフォルトでは、 AWS Batch マネージド型コンピューティング環境は、承認された最新バージョンの Amazon ECS 最適化 AMI をコンピュートリソースに使用します。ただし、マネージド型およびアンマネージド型のコンピューティング環境で使用する AMI を独自に作成することもできます。次のいずれかが必要な場合は、独自の AMI を作成することをお勧めします。

- AMI のルートまたはデータボリュームのストレージサイズを増やす
- サポートされている Amazon EC2 インスタンスタイプのインスタンスストレージボリュームを追加する
- Amazon ECS コンテナエージェントをカスタマイズする
- Docker をカスタマイズする
- サポートされている Amazon EC2 インスタンスタイプで、コンテナから GPU ハードウェアにアクセスできるように GPU ワークロードの AMI を設定する

> **Note**
>
> コンピューティング環境を作成した後は、コンピューティング環境の AMI AWS Batch をアップグレードしません。 AWS Batch また、Amazon ECS に最適化された AMI の新しいバージョンが利用可能になっても、コンピューティング環境の AMI は更新されません。ゲストオペレーティングシステムの管理はユーザーの責任です。これには、アップデートとセキュリティパッチが含まれます。また、コンピューティングリソースにインストールするその他のアプリケーションソフトウェアやユーティリティについても責任を負うものとします。 AWS Batch ジョブに新しい AMI を使用するには、次の手順を実行します。
>
> 1. 新しい AMI を使用して新しいコンピューティング環境を作成します。
> 2. コンピューティング環境を既存のジョブキューに追加します。
> 3. 古いコンピューティング環境をジョブキューから削除します。

4. 以前のコンピューティング環境を削除します。

2022 年 4 月に AWS Batch 、コンピューティング環境の更新サポートが強化されました。詳細については、「コンピューティング環境を更新します。」を参照してください。コンピューティング環境の拡張アップデートを使用して AMI を更新するには、次のルールに従います。

- service role (`serviceRole`) パラメーターを設定しないか、AWSServiceRoleForBatchサービスにリンクされたロールに設定してください。
- 割り当て戦略 (`allocationStrategy`) パラメータを `BEST_FIT_PROGRESSIVE`、`SPOT_CAPACITY_OPTIMIZED`、または `SPOT_PRICE_CAPACITY_OPTIMIZED` に設定します。
- 最新のイメージバージョンへの更新 (`updateToLatestImageVersion`) パラメータを `true` に設定します。
- `imageId`、`imageIdOverride`(`ec2Configuration`) または起動テンプレート (`launchTemplate`) には AMI ID を指定しないでください。AMI ID を指定しない場合は、 AWS Batch インフラストラクチャの更新が開始された時点でサポートされる最新の Amazon ECS 最適化 AMI AWS Batch を選択します。代わりに、`imageId` または `imageIdOverride` パラメータを使用してAMI IDを指定できます。あるいは、`LaunchTemplate` プロパティによって識別される起動テンプレートを指定できます。これらのプロパティのいずれかを変更すると、インフラストラクチャの更新が開始されます。AMI ID が起動テンプレートで指定されている場合、`imageId` または `imageIdOverride` パラメータで AMI ID を指定しても AMI ID を置き換えることはできません。AMI ID は、別の起動テンプレートを指定することでのみ置き換えることができます。起動テンプレートのバージョンが `$Default` または `$Latest` に設定されている場合、AMI ID は起動テンプレートの新しいデフォルトバージョンを設定 (`$Default` の場合) するか、起動テンプレートに新しいバージョンを追加 (`$Latest` の場合) することで置き換えることができます。

これらのルールに従うと、インフラストラクチャの更新を開始する更新により、AMI ID が再選択されます。起動テンプレート (`launchTemplate`) の `version` 設定が `$Latest` または `$Default` に設定されている場合、`launchTemplate` が更新されていなくても、起動テンプレートの最新バージョンまたはデフォルトバージョンがインフラストラクチャの更新時に評価されます。

トピック

コンピューティングリソースの AMI 仕様

AWS Batch 基本的なコンピュートリソースAMI 仕様は以下のとおりです。

必須

- HVM 仮想化タイプの AMI で、バージョン 3.10 以上の Linux カーネルを実行している最新の Linux ディストリビューション。Windows コンテナはサポートされていません。

  > **Important**
  >
  > マルチノードの並列ジョブは、`ecs-init` パッケージがインストールされた Amazon Linux インスタンスで起動されたコンピューティングリソースでのみ実行できます。コンピューティング環境を作成するときに、デフォルトの Amazon ECS 最適化 AMI を使用することが推奨されます。これを行うには、カスタム AMI を指定しません。詳細については、マルチノードの並列ジョブを参照してください。

- Amazon ECS コンテナエージェント。最新バージョンの使用をお勧めします。詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS コンテナエージェントのインストールを参照してください。
- `awslogs` ログドライバは、Amazon ECS コンテナエージェントが開始するときの `ECS_AVAILABLE_LOGGING_DRIVERS` 環境変数として利用可能なログドライバとして指定する必要があります。詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS コンテナエージェントの構成を参照してください。
- バージョン 1.9 以上を実行する Docker デーモン、および Docker ランタイムの依存関係。詳細については、Docker ドキュメントのランタイムの依存関係を確認するを参照してください。

> **Note**
>
> 対応する Amazon ECS エージェントバージョンに同梱されており、そのバージョンでテストされた Docker バージョンをお勧めします。Amazon ECS は、上の Amazon ECS に最適化された AMI の Linux バリアントの変更ログを提供しています。 GitHub詳細については、「Changelog」を参照してください。

推奨

- Amazon ECS エージェントを実行およびモニタリングするための初期化およびナニープロセス。Amazon ECS に最適化した AMI は `ecs-init` 起動プロセスを使用し、その他のオペレーティングシステムは `systemd` を使用する場合があります。詳細については、Amazon Elastic Container Service 開発者ガイドのコンテナインスタンスのユーザーデータ設定スクリプトの例を参照してください。の詳細についてはecs-init、のプロジェクトを参照してください。ecs-init GitHub少なくとも、マネージド型のコンピューティング環境ではブート時に Amazon ECS エージェントをスタートする必要があります。Amazon ECS エージェントがコンピューティングリソースで実行されていない場合、 AWS Batchそのエージェントはからのジョブを受け入れることができません。

Amazon ECS に最適化された AMI は、これらの要件および推奨事項に従って事前設定されています。コンピューティングリソースには、Amazon ECS に最適化された AMI または `ecs-init` パッケージがインストールされた Amazon Linux AMI を使用することをお勧めします。アプリケーションに特定のオペレーティングシステムや、その AMI でまだ使用できない Docker バージョンが必要な場合は、別の AMI を選択します。詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS に最適化された AMIを参照してください。

コンピューティングリソース AMI の作成

コンピューティングリソースのカスタム AMI を独自に作成して、マネージド型およびアンマネージド型のコンピューティング環境で使用できます。手順については、コンピューティングリソースの AMI 仕様を参照してください。カスタム AMI を作成したら、その AMI を使用するコンピューティング環境を作成し、その環境をジョブキューに関連付けることができます。最後に、そのキューへのジョブの送信をスタートできます。

コンピューティングリソースのカスタム AMI を作成するには

1. 基本 AMI を選択してスタートします。基本 AMI は、HVM 仮想化を使用する必要があります。基本 AMI を Windows AMI にすることはできません。

   > **Note**
   >
   > コンピューティング環境用に選択する AMI は、そのコンピューティング環境で使用するインスタンスタイプのアーキテクチャと一致している必要があります。たとえば、コンピューティング環境で A1 インスタンスタイプを使用する場合、選択するコンピューティングリソース AMI で Arm インスタンスをサポートしている必要があります。Amazon ECS は、Amazon ECS に最適化された Amazon Linux 2 AMI の、x86 と Arm の両バージョンを提供しています。詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS に最適化された Amazon Linux 2 AMIを参照してください。

   Amazon ECS に最適化された Amazon Linux 2 AMI は、マネージド型のコンピューティング環境のコンピューティングリソースのデフォルト AMI です。Amazon ECS 最適化 Amazon Linux 2 AMI は、 AWS エンジニア AWS Batch によって事前設定され、 でテストされています。これは最小限の AMI であり、 を使用して、 で実行されているコンピューティングリソース AWS をすばやく取得できます。詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS に最適化された AMIを参照してください。

   または、別の Amazon Linux 2 バリアントを選択し、次のコマンドを使用して `ecs-init` パッケージをインストールできます。詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon Linux 2 EC2 インスタンスへの Amazon ECS コンテナエージェントのインストールを参照してください。

   ```
   $ sudo amazon-linux-extras disable docker
   $ sudo amazon-linux-extras install ecs-init
   ```

   例えば、 AWS Batch コンピューティングリソースで GPU ワークロードを実行する場合は、Amazon Linux Deep Learning AMI から開始できます。次に、 AWS Batch ジョブを実行するように AMI を設定します。詳細については、「GPU ワークロードの AMI の使用」を参照してください。

> **Important**
>
> `ecs-init` パッケージをサポートしていない基本 AMI も選択できます。ただし、その場合はブート時に Amazon ECS エージェントを開始して実行を続けるように設定する必要があります。Amazon ECS コンテナエージェントを開始してモニタリングする、`systemd` を使用したユーザーデータ設定スクリプトの例をいくつか用意しています。詳細については、Amazon Elastic Container Service デベロッパーガイドのコンテナインスタンスのユーザーデータ設定スクリプトの例を参照してください。

2. AMI の適切なストレージオプションがある、選択された基本 AMI からインスタンスを起動します。アタッチされた Amazon EBS ボリュームまたはインスタンスストレージボリューム (選択したインスタンスタイプでサポートされている場合) のサイズと数を設定できます。詳細については、Amazon EC2 ユーザーガイド」の「インスタンスと Amazon EC2 インスタンスストアの起動」を参照してください。 Amazon EC2
3. SSH を使用してインスタンスに接続し、必要に応じて設定タスクを実行します。これには、以下のステップのいずれかまたはすべてが含まれる場合があります。

   - Amazon ECS コンテナエージェントをインストールする 詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS コンテナエージェントのインストールを参照してください。
   - インスタンスストアボリュームをフォーマットするスクリプトを設定する。
   - インスタンスストアボリュームまたは Amazon EFS ファイルシステムを `/etc/fstab` ファイルに追加し、ブート時にマウントする。
   - Docker オプションを設定する (デバッグの有効化、基本イメージサイズの調整など)。
   - パッケージのインストールやファイルのコピー。

   詳細については、「Amazon EC2 ユーザーガイド」の「SSH を使用した Linux インスタンスへの接続」を参照してください。 Amazon EC2
4. Amazon ECS コンテナエージェントをインスタンスで開始する場合は、AMI を作成する前にインスタンスを停止して永続的なデータチェックポイントを削除する必要があります。この操作を行わないと、AMI から起動されたインスタンスでエージェントが起動しません。

   a. Amazon ECS コンテナエージェントを停止します。

      - Amazon ECS 対応 Amazon Linux 2 AMI:

```
sudo systemctl stop ecs
```

- Amazon ECS に最適化された Amazon Linux AMI

```
sudo stop ecs
```

b. 永続的なデータチェックポイントファイルを削除します。デフォルトでは、このファイルは `/var/lib/ecs/data/` ディレクトリにあります。ファイルが存在する場合は、次のコマンドを使用してこれらのファイルを削除します。

```
sudo rm -rf /var/lib/ecs/data/*
```

5. 実行中のインスタンスから新しい AMI を作成します。詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EBS Backed Linux AMI の作成」を参照してください。 Amazon EC2

で新しい AMI を使用するには AWS Batch

1. 新しい AMI が作成されたら、新規 AMI でコンピューティング環境を作成します。これを行うには、
イメージタイプを選択し、イメージ ID にカスタム AMI ID を入力します。
コンピューティング環境を作成するときにボックスを上書きします。 AWS Batch 詳細については、the section called “EC2 リソースを使用して、マネージド型のコンピューティング環境を作成するには”を参照してください。

> **Note**
>
> コンピューティング環境用に選択する AMI は、そのコンピューティング環境で使用するインスタンスタイプのアーキテクチャと一致している必要があります。たとえば、コンピューティング環境で A1 インスタンスタイプを使用する場合、選択するコンピューティングリソース AMI で Arm インスタンスをサポートしている必要があります。Amazon ECS は、Amazon ECS に最適化された Amazon Linux 2 AMI の、x86 と Arm の両バージョンを提供しています。詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS に最適化された Amazon Linux 2 AMIを参照してください。

2. ジョブキューを作成し、新しいコンピューティング環境を関連付けます。詳細については、「ジョブキューの作成」を参照してください。

> **Note**
>
> ジョブキューに関連付けられているすべてのコンピューティング環境は、同じアーキテクチャを共有する必要があります。 AWS Batch は、1 つのジョブキューでのコンピューティング環境アーキテクチャタイプの混在をサポートしていません。

3. (オプション) 新しいジョブキューにサンプルジョブを送信します。詳細については、ジョブ定義の例、シングルノードのジョブ定義を作成する 、およびジョブの送信を参照してください。

GPU ワークロードの AMI の使用

ユーザーのAWS Batch コンピューティングリソースで GPU ワークロードを実行するには、GPU 対応の AMI を使用する必要があります。詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS での GPU の使用およびAmazon ECS に最適化された AMIを参照してください。

マネージド型のコンピューティング環境では、コンピューティング環境で p2、p3、p4、p5、g3、g3s またはg5のインスタンスタイプまたはインスタンスファミリーが指定されている場合、Amazon ECS GPU に最適化された AMI が使用されます。

アンマネージド型のコンピューティング環境では、 Amazon ECS GPU に最適化された AMI をお勧めします。AWS Command Line Interface または AWS Systems Managerパラメータストアの GetParameter、GetParameters、および GetParametersByPath オペレーションを使用して、推奨される Amazon ECS GPU に最適化された AMI のメタデータを取得できます。

> **Note**
>
> p5のインスタンスファミリーは、Amazon ECS GPUに最適化されたAMIに等しいバージョン、またはAmazon ECS GPUに最適化されたAMIの20230912 よりも遅いバージョンでのみサポートされており、p2およびg2 のインスタンスタイプとは互換性がありません。p5 インスタンスを使用する必要がある場合は、コンピューティング環境に p2 または g2の インスタンスが含まれておらず、最新のデフォルトの Batch AMI を使用していることを確認してください。新しいコンピューティング環境を作成すると最新の AMI が使用されますが、ユーザーがp5を追加するようにコンピューティング環境を更新する場合は、ComputeResource プロパティでupdateToLatestImageVersion をtrue に設定することにより、最新の AMI を使用していることを確認できます。AMIのGPUインスタンスと互換性の詳細について

は、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS での GPU との連動を参照してください。

次に、GetParameter コマンドの使用方法を示します。

AWS CLI

```
$ aws ssm get-parameter --name /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/
recommended \
                        --region us-east-2 --output json
```

出力には、Value パラメータのAMI 情報が含まれています。

```
{
    "Parameter": {
        "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended",
        "LastModifiedDate": 1555434128.664,
        "Value": "{\"schema_version\":1,\"image_name\":\"amzn2-ami-ecs-gpu-
hvm-2.0.20190402-x86_64-ebs\",\"image_id\":\"ami-083c800fe4211192f\",\"os\":\"Amazon
 Linux 2\",\"ecs_runtime_version\":\"Docker version 18.06.1-ce\",\"ecs_agent_version
\":\"1.27.0\"}",
        "Version": 9,
        "Type": "String",
        "ARN": "arn:aws:ssm:us-east-2::parameter/aws/service/ecs/optimized-ami/
amazon-linux-2/gpu/recommended"
    }
}
```

Python

```
from __future__ import print_function

import json
import boto3

ssm = boto3.client('ssm', 'us-east-2')

response = ssm.get_parameter(Name='/aws/service/ecs/optimized-ami/amazon-linux-2/
gpu/recommended')
jsonVal = json.loads(response['Parameter']['Value'])
print("image_id   = " + jsonVal['image_id'])
```

```
print("image_name = " + jsonVal['image_name'])
```

出力には、AMI ID と AMI 名のみが含まれます。

```
image_id   = ami-083c800fe4211192f
image_name = amzn2-ami-ecs-gpu-hvm-2.0.20190402-x86_64-ebs
```

GetParameters の使用例を以下に示します。

AWS CLI

```
$ aws ssm get-parameters --names  /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/
recommended/image_name \
                                 /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/
recommended/image_id \
                         --region us-east-2 --output json
```

出力にはパラメータそれぞれの完全なメタデータが含まれています。

```
{
    "InvalidParameters": [],
    "Parameters": [
        {
            "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/
image_id",
            "LastModifiedDate": 1555434128.749,
            "Value": "ami-083c800fe4211192f",
            "Version": 9,
            "Type": "String",
            "ARN": "arn:aws:ssm:us-east-2::parameter/aws/service/ecs/optimized-ami/
amazon-linux-2/gpu/recommended/image_id"
        },
        {
            "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/
image_name",
            "LastModifiedDate": 1555434128.712,
            "Value": "amzn2-ami-ecs-gpu-hvm-2.0.20190402-x86_64-ebs",
            "Version": 9,
            "Type": "String",
            "ARN": "arn:aws:ssm:us-east-2::parameter/aws/service/ecs/optimized-ami/
amazon-linux-2/gpu/recommended/image_name"
```

```
        }
    ]
}
```

Python

```
from __future__ import print_function

import boto3

ssm = boto3.client('ssm', 'us-east-2')

response = ssm.get_parameters(
            Names=['/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/
image_name',
                   '/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/
image_id'])
for parameter in response['Parameters']:
    print(parameter['Name'] + " = " + parameter['Value'])
```

出力には、AMI ID とAMI 名が含まれており、フルパス名が使用されています。

```
/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_id =
 ami-083c800fe4211192f
/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_name = amzn2-
ami-ecs-gpu-hvm-2.0.20190402-x86_64-ebs
```

以下の例は、GetParametersByPathコマンドの使用方法を示しています。

AWS CLI

```
$ aws ssm get-parameters-by-path --path /aws/service/ecs/optimized-ami/amazon-
linux-2/gpu/recommended \
                                 --region us-east-2 --output json
```

出力には、指定されたパスに基づくすべてのパラメータの完全なメタデータが含まれています。

```
{
    "Parameters": [
        {
```

```
            "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/
ecs_agent_version",
            "LastModifiedDate": 1555434128.801,
            "Value": "1.27.0",
            "Version": 8,
            "Type": "String",
            "ARN": "arn:aws:ssm:us-east-2::parameter/aws/service/ecs/optimized-ami/
amazon-linux-2/gpu/recommended/ecs_agent_version"
        },
        {
            "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/
ecs_runtime_version",
            "LastModifiedDate": 1548368308.213,
            "Value": "Docker version 18.06.1-ce",
            "Version": 1,
            "Type": "String",
            "ARN": "arn:aws:ssm:us-east-2::parameter/aws/service/ecs/optimized-ami/
amazon-linux-2/gpu/recommended/ecs_runtime_version"
        },
        {
            "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/
image_id",
            "LastModifiedDate": 1555434128.749,
            "Value": "ami-083c800fe4211192f",
            "Version": 9,
            "Type": "String",
            "ARN": "arn:aws:ssm:us-east-2::parameter/aws/service/ecs/optimized-ami/
amazon-linux-2/gpu/recommended/image_id"
        },
        {
            "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/
image_name",
            "LastModifiedDate": 1555434128.712,
            "Value": "amzn2-ami-ecs-gpu-hvm-2.0.20190402-x86_64-ebs",
            "Version": 9,
            "Type": "String",
            "ARN": "arn:aws:ssm:us-east-2::parameter/aws/service/ecs/optimized-ami/
amazon-linux-2/gpu/recommended/image_name"
        },
        {
            "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/
os",
            "LastModifiedDate": 1548368308.143,
            "Value": "Amazon Linux 2",
```

```
            "Version": 1,
            "Type": "String",
            "ARN": "arn:aws:ssm:us-east-2::parameter/aws/service/ecs/optimized-ami/
amazon-linux-2/gpu/recommended/os"
        },
        {
            "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/
schema_version",
            "LastModifiedDate": 1548368307.914,
            "Value": "1",
            "Version": 1,
            "Type": "String",
            "ARN": "arn:aws:ssm:us-east-2::parameter/aws/service/ecs/optimized-ami/
amazon-linux-2/gpu/recommended/schema_version"
        }
    ]
}
```

Python

```
from __future__ import print_function

import boto3

ssm = boto3.client('ssm', 'us-east-2')

response = ssm.get_parameters_by_path(Path='/aws/service/ecs/optimized-ami/amazon-
linux-2/gpu/recommended')
for parameter in response['Parameters']:
    print(parameter['Name'] + " = " + parameter['Value'])
```

出力には、名前のフルパスを使用して、指定されたパスにあるすべてのパラメータ名の値が含まれています。

```
/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/ecs_agent_version =
 1.27.0
/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/ecs_runtime_version =
 Docker version 18.06.1-ce
/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_id =
 ami-083c800fe4211192f
/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/image_name = amzn2-
ami-ecs-gpu-hvm-2.0.20190402-x86_64-ebs
/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/os = Amazon Linux 2
```

```
/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended/schema_version = 1
```

詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS に最適化された AMI メタデータの取得を参照してください。

Amazon Linux の廃止

Amazon Linux AMI (Amazon Linux 1 とも呼ばれる) は、2023 年 12 月 31 日にサポート終了となりました。 AWS Batch は Amazon Linux AMI のサポートを終了しました。これは、2024 年 1 月 1 日以降、セキュリティアップデートやバグ修正を一切受けられないためです。Amazon Linux の詳細については end-of-life、「よくあるご質問」を参照してください。

予期しないワークロードの中断を防ぐため、既存の Amazon Linux ベースのコンピューティング環境を Amazon Linux 2023 に更新し、引き続きセキュリティやその他の更新を受けることをお勧めします。

Amazon Linux AMI を使用するコンピューティング環境は、2023 年 12 月 31 end-of-life 日以降も引き続き機能する可能性があります。ただし、これらのコンピューティング環境には、 AWSから新しいソフトウェアアップデート、セキュリティパッチ、またはバグ修正が適用されなくなります。その後、Amazon Linux AMI end-of-life でこれらのコンピューティング環境を維持するのはお客様の責任です。最適なパフォーマンスとセキュリティを維持するために、 AWS Batch コンピューティング環境を Amazon Linux 2023 または Amazon Linux 2 に移行することをお勧めします。

Amazon Linux AMI AWS Batch から Amazon Linux 2023 または Amazon Linux 2 への移行に関するヘルプについては、「コンピューティング環境の更新-」を参照してください。 AWS Batch

起動テンプレートのサポート

AWS Batch はAmazon EC2 EC2 起動テンプレートの使用をサポートします。起動テンプレートを使用すると、カスタマイズされた AMIs を作成することなく、 AWS Batch コンピューティングリソースのデフォルト設定を変更できます。

> **Note**
>
> 起動テンプレートは AWS Fargate リソースではサポートされていません。

コンピューティング環境に関連付ける前に、起動テンプレートを作成する必要があります。起動テンプレートは、Amazon EC2 コンソールから作成することができます。または、 AWS CLI または

AWS SDK を使用できます。例えば、次の JSON ファイルは、デフォルトの AWS Batch コンピューティングリソース AMI の Docker データボリュームのサイズを変更し、暗号化するように設定する起動テンプレートを表します。

```
{
    "LaunchTemplateName": "increase-container-volume-encrypt",
    "LaunchTemplateData": {
        "BlockDeviceMappings": [
            {
                "DeviceName": "/dev/xvda",
                "Ebs": {
                    "Encrypted": true,
                    "VolumeSize": 100,
                    "VolumeType": "gp2"
                }
            }
        ]
    }
}
```

という名前のファイルに JSON を保存`lt-data.json`し、次の AWS CLI コマンドを実行することで、以前の起動テンプレートを作成できます。

```
aws ec2 --region <region> create-launch-template --cli-input-json file://lt-data.json
```

起動テンプレートの詳細については、Amazon EC2 ユーザーガイド」の「起動テンプレートからのインスタンスの起動」を参照してください。

起動テンプレートを使用してコンピューティング環境を作成する場合、次の既存のコンピューティング環境パラメータを起動テンプレートに移動できます。

> **Note**
>
> 上記のパラメータ (Amazon EC2 タグ以外) が起動テンプレートおよびコンピューティング環境設定の両方で指定されているとします。その場合は、コンピューティング環境パラメータが優先されます。Amazon EC2 タグは、起動テンプレートとコンピューティング環境設定間でマージされます。タグキーの対立がある場合、コンピューティング環境設定の値が優先されます。

- Amazon EC2 のキーペア
- Amazon EC2 AMI ID
- セキュリティグループ ID
- Amazon EC2 タグ

次の起動テンプレートパラメータは では無視されます AWS Batch。

- インスタンスタイプ (コンピューティング環境作成時にインスタンスタイプを指定)
- インスタンスロール (コンピューティング環境作成時にインスタンスロールを指定)
- ネットワークインターフェイスサブネット (コンピューティング環境の作成時に、サブネットを指定)
- インスタンス市場オプション (スポットインスタンス設定を制御するAWS Batch 必要があります)
- API の終了を無効にする (インスタンスのライフサイクルを制御するAWS Batch 必要があります)

AWS Batch は、インフラストラクチャの更新中にのみ、起動テンプレートを新しい起動テンプレートバージョンで更新します。詳細については、「コンピューティング環境を更新します。」を参照してください。

起動テンプレートの Amazon EC2 ユーザーデータ

起動テンプレート内の Amazon EC2 ユーザーデータを、cloud-init をインスタンス起動時に提供できます。ユーザーデータは、以下を含む一般的な設定シナリオを実行できます。ただし、これらに限定されるものではありません。

- ユーザーまたはグループを含む
- パッケージのインストール
- パーティションおよびファイルシステムの作成

起動テンプレートの Amazon EC2 ユーザーデータは、MIME マルチパートアーカイブの形式であることが必要です。これは、ユーザーデータがコンピューティングリソースの設定に必要な他の AWS Batch ユーザーデータとマージされるためです。複数のユーザーデータブロックと単一の MIME マルチパートファイルを組み合わせることができます。たとえば、Amazon ECS コンテナエージェントの設定情報を書き込むユーザーデータシェルスクリプトを使用して、Docker デーモンを設定するクラウドブートフックを組み合わせることができます。

を使用している場合は AWS CloudFormation、 AWS::CloudFormation::Initタイプを cfn-init ヘルパースクリプトとともに使用して、一般的な設定シナリオを実行できます。

MIME マルチパートファイルには次のコンポーネントが含まれます。

- コンテンツの種類およびパート境界の宣言: `Content-Type: multipart/mixed; boundary="==BOUNDARY=="`
- MIME バージョンの宣言: `MIME-Version: 1.0`
- 次のコンポーネントを含む 1 つ以上のユーザーデータブロック:
  - ユーザーデータブロックの始まりを示す開始境界: `--==BOUNDARY==`。この境界の前の行は空白にしておく必要があります。
  - ブロックのコンテンツの種類の宣言: `Content-Type: text/cloud-config; charset="us-ascii"`。コンテンツの種類の詳細については、Cloud-Init のドキュメントを参照してください。コンテンツタイプ宣言の後の行は空白にしておく必要があります。
  - ユーザーデータのコンテンツ (例えば、シェルコマンドや `cloud-init` ディレクティブのリスト)。
- MIME マルチパートファイルの終わりを示す、終了境界: `--==BOUNDARY==--`。この境界の前の行は空白にしておく必要があります。

自分で MIME マルチパートファイルを作成するときに使用できる例は、次の通りです。

> **Note**
>
> Amazon EC2 コンソールの起動テンプレートにユーザーデータを追加するには、ユーザーデータをプレーンテキストとして貼り付けます。または、ファイルからアップロードすることもできます。 AWS CLI または AWS SDK を使用する場合は、この base64 JSON ファイルに示すように、CreateLaunchテンプレート を呼び出すときに、まずユーザーデータをエンコードし、その文字列を UserDataパラメータの値として送信する必要があります。
>
> ```
> {
>     "LaunchTemplateName": "base64-user-data",
>     "LaunchTemplateData": {
>         "UserData":
>  "ewogICAgIkxhdW5jaFRlbXBsYXRlTmFtZSI6ICJpbmNyZWFzZS1jb250YWluZXItdm9sdW..."
>     }
> }
> ```

例

- 例: 既存の Amazon EFS ファイルシステムのマウント
- 例: デフォルトの Amazon ECS コンテナエージェント設定の上書き
- 例 : 既存の Amazon FSx for Lustre ファイルシステムのマウント

例: 既存の Amazon EFS ファイルシステムのマウント

Example

このサンプル MIME マルチパートファイルは、コンピューティングリソースを設定して `amazon-efs-utils` パッケージをインストールし、既存の Amazon EFS ファイルシステムを `/mnt/efs` にマウントします。

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="

--==MYBOUNDARY==
Content-Type: text/cloud-config; charset="us-ascii"

packages:
- amazon-efs-utils

runcmd:
- file_system_id_01=fs-abcdef123
- efs_directory=/mnt/efs

- mkdir -p ${efs_directory}
- echo "${file_system_id_01}:/ ${efs_directory} efs tls,_netdev" >> /etc/fstab
- mount -a -t efs defaults

--==MYBOUNDARY==--
```

例: デフォルトの Amazon ECS コンテナエージェント設定の上書き

Example

このサンプル MIME マルチパートファイルは、コンピューティングリソースのデフォルトの Docker イメージクリーンアップ設定を上書きします。

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="
```

```
--==MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
echo ECS_IMAGE_CLEANUP_INTERVAL=60m >> /etc/ecs/ecs.config
echo ECS_IMAGE_MINIMUM_CLEANUP_AGE=60m >> /etc/ecs/ecs.config

--==MYBOUNDARY==--
```

例：既存の Amazon FSx for Lustre ファイルシステムのマウント

Example

このサンプル MIME マルチパートファイルは、コンピューティングリソースを設定して Extras Library から `lustre2.10` パッケージをインストールし、`/scratch` にある既存の Fsx for Lustre ファイルシステムを `fsx` というマウント名でマウントします。この例は Amazon Linux 2 用です。他の Linux ディストリビューションのインストール手順については、Amazon FSx for Lustre ユーザーガイドのLustre Clientのインストールを参照してください。詳細については、Amazon FSx for Lustre ユーザーガイドのAmazon FSx ファイルシステムの自動マウントを参照してください。

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="

--==MYBOUNDARY==
Content-Type: text/cloud-config; charset="us-ascii"

runcmd:
- file_system_id_01=fs-0abcdef1234567890
- region=us-east-2
- fsx_directory=/scratch
- amazon-linux-extras install -y lustre2.10
- mkdir -p ${fsx_directory}
- mount -t lustre ${file_system_id_01}.fsx.${region}.amazonaws.com@tcp:fsx
 ${fsx_directory}

--==MYBOUNDARY==--
```

コンテナプロパティの volumes および mountPoints メンバーでは、マウントポイントをコンテナにマッピングする必要があります。

```
{
```

```
    "volumes": [
        {
            "host": {
                "sourcePath": "/scratch"
            },
            "name": "Scratch"
        }
    ],
    "mountPoints": [
        {
            "containerPath": "/scratch",
            "sourceVolume": "Scratch"
        }
    ],
}
```

コンピューティング環境の作成

でジョブを実行する前に AWS Batch、コンピューティング環境を作成する必要があります。が仕様に基づいて環境内の Amazon EC2 インスタンスまたは AWS Fargate リソース AWS Batch を管理するマネージドコンピューティング環境を作成できます。あるいは、アンマネージド型のコンピューティング環境を構築し、Amazon EC2 インスタンスの設定を環境内で処理することも可能です。

Important

Fargate Spot インスタンスは、以下のシナリオではサポートされません。

- ARM64 アーキテクチャの Amazon Linux コンテナの場合。
- Windows containers on AWS Fargate

このようなシナリオでは、Fargate Spot コンピューティング環境のみを使用するジョブキューにジョブが送信されると、ジョブキューはブロックされます。

目次

- Amazon EKS リソースを使用してマネージドコンピューティング環境を作成するには

AWS Fargate リソースを使用してマネージドコンピューティング環境を作成するには

1. https://console.aws.amazon.com/batch/ で AWS Batch コンソールを開きます。
2. ナビゲーションバーから、 AWS リージョン 使用する を選択します。
3. ナビゲーションペインで、コンピューティング環境] を選択します。
4. 作成] を選択します。
5. コンピューティング環境を設定します。

   > **Note**
   >
   > Windows containers on AWS Fargate ジョブのコンピューティング環境には、少なくとも 1 つの vCPU が必要です。

   a. コンピュート環境設定 で、Fargate を選択します。

   b. 名前 で、コンピューティング環境の一意な名前を指定します。名前の長さは最大 128 文字です。大文字、小文字、数字、ハイフン (-)、アンダースコア (_) を含めることができます。

   c. サービスロール で、サービスがユーザーに代わって必要な AWS API オペレーションを呼び出せるようにする AWS Batch サービスにリンクされたロールを選択します。例えば、[AWSServiceRoleForBatch] を選択します。詳細については、「のサービスにリンクされたロールのアクセス許可 AWS Batch」を参照してください。

   d. (オプション) タグ を展開します。タグを追加するには、タグの追加 を選択します。次に、キー 名を入力し、オプションで 値 を入力します。タグを追加] を選択します。

   e. 次のページ を選択します。

6. インスタンス設定 セクション:

   a. (オプション) Fargate Spot キャパシティを使用 では、Fargate Spot をオンにします。Fargate Spot については、Amazon EC2 スポットと Fargate_Spot を使用するを参照してください。

   b. 最大 vCPU] では、ジョブキューの需要にかかわらず、コンピューティング環境でスケールアウトできる vCPU の最大数を選択します。

c. 次のページ を選択します。

7. ネットワーキングを設定します。

> **Important**
>
> コンピューティングリソースには、Amazon ECS サービスエンドポイントと通信するために外部ネットワークアクセスが必要です。これは、インターフェイス VPC エンドポイントを介して、またはパブリック IP アドレスを持つコンピュートリソースを通じて可能になります。
> インターフェイス VPC エンドポイントの詳細については、Amazon Elastic Container Service 開発者ガイドのAmazon ECS インターフェイス VPC エンドポイント (AWS PrivateLink)を参照してください。
> インターフェイス VPC エンドポイントが設定されておらず、コンピューティングリソースがパブリック IP アドレスを持たない場合は、ネットワークアドレス変換 (NAT) を使用してこのアクセスを提供する必要があります。詳細については、Amazon VPC ユーザーガイドに含まれるNAT ゲートウェイを参照してください。詳細については、the section called “「VPC を作成する」”を参照してください。

a. 仮想プライベートクラウド (VPC) ID では、インスタンスを起動する VPC を選択します。

b. サブネット では、使用するサブネットを選択します。デフォルトでは、選択した VPC 内のすべてのサブネットを使用できます。

> **Note**
>
> AWS Batch Fargate の は、現在ローカルゾーンをサポートしていません。詳細については、Amazon Elastic Container Service 開発者ガイドのAmazon ECS クラスターのローカルゾーン、波長ゾーン、および AWS Outpostsを参照してください。

c. セキュリティグループ] では、インスタンスにアタッチするセキュリティグループを選択します。デフォルトでは、VPC のデフォルトのセキュリティグループが選択されます。

d. 次のページ を選択します。

8. レビュー では、設定手順を確認します。変更する必要がある場合は、編集] を選択します。完了したら、コンピューティング環境の作成 を選択します。

EC2 リソースを使用して、マネージド型のコンピューティング環境を作成するには

1. https://console.aws.amazon.com/batch/ で AWS Batch コンソールを開きます。
2. ナビゲーションバーから、 AWS リージョン 使用する を選択します。
3. ナビゲーションペインで、コンピューティング環境] を選択します。
4. 作成] を選択します。
5. 環境を設定します。

   a. コンピューティング環境設定 で、Amazon Elastic Compute Cloud (Amazon EC2) を選択します。

   b. オーケストレーションタイプ で、マネージド を選択します。

   c. 名前 で、コンピューティング環境の一意な名前を指定します。名前の長さは最大 128 文字です。大文字、小文字、数字、ハイフン (-)、アンダースコア (_) を含めることができます。

   d. (オプション) サービスロール で、サービスがユーザーに代わって必要な AWS API オペレーションを呼び出せるようにする AWS Batch サービスにリンクされたロールを選択します。例えば、[AWSServiceRoleForBatch] を選択します。詳細については、「のサービスにリンクされたロールのアクセス許可 AWS Batch」を参照してください。

   e. インスタンスロールの場合、新しいインスタンスプロファイルを作成するか、必要な IAM アクセス許可がアタッチされた既存のプロファイルを使用するかを選択します。このインスタンスプロファイルにより、コンピューティング環境用に作成された Amazon ECS コンテナインスタンスは、ユーザーに代わって必要な AWS API オペレーションを呼び出すことができます。詳細については、「Amazon ECS インスタンスロール」を参照してください。新しいインスタンスプロファイルを作成することを選択した場合は、必要なロール (`ecsInstanceRole`) が作成されます。

   f. (オプション) タグ を展開します。

   g. (オプション) EC2 タグ で、タグを追加 を選択して、コンピューティング環境で起動されるリソースにタグを追加します。次に、キー 名を入力し、オプションで 値 を入力します。タグを追加] を選択します。

   h. (オプション) タグ には タグを追加 を選択します。次に、キー 名を入力し、オプションで 値 を入力します。タグを追加] を選択します。

      詳細については、AWS Batch リソースのタグ付けを参照してください。

   i. 次のページ を選択します。

6. インスタンス設定 セクション:

   a. (オプション) スポットインスタンスの使用を有効にする でスポットをオンにします。詳細については、スポットインスタンスを参照してください。

   b. (スポットの場合のみ) 上限のオンデマンド価格の割合 (%) で、インスタンス起動前のインスタンスタイプのオンデマンド価格と対比したスポットインスタンス価格の最大パーセンテージを選択します。たとえば、上限価格が 20% の場合、その EC2 インスタンスのスポット料金は現在のオンデマンド料金の 20% 未満にする必要があります。支払い額は常に最低 (市場料金) となり、最大パーセンテージを超えることはありません。このフィールドを空のままにした場合、デフォルト値はオンデマンド料金の 100% です。

   c. (スポットの場合のみ) スポットフリートロール で、既存の Amazon EC2 スポットフリート IAM ロールを選択してスポットコンピューティング環境に適用します。既存の Amazon EC2 スポットフリート IAM ロールが存在しない場合には、まずこのロールを作成する必要があります。詳細については、「Amazon EC2 スポットフリートロール」を参照してください。

      > **Important**
      >
      > 作成時にスポットインスタンスにタグを付けるには、Amazon EC2 スポットフリート IAM ロールで新しい AmazonEC2SpotFleetTaggingRole 管理ポリシーを使用する必要があります。AmazonEC2SpotFleetRole 管理ポリシーには、スポットインスタンスにタグを付けるために必要なアクセス許可がありません。詳細については、「作成時にタグが付けられていないスポットインスタンス」および「the section called “ リソースのタグ付け”」を参照してください。

   d. 最小 vCPU で、ジョブキューの需要にかかわらず、コンピューティング環境で維持する vCPU の最小数を選択します。

   e. 必要な vCPU で、コンピューティング環境の起動に必要な vCPU の数を選択します。ジョブキューの需要が増えると、 AWS Batch はコンピューティング環境で必要な vCPU の数を増やし、vCPU の最大数まで EC2 インスタンスを追加できます。需要が減ると、 AWS Batch はコンピューティング環境で必要な vCPU の数を減らし、vCPU の最小数までインスタンスを削減できます。

   f. 最大 vCPU] では、ジョブキューの需要にかかわらず、コンピューティング環境でスケールアウトできる vCPU の最大数を選択します。

   g. 許可されたインスタンスタイプ] では、起動できる Amazon EC2 インスタンスタイプを選択します。インスタンスファミリーを指定してそのファミリー内のいずれかのインスタンスタ

イプ (c5、c5n、p3など) を起動できます。または、ファミリー内の特定のサイズを指定することもできます (c5.8xlarge)。メタルインスタンスタイプはインスタンスファミリーに含まれていません。たとえば、c5 は c5.metal を含んでいません。また、(C4、M4、および R4 インスタンスファミリーから) optimal を選択して、ジョブキューの需要に見合ったインスタンスタイプを使用することもできます。

> **Note**
>
> コンピューティング環境を作成する際、そのコンピューティング環境で選択するインスタンスタイプで同じアーキテクチャを使用する必要があります。例えば、x86 と ARM インスタンスを同じコンピューティング環境で使用することはできません。

> **Note**
>
> AWS Batch は、ジョブキューの必要量に基づいて GPUs をスケーリングします。GPU スケジューリングを使用するには、コンピューティング環境に p2、p3、p4、p5、g3、g3s、g4または g5 ファミリーのインスタンスタイプを含める必要があります。

> **Note**
>
> 現在、optimal は C4、M4、および R4 インスタンスファミリーのインスタンスタイプを使用しています。これらのインスタンスファミリーのインスタンスタイプ AWS リージョン がない では、、C5、M5および インスタンスファミリーのR5インスタンスタイプが使用されます。

h. 追加設定] を展開します。

i. (オプション) プレイスメントグループ では、プレイスメントグループ名を入力して、コンピューティング環境内のリソースをグループ化します。

j. (オプション) EC2 キーペア で、インスタンス接続時のセキュリティ認証情報としてパブリックキーとプライベートキーのペアを選択します。Amazon EC2 キーペアの詳細については、Amazon EC2 キーペアおよび Linux インスタンスを参照してください。

k. 配分戦略] では、許可されるインスタンスタイプのリストからインスタンスタイプを選択するときに使用する配分戦略を選択します。EC2 のオンデマンドコンピューティング環境では BEST_FIT_PROGRESSIVE を、EC2 のスポットコンピューティング環境では SPOT_CAPACITY_OPTIMIZED または SPOT_PRICE_CAPACITY_OPTIMIZED を選択するのが一般的です。詳細については、「the section called “配分戦略”」を参照してください。

l. (オプション) EC2 設定 では、イメージタイプとイメージ ID オーバーライド値を選択して、 がコンピューティング環境のインスタンスの Amazon マシンイメージ (AMIsを選択 AWS Batch するための情報を提供します。各イメージタイプ にイメージ ID オーバーライドが指定されていない場合、 は最新の Amazon ECS 最適化 AMI AWS Batch を選択します。イメージタイプが指定されていない場合、デフォルトは GPU 以外の AWS Graviton インスタンス用の Amazon Linux 2 です。

> **Important**
>
> カスタム AMI を使用するには、イメージタイプを選択し、イメージ ID のオーバーライド ボックスにカスタム AMI ID を入力します。

Amazon Linux 2

すべての AWS Graviton ベースのインスタンスファミリー (、、C6gM6g、 などT4g) のデフォルトでありR6g、すべての非 GPU インスタンスタイプに使用できます。

Amazon Linux 2 (GPU)

すべての GPU インスタンスファミリー (P4や などG4) のデフォルトであり、すべての AWS Graviton ベース以外のインスタンスタイプに使用できます。

Amazon Linux

非 GPU、非 AWS Graviton インスタンスファミリーに使用できます。Amazon Linux AMI の標準サポートは終了しました。詳細については、Amazon Linux AMI を参照してください。

> **Note**
>
> コンピューティング環境用に選択する AMI は、そのコンピューティング環境で使用するインスタンスタイプのアーキテクチャと一致している必要があります。たとえば、コンピューティング環境で A1 インスタンスタイプを使用する場合、選択

するコンピューティングリソース AMI で Arm インスタンスをサポートしている必要があります。Amazon ECS は、Amazon ECS に最適化された Amazon Linux 2 AMI の、x86 と Arm の両バージョンを提供しています。詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS に最適化された Amazon Linux 2 AMIを参照してください。

m. (オプション)起動テンプレート] では、既存の Amazon EC2 起動テンプレートを選択して、コンピューティングリソースを設定します。テンプレートのデフォルトバージョンは自動的に設定されます。詳細については、起動テンプレートのサポートを参照してください。

> **Note**
>
> 起動テンプレートで、作成したカスタム AMI を指定できます。

n. (オプション) 起動テンプレートのバージョン] では、`$Default` または `$Latest` を使用するか、あるいは起用するバージョン番号を指定します。

> **Important**
>
> 起動テンプレートのバージョンパラメータが `$Default` または `$Latest` の場合、指定された起動テンプレートのデフォルトまたは最新バージョンがインフラストラクチャの更新時に評価されます。デフォルトで別の AMI ID が選択されている場合、または起動テンプレートの最新バージョンが選択されている場合、その AMI ID が更新に使用されます。詳細については、the section called “AMI ID のアップデート”を参照してください。

o. 次のページ を選択します。

7. ネットワーク設定 セクションで:

> **Important**
>
> コンピューティングリソースには、Amazon ECS サービスエンドポイントと通信するために外部ネットワークアクセスが必要です。これは、インターフェイス VPC エンドポイントを介して、またはパブリック IP アドレスを持つコンピュートリソースを通じて可能になります。
> インターフェイス VPC エンドポイントの詳細については、Amazon Elastic Container Service 開発者ガイドのAmazon ECS インターフェイス VPC エンドポイント (AWS PrivateLink)を参照してください。

インターフェイス VPC エンドポイントが設定されておらず、コンピューティングリソースがパブリック IP アドレスを持たない場合は、ネットワークアドレス変換 (NAT) を使用してこのアクセスを提供する必要があります。詳細については、Amazon VPC ユーザーガイドに含まれるNAT ゲートウェイを参照してください。詳細については、the section called “「VPC を作成する」”を参照してください。

a. 仮想プライベートクラウド (VPC) ID では、インスタンスを起動する先の VPC を選択します。

b. サブネット では、使用するサブネットを選択します。デフォルトでは、選択した VPC 内のすべてのサブネットを使用できます。

> **Note**
>
> AWS Batch Amazon EC2 の はローカルゾーンをサポートしています。詳細については、「Amazon EC2 ユーザーガイド」の「Local Zones」と「Amazon Elastic Container Service デベロッパーガイド」の「Local Zones」および「Amazon ECS クラスター AWS Outposts」を参照してください。 Amazon EC2

c. (オプション) セキュリティグループ で、インスタンスにアタッチするセキュリティグループを選択します。デフォルトでは、VPC のデフォルトのセキュリティグループが選択されます。

8. 次のページ を選択します。
9. レビュー では、設定手順を確認します。変更する必要がある場合は、編集] を選択します。完了したら、コンピューティング環境の作成 を選択します。

EC2 リソースを使用して、アンマネージド型のコンピューティング環境を作成するには

1. https://console.aws.amazon.com/batch/ で AWS Batch コンソールを開きます。
2. ナビゲーションバーから、 AWS リージョン 使用する を選択します。
3. [コンピューティング環境] ページで、 [作成] を選択します。
4. 環境を設定します。

a. コンピューティング環境設定 で、Amazon Elastic Compute Cloud (Amazon EC2) を選択します。

b. オーケストレーションタイプ で、アンマネージド を選択します。

5. 名前 で、コンピューティング環境の一意な名前を指定します。名前の最大長は 128 文字です。大文字、小文字、数字、ハイフン (-)、アンダースコア (_) を含めることができます。

6. (オプション) サービスロール で、サービスがユーザーに代わって必要な AWS API オペレーションを呼び出せるようにする AWS Batch ロールを選択します。例えば、[AWSBatchServiceRole] を選択します。詳細については、the section called “サービスリンクロールの使用”を参照してください。

7. 最大 vCPU] では、ジョブキューの需要にかかわらず、コンピューティング環境でスケールアウトできる vCPU の最大数を選択します。

8. (オプション) タグ を展開します。タグを追加するには、タグの追加 を選択します。次に、キー名を入力し、オプションで 値 を入力します。タグを追加] を選択します。詳細については、AWS Batch リソースのタグ付けを参照してください。

9. 次のページ を選択します。

10. レビュー では、設定手順を確認します。変更する必要がある場合は、編集] を選択します。完了したら、コンピューティング環境の作成 を選択します。

Amazon EKS リソースを使用してマネージドコンピューティング環境を作成するには

1. https://console.aws.amazon.com/batch/ で AWS Batch コンソールを開きます。

2. ナビゲーションバーから、 AWS リージョン 使用する を選択します。

3. ナビゲーションペインで、コンピューティング環境] を選択します。

4. 作成] を選択します。

5. コンピューティング環境の設定 で、Amazon Elastic Kubernetes Service (Amazon EKS) を選択します。

6. 名前 で、コンピューティング環境の一意な名前を指定します。名前の最大長は 128 文字です。大文字、小文字、数字、ハイフン (-)、アンダースコア (_) を含めることができます。

7. インスタンスロール では、必要な IAM アクセス許可がアタッチされた既存のインスタンスプロファイルを使用することを選択します。

> **Note**
>
> AWS Batch コンソールでコンピューティング環境を作成するには、`eks:ListClusters`と `の``eks:DescribeCluster`アクセス許可を持つインスタンスプロファイルを選択します。

8. EKS クラスター で、既存の Amazon EKS クラスターを選択します。
9. 名前空間 で、クラスター内の AWS Batch プロセスをグループ化する Kubernetes 名前空間を入力します。
10. (オプション) タグ を展開します。タグの追加 を選択し、キーと値のペアを入力します。
11. 次のページ を選択します。
12. (オプション) EC2 スポットインスタンスを使用 で、スポットインスタンスの使用を有効にするをオンにして Amazon EC2 スポットインスタンスを使用します。
13. (スポットの場合のみ) 上限のオンデマンド価格の割合 (%) で、インスタンス起動前のインスタンスタイプのオンデマンド価格と対比したスポットインスタンス価格の最大パーセンテージを選択します。たとえば、上限価格が 20% の場合、その EC2 インスタンスのスポット料金は現在のオンデマンド料金の 20% 未満にする必要があります。支払い額は常に最低 (市場料金) となり、最大パーセンテージを超えることはありません。このフィールドを空のままにした場合、デフォルト値はオンデマンド料金の 100% です。
14. (スポットのみ) スポットフリートロール で、SPOT コンピューティング環境用の Amazon EC2 スポットフリート IAM ロールを選択します。

> **Important**
>
> このロールは、配分戦略が `BEST_FIT` に設定されている場合、または指定されていない場合に必要です。

15. (オプション) 最小 vCPU で、ジョブキューの需要にかかわらず、コンピューティング環境で維持する vCPU の最小数を選択します。
16. (オプション) 最大 vCPU で、ジョブキューの需要にかかわらず、コンピューティング環境でスケールアウトできる vCPU の最大数を選択します。
17. 許可されたインスタンスタイプ] では、起動できる Amazon EC2 インスタンスタイプを選択します。インスタンスファミリーを指定してそのファミリー内のいずれかのインスタンスタイプ (`c5`、`c5n`、`p3`など) を起動できます。または、ファミリー内の特定のサイズを指定することもできます (例えば、`c5.8xlarge`)。メタルインスタンスタイプはインスタンスファミリーに含ま

れていません。たとえば、c5 は c5.metal を含んでいません。また、(C4、M4、および R4 インスタンスファミリーから) optimal を選択して、ジョブキューの需要に見合ったインスタンスタイプを使用することもできます。

> **Note**
>
> コンピューティング環境を作成する際、そのコンピューティング環境で選択するインスタンスタイプで同じアーキテクチャを使用する必要があります。例えば、x86 と ARM インスタンスを同じコンピューティング環境で使用することはできません。

> **Note**
>
> AWS Batch はGPUs をスケーリングします。GPU スケジューリングを使用するには、コンピューティング環境に p2、p3、p4、p5、g3、g3s、g4または g5 ファミリーのインスタンスタイプを含める必要があります。

> **Note**
>
> 現在、optimal は C4、M4、および R4 インスタンスファミリーのインスタンスタイプを使用しています。これらのインスタンスファミリーのインスタンスタイプ AWS リージョン がない では、、C5、M5および インスタンスファミリーのR5インスタンスタイプが使用されます。

18. (オプション) 追加設定] を展開します。

    a. (オプション) プレイスメントグループ では、プレイスメントグループ名を入力して、コンピューティング環境内のリソースをグループ化します。

    b. 配分戦略 で、BEST_FIT_PROGRESSIVE を選択します。

    c. (オプション) Amazon マシンイメージ (AMI) 設定 で、Amazon マシンイメージ (AMI) 設定の追加 を選択します。次に、イメージタイプ を選択し、イメージ ID のオーバーライド と Kubernetes バージョンを入力します。

> **Important**
>
> カスタム AMI を使用するには、イメージタイプを選択し、イメージ ID のオーバーライド ボックスにカスタム AMI ID を入力します。

> **Note**
>
> 各イメージタイプ にイメージ ID オーバーライドが指定されていない場合、 は最新の Amazon ECS 最適化 AMI AWS Batch を選択します。イメージタイプが指定されていない場合、デフォルトは GPU 以外の AWS Graviton インスタンスの Amazon Linux 2 です。
>
> Amazon Linux 2
>
> すべての AWS Graviton ベースのインスタンスファミリー (、、C6gM6g、 などT4g) のデフォルトでありR6g、すべての非 GPU インスタンスタイプに使用できます。
>
> Amazon Linux 2 (GPU)
>
> すべての GPU インスタンスファミリー (P4や などG4) のデフォルト。 およびは、すべての AWS Graviton ベース以外のインスタンスタイプに使用できます。

d. (オプション) 起動テンプレート では、既存の起動テンプレートを選択します。

e. (オプション) 起動テンプレートのバージョン では、**$Default** または **$Latest** を使用するか、あるいは起用するバージョン番号を指定します。

19. 次のページ を選択します。

20. 仮想プライベートクラウド (VPC) ID で、インスタンスを起動する先の VPC を選択します。

21. サブネット では、使用するサブネットを選択します。デフォルトでは、選択した VPC 内のすべてのサブネットを使用できます。

> **Note**
>
> AWS Batch Amazon EKS の はローカルゾーンをサポートします。詳細については、「Amazon EKS ユーザーガイド」の「Amazon EKS と AWS ローカルゾーン」を参照してください。

22. (オプション) セキュリティグループ で、インスタンスにアタッチするセキュリティグループを選択します。デフォルトでは、VPC のデフォルトのセキュリティグループが選択されます。
23. 次のページ を選択します。
24. レビュー では、設定手順を確認します。変更する必要がある場合は、編集] を選択します。完了したら、コンピューティング環境の作成 を選択します。

コンピューティング環境テンプレート

次の例では、空のコンピューティング環境テンプレートを示しています。このテンプレートを使ってコンピューティング環境を作成し、それをファイルに保存して AWS CLI `--cli-input-json` オプションで使用できます。これらのパラメータの詳細については、「CreateComputeEnvironment」(AWS BatchAPI リファレンス) を参照してください。

```
{
    "computeEnvironmentName": "",
    "type": "UNMANAGED",
    "state": "DISABLED",
    "unmanagedvCpus": 0,
    "computeResources": {
        "type": "EC2",
        "allocationStrategy": "BEST_FIT_PROGRESSIVE",
        "minvCpus": 0,
        "maxvCpus": 0,
        "desiredvCpus": 0,
        "instanceTypes": [
            ""
        ],
        "imageId": "",
        "subnets": [
            ""
        ],
        "securityGroupIds": [
            ""
        ],
        "ec2KeyPair": "",
        "instanceRole": "",
        "tags": {
            "KeyName": ""
        },
        "placementGroup": "",
```

```
        "bidPercentage": 0,
        "spotIamFleetRole": "",
        "launchTemplate": {
            "launchTemplateId": "",
            "launchTemplateName": "",
            "version": ""
        },
        "ec2Configuration": [
            {
                "imageType": "",
                "imageIdOverride": "",
                "imageKubernetesVersion": ""
            }
        ]
    },
    "serviceRole": "",
    "tags": {
        "KeyName": ""
    },
    "eksConfiguration": {
        "eksClusterArn": "",
        "kubernetesNamespace": ""
    }
}
```

> **Note**
>
> 先のコンピューティング環境テンプレートは、以下の AWS CLI コマンドで生成できます。
>
> ```
> $ aws batch create-compute-environment --generate-cli-skeleton
> ```

コンピューティング環境のパラメータ

コンピューティング環境は、コンピューティング環境の名前、タイプ、状態、コンピューティングリソース定義 (マネージドコンピューティング環境の場合)、Amazon EKS 設定 (Amazon EKS リソースを使用する場合)、 への IAM アクセス許可の提供に使用するサービスロール AWS Batch、コンピューティング環境のタグなど、いくつかの基本コンポーネントに分割されます。

トピック

コンピューティング環境の名前

`computeEnvironmentName`

コンピューティング環境の名前です。名前の最大長は 128 文字です。大文字、小文字、数字、ハイフン (-)、アンダースコア (_) を含めることができます。

型: 文字列

必須: はい

タイプ

`type`

コンピューティング環境のタイプ。定義した EC2 または Fargate コンピューティングリソースを が AWS Batch 管理MANAGEDすることを選択します。詳細については、「コンピューティングリソース」を参照してください。独自の EC2 コンピューティングリソースを管理する場合は、UNMANAGED を選択します。

タイプ: 文字列

有効な値: `MANAGED` | `UNMANAGED`

必須: はい

状態

`state`

コンピューティング環境の状態。

状態が の場合ENABLED、ス AWS Batch ケジューラは環境内にジョブを配置しようとします。これらのジョブは、コンピューティングリソース上の関連するジョブキューから取得されます。コンピューティング環境がマネージド型の場合、そのインスタンスはジョブキューの需要に基づいて自動的にスケールアウトまたはスケールインされます。

状態が の場合DISABLED、ス AWS Batch ケジューラは環境内にジョブを配置しようとしません。状態が `STARTING` または `RUNNING` のジョブは、引き続き正常に進行します。`DISABLED` 状態のマネージド型コンピューティング環境はスケールアウトされません。

> **Note**
>
> `DISABLED` 状態のコンピューティング環境では、引き続き課金が発生する可能性があります。追加料金が発生しないようにするには、コンピューティング環境をオフにしてから削除してください。詳細については、 AWS Batch 「 API リファレンスDeleteComputeEnvironment」の「」および「 ユーザーガイド」の「予期しない料金の回避」を参照してください。 AWS Billing

インスタンスがアイドル状態になると、インスタンスはその `minvCpus` 値までスケールダウンします。ただし、インスタンスのサイズは変更されません。たとえば、`minvCpus` の値が `4` で、`desiredvCpus` の値が`36` である `c5.8xlarge` インスタンスを考えてみましょう。このインスタンスは、`c5.large` インスタンスにスケールダウンしません。

タイプ: 文字列

有効な値: `ENABLED` | `DISABLED`

必須: いいえ

コンピューティングリソース

`computeResources`

コンピューティング環境によって管理されるコンピューティングリソースの詳細。詳細については、コンピューティング環境を参照してください。

型: ComputeResource オブジェクト

必須: このパラメータは、マネージド型のコンピューティング環境では必須です。

`type`

コンピューティング環境のタイプ。EC2 オンデマンドインスタンス (EC2) と EC2 スポットインスタンス (SPOT) を使用するか、マネージドコンピューティング環境で Fargate 容量 (FARGATE) および Fargate スポット容量 (FARGATE_SPOT) を使用するかを選択できます。SPOT を選択した場合は、`spotIamFleetRole` パラメータで Amazon EC2 スポットフリートロールも指定する必要があります。詳細については、Amazon EC2 スポットフリートロールを参照してください。

有効な値: EC2 | SPOT | FARGATE | FARGATE_SPOT

必須: はい

`allocationStrategy`

最適な EC2 インスタンスタイプのインスタンスを十分に配分できない場合に、コンピューティングリソースに使用する配分戦略です。これは、 または AWS リージョン Amazon EC2 サービスの制限 でインスタンスタイプの可用性が原因である可能性があります。詳細については、「配分戦略」を参照してください。

> **Note**
>
> このパラメータは、Fargate リソースで実行されているジョブには適用されません。

`BEST_FIT` (デフォルト)

AWS Batch は、低コストのインスタンスタイプを優先して、ジョブのニーズに最適なインスタンスタイプを選択します。選択したインスタンスタイプの追加のインスタンスが使用できない場合、 は追加のインスタンスが使用可能になるまで AWS Batch 待機します。十分なインスタンスを利用できない場合、またはAmazon EC2 サービスの制限に達した場

合、現在実行中のジョブが完了するまで追加のジョブは実行されません。この配分戦略により、コストは低くなりますが、スケーリングは制限される場合があります。BEST_FIT でスポットフリートを使用する場合は、スポットフリートの IAM ロールを指定する必要があります。BEST_FIT 配分戦略を使用するコンピューティングリソースはインフラストラクチャの更新をサポートしておらず、一部のパラメータを更新できません。詳細については、コンピューティング環境を更新します。を参照してください。

> **Note**
>
> Amazon EKS リソースを使用するコンピューティング環境では、BEST_FIT はサポートされません。

`BEST_FIT_PROGRESSIVE`

キュー内のジョブの要件を満たすのに十分な大きさのインスタントタイプを追加で選択します。単位 vCPU あたりのコストが低いインスタンスタイプが優先されます。以前に選択したインスタンスタイプの追加のインスタンスが使用できない場合、 は新しいインスタンスタイプ AWS Batch を選択します。

`SPOT_CAPACITY_OPTIMIZED`

(スポットインスタンスのコンピューティング環境でのみ利用可能) キュー内のジョブの要件を満たすのに十分な大きさの追加のインスタンスタイプを選択します。中断されにくいインスタンスタイプが優先されます。

`SPOT_PRICE_CAPACITY_OPTIMIZED`

(スポットインスタンスのコンピューティング環境でのみ利用可能) 価格とキャパシティで最適化された配分戦略では、価格とキャパシティの両方を考慮して、中断される可能性が最も低く、価格ができるだけ低いスポットインスタンスプールを選択します。

> **Note**
>
> ほとんどの場合、SPOT_CAPACITY_OPTIMIZED ではなく SPOT_PRICE_CAPACITY_OPTIMIZED を使用することをお勧めします。

オンデマンドインスタンスまたはスポットインスタンスを使用する BEST_FIT_PROGRESSIVE、SPOT_CAPACITY_OPTIMIZED、および

SPOT_PRICE_CAPACITY_OPTIMIZED戦略とスポットインスタンスを使用する BEST_FIT戦略maxvCpusでは、容量要件を満たすために を超える必要がある AWS Batch 場合があります。このイベントでは、 が複数のインスタンスmaxvCpusを超える AWS Batch ことはありません。

有効な値: BEST_FIT | BEST_FIT_PROGRESSIVE | SPOT_CAPACITY_OPTIMIZED | SPOT_PRICE_CAPACITY_OPTIMIZED

必須: いいえ

minvCpus

コンピューティング環境が DISABLED であっても、コンピューティング環境で維持する vCPU の最小数。

> **Note**
> このパラメータは、Fargate リソースで実行されているジョブには適用されません。

タイプ: 整数

必須: いいえ

maxvCpus

AWS Batch コンピューティング環境における vCPU の最大数。

> **Note**
> オンデマンドインスタンスまたはスポットインスタンスを使用する BEST_FIT_PROGRESSIVESPOT_CAPACITY_OPTIMIZED、、および SPOT_PRICE_CAPACITY_OPTIMIZED配分戦略とスポットインスタンスを使用する BEST_FIT戦略maxvCpusでは、容量要件を満たすために を超える必要がある AWS Batch 場合があります。このイベントでは、 が複数のインスタンスmaxvCpusを超える AWS Batch ことはありません。例えば、 は、コンピューティング環境で指定されたインスタンスの中から 1 つ以上のインスタンス AWS Batch のみを使用します。

タイプ: 整数

必須: いいえ

`desiredvCpus`

コンピューティング環境で必要な vCPUS の数。 は、ジョブキューの需要に基づいて、この値を最小値と最大値の間で AWS Batch 変更します。

> **Note**
> このパラメータは、Fargate リソースで実行されているジョブには適用されません。

タイプ: 整数

必須: いいえ

`instanceTypes`

起動される可能性のあるインスタンスタイプ。このパラメータは、Fargate リソースで実行されているジョブには適用されません。指定しません。インスタンスファミリーを指定してそのファミリー内のいずれかのインスタンスタイプ (`c5`、`c5n`、`p3`など) を起動できます。または、ファミリー内の特定のサイズを指定することもできます (`c5.8xlarge`)。メタルインスタンスタイプはインスタンスファミリーに含まれません (たとえば、`c5` に `c5.metal` は含まれません)。また、`optimal` を選択して (C4、M4、および R4 インスタンスファミリーから) ジョブキューの需要に見合ったインスタンスタイプを使用することもできます。

> **Note**
> コンピューティング環境を作成する際、そのコンピューティング環境で選択するインスタンスタイプで同じアーキテクチャを使用する必要があります。例えば、x86 と ARM インスタンスを同じコンピューティング環境で使用することはできません。

> **Note**
> 現在、`optimal` は C4、M4、および R4 インスタンスファミリーのインスタンスタイプを使用しています。これらのインスタンスファミリーのインスタンスタイプを持たない AWS リージョン では、C5、M5、および R5 インスタンスファミリーのインスタンスタイプが使用されます。

タイプ：文字列の配列

必須: はい

`imageId`

このパラメータは廃止されました。

コンピューティング環境で起動されるインスタンスに使用する Amazon マシンイメージ (AMI) の ID。このパラメータは、`Ec2Configuration` 構造の `imageIdOverride` メンバーによってオーバーライドされます。

> **Note**
> このパラメータは、Fargate リソースで実行されているジョブには適用されません。

> **Note**
> コンピューティング環境用に選択する AMI は、そのコンピューティング環境で使用するインスタンスタイプのアーキテクチャと一致している必要があります。たとえば、コンピューティング環境で A1 インスタンスタイプを使用する場合、選択するコンピューティングリソース AMI で Arm インスタンスをサポートしている必要があります。Amazon ECS は、Amazon ECS に最適化された Amazon Linux 2 AMI の、x86 と Arm の両バージョンを提供しています。詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS に最適化された Amazon Linux 2 AMIを参照してください。

タイプ: 文字列

必須: いいえ

`subnets`

コンピューティングリソースを起動する先の VPC サブネット。これらのサブネットは、同じ VPC 内にある必要があります。Fargate のコンピューティングリソースには、最大 16 個のサブネットを含めることができます。詳細については、Amazon VPC ユーザーガイドのVPC とサブネットを参照してください。

> **Note**
>
> AWS Batch Amazon EC2 の と Amazon EKS AWS Batch の は、ローカルゾーンをサポートします。詳細については、https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html?icmpid=docs_ec2_console#concepts-local-zonesAmazon EC2 ユーザーガイド」のAWS 「Local Zones」、「Amazon EKS ユーザーガイド」の「Amazon EKS および Local Zones」、および「Amazon Elastic Container Service デベロッパーガイド」の「Local Zones」、「Wavelength Zones」、および「Amazon ECS クラスター AWS Outposts」を参照してください。AWS Batch Fargate の は、現在ローカルゾーンをサポートしていません。

コンピューティング環境を更新するときに VPC サブネットの空のリストを指定した場合、結果の動作は Fargate と EC2 コンピューティングリソースで異なります。Fargate コンピューティングリソースの場合、空のリストを指定すると、このパラメータが指定されていないものとして処理され、変更も行われません。EC2 コンピューティングリソースの場合、空のリストを指定すると、コンピューティングリソースから VPC サブネットが削除されます。VPC サブネットを変更するには、コンピューティング環境のインフラストラクチャを更新する必要があります。これは、Fargate と EC2 コンピューティングリソースの両方に当てはまります。詳細については、コンピューティング環境を更新します。を参照してください。

タイプ：文字列の配列

必須: はい

`securityGroupIds`

コンピューティング環境で起動されるインスタンスに関連付けられる Amazon EC2 セキュリティグループ。`securityGroupIds` で、または `launchTemplate` で参照されている起動テンプレートを使用して、1 つ以上のセキュリティグループを指定する必要があります 。このパラメータは、Fargate リソースで実行されるジョブに必要であり、少なくとも 1 つのセキュリティグループを含める必要があります。(Fargate は起動テンプレートをサポートしていません。) `securityGroupIds` と `launchTemplate` の両方を使用してセキュリティグループを指定した場合は、`securityGroupIds` の値が使用されます。

コンピューティング環境を更新する場合、セキュリティグループの空のリストを指定すると、Fargate と EC2 のコンピューティングリソースで結果の動作が異なります。Fargate コンピューティングリソースの場合、空のリストを指定すると、このパラメータが指定されていないものとして処理され、変更も行われません。EC2 コンピューティングリソースの場合、空

のリストを指定すると、コンピューティングリソースからセキュリティグループが削除されます。セキュリティグループを変更するには、コンピューティング環境のインフラストラクチャを更新する必要があります。これは、Fargate と EC2 コンピューティングリソースの両方に当てはまります。詳細については、コンピューティング環境を更新します。を参照してください。

タイプ：文字列の配列

必須: はい

`ec2KeyPair`

コンピューティング環境で起動されるインスタンスに使用する EC2 キーペア。このキーペアを使用して、SSH でインスタンスにログインできます。コンピューティング環境を更新する場合、EC2 キーペアを変更するには、コンピューティング環境のインフラストラクチャを更新する必要があります。詳細については、コンピューティング環境を更新します。を参照してください。

> **Note**
>
> このパラメータは、Fargate リソースで実行されているジョブには適用されません。

タイプ: 文字列

必須: いいえ

`instanceRole`

コンピューティング環境内の Amazon EC2 インスタンスにアタッチされる Amazon ECS インスタンスプロファイル。このパラメータは、Fargate リソースで実行されているジョブには適用されません。指定しません。インスタンスプロファイルの短縮名、または、完全な Amazon リソースネーム (ARN) を指定できます。例えば、`ecsInstanceRole`、`arn:aws:iam::`*`aws_account_id`*`:instance-profile/ecsInstanceRole` などです。詳細については、Amazon ECS インスタンスロールを参照してください。

コンピューティング環境を更新する場合、この設定を変更するには、コンピューティング環境のインフラストラクチャを更新する必要があります。詳細については、コンピューティング環境を更新します。を参照してください。

型: 文字列

必須: いいえ

`tags`

コンピューティング環境で起動される EC2 インスタンスにタグとして適用するキーバリューペア。たとえば、タグとして `"Name": "AWS Batch Instance - C4OnDemand"` を指定し、その名前をコンピューティング環境の各インスタンスに使用できます。これは、Amazon EC2 コンソールで AWS Batch インスタンスを認識するのに役立ちます。これらのタグは、API オペレーションの使用 AWS BatchListTagsForResource時には表示されません。

コンピューティング環境を更新する場合、EC2 タグを変更するには、コンピューティング環境のインフラストラクチャを更新する必要があります。詳細については、コンピューティング環境を更新します。を参照してください。

> **Note**
>
> このパラメータは、Fargate リソースで実行されているジョブには適用されません。

型: 文字列間のマッピング

必須: いいえ

`placementGroup`

コンピューティングリソースに関連付ける Amazon EC2 プレイスメントグループ。このパラメータは、Fargate リソースで実行されているジョブには適用されません。指定しません。マルチノードの並列ジョブをコンピューティング環境に送信する場合、クラスタープレイスメントグループを作成してコンピューティングリソースに関連付けることを検討してください。これにより、論理的にグループ化されたインスタンス上のマルチノードの並列ジョブが、潜在的に高度なネットワークフローを備えた単一のアベイラビリティーゾーン内に保持されます。詳細については、Linux インスタンス用 Amazon EC2 ユーザーガイドのプレイスメントグループを参照してください。

> **Note**
>
> このパラメータは、Fargate リソースで実行されているジョブには適用されません。

タイプ: 文字列

必須: いいえ

`bidPercentage`

インスタンス起動前のインスタンスタイプのオンデマンド料金と対比した EC2 スポットインスタンス料金の最大パーセンテージ。たとえば、最大割合が 20% の場合、その EC2 インスタンスのスポット料金は現在のオンデマンド価格の 20% 未満にする必要があります。支払い額は常に最低 (市場料金) となり、最大パーセンテージを超えることはありません。このフィールドを空のままにした場合、デフォルト値はオンデマンド料金の 100% です。大部分のユースケースでは、このフィールドは空欄のままにしておくことをお勧めします。

コンピューティング環境を更新する場合、入札パーセンテージを変更するには、コンピューティング環境のインフラストラクチャを更新する必要があります。詳細については、コンピューティング環境を更新します。を参照してください。

> **Note**
>
> このパラメータは、Fargate リソースで実行されているジョブには適用されません。

必須: いいえ

`spotIamFleetRole`

SPOT コンピューティング環境に適用する Amazon EC2 スポットフリートの IAM ロールの Amazon リソースネーム (ARN)。このロールは、配分戦略が BEST_FIT に設定されている場合、または配分戦略が指定されていない場合に必要です。詳細については、Amazon EC2 スポットフリートロールを参照してください。

> **Note**
>
> このパラメータは、Fargate リソースで実行されているジョブには適用されません。

> **Important**
>
> 作成時にスポットインスタンスにタグを付けるには、ここで指定するスポットフリート IAM ロールで、新しい AmazonEC2SpotFleetTaggingRole 管理ポリシーを使用する必要があります。以前に推奨された AmazonEC2SpotFleetRole 管理ポリシーには、スポットインスタンスにタグを付けるために必要なアクセス許可がありません。詳細に

ついては、「作成時にタグが付けられていないスポットインスタンス」を参照してください。

型: 文字列

必須: このパラメータは、SPOT コンピューティング環境では必須です。

`launchTemplate`

コンピューティングリソースに関連付けるオプションの起動テンプレート。このパラメータは、Fargate リソースで実行されているジョブには適用されません。指定しません。CreateComputeEnvironment または UpdateComputeEnvironment API オペレーションで指定したその他のコンピューティングリソースパラメータによって、起動テンプレートの同じパラメータが上書きされます。起動テンプレートを使用するには、リクエストで起動テンプレート ID または起動テンプレート名のいずれか 1 つを指定します。詳細については、起動テンプレートのサポートを参照してください。

コンピューティング環境を更新するときに、カスタム起動テンプレートを削除してデフォルトの起動テンプレートを使用するには、起動テンプレート仕様の `launchTemplateId` または `launchTemplateName` メンバーを空の文字列に設定します。コンピューティング環境から起動テンプレートを削除しても、起動テンプレートで指定された AMI が使用されていた場合、その AMI は削除されません。起動テンプレートで選択された AMI を更新するには、`updateToLatestImageVersion` パラメータを `true` に設定する必要があります。コンピューティング環境を更新する場合、起動テンプレートを変更するには、コンピューティング環境のインフラストラクチャを更新する必要があります。詳細については、コンピューティング環境を更新します。を参照してください。

タイプ: LaunchTemplateSpecification

オブジェクト

必須: いいえ

`launchTemplateId`

起動テンプレートの ID。

タイプ: 文字列

必須: いいえ

`launchTemplateName`

起動テンプレートの名前。

タイプ: 文字列

必須: いいえ

`version`

起動テンプレートのバージョン番号 (`$Latest` または `$Default`)。

値が `$Latest` の場合、起動テンプレートの最新バージョンを使用します。値が `$Default` の場合、起動テンプレートのデフォルトバージョンを使用します。インフラストラクチャの更新中に、コンピューティング環境に `$Latest`または のいずれかが指定されている場合、 `$Default`は起動テンプレートのバージョンを AWS Batch 再評価し、別のバージョンの起動テンプレートを使用する場合があります。これは、起動テンプレートがアップデートで指定されていない場合も同じです。

デフォルト: `$Default`。

タイプ: 文字列

必須: いいえ

`ec2Configuration`

EC2 コンピューティング環境で、インスタンスの Amazon マシンイメージ (AMI) を選択するために使用される情報を提供します。もし `Ec2Configuration` が指定されなかった場合、デフォルトは Amazon Linux 2 (`ECS_AL2`) です。2021 年 3 月 31 日以前は、このデフォルトは GPU 以外の AWS Graviton 以外のインスタンスの Amazon Linux (`ECS_AL1`) でした。

コンピューティング環境を更新する場合、このパラメータを変更するには、コンピューティング環境のインフラストラクチャを更新する必要があります。詳細については、コンピューティング環境を更新します。 を参照してください。

> **Note**
>
> このパラメータは、Fargate リソースで実行されているジョブには適用されません。

型: Ec2Configuration オブジェクトの配列

必須: いいえ

`imageIdOverride`

コンピューティング環境で起動され、イメージタイプが一致するインスタンスに使用される AMI ID。この設定は、`computeResource` オブジェクト内の `imageId` セットをオーバーライドします。

タイプ: 文字列

必須: いいえ

`imageKubernetesVersion`

コンピューティング環境の Kubernetes のバージョン。値を指定しない場合、 AWS Batch でサポートされる最新バージョンが使用されます。

型: 文字列

長さの制限：最小長は 1 です。最大長は 256 です。

必須: いいえ

`imageType`

AMI の選択のために、インスタンスタイプと一致させるイメージタイプ。サポートされている値は、`ECS` リソースと `EKS` リソースで異なります。

ECS

`imageIdOverride` パラメータが指定されていない場合、最新の Amazon ECS に最適化されたAmazon Linux 2 AMI (`ECS_AL2`) が使用されます。新しいイメージタイプが更新で指定されているが、 `imageId`も `imageIdOverride`パラメータも指定されていない場合、 でサポートされているそのイメージタイプの最新の Amazon ECS 最適化 AMI AWS Batch が使用されます。

ECS_AL2

Amazon Linux 2: 非 GPU インスタンスファミリーのデフォルトです。

ECS_AL2_NVIDIA

Amazon Linux 2 (GPU): すべての GPU インスタンスファミリー (`P4`や などG4) のデフォルトであり、すべての AWS Graviton ベース以外のインスタンスタイプに使用できます。

ECS_AL1

Amazon Linux。Amazon Linux は標準サポート end-of-life の に達しました。詳細については、Amazon Linux AMI を参照してください。

EKS

`imageIdOverride` パラメータが指定されていない場合、最新の Amazon EKS に最適化された Amazon Linux AMI (EKS_AL2) が使用されます。新しいイメージタイプが更新で指定されているが、 `imageId`も `imageIdOverride`パラメータも指定されていない場合、 が AWS Batch サポートするそのイメージタイプの最新の Amazon EKS 最適化 AMI が使用されます。

EKS_AL2

Amazon Linux 2: 非 GPU インスタンスファミリーのデフォルトです。

EKS_AL2_NVIDIA

Amazon Linux 2 (高速) : すべての GPU インスタンスファミリー (例: P4および G4) のデフォルトであり、すべての AWS Graviton ベース以外のインスタンスタイプに使用できます。

型: 文字列

長さの制限：最小長は 1 です。最大長は 256 です。

必須: はい

Amazon EKS 設定

AWS Batch コンピューティング環境をサポートする Amazon EKS クラスターの構成。コンピューティング環境を作成する前に、クラスターが存在している必要があります。

`eksClusterArn`

Amazon EKS クラスターの Amazon リソースネーム (ARN)。例は `arn:aws:eks:`*`us-east-1`*`:`*`123456789012`*`:cluster/`*`ClusterForBatch`* です。

型: 文字列

必須: はい

`kubernetesNamespace`

この名前空間の Amazon EKS cluster. AWS Batch manages ポッドの名前空間。値を空または null にすることはできません。64 文字未満でなければならず、`default` に設定できません。また、`kube-`で始めることはできません。次の正規表現と一致する必要があります: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`。詳細については、Kubernetes ドキュメントの名前空間を参照してください。

型: 文字列

必須: はい

タイプ: EksConfiguration Object

必須: いいえ

サービスロール

`serviceRole`

がユーザーに代わって他の AWS サービスを呼び出す AWS Batch ことができる IAM ロールの完全な Amazon リソースネーム (ARN)。詳細については、「のサービスにリンクされたロールの使用 AWS Batch」を参照してください。サービスロールは指定しないことをお勧めします。これにより、 はAWSServiceRoleForBatchサービスにリンクされたロール AWS Batch を使用します。

> **Important**
>
> アカウントで AWS Batch サービスにリンクされたロール (AWSServiceRoleForBatch) がすでに作成されている場合、ここでロールを指定しない限り、そのロールはコンピューティング環境にデフォルトで使用されます。 AWS Batch サービスにリンクされたロールがアカウントに存在せず、ここでロールが指定されていない場合、サービスは AWS Batch サービスにリンクされたロールをアカウントに作成しようとします。AWSServiceRoleForBatch サービスにリンクされたロールの詳細については、のサービスにリンクされたロールのアクセス許可 AWS Batchを参照してください。AWSServiceRoleForBatch サービスにリンクされたロールを使用してコンピューティング環境を作成した場合、通常の IAM ロールを使用するように変更することはできません。同様に、コンピューティング環境が通常の IAM ロールで作成されている場合、AWSServiceRoleForBatch サービスにリンクされたロールを使用するように変更することはできません。インフラストラクチャの更新が必要となるコンピューティング環境の

パラメータを更新するには、AWSServiceRoleForBatch サービスにリンクされたロールを使用する必要があります。詳細については、コンピューティング環境を更新します。を参照してください。

指定したロールに / 以外のパスがある場合は、必ず完全なロール ARN を指定するか (推奨)、ロール名の前にパスを付けてください。

> **Note**
>
> AWS Batch サービスロールの作成方法によっては、その Amazon リソースネーム (ARN) に`service-role`パスプレフィックスが含まれている場合があります。サービスロールの名前のみを指定すると、 AWS Batch は ARN が`service-role`パスプレフィックスを使用しないと仮定します。このため、コンピューティング環境を作成するときには、サービスロールに完全 ARN を指定することが推奨されます。

タイプ: 文字列

必須: いいえ

タグ

`tags`

コンピューティング環境に関連したキーバリューペアのタグ。詳細については、AWS Batch リソースのタグ付けを参照してください。

型: 文字列間のマッピング

必須: いいえ

EC2 設定

AWS Batch EC2 および EC2 スポットコンピューティング環境用に Amazon ECS に最適化された AMI を使用しています。デフォルトはAmazon Linux 2 (ECS_AL2) です。2021 年 3 月 31 日より前のバージョンでは、GPU AWS インスタンスもグラビトンインスタンスもデフォルトは Amazon Linux (ECS_AL1) でした。

> **Note**
>
> AWS Batch Amazon Linux 2023 もサポートしています。

Amazon Linux AMI (Amazon Linux 1 とも呼ばれる) は、2023 年 12 月 31 日にサポート終了となりました。 AWS Batch は Amazon Linux AMI のサポートを終了しました。これは、2024 年 1 月 1 日以降、セキュリティアップデートやバグ修正が行われないためです。Amazon Linux の詳細については end-of-life、「よくあるご質問」を参照してください。

予期しないワークロードの中断を防ぐため、既存の Amazon Linux ベースのコンピューティング環境を Amazon Linux 2023 に更新し、引き続きセキュリティやその他の更新を受けることをお勧めします。

Amazon Linux AMI を使用するコンピューティング環境は、2023 年 12 月 31 end-of-life 日以降も引き続き機能する可能性があります。ただし、これらのコンピューティング環境には、 AWSから新しいソフトウェアアップデート、セキュリティパッチ、またはバグ修正が適用されなくなります。その後、Amazon Linux AMI end-of-life でこれらのコンピューティング環境を維持するのはお客様の責任です。最適なパフォーマンスとセキュリティを維持するために、 AWS Batch コンピューティング環境を Amazon Linux 2023 または Amazon Linux 2 に移行することをお勧めします。

Amazon Linux AMI AWS Batch から Amazon Linux 2023 または Amazon Linux 2 への移行に関するヘルプについては、「コンピューティング環境の更新」を参照してください- AWS Batch

配分戦略

マネージドコンピューティング環境を作成すると、 AWS Batch `instanceTypes`ジョブのニーズに最適なインスタンスタイプを指定から選択します。割り当て戦略は、 AWS Batch 追加の容量が必要な場合の動作を定義します。このパラメータは、Fargate リソースで実行されているジョブには適用されません。このパラメータを指定しないようにしてください。

`BEST_FIT` (デフォルト)

AWS Batch 最も低コストのインスタンスタイプを優先して、ジョブのニーズに最も適したインスタンスタイプを選択します。選択したインスタンスタイプのインスタンスが他にない場合は、AWS Batch 追加のインスタンスが使用可能になるまで待ちます。十分なインスタンスが利用できない場合、またはユーザーが Amazon EC2 Service Quotasに達した場合、現在実行中のジョブが完了するまで追加のジョブは実行されません。この配分戦略により、コストは低くなります

が、スケーリングは制限される場合があります。BEST_FIT でスポットフリートを使用する場合は、スポットフリートの IAM ロールを指定する必要があります。BEST_FIT コンピューティング環境の更新時にはサポートされません。詳細については、「コンピューティング環境を更新します。」を参照してください。

> **Note**
>
> AWS Batch AWS アカウント内のリソースを管理します。BEST_FIT 配分戦略を採用したコンピューティング環境では、もともとデフォルトで起動設定が使用されていました。ただし、 AWS 新しいアカウントでの起動設定の使用は、時間が経つにつれて制限されます。そのため、2024 年 4 月下旬から、新しく作成された BEST_FIT コンピューティング環境では、デフォルトで起動テンプレートが使用されるようになります。サービスロールに起動テンプレートを管理する権限がない場合でも、 AWS Batch 引き続き起動設定を使用できます。既存のコンピューティング環境は引き続き起動設定を使用します。

`BEST_FIT_PROGRESSIVE`

AWS Batch キュー内のジョブの要件を満たすのに十分な大きさのインスタンスタイプを追加で選択します。単位 vCPU あたりのコストが低いインスタンスタイプが優先されます。以前選択したインスタンスタイプの追加のインスタンスが利用できない場合、 AWS Batch は新しいインスタンスタイプを選択します。

`SPOT_CAPACITY_OPTIMIZED`

AWS Batch キュー内のジョブの要件を満たすのに十分な大きさのインスタンスタイプを 1 つ以上選択します。中断されにくいインスタンスタイプが推奨されます。この配分戦略は、スポットインスタンスのコンピューティングリソースでのみ使用できます。

`SPOT_PRICE_CAPACITY_OPTIMIZED`

料金とキャパシティの最適化配分戦略では、料金とキャパシティの両方を考慮して、中断される可能性が最も低く、料金が最も低いスポットインスタンスプールを選択します。この配分戦略は、スポットインスタンスのコンピューティングリソースでのみ使用できます。

> **Note**
>
> ほとんどの場合、`SPOT_CAPACITY_OPTIMIZED` ではなく `SPOT_PRICE_CAPACITY_OPTIMIZED` を使用することをお勧めします。

BEST_FIT_PROGRESSIVE および BEST_FIT ストラテジーはオンデマンドまたはスポットインスタンスを使用し、SPOT_CAPACITY_OPTIMIZED および SPOT_PRICE_CAPACITY_OPTIMIZED ストラテジーはスポットインスタンスを使用します。ただし、 AWS Batch maxvCpusキャパシティ要件を満たすにはそれを超える必要があるかもしれません。この場合、1 AWS Batch maxvCpus つのインスタンスしか超えないようにしてください。

コンピューティング環境を更新します。

EC2 リソースを使用するコンピュート環境を作成したら、コンピュート環境の多くの設定を直接更新できます。ただし、一部の設定を変更すると、AWS Batch コンピュート環境内のインスタンスを置き換える必要があります。

Fargate リソースを使用するコンピュート環境では、以下を更新できます。

- securityGroupIds
- subnets
- desiredvCpus
- maxvCpus
- minvCpus

AWS Batch には 2 つの更新メカニズムがあります。1 つ目は、コンピュート環境にインスタンスを追加または削除するスケーリングの更新です。2 つ目は、コンピュート環境内のインスタンスを置き換えるインフラストラクチャーの更新です。インフラストラクチャーの更新には、スケーリングの更新よりもずっと時間がかかります。

でコンピューティング環境を更新する場合 AWS Batch、必要な vCPUs (desiredvCpus)、最大仮想 CPU (maxvCpus)、最小仮vCPUs (minvCpus)、サービスロール (serviceRole)、状態 (state) の設定のみを変更すると、スケーリングが更新されます。

> **Note**
>
> desiredvCpus 設定を更新する場合、値は minvCpus と maxvCpus 値の間になければなりません。
> さらに、desiredvCpus 更新後の値は、現在の desiredvCpus 値以上にする必要があります。詳細については、the section called “desiredvCpus 設定を更新すると、エラーメッセージが表示されます”を参照してください。

UpdateComputeEnvironment API アクション AWS Batch で以下の設定のいずれかが変更されると、インフラストラクチャの更新が開始されます。インフラストラクチャを更新するには、サービスロールを AWSServiceRoleForBatch (デフォルト) に設定し、割り当て戦略は、BEST_FIT_PROGRESSIVE、SPOT_CAPACITY_OPTIMIZED、SPOT_PRICE_CAPACITY_OPTIMIZED または BEST_FIT にサポートされていません。サービスロールを除いて、スケーリングアップデートで変更できる設定はすべて、インフラストラクチャーアップデートでも変更できます。

> **Note**
>
> ほとんどの場合、SPOT_CAPACITY_OPTIMIZED ではなく SPOT_PRICE_CAPACITY_OPTIMIZED を使用することをおすすめします。

インフラストラクチャーの更新中、コンピューティング環境のステータスは UPDATING に変わります。新しいインスタンスは、更新された設定を使用して起動されます。新しいジョブは、新しいインスタンスでスケジュールされます。現在実行中のジョブは、インフラストラクチャー更新ポリシーに従ってディスパッチされます。詳細については、AWS Batch API リファレンスのUpdateComputeEnvironment とUpdatePolicy を参照してください。

UpdatePolicy データ型では、次のシナリオを想定してください：

> **Note**
>
> これらのシナリオでは、以下のことが当てはまります。インスタンスが終了すると、実行中のジョブは停止します。デフォルトでは、これらのジョブは再試行されません。インスタンスが終了した後にこれらのジョブの 1 つを再試行するには、ジョブの再試行戦略を設定します。詳細については、AWS Batch ユーザーガイドの the section called “ジョブの再試行の自動化”を参照してください。

- terminateJobsOnUpdate 設定が true に設定されている場合、実行中のジョブはインフラストラクチャーの更新中に終了します。この jobExecutionTimeoutMinutes の設定は無視されます。
- terminateJobsOnUpdate 設定がに設定されている場合、false インフラストラクチャーの更新後にジョブを実行できる時間が長くなります。この追加時間は jobExecutionTimeoutMinutes 設定で設定されます。デフォルトは 5 分 (300 秒) です。

コンピュート環境でキャパシティが利用可能になると、新しいインスタンスが更新された設定で起動され、新しいインスタンスでジョブが開始されます。古い設定のインスタンスで、すべてのジョブが完了すると、古いインスタンスは終了します。キャパシティが利用可能になるということは、必要な vCPUs の数が、最低でも最小のインスタンスタイプで必要な vCPUs の数を vCPUs の最大数より少なくなるということです。

インフラストラクチャの更新

コンピュート環境の一部の設定を変更するには、インフラストラクチャーの更新が必要です。以下の設定のいずれかが変更されると、インフラストラクチャーの更新が開始されます。

> **Important**
>
> コンピュート環境では、インフラストラクチャの更新を必要とする変更を行うには、AWSServiceRoleForBatch サービスにリンクされたロールを使用する必要があります。コンピュート環境がサービスにリンクされたロールを使用している場合、通常の IAM ロールを使用するように変更することはできません。コンピューティング環境がサービスにリンクされたロールを使用している場合、通常の IAM ロールを使用するように変更することはできません。そのため、インフラストラクチャーの更新は、サービスにリンクされたロールを使用して作成されたコンピュート環境でのみ実行できます。

- 割り当て戦略 (`allocationStrategy`、`BEST_FIT_PROGRESSIVE`、`SPOT_CAPACITY_OPTIMIZED` または `SPOT_PRICE_CAPACITY_OPTIMIZED` のいずれかである必要があります。元の割り当て戦略が `BEST_FIT` の場合、インフラ更新はサポートされません)。

  > **Note**
  >
  > ほとんどの場合、`SPOT_CAPACITY_OPTIMIZED` ではなく `SPOT_PRICE_CAPACITY_OPTIMIZED` を使用することをおすすめします。

- 入札率 (`bidPercentage`)
- EC2 設定(`ec2Configuration`)
- キーペア(`ec2KeyPair`)
- イメージ ID(`imageId`)
- インスタンスロール(`instanceRole`)

- インスタンスのタイプ(instanceTypes)
- 起動テンプレート(launchTemplate)
- 配置グループ(placementGroup)
- セキュリティグループ(securityGroupIds)
- VPC サブネット(subnets)
- EC2 タグ (tags)()
- コンピューティング環境タイプ (type、EC2 または SPOT のいずれかでも可)
- AWS Batch インフラストラクチャーの更新中にでサポートされている最新の AMI に更新するかどうか updateToLatestImageVersion

AMI ID のアップデート

インフラストラクチャーの更新中に、これら3つの設定のいずれかでAMIが指定されているかどうかに応じて、コンピュート環境のAMI IDが変更される場合があります。AMI は imageId (in computeResources)、imageIdOverride (in ec2Configuration)、または launchTemplate で指定されている起動テンプレートで指定されます。これらの設定のいずれにも AMI ID が指定されておらず、updateToLatestImageVersion 設定が true であるとします。その場合、がサポートしている最新の Amazon ECS 最適化 AMI AWS Batch がインフラストラクチャの更新に使用されます。

これらの設定の少なくとも 1 つで AMI ID が指定されている場合、更新は、更新前に使用された AMI ID が提供された設定によって異なります。コンピュート環境を作成する場合、AMI ID を選択する際の優先順位は、最初に起動テンプレート、次に imageId の設定、最後に imageIdOverride の設定です。ただし、使用される AMI ID が起動テンプレートからのものである場合、imageId または imageIdOverride の設定を更新しても AMI ID は更新されません。起動テンプレートから選択した AMI ID を更新する唯一の方法は、起動テンプレートを更新することです。起動テンプレートのバージョンパラメータが $Default または $Latest の場合、指定された起動テンプレートのデフォルトまたは最新バージョンが評価されます。デフォルトで別の AMI ID が選択されている場合、または起動テンプレートの最新バージョンが選択されている場合、その AMI ID が更新に使用されます。

起動テンプレートを使用して AMI ID を選択しなかった場合は、imageId または imageIdOverride のパラメータで指定されている AMI ID が使用されます。両方を指定すると、imageIdOverride パラメータで指定された AMI ID が使用されます。

コンピュート環境が imageId、imageIdOverride、または launchTemplate パラメータで指定された AMI ID を使用しており、AWS Batch でサポートされている最新の Amazon ECS 最適

化 AMI を使用するとします。次に、更新により AMI ID を提供した設定を削除する必要があります。このため `imageId`、そのパラメータには空の文字列を指定する必要があります。このため `imageIdOverride`、`ec2Configuration` パラメータには空の文字列を指定する必要があります。

AMI ID が起動テンプレートから取得された場合は、AWS Batch次のいずれかの方法でサポートされている最新の Amazon ECS 最適化 AMI に変更できます。

- `launchTemplateId` または `launchTemplateName` パラメータに空の文字列を指定して、起動テンプレートを削除します。これにより、AMI ID だけではなく、起動テンプレート全体が削除されます。
- 更新バージョンの起動テンプレートで AMI ID が指定されていない場合は、`updateToLatestImageVersion` パラメータをに設定する必要があります `true`。

Amazon EKS コンピュート環境

Amazon EKS AWS Batch での の開始方法 EKS コンピュート環境を作成するための簡単なガイドを提供します。このセクションでは、Amazon EKS コンピューティング環境について詳しく説明します。

トピック

- デフォルト AMI 選択
- サポートされる Kubernetes バージョン
- コンピュート環境の Kubernetes バージョンを更新しています。
- Kubernetes ノードの責任分担
- AWS Batch マネージドノードDaemonSetでの の実行
- 起動テンプレートを使用したマネージドノードのカスタマイズ

デフォルト AMI 選択

Amazon EKS コンピューティング環境を作成するときは、Amazon マシンイメージ (AMI) を指定する必要はありません。 AWS Batch は、CreateComputeEnvironmentリクエストで指定されたKubernetesバージョンとインスタンスタイプに基づいて Amazon EKS 最適化 AMI を選択します。一般的に、デフォルトの AMI 選択を使用することをお勧めします。Amazon EKS に最適化された AMI の詳細については、Amazon EKS ユーザーガイドのAmazon EKS に最適化された Amazon Linux AMI を参照してください。

次のコマンドを実行して、Amazon EKS コンピューティング環境でどの AMI タイプ AWS Batch が選択されているかを確認します。次の例は GPU 以外のインスタンスタイプです。

```
# compute CE example: indicates Batch has chosen the AL2 x86 or ARM EKS 1.29 AMI,
 depending on instance types
$ aws batch describe-compute-environments --compute-environments My-Eks-CE1 \
    | jq '.computeEnvironments[].computeResources.ec2Configuration'
[
  {
    "imageType": "EKS_AL2",
    "imageKubernetesVersion": "1.29"
  }
]
```

次の例は GPU インスタンスタイプです。

```
# GPU CE example: indicates Batch has choosen the AL2 x86 EKS Accelerated 1.29 AMI
$ aws batch describe-compute-environments --compute-environments My-Eks-GPU-CE \
    | jq '.computeEnvironments[].computeResources.ec2Configuration'
[
  {
    "imageType": "EKS_AL2_NVIDIA",
    "imageKubernetesVersion": "1.29"
  }
]
```

サポートされる Kubernetes バージョン

AWS Batch Amazon EKS の では、現在次のKubernetesバージョンがサポートされています。

- 1.29
- 1.28
- 1.27
- 1.26
- 1.25
- 1.24
- 1.23

CreateComputeEnvironment API オペレーションまたは API オペレーション UpdateComputeEnvironment を使用してコンピュート環境を作成または更新すると、次のようなエラーメッセージが表示されることがあります。この問題は、Kubernetes でサポートされていないバージョンを EC2Configuration で指定した場合に発生します。

```
At least one imageKubernetesVersion in EC2Configuration is not supported.
```

この問題を解決するには、コンピュート環境を削除し、サポートされている Kubernetes バージョンで再作成してください。

Amazon EKS クラスターでマイナーバージョンアップグレードを実行できます。たとえば、イナーバージョンがサポートされていない場合でも、クラスターを 1.xx から 1.yy マにアップグレードできます。

ただし、INVALID メジャーバージョンを更新すると、コンピュート環境のステータスがに変わることがあります。たとえば、1.xx から 2.yy へのメジャーバージョンアップグレードを実行する場合などです。メジャーバージョンが でサポートされていない場合は AWS Batch、次のようなエラーメッセージが表示されます。

```
reason=CLIENT_ERROR - ... EKS Cluster version [2.yy] is unsupported
```

コンピュート環境の Kubernetes バージョンを更新しています。

では AWS Batch、Amazon EKS クラスターのアップグレードをサポートするようにコンピューティング環境Kubernetesのバージョンを更新できます。コンピューティング環境Kubernetesのバージョンは、ジョブを実行するために が AWS Batch 起動するKubernetesノードの Amazon EKS AMI バージョンです。Amazon EKS クラスターのコントロールプレーンKubernetesのバージョンを更新する前または更新後に、Amazon EKS ノードでバージョンアップグレードを実行できます。コントロールプレーンをアップグレードした後に、ノードを更新することをお勧めします。詳細については、Amazon EKS ユーザーガイドの Amazon EKS クラスターのKubernetesバージョンを更新を参照してください。

コンピュート環境の Kubernetes バージョンをアップグレードするには、UpdateComputeEnvironment API オペレーションを使用します。

```
$ aws batch update-compute-environment \
    --compute-environment <compute-environment-name> \
    --compute-resources \
```

```
    'ec2Configuration=[{imageType=EKS_AL2,imageKubernetesVersion=1.23}]'
```

Kubernetes ノードの責任分担

コンピュート環境のメンテナンスは共同責任です。

- AWS Batch ノード、ラベル、テイント、名前空間、起動テンプレート、または Auto Scaling グループを変更または削除しないでください。 AWS Batch マネージドノードにテイントを追加しないでください。これらの変更を行うと、コンピュート環境がサポートされなくなり、アイドル状態のインスタンスなどの障害が発生します。
- ポッドを AWS Batch マネージドノードにターゲット設定しないでください。ポッドを管理対象ノードにすると、スケーリングが中断したり、ジョブキューが停止したりします。セルフマネージド型ノードまたはマネージド型ノードグループ AWS Batch で を使用しないワークロードを実行します。詳細については、Amazon EKS ユーザーガイドのマネージド型ノードグループを参照してください。
- AWS Batch マネージドノードで実行する DaemonSetをターゲットにすることができます。詳細については、「AWS Batch マネージドノードDaemonSetでの の実行」を参照してください。

AWS Batch は、コンピューティング環境 AMIs を自動的に更新しません。更新するのはあなたの責任です。モジュールを最新バージョンにアップグレードするには、次のコマンドを実行します。

```
$ aws batch update-compute-environment \
    --compute-environment <compute-environment-name> \
    --compute-resources 'updateToLatestImageVersion=true'
```

AWS Batch はバージョンを自動的にアップグレードしませんKubernetes。以下のコマンドを実行して、Kubernetes コンピューター環境のバージョンを *1.23* に更新します。

```
$ aws batch update-compute-environment \
    --compute-environment <compute-environment-name> \
    --compute-resources \
      'ec2Configuration=[{imageType=EKS_AL2,imageKubernetesVersion=1.23}]'
```

Kubernetes より新しいバージョンの AMI に更新する場合、更新時にジョブを終了するかどうか`terminateJobsOnUpdate`と、実行中のジョブが終了しない場合にインスタンスが置き換えられるまでの待機時間 (`jobExecutionTimeoutMinutes`.) を指定できます。詳細については、コンピューティング環境を更新します。 および UpdateComputeEnvironment API オペレーションで設定されているインフラストラクチャ更新ポリシー (UpdatePolicy) を参照してください。

AWS Batch マネージドノードDaemonSetでの の実行

AWS Batch AWS Batch はマネージドKubernetesノードにテイントを設定します。次の を使用してDaemonSet、 AWS Batch マネージドノードで実行する をターゲットにすることができますtolerations。

```
tolerations:
- key: "batch.amazonaws.com/batch-node"
  operator: "Exists"
```

そのための、もう 1 つの方法は、次の tolerations ように、実行することです。

```
tolerations:
- key: "batch.amazonaws.com/batch-node"
  operator: "Exists"
  effect: "NoSchedule"
- key: "batch.amazonaws.com/batch-node"
  operator: "Exists"
  effect: "NoExecute"
```

起動テンプレートを使用したマネージドノードのカスタマイズ

AWS Batch Amazon EKS の は、起動テンプレートをサポートしています。起動テンプレートでできることには制約があります。

Important

AWS Batch は を実行します/etc/eks/bootstrap.sh。/etc/eks/bootstrap.sh 起動テンプレートやcloud-init、user-data スクリプトでは実行しないでください。bootstrap.sh には --kubelet-extra-args パラメータ以外にもパラメータを追加できます。これを行うには、AWS_BATCH_KUBELET_EXTRA_ARGS /etc/aws-batch/batch.config ファイルに変数を設定します。次のコードを例として参照してください。

Note

が呼び出された後に起動テンプレートが変更された場合は、 CreateComputeEnvironmentを呼び出して、置き換える起動テンプレートのバージョンを評価するUpdateComputeEnvironment必要があります。

トピック

引数を **kubelet** さらに追加します。

AWS Batch は、kubeletコマンドへの引数の追加をサポートしています。kubelet によってサポートされるパラメータの全リストについては、Kubernetes のドキュメント を参照してください。次の例では、*--node-labels mylabel=helloworld* が kubelet コマンドラインに追加されています。

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="

--==MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
mkdir -p /etc/aws-batch

echo AWS_BATCH_KUBELET_EXTRA_ARGS=\"--node-labels mylabel=helloworld\" >> /etc/aws-
batch/batch.config

--==MYBOUNDARY==--
```

コンテナランタイムの設定

AWS Batch CONTAINER_RUNTIME 環境変数を使用して、管理対象ノードでコンテナランタイムを設定できます。次の例では、bootstrap.sh コンテナランタイムを実行時に containerd を設定しています。詳細については、 ドキュメントのcontainerd を参照してください。

> **Note**
>
> CONTAINER_RUNTIME 環境変数は bootstrap.sh の --container-runtime オプションと同等です。詳細については、 ドキュメントのOptions を参照してください。

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="

--==MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
mkdir -p /etc/aws-batch

echo CONTAINER_RUNTIME=containerd >> /etc/aws-batch/batch.config

--==MYBOUNDARY==--
```

Amazon EFS ボリュームをマウントする

起動テンプレートを使用してボリュームをノードにマウントできます。次の例では、cloud-config、packages および runcmd の設定が使用されています。詳細はドキュメントのクラウド設定例を参照してください。

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="

--==MYBOUNDARY==
Content-Type: text/cloud-config; charset="us-ascii"

packages:
- amazon-efs-utils

runcmd:
- file_system_id_01=fs-abcdef123
- efs_directory=/mnt/efs

- mkdir -p ${efs_directory}
- echo "${file_system_id_01}:/ ${efs_directory} efs _netdev,noresvport,tls,iam 0 0"
 >> /etc/fstab
- mount -t efs -o tls ${file_system_id_01}:/ ${efs_directory}

--==MYBOUNDARY==--
```

このボリュームをジョブで使用するには、eksProperties パラメータで に追加する必要がありますRegisterJobDefinition。以下は、応答の定義の例です。

```
{
    "jobDefinitionName": "MyJobOnEks_EFS",
    "type": "container",
    "eksProperties": {
        "podProperties": {
            "containers": [
                {
                    "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
                    "command": ["ls", "-la", "/efs"],
                    "resources": {
                        "limits": {
                            "cpu": "1",
                            "memory": "1024Mi"
                        }
                    },
                    "volumeMounts": [
                        {
                            "name": "efs-volume",
                            "mountPath": "/efs"
                        }
                    ]
                }
            ],
            "volumes": [
                {
                    "name": "efs-volume",
                    "hostPath": {
                        "path": "/mnt/efs"
                    }
                }
            ]
        }
    }
}
```

ノードでは、Amazon EFS ボリュームが /mnt/efs ディレクトリにマウントされます。Amazon EKS ジョブのコンテナでは、ボリュームは /efs ディレクトリにマウントされます。

IPv6 サポート

AWS Batch は、IPv6 アドレスを持つ Amazon EKS クラスターをサポートします。 AWS Batch サポートにカスタマイズは必要ありません。ただし、始める前に、Amazon EKS ユーザーガイド

のポッドとサービスへの IPv6 アドレスの割り当て で説明されている考慮事項と条件を確認することをお勧めします。

コンピューティングリソースメモリ管理

Amazon ECSコンテナエージェントが、コンテナインスタンスコンピュートリソースをクラスターコンピューティング環境に登録する場合、エージェントはタスクジョブの実行のために、コンテナインスタンスコンピューティングリソースが使用できるメモリ容量を決定する必要があります。プラットフォームのメモリオーバーヘッドとシステムカーネルが占めるメモリのため、この数値は、Amazon EC2 インスタンスとして公開されているインストール済みメモリ量とは異なります。例えば、`m4.large` インスタンスには 8 GiB のメモリがインストールされています。しかし、これはコンピューティングが登録されたときに、 ジョブに厳密に8192 MiBのメモリに常に変換されるとは限りません。

ジョブに8192 MiB を指定した場合、この要件を満たす使用可能なメモリが 8192 MiB以上のメモリがないとします。その場合、そのジョブをコンピューティング環境に置くことができなくなります。マネージド型コンピューティング環境を使用している場合、リクエストに対応するためには、AWS Batchはより大きなインスタンスタイプを起動する必要があります。

デフォルト AWS Batch コンピューティングリソースAMIも、Amazon ECS コンテナエージェントやその他の重要なシステムプロセス用に32 MiBのメモリを予約します。このメモリはジョブの割り当てには使用できません。詳細については、システムメモリの予約を参照してください。

Amazon ECS コンテナエージェントは、Docker `ReadMemInfo()`関数を使用してオペレーティングシステムで使用可能な合計メモリのクエリを実行します。Linuxは、合計メモリを判断するためにコマンドラインユーティリティを提供します。

Example - Linux合計メモリを決定

free コマンドは、オペレーティングシステムによって認識される合計メモリを復活します。

```
$ free -b
```

Amazon ECSに最適化されたAmazon Linux AMIを実行する `m4.large` インスタンスの出力例。

```
             total       used       free     shared    buffers     cached
Mem:    8373026816  348180480 8024846336      90112   25534464  205418496
-/+ buffers/cache:  117227520 8255799296
```

このインスタンスには 8373026816バイトの合計メモリがあります。つまり、7985 MiBがタスクに使用できます。

システムメモリの予約

タスクジョブでコンピューティングリソース上の全メモリを占有している場合、タスクジョブがメモリを必要とする重要なシステムプロセスと競い、システム障害の引き金となる可能性があります。Amazon ECS コンテナエージェントは、ECS_RESERVED_MEMORY と呼ばれる設定変数を提供します。この変数を使用して、タスクジョブに割り当てられたプールから特定のMiBメモリを削減できます。これにより、重要なシステムプロセスのメモリを効果的に確保することができます。

デフォルトのAWS Batchコンピューティングリソース AMI は、Amazon ECS コンテナエージェントやその他の重要なシステムプロセス用に、32MiBのメモリを予約します。

コンピューティングリソース メモリを表示

Amazon ECS コンソール または DescribeContainerInstances API操作で、コンテナインスタンスコンピューティングリソースが登録されているメモリ量を表示できます。特定のインスタンスタイプに対して、可能な限り大きなメモリをタスクジョブに割り当てて、リソース使用率を最大化しようとする場合は、そのコンピューティングリソースに使用可能なメモリを観察でき、その後にタスクジョブに可能な限り多くのメモリを割り当てることができます。

コンピューティングリソースのメモリを表示するために

1. コンソールをhttps://console.aws.amazon.com/ecs/v2で開きます。
2. クラスターを選択し、表示するコンピューティングリソースをホストするクラスターを選択します。

   コンピューティング環境のクラスター名は、クラスター環境名で始めます。
3. インフラストラクチャ を選択します。
4. コンテナインスタンスで、コンテナインスタンスを選択します。
5. リソースとネットワーキング セクションに、コンピューティングリソース用に登録され、使用できるメモリが表示されます。

   登録済み メモリの値は、Amazon ECSの初回起動時に登録されたコンピューティングリソースのメモリの値です。使用可能] メモリの値は。まだタスクジョブ に割り当てられていないメモリの値です。

Amazon EKSAWS Batchのメモリとv CPUに関する考慮事項

Amazon EKSのAWS Batchでは、コンテナで使用できるリソースを指定できます。たとえば、vCPUとメモリリソースのrequests値、またはlimits値を指定できます。

vCPU リソースを指定する際の制約は次のとおりです：

- 少なくとも 1つのvCPUrequests、またはlimits値 のいずれか指定する必要があります。
- １つのvCPU ユニットは、１つの物理コアまたは仮想コアに相当します。
- vCPUの値は、整数または0.25単位で入力する必要があります。
- 有効な最小値は0.25です。
- 両方が特定される場合、requestsの値はlimitsの値以下でなくてはなりません。このようにして、ソフトとハードのvCPU設定の両方を行うことができます。
- vCPU値をミリCPU形式で指定することはできません。たとえば、100mは有効な値ではありません。
- AWS Batchは、この requests 値をスケーリングの決定に使用します。requests値が指定されていない場合、limits値はrequests値にコピーされます。

メモリリソースを指定する際の制約は、次のとおりです：

- 少なくとも1つのメモリrequests または limits 値を指定する必要があります。
- メモリ値は mebibytes (MiBs) 内である必要があります。
- 両方を指定する場合、requests 値はlimits 値と等しくなければなりません。
- AWS Batchは、この requests 値をスケーリングの決定に使用します。requests 値が指定されていない場合、limits 値は requests 値にコピーされます。

GPUリソースを指定する際の制約は、次のとおりです：

- 両方を指定する場合、requests 値はlimits 値と等しくなければなりません。
- AWS Batchは、この requests 値をスケーリングの決定に使用します。requests値が指定されていない場合、limits値はrequests値にコピーされます。

ジョブ定義の例

Amazon EKS ジョブ定義の以下のAWS Batch では、ソフトvCPU共有を設定します。これにより、Amazon EKS のAWS Batchは、インスタンスタイプの vCPU容量のすべて使用できるようになります。ただし、実行中の他のジョブがある場合、そのジョブには最大数の 2 vCPUs が割り当てられます。メモリは、2 GBに制限されています。

```
{
    "jobDefinitionName": "MyJobOnEks_Sleep",
    "type": "container",
    "eksProperties": {
        "podProperties": {
            "containers": [
                {
                    "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
                    "command": ["sleep", "60"],
                    "resources": {
                        "requests": {
                            "cpu": "2",
                            "memory": "2048Mi"
                        }
                    }
                }
            ]
        }
    }
}
```

Amazon EKS AWS Batch の以下のジョブ定義の`request` 値は 1 で、最大の 4 vCPUs をジョブに割り当てます。

```
{
    "jobDefinitionName": "MyJobOnEks_Sleep",
    "type": "container",
    "eksProperties": {
        "podProperties": {
            "containers": [
                {
                    "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
                    "command": ["sleep", "60"],
                    "resources": {
                        "requests": {
```

```
                            "cpu": "1"
                        },
                        "limits": {
                            "cpu": "4",
                            "memory": "2048Mi"
                        }
                    }
                }
            ]
        }
    }
}
```

以下のAmazon EKS のAWS Batchジョブ定義では 、vCPU `limits` 値をに 1を、メモリ `limits` 値を 1 GB に設定します。

```
{
    "jobDefinitionName": "MyJobOnEks_Sleep",
    "type": "container",
    "eksProperties": {
        "podProperties": {
            "containers": [
                {
                    "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
                    "command": ["sleep", "60"],
                    "resources": {
                        "limits": {
                            "cpu": "1",
                            "memory": "1024Mi"
                        }
                    }
                }
            ]
        }
    }
}
```

AWS Batch Amazon EKS 上のジョブを Amazon EKS ポッドにAWS Batch 変換するときに、AWS Batch `limits` 値をその `requests` 値にコピーします。これは、`requests` 値が指定されていない場合です。前述のジョブ定義の例を送信すると、ポッドspec は次のようになります。

```
apiVersion: v1
```

```
kind: Pod
...
spec:
  ...
  containers:
    - command:
        - sleep
        - 60
      image: public.ecr.aws/amazonlinux/amazonlinux:2
      resources:
        limits:
          cpu: 1
          memory: 1024Mi
        requests:
          cpu: 1
          memory: 1024Mi
      ...
```

ノードCPUとメモリの予約

AWS Batch は、vCPU とメモリの予約に関してbootstrap.shファイルのデフォルトロジックに依存します。bootstrap.shファイルの詳細については、 の bootstrap.sh を参照してください。vCPUメモリリソースのサイズを決定する場合、以下の例を検討してください。

> **Note**
>
> 実行中のインスタンスがない場合、vCPUとメモリの予約が最初にAWS Batchスケーリングロジックと意思決定に影響を与える可能性があります。インスタンス実行されたら、AWS Batchは初期割り当てを調整します。

ノードCPU予約の例

CPU予約値は、インスタンスで使用可能なvCPUsの総数を使用してミリコア単位で計算されます。

vCPU番号	予約率
1	6%
2	1

vCPU番号	予約率
3～4	0.5%
4以上	0.25%

上記の値を使用すると、次のようになります:

- 仮想CPUが2つある `c5.large` インスタンスのvCPUs予約値は,70 m です。これは次の方法で計算されます：(1*60) + (1*10) = 70 m 。
- 96個のvCPUsを搭載した`c5.24xlarge` インスタンスのCPU予約値は、 310 mです。これは次の方法で計算されます：(1*60) + (1*10) + (2*5) + (92*2.5) = 310 m。

この例では、`c5.large` インスタンスでジョブを実行するために使用できるミリコアvCPU ユニットは、1930 (計算は2000 - 70) です。ジョブに 2 (2*1000 m) の vCPU ユニットが必要で、そのジョブが 単一の`c5.large` インスタンスに収まらないとします。ただし、`1.75` vCPU ユニットが必要なジョブには適しています。

ノードメモリ予約の例:

メモリ予約値は、以下を使用してメビバイト単位で計算されます：

- インスタンスキャパシティーは MB 単位です。たとえば、8 GB のインスタンスは 7,748 です MiB。
- `kubeReserved` 値。`kubeReserved` 値は、システムデーモン用に確保するメモリ量です。`kubeReserved` 値は次のように計算されます:((11 * インスタンスタイプでサポートされる最大ポッド数) + 255)。各インスタンスタイプによりサポートされるポッドの最大数のリストは、GitHub のeni-max-pods.txtを参照してください。
- `HardEvictionLimit` 値。使用可能なメモリが`HardEvictionLimit` 値を下回ると、インスタンスはポッドを削除しようとします。

割り当て可能なメモリの計算式は次のとおりです：(*`Instance_Capacity_IN_MIB`*)-(11 * (*`#####`*#))-255-(*`HardEvictionLimit ##`*))。

1つの `c5.large` インスタンスは最大29個のポッドをサポートします。`HardEvictionLimit` 値が100 MiBの8 GB `c5.large` インスタンスの場合、割り当て可能なメモリは7074ですMiB。これは次の方法で計算されます:(7748-(11 * 29) -255 -100) = 7074 MiB。この例では、8,192件のMiB のタスク

ジョブは 、8 gibibyte (GiB) インスタンスであるにもかかわらず、このインスタンスには当てはまりません。

DaemonSets

DaemonSets を使用する場合は、以下を考慮してください：

- Amazon EKS インスタンスでAWS Batchがまったく実行されていない場合、最初はDaemonSets AWS Batch のスケーリングロジックと意思決定に影響する可能性があります。AWS Batchは最初に0.5 のvCPU ユニットと、500 MiB を想定されるDaemonSetsに割り当てます。インスタンス実行されたら、AWS Batchは初期割り当てを調整します。
- DaemonSetがvCPUまたはメモリの制限を定義している場合、Amazon EKSジョブのAWS Batchが少なくなります。AWS Batchジョブに割り当てられる DaemonSets の数は、できるだけ少なくすることをお勧めします。

スケジューリングポリシー

スケジューリングポリシーを使用して、ジョブキュー内のコンピューティングリソースをユーザーやワークロード間で配分できます。スケジューリングポリシーを使用すると、ワークロードやユーザーに異なる公平配分識別子を割り当てることができます。AWS Batch は、各公平配分識別子に、一定期間に利用可能な総リソースに対する割合を割り当てます。

公平配分比率は、`shareDecaySeconds` および `shareDistribution` の値を使用して計算されます。ポリシーに割合の減衰時間を割り当てることで、公平配分の分析に時間を追加できます。減衰時間が長いほど、時間により多くの重みが与えられ、定義された重みよりも少なくなります。コンピューティング予約を指定することで、アクティブでない公平配分識別子に対してコンピューティングリソースを確保できます。詳細については、スケジューリングポリシーパラメータを参照してください。

トピック

- スケジューリングポリシーの作成
- スケジューリングポリシーパラメータ

スケジューリングポリシーの作成

スケジューリングポリシーを使用してジョブキューを作成する前に、スケジューリングポリシーを作成する必要があります。スケジューリングポリシーを作成するときは、キューの重みに 1 つ以上の公平配分識別子または公平配分識別子のプレフィックスを関連付けて、オプションで減衰期間とコンピューティング予約をポリシーに割り当てます。

スケジューリングポリシーを作成するには

1. https://console.aws.amazon.com/batch/ で AWS Batch コンソールを開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. ナビゲーションペインで、[Scheduling policies] (ポリシー)、[Create] (ポリシーの作成) の順に選択します。
4. [Name] (名前) に、スケジューリングポリシーの一意の名前を入力します。最大 128 文字の英字 (大文字と小文字)、数字、ハイフン、アンダースコアを使用できます。
5. (オプション) [Share decay seconds] (割合の減衰秒) に、スケジューリングポリシーの共有減衰時間の整数値を入力します。共有減衰時間が長いほど、ジョブをスケジュールするときに、コ

ンピューティングリソースの使用率を長時間考慮します。これにより、公平配分識別子を使用するジョブは、その公平配分識別子が最近コンピューティングリソースを使用していなかった場合に、その公平配分識別子の重みよりも多くのコンピューティングリソースを一時的に使用できるようになります。

6. (オプション) [Compute reservation] (コンピューティング予約) に、スケジューリングポリシーのコンピューティング予約を示す整数値を入力します。コンピューティング予約は、現在アクティブでない公平配分識別子に使用するために、一部の vCPUs を予備に保持します。

   予約比率は (*computeReservation*/100)^*ActiveFairShares* であり、ActiveFairShare は、アクティブな公平配分識別子の数です。

   たとえば、computeReservation の値が 50 の場合、AWS Batch は、公平配分識別子が 1 つしかない場合は、使用可能な最大 VCPU の 50% を確保し、2 つの公平配分識別子がある場合は 25%、公平配分識別子が 3 つある場合は 12.5% を予約します。あるcomputeReservation の値が 25 の場合、AWS Batchは、公平配分識別子が 1 つしかない場合は、使用可能な最大の VCPU の 25%、公平配分識別子が 2 つある場合は 6.25%、公平配分識別子が 3 つある場合は 1.56% を予約します。

7. 属性の共有セクションでは、スケジューリングポリシーに関連付ける各公平配分識別子の公平配分識別子と重みを指定できます。

   a. [Add share identifier] (配分識別子を追加) を選択します。

   b. [Share identifier] (配分識別子) に公平配分識別子を指定します。文字列が「**」で終わる場合、これはジョブの公平配分識別子の照合に使用される公平配分識別子のプレフィックスになります。スケジューリングポリシー内のすべての公平配分識別子と公平配分識別子のプレフィックスは一意でなければならず、重複することはできません。たとえば、同じスケジューリングポリシーに公平配分識別子のプレフィックス「UserA**」と公平配分識別子「UserA1」を含めることはできません。

   c. [Weight factor] (重み係数) には、公平配分識別子の相対的な重みを指定します。デフォルト値は 1.0 です。値が小さいほど、コンピューティングリソースの優先順位が高くなります。公平配分識別子のプレフィックスが使用されている場合、プレフィックスで始まる公平配分識別子を持つジョブは、重み係数を共有します。これにより、これらのジョブの重み係数が効果的に増加し、個々の優先度は下がりますが、公平配分識別子のプレフィックスには同じ重み係数が維持されます。

8. (オプション) タグセクションで、スケジューリングポリシーに関連付ける各タグのキーと値を指定します。詳細については、「AWS Batch リソースのタグ付け」を参照してください。

9. [Submit] (送信) を選択して終了し、スケジューリングポリシーを作成します。

スケジューリングポリシーテンプレート

以下に示すのは、空のスケジューリングポリシーテンプレートです。このテンプレートを使ってスケジューリングポリシーを作成し、それをファイルに保存して AWS CLI `--cli-input-json` オプションで使用できます。これらのパラメータの詳細については、「CreateComputeEnvironment」(AWS BatchAPI リファレンス内) を参照してください。

```
{
    "name": "",
    "fairsharePolicy": {
        "shareDecaySeconds": 0,
        "computeReservation": 0,
        "shareDistribution": [
            {
                "shareIdentifier": "",
                "weightFactor": 0.0
            }
        ]
    },
    "tags": {
        "KeyName": ""
    }
}
```

> **Note**
>
> 前述のジョブキューテンプレートは、以下の AWS CLI コマンドで生成できます。
>
> ```
> $ aws batch create-scheduling-policy --generate-cli-skeleton
> ```

スケジューリングポリシーパラメータ

スケジューリングポリシーは、スケジューリングポリシーの名前、公平配分ポリシー、タグの 3 つのベーシックコンポーネントに分かれています。

トピック

- スケジューリングポリシー名
- 公平配分ポリシー

- タグ

スケジューリングポリシー名

`name`

スケジュールポリシーの名前。最大 128 文字の英字 (大文字と小文字)、数字、ハイフン、アンダースコアを使用できます。

型: 文字列

必須: はい

公平配分ポリシー

`fairsharePolicy`

スケジューリングポリシーの公平配分ポリシー。

```
"fairsharePolicy": {
   "computeReservation": number,
   "shareDecaySeconds": number,
   "shareDistribution": [
      {
         "shareIdentifier": "string",
         "weightFactor": number
      }
   ]
}
```

型: オブジェクト

必須: いいえ

`computeReservation`

まだ使用されていない公平配分識別子に利用可能な最大 VCPU の一部を予約するために使用される値。

予約比率は (*computeReservation*/100)^*ActiveFairShares* であり、ActiveFairShare は、アクティブな公平配分識別子の数です。

たとえば、computeReservation の値が 50 の場合、AWS Batch は、アクティブな公平配分識別子が 1 つしかない場合は、使用可能な最大 VCPU の 50% を確保し、アクティブな公平配分識別子が 2 つある場合は 25%、アクティブな公平配分識別子が 3 つある場合は 12.5% を予約します。computeReservation の値が 25 の場合、AWS Batch は、アクティブな公平配分識別子が 1 つしかない場合は、使用可能な最大 VCPU の 25% を確保し、アクティブな公平配分識別子が 2 つある場合は 6.25%、アクティブな公平配分識別子が 3 つある場合は 1.56% を予約します。

型: 整数

値の範囲: 最小値 は 0 です。最大値は 99 です。

必須: いいえ

shareDecaySeconds

使用中の各公平配分識別子の公平配分の割合を計算するために使用する期間。値がゼロ (0) の場合、現在の使用量のみを測定する必要があることを示します。この減衰により、最近実行されたジョブは、以前に実行されたジョブよりも重み付けされます。

型: 整数

値の範囲: 最小値 は 0 です。最大値は 604,800 です (1 週間)。

必須: いいえ

shareDistribution

公平配分ポリシーの公平配分識別子の重みを含むオブジェクトの配列。含まれていない公平配分識別子のデフォルトの重みは 1.0 です。

```
"shareDistribution": [
   {
      "shareIdentifier": "string",
      "weightFactor": number
   }
]
```

型: 配列

必須: いいえ

`shareIdentifier`

公平配分識別子または公平配分識別子のプレフィックス。文字列が「**」で終わる場合、この文字列は、そのプレフィックスで始まる公平配分識別子に対して公平配分識別子のプレフィックスを指定します。たとえば、値が `UserA*` の場合、`weightFactor` は 1 であり、`UserA` で始まる公平配分識別子が 2 つあります。それらの公平配分識別子の重みは 2 になります。そのような公平配分識別子が 5 つある場合、それぞれの重みは 5 になります。

公平配分ポリシー内の公平配分識別子と公平配分識別子のプレフィックスのリストは、重複することはできません。たとえば、`UserA*` の公平配分識別子のプレフィックスと、`UserA-1` の公平配分識別子を同じ公平配分ポリシーに含めることはできません。

型: 文字列

必須: はい

`weightFactor`

公平配分識別子の重み係数。デフォルト値は 1.0 です。値が小さいほど、コンピューティングリソースの優先順位が高くなります。たとえば、重み係数 0.125 (1/8) の配分識別子を使用するジョブは、重み係数 1 の配分識別子を使用するジョブの 8 倍のコンピューティングリソースを取得します。

サポートされる最小値は 0.0001、最大値は 999.9999 です。

タイプ: 浮動小数点

必須: いいえ

タグ

`tags`

スケジューリングポリシーに関連付けるキーバリューペアのタグ。詳細については、「AWS Batch リソースのタグ付け」を参照してください。

タイプ: 文字列間のマッピング

必須: いいえ

AWS Batch コンソールで Step Functions ステートマシンを使って AWS Batch ジョブをオーケストレートする

AWS Batch コンソールを使用して、 ステートマシンおよびこれらのス Step Functions テートマシンが使用する 関数の詳細を表示できます。

セクション

ステートマシンの詳細の表示

AWS Batch コンソールには、AWS Batch ジョブを送信するワークフローステップが少なくとも 1 つ含まれているステートマシンが、現在の AWS リージョン から一覧表示されます。

ステートマシンを選択して、ワークフローをグラフィカル表示します。青でハイライトされたステップは、AWS Batch ジョブを表します。グラフコントロールを使用して、グラフをズームイン、ズームアウト、中央揃えにします。

> **Note**
>
> ステートマシンの定義で AWS Batch ジョブが JsonPath で動的に参照されている場合、関数の詳細を AWS Batch コンソールに表示することはできません。代わりに、ジョブ名が動的参照として表示され、グラフ内の対応するステップが灰色表示になります。

ステートマシンの詳細を表示するには

1. AWS Batch コンソールの[Workflow orchestration powered by Step Functions ページ] (Step Functions により強化されたワークフローオーケストレーション) を開きます。
2. ステートマシンを選択します。
   <result>

   AWS Batch コンソールに [詳細] ページが表示されます。
   </result>

詳細については、 AWS Step Functions デベロッパーガイドの Step Functions を参照してください。

ステートマシンの編集

ステートマシンを編集する場合、AWS Batch を使用して Step Functions コンソールの [Edit definition] (定義の編集) ページを開きます。

ステートマシンを編集するには

1. AWS Batch コンソールの[Workflow orchestration powered by Step Functions ページ] (Step Functions により強化されたワークフローオーケストレーション) を開きます。
2. ステートマシンを選択します。
3. [Edit] (編集) を選択します。

   Step Functions コンソールで [Edit definition] (定義の編集) ページが開きます。
4. ステートマシンを編集し、[Save] (保存) を選択します。

ステートマシンの編集の詳細については、AWS Step Functions デベロッパーガイドの 「Step Functions state machine language」を参照してください。

ステートマシンの実行

ステートマシンを実行する場合は、AWS Batch を使用して Step Functions コンソールの [New execution] (新しい実行) ページを開きます。

ステートマシンを実行するには

1. AWS Batch コンソールの[Workflow orchestration powered by Step Functions ページ] (Step Functions により強化されたワークフローオーケストレーション) を開きます。
2. ステートマシンを選択します。
3. [Execute] (実行) を選択します。

   Step Functions コンソールで [New execution] (新しい実行) ページが開きます。
4. (オプション) ステートマシンを編集し、[Start execution] (実行を開始) を選択します。

ステートマシンの実行の詳細については、AWS Step Functions デベロッパーガイドの 「ステップ関数ステートマシンの実行概念」を参照してください。

AWS Fargate 上の AWS Batch

AWS Fargate は、サーバーやAmazon EC2インスタンスのクラスターを管理することなくAWS Batch でコンテナを実行できるテクノロジーです。AWSFargateを使用すると、コンテナを実行するために仮想マシンのクラスターをプロビジョニング、設定、スケールする必要がありません。これにより、サーバータイプの選択、クラスターをスケールするタイミングの決定、クラスターのパッキングの最適化を行う必要がなくなります。

Fargate リソースを使用してジョブを実行する場合、アプリケーションをコンテナにパッケージ化し、CPU とメモリ要件を指定して、ネットワークとIAM ポリシーを定義して、アプリケーションを起動します。各 Fargate ジョブは、独自の分離境界を持ち、基本となるカーネル、CPU リソース、メモリリソース、Elastic Network Interface を別のジョブと共有しません。

目次

Fargateをいつ使うべきか

ほとんどのシナリオで Fargate を使用することをお勧めします。Fargate は、コンテナに指定したリソース要件に厳密に一致するようにコンピューティングを起動し、スケールします。Fargate を使用すると、追加のサーバーに対してオーバープロビジョニングまたは料金を支払う必要はありません。また、インスタンスタイプなど、インフラストラクチャ関連のパラメータの詳細について心配する必要はありません。コンピューティング環境をスケールアップする必要がある場合、Fargate リソースで実行されるジョブをより迅速に開始できます。通常、新しい Amazon EC2 インスタンスの作成に数分かかります。しかし、Fargate で実行されるジョブは約 30 秒でプロビジョニングできます。正確な所要時間は、コンテナイメージのサイズやジョブ数など、いくつかの要因によって異なります。

ただし、ジョブに次のいずれかが必要な場合は、Amazon EC2 を使用することをお勧めします。

- 16 個以上の vCPU
- 120 ギガバイト (GiB) 以上のメモリ
- 1 個のGPU

- 1 個のカスタム Amazon マシンイメージ (AMI)
- LinuxParameters パラメータのいずれか

ジョブの数が多い場合は、Amazon EC2 インフラストラクチャを使用することをお勧めします。たとえば、同時実行ジョブの数が Fargate スロットリング制限を超える場合です。これは、EC2 では、Fargate リソースよりも高いレートで EC2 リソースにジョブをディスパッチできるためです。さらに、EC2 を使用すると、同時に実行できるジョブが増えます。詳細については、Amazon Elastic Container Service デベロッパーガイドのAWS Fargate サービスクォータを参照してください。

Fargate でのJob 定義

Fargate での AWS Batch ジョブは、使用可能なすべてのジョブ定義パラメータをサポートしているわけではありません。一部のパラメータはまったくサポートされていません。また、その他のパラメータはFargate ジョブでは異なる動作をします。

次のリストでは、Fargate ジョブで有効でないか、または制限されていないジョブ定義パラメータについて説明します。

`platformCapabilities`

`FARGATE` と指定する必要があります。

```
"platformCapabilities": [ "FARGATE" ]
```

`type`

`container` と指定する必要があります。

```
"type": "container"
```

`containerProperties`のパラメータ

`executionRoleArn`

Fargate リソースで実行されているジョブには指定する必要があります。詳細については、Amazon Elastic Container Service デベロッパーガイドのタスク用の IAM ロールを参照してください。

```
"executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole"
```

`fargatePlatformConfiguration`

(オプション、Fargate ジョブ定義の場合のみ)。Fargate プラットフォームのバージョンを指定するか、最新のプラットフォームバージョンの場合は LATEST を指定します。`platformVersion` の可能な値はデフォルトで `1.3.0`、`1.4.0`、LATEST です。

```
"fargatePlatformConfiguration": { "platformVersion": "1.4.0" }
```

`instanceType`, `ulimits`

Fargate リソースで実行されているジョブには適用されません。

`memory`, `vcpus`

これらの設定は、`resourceRequirements` で指定する必要があります。

`privileged`

このパラメータを指定しないか、`false` を指定します。

```
"privileged": false
```

`resourceRequirements`

メモリと vCPU の要件は、サポートされている値を使用して指定する必要があります。GPU リソースは、Fargate リソースで実行されているジョブではサポートされていません。

GuardDuty Runtime Monitoring を使用すると、 GuardDuty セキュリティエージェントのメモリオーバーヘッドがわずかに発生します。したがって、メモリ制限にはセキュリティエージェントのサイズ GuardDutyを含める必要があります。 GuardDuty セキュリティエージェントのメモリ制限については、「 ユーザーガイド」の「CPU とメモリの制限」を参照してください。GuardDuty ベストプラクティスの詳細については、「Amazon ECS デベロッパーガイド」の「Runtime Monitoring を有効にした後で Fargate タスクのメモリ不足エラーを修正する方法」を参照してください。

```
"resourceRequirements": [
  {"type": "MEMORY", "value": "512"},
  {"type": "VCPU",   "value": "0.25"}
]
```

`linuxParameters`のパラメータ

`devices`, `maxSwap`, `sharedMemorySize`, `swappiness`, `tmpfs`

Fargate リソースで実行されているジョブには適用されません。

`logConfiguration`のパラメータ

`logDriver`

`awslogs` と `splunk` のみがサポートされています。詳細については、awslogs ログドライバーを使用するを参照してください。

`networkConfiguration` のメンバー

`assignPublicIp`

プライベートサブネットにインターネットへトラフィックを送るための NAT ゲートウェイが接続されていない場合、`assignPublicIp` はENABLEDでなければなりません。詳細については、AWS Batch 実行 IAM ロールを参照してください。

Fargate のジョブキュー

Fargate の AWS Batch ジョブキューは基本的に変更されません。`computeEnvironmentOrder` にリストされているコンピューティング環境は、すべて Fargate のコンピューティング環境 (`FARGATE` または `FARGATE_SPOT`) であることが唯一の制限事項です。EC2 と Fargate コンピューティング環境は混在できません。

Fargate のコンピューティング環境

Fargate の AWS Batch コンピューティング環境は、利用可能なすべてのコンピューティング環境パラメーターをサポートしていません。一部のパラメータはまったくサポートされていません。Fargate では特定の制限があるパラメータもあります。

次のリストでは、Fargate ジョブで有効でない、または制限されているコンピューティング環境パラメータについて説明します。

`type`

このパラメータを `MANAGED` に設定する必要があります。

```
"type": "MANAGED"
```

computeResources オブジェクトのパラメータ

allocationStrategy, bidPercentage, desiredvCpus, imageId, instanceTypes, ec2Configuration, ec2KeyPair, instanceRole, launchTemplate, minvCpus, placementGroup, spotIamFleetRole

これらは、Fargate コンピューティング環境には適用されないため、指定できません。

subnets

このパラメータにリストされているサブネットに NAT ゲートウェイが接続されていない場合は、ジョブ定義の assignPublicIp パラメータを ENABLED に設定する必要があります。

tags

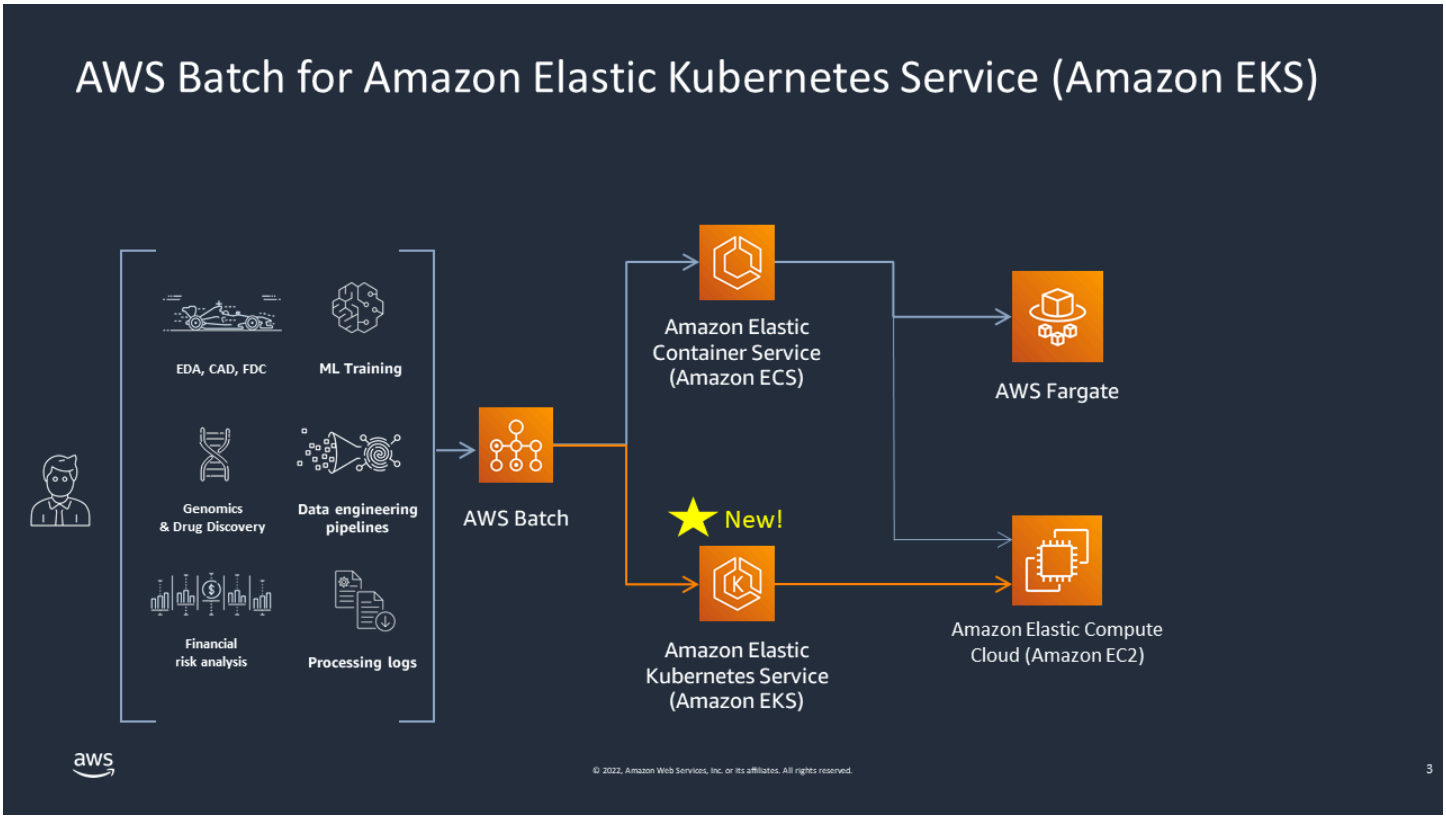
これは、Fargate コンピューティング環境では適用されないため、指定できません。Fargate のコンピューティング環境用のタグを指定するには、computeResources オブジェクトにない tags パラメータを使用します。

type

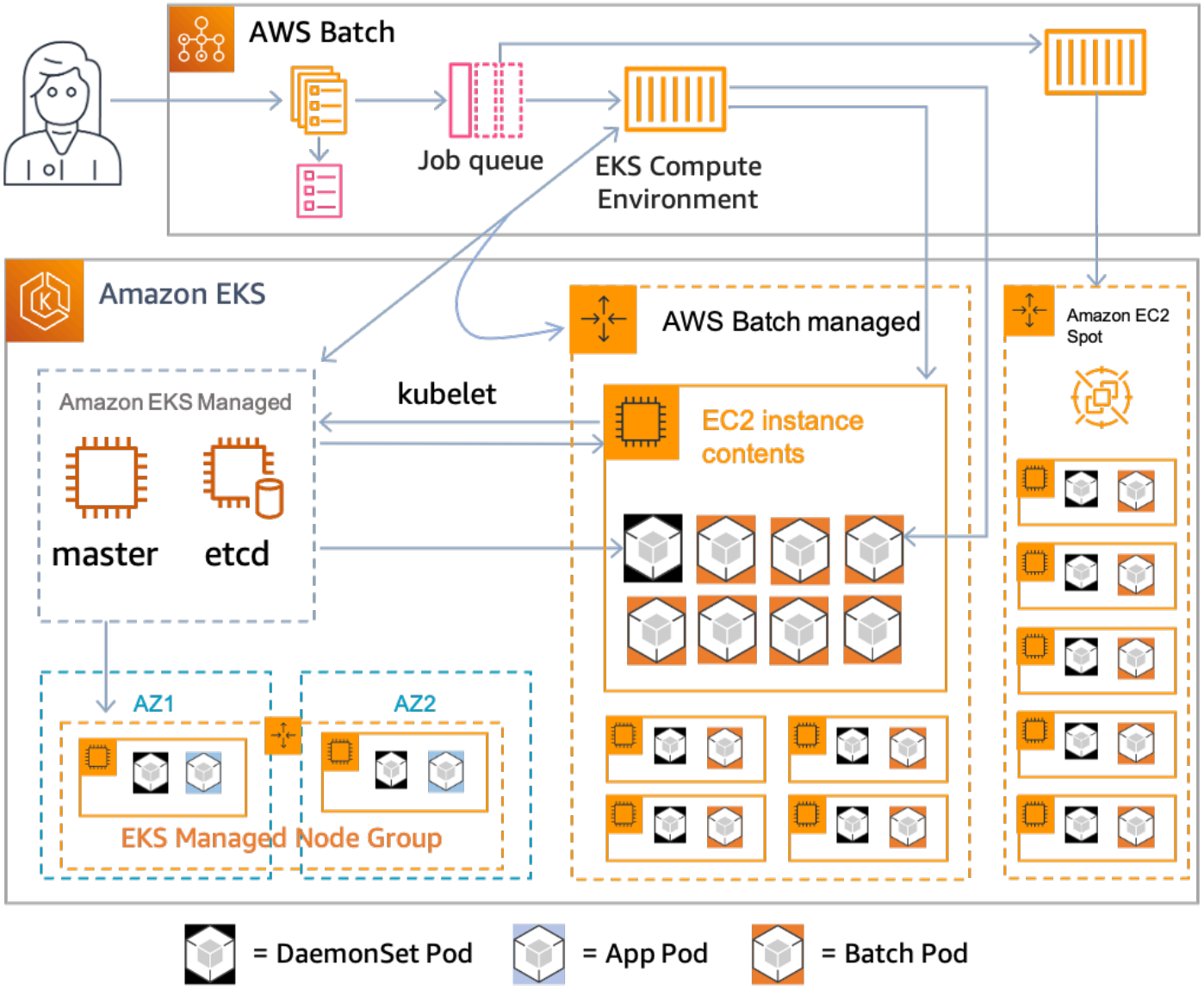
これは FARGATE または FARGATE_SPOT であることが必要です。

```
"type": "FARGATE_SPOT"
```

AWS Batch Amazon EKS での



AWS Batch は、マネージドバッチ機能を提供することで、Amazon EKS クラスターのバッチワークロードを簡素化します。これには、キューイング、依存関係の追跡、マネージドジョブの再試行と優先順位、ポッド管理、ノードスケーリングが含まれます。 は、複数のアベイラビリティーゾーンと複数の Amazon EC2 インスタンスタイプとサイズを処理 AWS Batch できます。 はAmazon EC2 スポットのベストプラクティスのいくつか AWS Batch を統合して、耐障害性のある方法でワークロードを実行するため、中断を減らすことができます。を使用すると、 AWS Batch わずかな夜間ジョブや数百万のミッションクリティカルなジョブを自信を持って実行できます。



AWS Batch は、Amazon Elastic Kubernetes Service (Amazon EKS) によって管理されるKubernetesクラスター内のバッチワークロードをオーケストレーションするマネージドサービスです。 AWS Batch は、クラスターの外部でこのオーケストレーションを「オーバーレイ」モデルを使用して実行します。 AWS Batch はマネージドサービスであるため、クラスターにインストールまたは管理するKubernetesコンポーネント (オペレーターやカスタムリソースなど) はありません。 AWS Batch は、 が Kubernetes API サーバーと AWS Batch 通信できるようにするロールベースのアクセスコントロール (RBAC) Kubernetes でクラスターを設定する必要があります。 は Kubernetes APIs を AWS Batch 呼び出してポッドとノードを作成、モニタリング、削除します。

AWS Batch には、ジョブ容量の割り当てに関する最適化により、ジョブキューの負荷に基づいてKubernetesノードをスケーリングするスケーリングロジックが組み込まれています。ジョブキューが空の場合、 はノードを設定した最小容量に AWS Batch スケールダウンします。デフォルトでは

ゼロです。 はこれらのノードのライフサイクル全体 AWS Batch を管理し、ノードにラベルとテイントを付けます。これにより、他のKubernetesワークロードは によって管理されるノードに配置されません AWS Batch。例外は です。これはDaemonSets、ジョブを適切に実行するために必要なモニタリングやその他の機能を提供するために AWS Batch ノードをターゲットにすることができます。さらに、管理されていないクラスター内のノードで、ジョブ、特にポッド AWS Batch を実行しません。これにより、クラスター上の他のアプリケーション用に別々のスケーリングロジックとサービスを使用できます。

にジョブを送信するには AWS Batch、 AWS Batch API と直接やり取りします。 はジョブを AWS Batch に変換しpodspecs、 AWS Batch Amazon EKS クラスターの によって管理されるノードにポッドを配置するリクエストを作成します。kubectlなどのツールを使用して、実行中のポッドやノードを表示できます。ポッドの実行が完了すると、 はKubernetesシステムへの負荷を軽減するために作成したポッド AWS Batch を削除します。

有効な Amazon EKS クラスターを に接続することで開始できます AWS Batch。次に、 AWS Batch ジョブキューをアタッチし、podspec同等の属性を使用して Amazon EKS ジョブ定義を登録します。最後に、ジョブ定義を参照する SubmitJob API オペレーションを使用してジョブを送信します。詳細については、「Amazon EKS AWS Batch での の開始方法」を参照してください。

Elastic Fabric Adapter

Elastic Fabric Adapter (EFA) は、ハイパフォーマンスコンピューティング (HPC) アプリケーションを高速化するネットワークデバイスです。 AWS Batch は、以下の条件が満たされている場合、EFA を使用するアプリケーションをサポートします。

- EFAsユーザーガイドAmazon EC2」の「サポートされているインスタンスタイプ」を参照してください。

  > **Tip**
  >
  > で EFAs をサポートするインスタンスタイプのリストを表示するには AWS リージョン、次のコマンドを実行します。次に、返されるリストと AWS Batch 、コンソールで使用可能なインスタンスタイプのリストを相互参照します。
  >
  > ```
  > $ aws ec2 describe-instance-types  --region us-east-1  --filters Name=network-info.efa-supported,Values=true  --query "InstanceTypes[*].[InstanceType]"  --output text | sort
  > ```

- EFAをサポートするオペレーティングシステムのリストについては、サポート対象のオペレーティングシステムを参照してください。
- AMI には EFA ドライバーがロードされています。
- EFA のセキュリティグループは、セキュリティグループとの間で送受信されるすべてのトラフィックを許可する必要があります。
- EFA を使用するすべてのインスタンスは、同じクラスタープレイスメントグループに属していなくてはなりません。
- ジョブ定義には、`hostPath` を `/dev/infiniband/uverbs0` に設定した`devices` メンバーを含めて、EFA デバイスがコンテナにパススルーされるようにする必要があります。`containerPath` が特定される場合は、それも `/dev/infiniband/uverbs0` に設定する必要があります。`permissions` を設定する場合は、`READ` | `WRITE` | `MKNOD` に設定する必要があります。

  LinuxParameters メンバーの場所は、マルチノードの並列ジョブと単一ノードのコンテナジョブでは異なります。以下の例は、その違いを示すものですが、必要な値が不足しています。

Example マルチノードの並列ジョブの例

```
{
  "jobDefinitionName": "EFA-MNP-JobDef",
  "type": "multinode",
  "nodeProperties": {
    ...
    "nodeRangeProperties": [
      {
        ...
        "container": {
          ...
          "linuxParameters": {
            "devices": [
              {
                "hostPath": "/dev/infiniband/uverbs0",
                "containerPath": "/dev/infiniband/uverbs0",
                "permissions": [
                    "READ", "WRITE", "MKNOD"
                ]
              },
            ],
          },
        },
      },
    ],
  },
}
```

Example 単一ノードのコンテナジョブの例

```
{
  "jobDefinitionName": "EFA-Container-JobDef",
  "type": "container",
  ...
  "containerProperties": {
    ...
    "linuxParameters": {
      "devices": [
        {
          "hostPath": "/dev/infiniband/uverbs0",
        },
```

```
      ],
    },
  },
}
```

EFA の詳細については、「Amazon EC2 ユーザーガイド」の「Elastic Fabric Adapter」を参照してください。 Amazon EC2

AWS Batch IAM ポリシー、ロール、アクセス権限

デフォルトでは、ユーザーには AWS Batch リソースを作成または変更する、または AWS Batch API、AWS Batch コンソール、またはAWS CLIを使用するタスクを実行する権限がありません。ユーザーがこれらのリソースを利用するには、特定のリソースと APIアクションを使用する許可を付与するIAM ポリシーを作成する必要があります。続いて、こうしたアクセス権限が必要なユーザーまたはグループにそのポリシーをアタッチします。

ポリシーをユーザーまたはユーザーグループにアタッチする場合、ポリシーによって特定リソースの特定タスクを実行するユーザーの権限が許可または拒否されます。詳細については、IAM ユーザーガイド の アクセス許可とポリシーを参照してください。カスタム IAM ポリシーの管理と作成の詳細については、IAM ポリシーの管理を参照してください。

AWS Batchは AWS のサービス ユーザーに代わって他のユーザーを呼び出します。そのため、AWS Batchは、ユーザーの認証情報を使用して認証する必要があります。より具体的には、AWS Batch は、これらのアクセス許可を指定するIAM ロールとポリシーを作成します。次に、それらの作成時に、そのロールをコンピューティング環境に関連付けます。詳細については、IAM ユーザーガイドのAmazon ECS インスタンスロール、IAM ロール、サービスにリンクされたロールの使用、AWS サービスに対してアクセス許可を委任するロールの作成を参照してください。

ご利用開始にあたって

IAM ポリシーは、1 つ以上の AWS Batch アクションを使用するアクセス許可を付与または拒否する必要があります。

トピック

- ポリシーの構造
- AWS Batch API アクションでサポートされるリソースレベルのアクセス許可
- ポリシーの例
- AWS Batch 管理ポリシー
- AWS Batch IAM ポリシーの作成
- Amazon ECS インスタンスロール
- Amazon EC2 スポットフリートロール
- EventBridge IAM ロール

ポリシーの構造

次のトピックでは、IAM ポリシーの簡単な構造について説明します。

トピック

ポリシー構文

IAM ポリシーは 1 つ以上のステートメントで構成される JSON ドキュメントです。各ステートメントは次のように構成されます。

```
{
  "Statement":[{
    "Effect":"effect",
    "Action":"action",
    "Resource":"arn",
    "Condition":{
      "condition":{
    "key":"value"
    }
      }
    }
  ]
}
```

ステートメントはさまざまな工レメントで構成されます。

- Effect: effect は、`Allow` または `Deny` にすることができます。デフォルトでは、 ユーザーはリソースおよび API アクションを使用するアクセス許可がありません。そのため、すべてのリクエストが拒否されます。明示的な許可はデフォルトに上書きされます。明示的な拒否はすべての許可に上書きされます。
- Action] (アクション): action は、アクセス許可を付与または拒否する対象とする、特定の API アクションです。アクションを指定する方法に関する指示は、AWS Batch のアクション を参照してください。

- Resource] (リソース): アクションによって影響を及ぼされるリソースです。AWS Batch API アクションの中には、アクションによって作成/変更できるリソースをポリシー内で特定できるものもあります。ステートメントでリソースを指定するには、Amazon リソースネーム (ARN) を使用します。詳細については、AWS Batch API アクションでサポートされるリソースレベルのアクセス許可およびAWS Batch 用の Amazon リソースネーム を参照してください。AWS Batch API オペレーションで、現在リソースレベルのアクセス許可がサポートされていない場合、ワイルドカード (*) を含めて、アクションがすべてのリソースに影響するように指定する必要があります。
- Condition] (条件): condition はオプションです。ポリシーの発効条件を指定するために使用します。

AWS Batch のIAMポリシーステートメント例についての詳細は、AWS Batch IAM ポリシーの作成を参照してください。

AWS Batch のアクション

IAM ポリシーステートメントで、IAM をサポートするすべてのサービスから任意の API アクションを指定できます。AWS Batch の場合、API アクションの名前に以下のプレフィックスを使います : `batch:` (例、`batch:SubmitJob` および `batch:CreateComputeEnvironment`)。

単一のステートメントで複数のアクションを指定するには、各アクションをカンマで区切ります。

```
"Action": ["batch:action1", "batch:action2"]
```

またワイルドカード (*) を含めて、複数のアクションを指定することもできます。例えば、Describe という単語で始まる名前を用いて、すべてのアクションを指定できます。

```
"Action": "batch:Describe*"
```

すべての AWS Batch API アクションを指定するには、ワイルドカード (*) を含みます。

```
"Action": "batch:*"
```

AWS Batchのアクションのリストについては、AWS Batch API リファレンスのアクションを参照してください

AWS Batch 用の Amazon リソースネーム

各 IAM ポリシーステートメントは、ユーザーがAmazon リソースネーム (ARN) を使用して指定したリソースに適用されます。

Amazon リソースネーム (ARN) には、次の一般的な構文があります:

```
arn:aws:[service]:[region]:[account]:resourceType/resourcePath
```

service

サービス (例: batch)。

region

リソースのAWS リージョン(例: us-east-2)。

account

ハイフンなしの AWS アカウント ID (例: 123456789012)。

resourceType

リソースの種類 (例: compute-environment)。

resourcePath

リソースを識別するパス。パスにワイルドカードの * が使用できます。

AWS Batch API オペレーションは、現在、複数の API オペレーションにおけるリソースレベルのアクセス許可をサポートしています。詳細については、AWS Batch API アクションでサポートされるリソースレベルのアクセス許可を参照してください。すべてのリソースを指定する場合、または特定の API アクションが ARN をサポートしていない場合は、Resource エレメントに (*) ワイルドカードを含めます。

```
"Resource": "*"
```

ユーザーが必要なアクセス許可を持っているかどうかを確認する

IAM ポリシーを本稼働環境に置く前に、そのポリシーがユーザーに対し、特定の API アクションおよび必要なリソースを使用のアクセス許可を付与しているかどうかを確認することをお勧めします。

これを行うには、まずテスト目的のユーザーを作成して、IAM ポリシーをテストユーザーにアタッチします。次に、テストユーザーとしてリクエストを作成します。テストリクエストは、コンソールまたは AWS CLI を使用して行うことができます。

> **Note**
>
> IAM ポリシーシミュレーターを使用してポリシーをテストすることもできます。ポリシーシミュレーターの詳細については、IAM ユーザーガイドのIAM ポリシーシミュレーターで作業するを参照してください。

ポリシーが想定したアクセス許可をユーザーに付与していない場合、または過度に許可されている場合、必要に応じてポリシーを調整できます。必要な結果を得るまで再テストします。

> **Important**
>
> ポリシーの変更が反映され、有効になるには数分間かかります。したがって、ポリシーの更新をテストする前に、合格するには、少なくとも5分みておくことをお勧めします。

認証チェックが失敗した場合、リクエストでは診断情報でエンコードされたメッセージが返されます。`DecodeAuthorizationMessage` アクションを使用してメッセージをデコードできます。詳細については、AWS Security Token Service API リファレンス のDecodeAuthorizationMessage、およびAWS CLI コマンドリファレンスのdecode-authorization-messageを参照してください。

AWS Batch API アクションでサポートされるリソースレベルのアクセス許可

リソースレベルのアクセス許可という用語は、ユーザーがアクションを実行可能なリソースを指定できることを示します。AWS Batch では、リソースレベルのアクセス許可が部分的にサポートされています。いくつかの AWS Batch アクションでは、満たすべき条件に基づいて、ユーザーがそのアクションを使用できるようになるタイミングを制御できます。ユーザーが使用できる特定のリソースに基づいて制御することもできます。たとえば、特定のジョブ定義がある特定のジョブキューでのみ、ジョブを送信するアクセス許可をユーザーに付与できます。

現在、リソースレベルのアクセス許可をサポートしている AWS Batch ルール API アクションのリストを以下に示します。各アクションでサポートされるリソース、リソース ARN、条件キーについても説明しています。

> **Important**
>
> AWS Batch API アクションがこの表に示されていない場合、リソースレベルのアクセス許可をサポートしていません。AWS Batch API アクションがリソースレベルのアクセス許可をサポートしていない場合、アクションを使用するアクセス許可をユーザーに付与できます。ただし、ポリシーステートメントのリソース要素にワイルドカード (*) を含める必要があります。

アクション

CancelJob, CreateComputeEnvironment, CreateJobQueue, CreateSchedulingPolicy, DeleteComputeEnvironment, DeleteJobQueue, DeleteSchedulingPolicy, DeregisterJobDefinition, ListTagsForResource, RegisterJobDefinition, SubmitJob, TagResource, TerminateJob, UntagResource, UpdateComputeEnvironment, UpdateSchedulingPolicy, UpdateJobQueue

CancelJob

AWS Batch キュー内のジョブをキャンセルします。

リソース]

ジョブ

arn:aws:batch:*region*:*account*:job/*jobId*

条件キー

`aws:ResourceTag/${TagKey}` (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

CreateComputeEnvironment

AWS Batchコンピューティング環境を作成します。

リソース]

コンピューティング環境

arn:aws:batch:*region*:*account*:compute-environment/*compute-environment-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

条件キー

aws:RequestTag/${TagKey} (文字列)

リクエストで渡されたタグに基づいてアクションをフィルタリングします。

aws:TagKeys (文字列)

リクエストで渡されたタグキーに基づいてアクションをフィルタリングします。

CreateJobQueue

AWS Batch ジョブキューを作成します。

リソース]

コンピューティング環境

arn:aws:batch:*region*:*account*:compute-environment/*compute-environment-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

ジョブキュー

arn:aws:batch:*region*:*account*:job-queue/*queue-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

スケジューリングポリシー

arn:aws:batch:*region*:*account*:scheduling-policy/*scheduling-policy-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

条件キー

aws:RequestTag/${TagKey} (文字列)

リクエストで渡されたタグに基づいてアクションをフィルタリングします。

aws:TagKeys (文字列)

リクエストで渡されたタグキーに基づいてアクションをフィルタリングします。

DeleteComputeEnvironment

AWS Batch コンピューティング環境を削除します。

リソース]

コンピューティング環境

arn:aws:batch:*region*:*account*:compute-environment/*compute-environment-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

CreateSchedulingPolicy

AWS Batchスケジューリングポリシーを作成します。

リソース]

スケジューリングポリシー

arn:aws:batch:*region*:*account*:scheduling-policy/*scheduling-policy-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

条件キー

aws:RequestTag/${TagKey} (文字列)

リクエストで渡されたタグに基づいてアクションをフィルタリングします。

aws:TagKeys (文字列)

リクエストで渡されたタグキーに基づいてアクションをフィルタリングします。

DeleteJobQueue

指定されたジョブキューを削除します。ジョブキューを削除すると、最終的にキュー内のすべてのジョブが削除されます。ジョブは、毎秒約 16 ジョブの割合で削除されます。

リソース]

ジョブキュー

arn:aws:batch:*region*:*account*:job-queue/*queue-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

DeleteSchedulingPolicy

指定されたスケジューリングポリシーを削除します。

リソース]

スケジューリングポリシー

arn:aws:batch:*region*:*account*:scheduling-policy/*scheduling-policy-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

DeregisterJobDefinition

AWS Batch ジョブの定義を登録解除します。

リソース]

ジョブ定義

arn:aws:batch:*region*:*account*:job-definition/*definition-name*:*revision*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

ListTagsForResource

指定したリソースのタグを一覧表示します。

リソース]

コンピューティング環境

arn:aws:batch:*region*:*account*:compute-environment/*compute-environment-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

ジョブ

arn:aws:batch:*region*:*account*:job/*jobId*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

ジョブ定義

arn:aws:batch:*region*:*account*:job-definition/*definition-name*:*revision*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

ジョブキュー

arn:aws:batch:*region*:*account*:job-queue/*queue-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

スケジューリングポリシー

arn:aws:batch:*region*:*account*:scheduling-policy/*scheduling-policy-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

RegisterJobDefinition

AWS Batch定義を登録します。

リソース]

ジョブ定義

arn:aws:batch:*region*:*account*:job-definition/*definition-name*

条件キー

batch:AWSLogsCreateGroup (ブール)

このパラメータが true の場合、awslogs-group がログに対して作成されます。

batch:AWSLogsGroup (文字列)

ログが配置されている awslogs グループ。

batch:AWSLogsRegion (文字列)

ログが送信されるリージョン。

batch:AWSLogsStreamPrefix (文字列)

awslogs ログストリームのプレフィックス。

`batch:Image` (文字列)

ジョブを開始するときに使用される Docker イメージ。

`batch:LogDriver` (文字列)

ジョブに使用されるログドライバー。

`batch:Privileged` (ブール)

このパラメータが true のとき、コンテナには、ホストコンテナインスタンスに対する昇格されたアクセス許可が付与されます。

`batch:User` (文字列)

ジョブのコンテナ内で使用するユーザー名または数値 uid。

`aws:RequestTag/${TagKey}` (文字列)

リクエストで渡されたタグに基づいてアクションをフィルタリングします。

`aws:TagKeys` (文字列)

リクエストで渡されたタグキーに基づいてアクションをフィルタリングします。

SubmitJob

ジョブ定義から AWS Batch ジョブを送信します。

リソース]

ジョブ

arn:aws:batch:*region*:*account*:job/*jobId*

条件キー

`aws:ResourceTag/${TagKey}` (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

ジョブ定義

arn:aws:batch:*region*:*account*:job-definition/*definition-name*[:*revision*]

条件キー

`aws:ResourceTag/${TagKey}` (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

> **Note**
>
> このキーは、ジョブ定義の Amazon リソースネーム (ARN) が arn:aws:batch:*region*:*account_number*:job-definition/*definition-name*:*revision*. の形式の場合にのみ使用できます。

ジョブキュー

arn:aws:batch:*region*:*account*:job-queue/*queue-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

TagResource

指定されたリソースをタグ付けします。

リソース]

コンピューティング環境

arn:aws:batch:*region*:*account*:compute-environment/*compute-environment-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

ジョブ

arn:aws:batch:*region*:*account*:job/*jobId*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

ジョブ定義

arn:aws:batch:*region*:*account*:job-definition/*definition-name*:*revision*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

ジョブキュー

arn:aws:batch:*region*:*account*:job-queue/*queue-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

スケジューリングポリシー

arn:aws:batch:*region*:*account*:scheduling-policy/*scheduling-policy-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

条件キー

aws:RequestTag/${TagKey} (文字列)

リクエストで渡されたタグに基づいてアクションをフィルタリングします。

aws:TagKeys (文字列)

リクエストで渡されたタグキーに基づいてアクションをフィルタリングします。

TerminateJob

AWS Batch ジョブキュー内のジョブを終了します。

リソース]

ジョブ

arn:aws:batch:*region*:*account*:job/*jobId*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

UntagResource

指定されたリソースのタグを解除します。

リソース]

コンピューティング環境

arn:aws:batch:*region*:*account*:compute-environment/*compute-environment-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

ジョブ

arn:aws:batch:*region*:*account*:job/*jobId*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

ジョブ定義

arn:aws:batch:*region*:*account*:job-definition/*definition-name*:*revision*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

ジョブキュー

arn:aws:batch:*region*:*account*:job-queue/*queue-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

スケジューリングポリシー

arn:aws:batch:*region*:*account*:scheduling-policy/*scheduling-policy-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

条件キー

aws:TagKeys (文字列)

リクエストで渡されたタグキーに基づいてアクションをフィルタリングします。

UpdateComputeEnvironment

AWS Batch コンピューティング環境を更新します。

リソース]

コンピューティング環境

arn:aws:batch:*region*:*account*:compute-environment/*compute-environment-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

UpdateJobQueue

ジョブキューを更新します。

リソース]

ジョブキュー

arn:aws:batch:*region*:*account*:job-queue/*queue-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

スケジューリングポリシー

arn:aws:batch:*region*:*account*:scheduling-policy/*scheduling-policy-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

UpdateSchedulingPolicy

スケジューリングポリシーを更新します。

リソース]

スケジューリングポリシー

arn:aws:batch:*region*:*account*:scheduling-policy/*scheduling-policy-name*

条件キー

aws:ResourceTag/${TagKey} (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

AWS Batch API のアクションの条件キー

AWS Batch は、IAM ポリシーの Condition 要素で使用される以下の条件キーを定義します。これらのキーを使用して、ポリシーステートメントが適用される条件を絞り込むことができます。すべてのサービスで使用できるグローバル条件キーを確認するには、IAM ユーザーガイドの使用できるグローバル条件キーを参照してください。

batch:AWSLogsCreateGroup (ブール)

このパラメータが true の場合、awslogs-group がログに対して作成されます。

`batch:AWSLogsGroup` (文字列)

ログが配置されている awslogs グループ。

`batch:AWSLogsRegion` (文字列)

ログが送信される AWS リージョン。

`batch:AWSLogsStreamPrefix` (文字列)

awslogs ログストリームのプレフィックス。

`batch:Image` (文字列)

ジョブを開始するときに使用される Docker イメージ。

`batch:LogDriver` (文字列)

ジョブに使用されるログドライバー。

`batch:Privileged` (ブール)

このパラメータが true のとき、ジョブコンテナには、ホストコンテナインスタンスに対する昇格されたアクセス許可 (ルートユーザーと同様) が付与されます。

`aws:ResourceTag/${TagKey}` (文字列)

リソースに関連付けられているタグに基づいてアクションをフィルタリングします。

`aws:RequestTag/${TagKey}` (文字列)

リクエストで渡されたタグに基づいてアクションをフィルタリングします。

`batch:ShareIdentifier` (文字列)

SubmitJob に送られた shareIdentifier パラメータに基づいたアクションをフィルタリングします。

`aws:TagKeys` (文字列)

リクエストで渡されたタグキーに基づいてアクションをフィルタリングします。

`batch:User` (文字列)

ジョブのコンテナ内で使用するユーザー名または数値ユーザー ID (uid)。

ポリシーの例

以下の例では、ユーザーの AWS Batch に対するアクセス権限を制御するために使用できるポリシーステートメントについて説明しています。

例

- 読み取り専用アクセス
- POSIX ユーザー、Docker イメージ、特権レベル、ジョブ送信のロールに制限する
- ジョブ送信時にジョブ定義プレフィックスを制限する
- ジョブキューへの制限
- すべての条件キーが文字列に一致した場合はアクションを拒否する
- いずれかの条件キーが文字列に一致した場合はアクションを拒否する
- batch:ShareIdentifier 条件キーを使用する

読み取り専用アクセス

次のポリシーでは、名前が `Describe` および `List` で始まるすべての AWS Batch API アクションを使用できるアクセス権限をユーザーに付与します。

別のステートメントでアクセス権限を付与されない限り、ユーザーにはそのリソースに対してアクションを実行するアクセス権限がありません。デフォルトでは、API アクションを使用する権限は拒否されます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "batch:Describe*",
                "batch:List*"
            ],
            "Resource": "*"
        }
    ]
}
```

POSIX ユーザー、Docker イメージ、特権レベル、ジョブ送信のロールに制限する

次のポリシーは、POSIX ユーザーが自身の制限されたジョブ定義のセットを管理することを許可します。

最初と 2 番目のステートメントを使用して、名前が *JobDefA_* で始まるジョブ定義を登録および登録解除します。

また、最初のステートメントでは、条件付きコンテキストキーを使用して POSIX ユーザー、特権ステータス、コンテナイメージ値をジョブ定義の `containerProperties` 内に制限します。詳細については、AWS Batch API リファレンスの RegisterJobDefinition を参照してください。この例では、POSIX ユーザーが nobody に設定されている場合にのみ、ジョブ定義を登録できます。特権フラグは `false` に設定されています。最後に、イメージは Amazon ECR レポジトリの `myImage` に設定されています。

> **Important**
>
> Docker は、コンテナイメージ内から `user` パラメータをユーザー `uid` に解決します。ほとんどの場合、これはコンテナイメージ内の `/etc/passwd` ファイルにあります。ジョブ定義と関連付けられたすべての IAM ポリシーの両方で直接 `uid` 値を使用することで、この名前解決を回避できます。AWS Batch API オペレーションおよび `batch:User` IAM 条件キーのいずれにおいても、数値がサポートされます。

3 番目のステートメントを使用して、ジョブ定義を特定のロールのみに制限します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "batch:RegisterJobDefinition"
            ],
            "Resource": [
                "arn:aws:batch:<aws_region>:<aws_account_id>:job-definition/JobDefA_*"
            ],
            "Condition": {
                "StringEquals": {
```

```
                    "batch:User": [
                        "nobody"
                    ],
                    "batch:Image": [
                        "<aws_account_id>.dkr.ecr.<aws_region>.amazonaws.com/myImage"
                    ]
                },
                "Bool": {
                    "batch:Privileged": "false"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "batch:DeregisterJobDefinition"
            ],
            "Resource": [
                "arn:aws:batch:<aws_region>:<aws_account_id>:job-definition/JobDefA_*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::<aws_account_id>:role/MyBatchJobRole"
            ]
        }
    ]
}
```

ジョブ送信時にジョブ定義プレフィックスを制限する

次のポリシーを使用して、*JobDefA* で始まるすべてのジョブ定義のジョブキューにジョブを送信します。

> **Important**
>
> ジョブ送信へのリソースレベルアクセスに絞り込む場合、ジョブキューおよびジョブ定義の両方のリソースタイプを指定する必要があります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "batch:SubmitJob"
      ],
      "Resource": [
        "arn:aws:batch:<aws_region>:<aws_account_id>:job-definition/JobDefA_*",
        "arn:aws:batch:<aws_region>:<aws_account_id>:job-queue/*"
      ]
    }
  ]
}
```

ジョブキューへの制限

次のポリシーを使用して、任意のジョブ定義名で queue1 という名前の特定のジョブキューへのジョブの送信をユーザーに許可します。

> **Important**
>
> ジョブ送信へのリソースレベルアクセスに絞り込む場合、ジョブキューおよびジョブ定義の両方のリソースタイプを指定する必要があります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "batch:SubmitJob"
      ],
      "Resource": [
        "arn:aws:batch:<aws_region>:<aws_account_id>:job-definition/*",
        "arn:aws:batch:<aws_region>:<aws_account_id>:job-queue/queue1"
      ]
    }
  ]
```

```
}
```

すべての条件キーが文字列に一致した場合はアクションを拒否する

次のポリシーでは、batch:Image (コンテナイメージ ID) 条件キーが "*string1*" で、batch:LogDriver (コンテナログドライバー) 条件キーが "*string2*" の場合、RegisterJobDefinition API オペレーションへのアクセスを拒否します。AWS Batch各コンテナの条件キーを評価します。マルチノードの並列ジョブのように、ジョブが複数のコンテナにまたがる場合、コンテナの構成が異なる可能性があります。1 つのステートメントで複数の条件キーを評価する場合、条件キーは AND ロジックを使用して結合されます。そのため、1 つのコンテナで複数の条件キーのいずれかが一致しない場合、そのコンテナに Deny の効果は適用されません。それどころか、同じジョブ内の別のコンテナが拒否される可能性があります。

AWS Batch の条件キーのリストについては、サービス認可リファレンスのAWS Batch の条件キーを参照してください。この方法は、batch:ShareIdentifier を除くすべての batch 条件キーで使用できます。batch:ShareIdentifier 条件キーは、ジョブ定義ではなく、ジョブに対して定義されます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "batch:RegisterJobDefinition"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Deny",
      "Action": "batch:RegisterJobDefinition",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "batch:Image": "string1",
          "batch:LogDriver": "string2"
        }
      }
    }
```

```
  ]
}
```

いずれかの条件キーが文字列に一致した場合はアクションを拒否する

次のポリシーは、batch:Image (コンテナイメージ ID) 条件キーが "*string1*" または batch:LogDriver (コンテナログドライバー) 条件キーが "*string2*" の場合、RegisterJobDefinition API オペレーションへのアクセスを拒否します。マルチノードの並列ジョブのように、ジョブが複数のコンテナにまたがる場合、コンテナの構成が異なる可能性があります。1 つのステートメントで複数の条件キーを評価する場合、条件キーは AND ロジックを使用して結合されます。そのため、1 つのコンテナで複数の条件キーのいずれかが一致しない場合、そのコンテナに Deny の効果は適用されません。それどころか、同じジョブ内の別のコンテナが拒否される可能性があります。

AWS Batch の条件キーのリストについては、サービス認可リファレンスのAWS Batch の条件キーを参照してください。この方法は、batch:ShareIdentifier を除くすべての batch 条件キーで使用できます。(batch:ShareIdentifier 条件キーは、ジョブ定義ではなく、ジョブに対して定義されます)。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "batch:RegisterJobDefinition"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Deny",
      "Action": [
        "batch:RegisterJobDefinition"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
```

```
          "batch:Image": [
            "string1"
          ]
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": [
        "batch:RegisterJobDefinition"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "batch:LogDriver": [
            "string2"
          ]
        }
      }
    }
  ]
}
```

batch:ShareIdentifier 条件キーを使用する

次のポリシーを使用して、jobDefA ジョブ定義を使用するジョブを lowCpu 配分識別子で jobqueue1 ジョブキューに送信します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "batch:SubmitJob"
      ],
      "Resource": [
        "arn:aws::batch:<aws_region>:<aws_account_id>:job-definition/JobDefA",
        "arn:aws::batch:<aws_region>:<aws_account_id>:job-queue/jobqueue1"
      ],
      "Condition": {
```

```
        "StringEquals": {
          "batch:ShareIdentifier": [
            "lowCpu"
          ]
        }
      }
    }
  ]
}
```

AWS Batch 管理ポリシー

AWS Batch は、 ユーザーにアタッチできる管理ポリシーを提供し、このポリシーにより、AWS Batch のリソースおよび API オペレーションを使用する権限を付与します。このポリシーを直接適用することも、独自のポリシーを作成する開始点として使用することもできます。これらのポリシーに記載されている各 API オペレーションの詳細については、AWS Batch API リファレンスのアクションを参照してください。

AWSBatchFullAccess

このポリシーでは、管理者権限による AWS Batch へのフルアクセスを許可します。

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect":"Allow",
      "Action":[
        "batch:*",
        "cloudwatch:GetMetricStatistics",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeKeyPairs",
        "ec2:DescribeVpcs",
        "ec2:DescribeImages",
        "ec2:DescribeLaunchTemplates",
        "ec2:DescribeLaunchTemplateVersions",
        "ecs:DescribeClusters",
        "ecs:Describe*",
        "ecs:List*",
        "eks:DescribeCluster",
        "eks:ListClusters",
```

```
        "logs:Describe*",
        "logs:Get*",
        "logs:TestMetricFilter",
        "logs:FilterLogEvents",
        "iam:ListInstanceProfiles",
        "iam:ListRoles"
      ],
      "Resource":"*"
    },
    {
      "Effect":"Allow",
      "Action":[
        "iam:PassRole"
      ],
      "Resource":[
        "arn:aws:iam::*:role/AWSBatchServiceRole",
        "arn:aws:iam::*:role/service-role/AWSBatchServiceRole",
        "arn:aws:iam::*:role/ecsInstanceRole",
        "arn:aws:iam::*:instance-profile/ecsInstanceRole",
        "arn:aws:iam::*:role/iaws-ec2-spot-fleet-role",
        "arn:aws:iam::*:role/aws-ec2-spot-fleet-role",
        "arn:aws:iam::*:role/AWSBatchJobRole*"
      ]
    },
    {
      "Effect":"Allow",
      "Action":[
        "iam:CreateServiceLinkedRole"
      ],
      "Resource":"arn:aws:iam::*:role/*Batch*",
      "Condition": {
        "StringEquals": {
          "iam:AWSServiceName": "batch.amazonaws.com"
          }
      }
    }
  ]
}
```

AWS Batch IAM ポリシーの作成

アカウントのユーザーがアクセスできる呼び出しやリソースを制限する特定の IAM ポリシーを作成できます。次に、このポリシーをユーザーにアタッチできます。

ポリシーをユーザーまたはグループにアタッチすると、ポリシーによって特定リソースについて特定タスクを実行する権限が許可または拒否されます。詳細については、IAM ユーザーガイド の アクセス許可とポリシーを参照してください。カスタム IAM ポリシーを管理および作成する方法については、IAM ポリシーの管理を参照してください。

Amazon ECS インスタンスロール

AWS Batch コンピューティング環境には、Amazon ECS コンテナインスタンスが自動入力されます。Amazon ECS コンテナエージェントがローカルで実行されます。Amazon ECS コンテナエージェントは、ユーザーに代わり様々な AWS API オペレーションの呼び出しを行います。そのため、エージェントを実行するコンテナインスタンスには、エージェントがユーザーに属していることこれらのサービスに伝えるIAM ポリシーとロールが必要です。コンテナインスタンスを起動するときに使用する IAM ロールとインスタンスプロファイルを作成する必要があります。それ以外の場合、コンピューティング環境を作成してコンテナインスタンスを起動することはできません。この要件が適用されるコンテナインスタンスの起動には、Amazon が提供する、Amazon ECS に最適化された AMI が使用されている場合と使用されていない場合があります。詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS コンテナインスタンス IAM ロールを参照してください。

コンソールの初回実行時には、Amazon ECS インスタンスのロールおよびインスタンスプロファイルが自動的に作成されます。ただし、次の手順を使用して、アカウントに既に Amazon ECS インスタンスロールおよびインスタンスプロファイルが存在するか確認することができます。以下の手順では、マネージド IAM ポリシーをアタッチする方法についても説明します。

IAM コンソールで **`ecsInstanceRole`** を確認するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで Roles] (ロール) を選択します。
3. ロールのリストで `ecsInstanceRole` を検索します。ロールが見つからない場合、以下のステップを使用してロールを作成します。

    a. ロールの作成] を選択します。
    b. 信頼できるエンティティタイプ で、AWS のサービス を選択します。
    c. 一般的なユースケース で EC2 を選択します。
    d. 次へ をクリックします。
    e. アクセス権限ポリシー については、AmazonEC2ContainerServiceforEC2Role を検索してください。

f. AmazonEC2ContainerServiceforEC2Role の横にあるチェックボックスを選択し、次へ を選択します。

g. ロール名 には、ecsInstanceRole を入力し、そしてロールの作成] を選択します。

> **Note**
>
> AWS Management Console を使用して Amazon EC2 のロールを作成する場合、コンソールは、インスタンスプロファイルを自動的に作成し、そのインスタンスプロファイルにロールと同じ名前を付けます。

または、AWS CLI を使用してecsInstanceRole IAM ロールを作成することもできます。次の実行ポリシーと信頼ポリシーを持つ AWS という名前の IAM ロールを作成します。

IAM ロールおよびインスタンスプロファイルを作成するには (AWS CLI)

1. 以下の信頼ポリシーを作成し、ecsInstanceRole-role-trust-policy.json という名前のテキストファイルに保存します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "Service": "ec2.amazonaws.com"},
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. ecsInstanceRoleのロールを作成するには、ロール作成コマンドを使用します。信頼ポリシーファイルの場所を assume-role-policy-document パラメータに指定します。

```
$ aws iam create-role \
    --role-name ecsInstanceRole \
    --assume-role-policy-document file://ecsInstanceRole-role-trust-policy.json
```

以下に、応答の例を示します。

```
{
```

```
    "Role": {
         "Path": "/",
         "RoleName: "ecsInstanceRole",
         "RoleId": "AROAT46P5RDIY4EXAMPLE",
         "Arn": "arn:aws:iam::123456789012:role/ecsInstanceRole".
        "CreateDate": "2022-12-12T23:46:37.247Z",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service: "ec2.amazonaws.com"
                    }
                    "Action": "sts:AssumeRole",
                 }
              ]
          }
        }
```

3. [インスタンスプロファイルの作成]コマンドを使用して、 ecsInstanceRoleという名前のインスタンスプロファイルを作成します。

> **Note**
>
> AWS CLI および AWS API で別個のアクションとして、ロールとインスタンスプロファイルを作成する必要があります。

```
$ aws iam create-instance-profile --instance-profile-name ecsInstanceRole
```

以下に、応答の例を示します。

```
{
    "InstanceProfile": {
        "Path": "/",
        "InstanceProfileName": "ecsInstanceRole",
        "InstanceProfileId": "AIPAT46P5RDITREXAMPLE",
        "Arn": "arn:aws:iam::123456789012:instance-profile/ecsInstanceRole",
        "CreateDate": "2022-06-30T23:53:34.093Z",
        "Roles": [],    }
```

```
}
```

4. [インスタンスプロフィールにロールを追加] コマンドを使用して、ecsInstanceRoleのロールをecsInstanceRoleインスタンスプロファイルにアタッチします。

```
aws iam add-role-to-instance-profile \
    --role-name ecsInstanceRole --instance-profile-name ecsInstanceRole
```

5. ポリシーにロールを追加 コマンドを使用して、AmazonEC2ContainerServiceforEC2RoleAWS 管理ポリシーを ecsInstanceRole ロールにアタッチします。

```
$ aws iam attach-role-policy \
    --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonEC2ContainerServiceforEC2Role \
    --role-name ecsInstanceRole
```

Amazon EC2 スポットフリートロール

Amazon EC2 スポットフリートインスタンスを使用するマネージド型のコンピューティング環境環境を作成する場合は、AmazonEC2SpotFleetTaggingRole ポリシーを作成しなければなりません。このポリシーは、ユーザーに代わりインスタンスの起動、タグ付けおよび終了を行うためのスポットフリート許可を付与します。スポットフリートのリクエストでロールを指定します。また、Amazon EC2 スポットおよびスポットフリートにサービスにリンクされたロール AWSServiceRoleForEC2Spot] および AWSServiceRoleForEC2SpotFleet] がある必要があります。次の手順に従って、これらすべてのロールを作成します。詳細については、IAM ユーザーガイドのサービスにリンクされたロールの使用およびAWS サービスに対してアクセス許可を委譲するロールの作成を参照してください。

トピック

- AWS Management Console において、Amazon EC2 スポットフリートロールを作成する
- AWS CLI を使用して、Amazon EC2 スポットフリートのロールを作成する

AWS Management Console において、Amazon EC2 スポットフリートロールを作成する

Amazon EC2 スポットフリートの **AmazonEC2SpotFleetTaggingRole** IAM サービスにリンクされたロールを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. アクセス管理] のために、ロール を選択します。
3. ロール] には、ロールの作成 を選択します。
4. 信頼されるエンティティを選択] (信頼されるエンティティタイプにある) から、AWS のサービス]を選択します。
5. その他のユースケース AWS のサービス には 、EC2 を選択し、次に EC2 - スポットフリートタギング] を選択します。
6. 次へ を選択します。
7. ポリシー名 の権限ポリシー から、AmazonEC2SpotFleetTaggingRoleを確認します 。
8. 次へ を選択します。
9. 名前、確認および作成については:
    a. ロールを識別するために、名前タグ に名前を入力します。
    b. 説明 には、ポリシーの簡単な説明を入力します。
    c. (オプション) ステップ 1: 信頼できるエンティティの選択 では編集 を選択して、コードを変更します。
    d. (オプション)ステップ 2: 権限の追加 では、編集 を選択してコードを変更します。
    e. (オプション) タグを追加 で タグを追加 を選択し、リソースにタグを追加します。
    f. ロールの作成] を選択します。

> **Note**
>
> これまでは、Amazon EC2 スポットフリートロールに対し 2つの管理ポリシーがありました。
>
> - AmazonEC2SpotFleetRole: これは、スポットフリートロール用のオリジナルの管理ポリシーです。ただし、これを AWS Batchと一緒に使用することはもう推奨されていません。このポリシーは、AWSServiceRoleForBatch のサービスにリンクされたロールを使用す

るために必要なコンピューティング環境でのスポットインスタンスのタグ付けをサポートしていません。以前に、このポリシーを使用してスポットフリートロールを作成した場合は、新しい推奨ポリシーをそのロールに適用してください。詳細については、作成時にタグが付けられていないスポットインスタンスを参照してください。

- AmazonEC2SpotFleetTaggingRole: このロールでは、Amazon EC2 スポットインスタンスにタグを付けるために必要なすべてのアクセス権限が提供されます。このロールを使用して、AWS Batchコンピューティング環境でのスポットインスタンスのタグ付けを許可します。

AWS CLI を使用して、Amazon EC2 スポットフリートのロールを作成する

スポットフリートのコンピューティング環境の AmazonEC2SpotFleetTaggingRole IAM ロールを作成するには

1. AWS CLI を使用して次のコマンドを実行します。

```
$ aws iam create-role --role-name AmazonEC2SpotFleetTaggingRole \
    --assume-role-policy-document '{
    "Version":"2012-10-17",
    "Statement":[
       {
         "Sid":"",
         "Effect":"Allow",
         "Principal": {
           "Service":"spotfleet.amazonaws.com"
          },
          "Action":"sts:AssumeRole"
        }
      ]
    }'
```

2. AmazonEC2SpotFleetTaggingRole マネージド IAM ポリシーを AmazonEC2SpotFleetTaggingRole ロールにアタッチするには、以下のコマンドを AWS CLI で実行します。

```
$ aws iam attach-role-policy \
  --policy-arn \
    arn:aws:iam::aws:policy/service-role/AmazonEC2SpotFleetTaggingRole \
  --role-name \
```

```
    AmazonEC2SpotFleetTaggingRole
```

Amazon EC2 スポットの **AWSServiceRoleForEC2Spot** IAMのサービスにリンクされたロールを作成するには

> **Note**
>
> AWSServiceRoleForEC2Spot IAM サービスにリンクされたロールがすでに存在する場合は、次のようなエラーメッセージが表示されます。
>
> ```
> An error occurred (InvalidInput) when calling the CreateServiceLinkedRole
>  operation:
> Service role name AWSServiceRoleForEC2Spot has been taken in this account,
>  please try a different suffix.
> ```

- AWS CLI を使用して次のコマンドを実行します。

```
$ aws iam create-service-linked-role --aws-service-name spot.amazonaws.com
```

Amazon EC2 スポットフリートの **AWSServiceRoleForEC2SpotFleet** IAM サービスにリンクされたロールを作成するには

> **Note**
>
> AWSServiceRoleForEC2SpotFleet IAM サービスにリンクされたロールがすでに存在する場合は、次のようなエラーメッセージが表示されます。
>
> ```
> An error occurred (InvalidInput) when calling the CreateServiceLinkedRole
>  operation:
> Service role name AWSServiceRoleForEC2SpotFleet has been taken in this account,
>  please try a different suffix.
> ```

- AWS CLI を使用して次のコマンドを実行します。

```
$ aws iam create-service-linked-role --aws-service-name spotfleet.amazonaws.com
```

EventBridge IAM ロール

Amazon EventBridgeは、 AWS リソースにおける変更を示すシステムイベントのストリームをほぼリアルタイムで提供します。AWS Batch ジョブは EventBridgeターゲットとして使用できます。すぐに設定できる簡単なルールを使用して、それらのルールに対応して、イベントを一致させ AWS Batch ジョブを送信できます。CloudWatch Events ルールおよびターゲットを使用して AWS Batch ジョブを送信できるようにするには、CloudWatch Eventsに、ユーザーに代わって AWS Batch ジョブを実行する権限が必要です。

> **Note**
>
> AWS Batch キューをターゲットとして指定するルールを EventBridgeコンソールで作成する場合、このロールを作成することができます。チュートリアル例については、AWS Batch EventBridge ターゲットとしての ジョブを参照してください。IAM コンソールを使って、ユーザーはEventBridge ロールをマニュアルで作成できます。詳細については、IAM ユーザーガイドのIAM ユーザーガイドの作成 (コンソール) を参照してください。

EventBridge IAMsのIAM ロールの信頼関係では、`events.amazonaws.com` サービスプリンシパルでロールを継承することを許可する必要があります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "events.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

EventBridge IAM ロールにアタッチされているポリシーが、リソースへの batch:SubmitJob 許可を与えていることを確認してください。次の例では、AWS Batch が、AWSBatchServiceEventTargetRoleの管理ポリシーを提供してこれらのアクセス権限を付与しています。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "batch:SubmitJob"
       ],
      "Resource": "*"
    }
  ]
}
```

AWS Batch Amazon のイベントストリーム EventBridge

Amazon の AWS Batch イベントストリームを使用して EventBridge 、ジョブキュー内のジョブの現在の状態に関するほぼリアルタイムの通知を受け取ることができます。

を使用して EventBridge 、 AWS Batch サービスに関する詳細なインサイトを得ることができます。具体的には、ジョブの進行状況の確認、 AWS Batch カスタムワークフローの構築、使用状況レポートまたはメトリクスの生成、独自のダッシュボードの構築に使用できます。 AWS Batch および では EventBridge、 AWS Batch ジョブのステータス変更を継続的にポーリングするスケジューリングおよびモニタリングコードは必要ありません。代わりに、さまざまな Amazon EventBridge ターゲットを使用して、 AWS Batch ジョブ状態の変更を非同期的に処理できます。これには、 AWS Lambda、Amazon Simple Queue Service、Amazon Simple Notification Service、または Amazon Kinesis Data Streams が含まれます。

イベントストリームからの AWS Batch イベントは、少なくとも 1 回確実に配信されます。重複したイベントが送信された場合でも、イベントには重複を識別できるだけの十分な情報が備わっています。これにより、イベントのタイムスタンプとジョブの状態を比較できます。

AWS Batch ジョブは EventBridge ターゲットとして使用できます。シンプルなルールを使用すると、イベントを照合し、それに応じて AWS Batch ジョブを送信できます。詳細については、「Amazon EventBridge ユーザーガイド」の「 とは EventBridge」を参照してください。EventBridge を使用して、 cronまたは rate 式を使用して特定の時間に自己トリガーする自動アクションをスケジュールすることもできます。詳細については、「Amazon ユーザーガイド」の「スケジュールに従って実行される Amazon EventBridge ルールの作成」を参照してください。EventBridge チュートリアル例については、AWS Batch EventBridge ターゲットとしての ジョブを参照してください。スケ EventBridge ジューラの使用の詳細については、「Amazon ユーザーガイド」の「Amazon ス EventBridge ケジューラの設定」を参照してください。 EventBridge

トピック

- AWS Batch イベント
- での AWS ユーザー通知の使用 AWS Batch
- AWS Batch EventBridge ターゲットとしての ジョブ
- チュートリアル：AWS Batch EventBridge を聴く
- チュートリアル:ジョブ停止イベントに Amazon シンプル 通知サービス アラートを送信

AWS Batch イベント

AWS Batch はジョブステータス変更イベントを に送信します EventBridge。 はジョブの状態 AWS Batch を追跡します。以前に提出されたジョブの状態が変わった場合、イベントが起動します。例えば、RUNNING ステータスにあるジョブが FAILED ステータスに移動した場合などです。これらのイベントはジョブ状態の変更イベントとして分類されます。

> **Note**
>
> AWS Batch 将来、他のイベントタイプ、ソース、詳細が追加される可能性があります。イベントの JSON データをプログラムで逆シリアル化する場合は、不明なプロパティが追加されているときにアプリケーションで対応する準備ができていることを確認してください。これは、これらのプロパティが追加された場合に,問題が発生するのを防ぐためです。

ジョブ状態変更イベント

(以前に送信された) 既存のジョブで状態が変更されると、イベントが作成されます。 AWS Batch ジョブの状態の詳細については、「」を参照してくださいジョブの状態。

> **Note**
>
> 最初のジョブの送信では、イベントは作成されません。

Example ジョブ状態変更イベント

タスク状態変更イベントは、次の形式で配信されます。detail セクションは、 DescribeJobs API AWS Batch リファレンスの API オペレーションから返される JobDetail オブジェクトに似ています。 EventBridge パラメータの詳細については、「Amazon EventBridge ユーザーガイド」の「イベントとイベントパターン」を参照してください。

```
{
    "version": "0",
    "id": "c8f9c4b5-76e5-d76a-f980-7011e206042b",
    "detail-type": "Batch Job State Change",
    "source": "aws.batch",
    "account": "123456789012",
    "time": "2022-01-11T23:36:40Z",
    "region": "us-east-1",
```

```
    "resources": [
        "arn:aws:batch:us-east-1:123456789012:job/4c7599ae-0a82-49aa-ba5a-4727fcce14a8"
    ],
    "detail": {
        "jobArn": "arn:aws:batch:us-east-1:123456789012:job/4c7599ae-0a82-49aa-
ba5a-4727fcce14a8",
        "jobName": "event-test",
        "jobId": "4c7599ae-0a82-49aa-ba5a-4727fcce14a8",
        "jobQueue": "arn:aws:batch:us-east-1:123456789012:job-queue/
PexjEHappyPathCanary2JobQueue",
        "status": "RUNNABLE",
        "attempts": [],
        "createdAt": 1641944200058,
        "retryStrategy": {
            "attempts": 2,
            "evaluateOnExit": []
        },
        "dependsOn": [],
        "jobDefinition": "arn:aws:batch:us-east-1:123456789012:job-definition/first-
run-job-definition:1",
        "parameters": {},
        "container": {
            "image": "137112412989.dkr.ecr.us-east-1.amazonaws.com/amazonlinux:latest",
            "command": [
                "sleep",
                "600"
            ],
            "volumes": [],
            "environment": [],
            "mountPoints": [],
            "ulimits": [],
            "networkInterfaces": [],
            "resourceRequirements": [
                {
                    "value": "2",
                    "type": "VCPU"
                }, {
                    "value": "256",
                    "type": "MEMORY"
                }
            ],
            "secrets": []
        },
        "tags": {
```

```
            "resourceArn": "arn:aws:batch:us-
east-1:123456789012:job/4c7599ae-0a82-49aa-ba5a-4727fcce14a8"
        },
        "propagateTags": false,
        "platformCapabilities": []
    }
}
```

ジョブキューのブロックされたイベント

が RUNNABLE状態のジョブ AWS Batch を検出してキューをブロックするたびに、Amazon CloudWatch Events でイベントが作成されます。サポートされているブロックされたキューの原因の詳細については、「ブロックされたジョブキューメッセージの例」を参照してください。API DescribeJobsアクションの statusReasonフィールドでも同じ理由を使用できます。

Example ジョブ状態変更イベント

タスク状態変更イベントは、次の形式で配信されます。detail セクションは、 DescribeJobs API AWS Batch リファレンスの API オペレーションから返される JobDetail オブジェクトに似ています。パラメータの詳細については EventBridge、「Amazon EventBridge ユーザーガイド」の「イベントとイベントパターン」を参照してください。

```
{
    "version": "0",
    "id": "c8f9c4b5-76e5-d76a-f980-7011e206042b",
    "detail-type": "Batch Job Queue Blocked",
    "source": "aws.batch",
    "account": "123456789012",
    "time": "2022-01-11T23:36:40Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:batch:us-east-1:123456789012:job/4c7599ae-0a82-49aa-
ba5a-4727fcce14a8",
        "arn:aws:batch:us-east-1:123456789012:job-queue/PexjEHappyPathCanary2JobQueue"
    ],
    "detail": {
        "jobArn": "arn:aws:batch:us-east-1:123456789012:job/4c7599ae-0a82-49aa-
ba5a-4727fcce14a8",
        "jobName": "event-test",
        "jobId": "4c7599ae-0a82-49aa-ba5a-4727fcce14a8",
        "jobQueue": "arn:aws:batch:us-east-1:123456789012:job-queue/
PexjEHappyPathCanary2JobQueue",
```

```
        "status": "RUNNABLE",
        "statusReason": "blocked-reason"
        "attempts": [],
        "createdAt": 1641944200058,
        "retryStrategy": {
            "attempts": 2,
            "evaluateOnExit": []
        },
        "dependsOn": [],
        "jobDefinition": "arn:aws:batch:us-east-1:123456789012:job-definition/first-
run-job-definition:1",
        "parameters": {},
        "container": {
            "image": "137112412989.dkr.ecr.us-east-1.amazonaws.com/amazonlinux:latest",
            "command": [
                "sleep",
                "600"
            ],
            "volumes": [],
            "environment": [],
            "mountPoints": [],
            "ulimits": [],
            "networkInterfaces": [],
            "resourceRequirements": [
                {
                    "value": "2",
                    "type": "VCPU"
                }, {
                    "value": "256",
                    "type": "MEMORY"
                }
            ],
            "secrets": []
        },
        "tags": {
            "resourceArn": "arn:aws:batch:us-
east-1:123456789012:job/4c7599ae-0a82-49aa-ba5a-4727fcce14a8"
        },
        "propagateTags": false,
        "platformCapabilities": []
    }
}
```

での AWS ユーザー通知の使用 AWS Batch

AWS ユーザー通知を使用すると、 AWS Batch イベントの通知を受ける配信チャネルを設定できます。指定したルールにイベントが一致すると、通知を受け取ります。イベントの通知は、E メール、AWS Chatbot チャット通知、AWS Console Mobile Application プッシュ通知など、複数のチャネルを通じて受け取ることができます。「コンソール通知センター」で通知を確認することもできます。ユーザー通知は集約をサポートしているため、特定のイベント中に受け取る通知の数を減らすことができます。

でユーザー通知を設定するには AWS Batch：

1. AWS Batch コンソールを開きます。
2. Dashboard を選択します。
3. 通知の設定 を選択します。
4. AWS ユーザー通知 で、通知設定の作成 を選択します。

ユーザー通知を設定および表示する方法については、AWS 「ユーザー通知の開始方法」を参照してください。

AWS Batch EventBridge ターゲットとしての ジョブ

Amazon EventBridge は、Amazon Web Services リソースの変更を示すシステムイベントのストリームをほぼリアルタイムで提供します。通常、 AWS Batch Amazon Elastic Container Service、Amazon Elastic Kubernetes Service、および AWS Fargate ジョブでは、 EventBridge ターゲットとして使用できます。シンプルなルールを使用すると、イベントを照合し、それに応じて AWS Batch ジョブを送信できます。詳細については、「Amazon EventBridge ユーザーガイド」の「 とは EventBridge」を参照してください。

EventBridge を使用して、 cronまたは rate 式を使用して特定の時間に呼び出される自動アクションをスケジュールすることもできます。詳細については、「Amazon ユーザーガイド」の「スケジュールに従って実行される Amazon EventBridge ルールの作成」を参照してください。 EventBridge

イベントがイベントパターンに一致したときに実行されるルールを作成する方法については、「Amazon ユーザーガイド」の「イベントに反応する Amazon EventBridge ルールの作成」を参照してください。 EventBridge

EventBridge ターゲットとしての AWS Batch ジョブの一般的なユースケースには、次のユースケースが含まれます。

- スケジュールされたジョブは、一定の時間間隔で発生します。例えば、cron ジョブは、Amazon EC2 スポットインスタンスのコストが低い時間帯にのみ発生します。
- AWS Batch ジョブは、 にログインされた API オペレーションに応答して実行されます CloudTrail。例えば、指定したAmazon S3 バケットにオブジェクトがアップロードされるたびにジョブが送信されます。この場合、 EventBridge 入力トランスフォーマーはオブジェクトのバケット名とキー名をパラメータに渡 AWS Batch します。

> **Note**
>
> このシナリオでは、すべての関連 AWS リソースが同じリージョンに存在する必要があります。これには、Amazon S3 バケット、 EventBridge ルール、 CloudTrail ログなどのリソースが含まれます。

EventBridge ルールとターゲットを使用して AWS Batch ジョブを送信する前に、 EventBridge サービスには AWS Batch ジョブを実行するためのいくつかのアクセス許可が必要です。 EventBridge コンソールで AWS Batch ジョブをターゲットとして指定するルールを作成する場合、このロールを作成することもできます。このロールに必要なサービスプリンシパルと IAM アクセス権限の詳細については、EventBridge IAM ロールを参照してください。

スケジュールされた AWS Batch ジョブの作成

次の手順では、スケジュールされた AWS Batch ジョブと必要な EventBridge IAM ロールを作成する方法について説明します。

でスケジュールされた AWS Batch ジョブを作成するには EventBridge

> **Note**
>
> この手順は、Amazon ECS、Amazon EKS、および AWS Fargate ジョブのすべての AWS Batch で機能します。

1. https://console.aws.amazon.com/events/ で Amazon EventBridge コンソールを開きます。
2. ナビゲーションバーから、 AWS リージョン 使用する を選択します。
3. ナビゲーションペインで Rules] (ルール) を選択します。
4. ルールの作成 を選択します。

5. 名前 で、コンピューティング環境の一意な名前を指定します。名前は最大 64 文字まで入力できます。大文字、小文字、数字、ハイフン (-)、アンダースコア (_) を含めることができます。

   > **Note**
   >
   > ルールには、同じリージョン内および同じイベントバス上の別のルールと同じ名前を付けることはできません。

6. (オプション) 説明に、ルールの説明を入力します。
7. イベントバス] では、このルールに関連付けるイベントバスを選択します。このルールをアカウントからのイベントと一致させるには、デフォルト を選択します。アカウントの AWS のサービス がイベントを発行すると、常にアカウントのデフォルトのイベントバスに移動します。
8. (オプション) すぐに実行しないバスのルールについては、そのルールをオフにします。
9. ルールタイプ では、スケジュール] を選択します。
10. 続行してルールを作成する または 次へ を選択します。
11. スケジュールパターン] では、次のいずれかを実行します。

    - 毎月第一月曜日の太平洋標準日午前 8:00 など、毎月最初の月曜日の PST に cron 式を入力します。詳細については、「Amazon EventBridge ユーザーガイド」の「Cron 式」を参照してください。
    - 一定の間隔 (10 分ごとなど) で実行するスケジュールを選択してから、rate 式を入力します。

12. 次へ を選択します。
13. ターゲットタイプ] には、AWS のサービス を選択します。
14. ターゲットの選択 で、バッチジョブのキュー を選択します。次を設定します。

    - ジョブキュー]: ジョブをスケジュールするジョブキューの Amazon リソースネーム (ARN) を入力します。
    - ジョブ定義]: ジョブに使用するジョブ定義の名前、改正、または完全な ARN を入力します。
    - ジョブ名]: ジョブの名前を入力します。
    - 配列サイズ]: (オプション) 複数のコピーを実行するためのジョブの配列サイズを入力します。詳細については、配列ジョブを参照してください。
    - ジョブの試行]: (オプション) ジョブが失敗したときに再試行する回数を入力します。詳細については、「ジョブの再試行の自動化」を参照してください。

15. バッチジョブキューのターゲットタイプの場合、 にはターゲットにイベントを送信するアクセス許可 EventBridge が必要です。 EventBridge は、ルールの実行に必要な IAM ロールを作成できます。次のいずれかを行います。

    - 自動的に IAM ロールを作成するには、この特定のリソースに対して新しいロールを作成するを選択します。
    - 以前に作成した IAM ロールを使用するには、既存のロールの使用 を選択します。

16. （オプション）追加設定 を展開します。

    a. ターゲット入力の設定 で、イベントからのテキストをターゲットに渡す前にどのように処理するかを選択します。

    b. イベントの最大保存期間 では、未処理のイベントを保存する時間間隔を指定します。

    c. 再試行 では、イベントを再試行する回数を入力します。

    d. デッドレターキュー では、未処理イベントの取扱いに関するオプションを選択します。必要に応じて、デッドレターキューに Amazon SQS キューを指定します。

17. (オプション) 別のターゲットを追加] を選択して、このルールに別のターゲットを追加します。

18. 次へ を選択します。

19. (オプション) タグ で 新しいタグを追加 を選択し、ルールのリソースラベルを追加します。詳細については、「Amazon EventBridge tags」を参照してください。

20. 次へ を選択します。

21. レビューと作成では、設定手順を確認してください。変更する必要がある場合は、編集] を選択します。完了したら、ルールの作成 を選択します。

ルールの作成の詳細については、「Amazon ユーザーガイド」の「スケジュールに従って実行される Amazon EventBridge ルールの作成」を参照してください。 EventBridge

イベントパターンを持つルールの作成

以下の手順では、イベントパターンを使用してルールを作成する方法について説明します。

イベントが定義されたパターンと一致するときにイベントをターゲットに送信するルールを作成するには

> **Note**
>
> この手順は、Amazon ECS、Amazon EKS、および AWS Fargate ジョブのすべての AWS Batch で機能します。

1. https://console.aws.amazon.com/events/ で Amazon EventBridge コンソールを開きます。
2. ナビゲーションバーから、 AWS リージョン 使用する を選択します。
3. ナビゲーションペインで Rules] (ルール) を選択します。
4. ルールの作成 を選択します。
5. 名前 で、コンピューティング環境の一意な名前を指定します。名前は最大 64 文字まで入力できます。大文字、小文字、数字、ハイフン (-)、アンダースコア (_) を含めることができます。

   > **Note**
   >
   > ルールには、同じリージョン内および同じイベントバス上の別のルールと同じ名前を付けることはできません。

6. (オプション) 説明に、ルールの説明を入力します。
7. イベントバス] では、このルールに関連付けるイベントバスを選択します。このルールをアカウントからのイベントと一致させるには、デフォルト を選択します。アカウントの AWS のサービス がイベントを発行すると、常にアカウントのデフォルトのイベントバスに移動します。
8. (オプション) すぐに実行しないバスのルールについては、そのルールをオフにします。
9. ルールタイプ では、イベントパターンを持つルール] を選択します。
10. 次へ をクリックします。
11. イベントソース で、AWS イベント または EventBridge パートナーイベント を選択します。
12. (オプション) サンプルイベント では:
    a. サンプルイベントタイプ では、AWS イベント を選択します。
    b. サンプルイベントでは、 バッチジョブの状態変更 を選択します。
13. 作成方法 では、パターンフォームの使用 を選択します。

14. イベントパターン では:

    a. イベントソース では、AWS のサービス を選択します。

    b. AWS のサービス では、バッチ を選択します。

    c. イベントタイプ では、バッチジョブの状態変更 を選択します。

15. 次へ を選択します。

16. ターゲットタイプ] には、AWS のサービス を選択します。

17. ターゲットタイプを選択 で、ターゲットタイプを選択します。たとえば、バッチジョブのキュー を選択します。以下を指定します。

    - ジョブキュー]: ジョブをスケジュールするジョブキューの Amazon リソースネーム (ARN) を入力します。
    - ジョブ定義]: ジョブに使用するジョブ定義の名前、改正、または完全な ARN を入力します。
    - ジョブ名]: ジョブの名前を入力します。
    - 配列サイズ]: (オプション) 複数のコピーを実行するためのジョブの配列サイズを入力します。詳細については、配列ジョブを参照してください。
    - ジョブの試行]: (オプション) ジョブが失敗したときに再試行する回数を入力します。詳細については、「ジョブの再試行の自動化」を参照してください。

18. バッチジョブキューのターゲットタイプの場合、 にはターゲットにイベントを送信するアクセス許可 EventBridge が必要です。 EventBridge は、ルールの実行に必要な IAM ロールを作成できます。次のいずれかを行います。

    - 自動的に IAM ロールを作成するには、この特定のリソースに対して新しいロールを作成する を選択します。
    - 以前に作成した IAM ロールを使用するには、既存のロールの使用 を選択します。

19. (オプション) 追加設定 を展開します。

    a. ターゲット入力の設定 で、イベントからのテキストの処理方法を選択します。

    b. イベントの最大保存期間 では、未処理のイベントを保存する時間間隔を指定します。

    c. 再試行 では、イベントを再試行する回数を入力します。

    d. デッドレターキュー では、未処理イベントの取扱いに関するオプションを選択します。必要に応じて、デッドレターキューに Amazon SQS キューを指定します。

20. (オプション) 別のターゲットを追加 を選択して、別のターゲットを追加します。

21. 次へ を選択します。

22. (オプション) タグ で 新しいタグを追加 を選択し、リソースラベルを追加します。詳細については、「Amazon ユーザーガイド」の「Amazon EventBridge タグ」を参照してください。 EventBridge
23. 次へ を選択します。
24. レビューと作成では、設定手順を確認してください。変更する必要がある場合は、編集] を選択します。完了したら、ルールの作成 を選択します。

    ルールの作成の詳細については、「Amazon ユーザーガイド」の「イベントに反応する Amazon EventBridge ルールの作成」を参照してください。 EventBridge

EventBridge 入力トランスフォーマーを使用してスケジュールに従ってイベント情報を AWS Batch ターゲットに渡す

EventBridge 入力トランスフォーマーを使用して、ジョブ送信 AWS Batch でイベント情報を に渡すことができます。これは、他の AWS イベント情報の結果としてジョブを呼び出す場合は、特に重要です。例えば、Amazon S3 バケットへのオブジェクトのアップロード時など。または、コンテナのコマンドで、パラメータの置換値を使用したジョブ定義を使用できます。 EventBridge Input Transformer は、イベントデータに基づいてパラメータ値を指定できます。

その後、開始する AWS Batch イベントからの情報を解析し、 `parameters` オブジェクトに変換するイベントターゲットを作成します。ジョブが実行されると、トリガーイベントからのパラメータがジョブコンテナのコマンドに渡されます。

> **Note**
>
> このシナリオでは、すべての AWS リソース (Amazon S3 バケット、 EventBridge ルール、CloudTrail口グなど) が同じリージョンにある必要があります。

入力トランスフォーマーを使用する AWS Batch ターゲットを作成するには

1. https://console.aws.amazon.com/events/ で Amazon EventBridge コンソールを開きます。
2. ナビゲーションバーから、 AWS リージョン 使用する を選択します。
3. ナビゲーションペインで Rules] (ルール) を選択します。
4. ルールの作成 を選択します。

5. 名前 で、コンピューティング環境の一意な名前を指定します。名前は最大 64 文字まで入力できます。大文字、小文字、数字、ハイフン (-)、アンダースコア (_) を含めることができます。

> **Note**
>
> ルールは、同じ AWS リージョン および同じイベントバス内の別のルールと同じ名前にすることはできません。

6. (オプション) 説明 に、ルールの説明を入力します。
7. イベントバス] では、このルールに関連付けるイベントバスを選択します。このルールをアカウントからのイベントと一致させるには、デフォルト を選択します。アカウントの AWS のサービス がイベントを発行すると、常にアカウントのデフォルトのイベントバスに移動します。
8. (オプション) すぐに実行しないバスのルールについては、そのルールをオフにします。
9. ルールタイプ では、スケジュール] を選択します。
10. 続行してルールを作成する または 次へ を選択します。
11. スケジュールパターン] では、次のいずれかを実行します。
    - 毎月第一月曜日の太平洋標準日午前 8:00 など、毎月最初の月曜日の PST で、cron 式を入力します。詳細については、「Amazon EventBridge ユーザーガイド」の「cron 式」を参照してください。
    - 一定の間隔 (10 分ごとなど) で実行するスケジュールを選択してから、rate 式を入力します。
12. 次へ を選択します。
13. ターゲットタイプ] には、AWS のサービス を選択します。
14. ターゲットの選択 で、バッチジョブのキュー を選択します。次を設定します。
    - ジョブキュー]: ジョブをスケジュールするジョブキューの Amazon リソースネーム (ARN) を入力します。
    - ジョブ定義]: ジョブに使用するジョブ定義の名前、改正、または完全な ARN を入力します。
    - ジョブ名]: ジョブの名前を入力します。
    - 配列サイズ]: (オプション) 複数のコピーを実行するためのジョブの配列サイズを入力します。詳細については、配列ジョブを参照してください。
    - ジョブの試行]: (オプション) ジョブが失敗したときに再試行する回数を入力します。詳細については、「ジョブの再試行の自動化」を参照してください。

15. バッチジョブキューのターゲットタイプの場合、 にはターゲットにイベントを送信するアクセス許可 EventBridge が必要です。 EventBridge は、ルールの実行に必要な IAM ロールを作成できます。次のいずれかを行います。

    - 自動的に IAM ロールを作成するには、この特定のリソースに対して新しいロールを作成する を選択します。
    - 以前に作成した IAM ロールを使用するには、既存のロールの使用 を選択します。

16. （オプション）追加設定 を展開します。
17. 追加設定 セクションの ターゲット入力の設定] で 入力トランスフォーマー] を選択します。
18. 入力トランスフォーマーの設定 を選択します。
19. (オプション) サンプルイベント では:

    a. サンプルイベントタイプ では、AWS イベント を選択します。
    b. サンプルイベントでは、 バッチジョブの状態変更 を選択します。

20. ターゲット入力トランスフォーマー の 入力パス で、トリガーするイベントから解析する値を指定します。例えば、バッチジョブの状態変更 イベントを解析するには、次の JSON 形式を使用します。

```
{
    "instance": "$.detail.jobId",
    "state": "$.detail.status"
}
```

21. テンプレート] では、以下を入力します。

```
{
    "instance": <jobId> ,
    "status": <status>
}
```

22. 確認] を選択します。
23. イベントの最大保存期間 では、未処理のイベントを保存する時間間隔を指定します。
24. 再試行 では、イベントを再試行する回数を入力します。
25. デッドレターキュー では、未処理イベントの取扱いに関するオプションを選択します。必要に応じて、デッドレターキューに Amazon SQS キューを指定します。
26. (オプション) 別のターゲットを追加 を選択して、別のターゲットを追加します。
27. 次へ を選択します。

28. (オプション) タグ で 新しいタグを追加 を選択し、リソースラベルを追加します。詳細については、「Amazon ユーザーガイド」の「Amazon EventBridge タグ」を参照してください。EventBridge
29. 次へ を選択します。
30. レビューと作成では、設定手順を確認してください。変更する必要がある場合は、編集] を選択します。完了したら、ルールの作成 を選択します。

チュートリアル：AWS Batch EventBridge を聴く

このチュートリアルでは、AWS Batch ジョブイベントを聴いて、CloudWatch Logs ログストリームに書き込むシンプルな AWS Lambda 関数を設定します。

前提条件

このチュートリアルでは、コンピューティング作業環境と、ジョブを受け付けることができるジョブキューがあることを前提としています。イベントをキャプチャするコンピューティング作業環境とジョブキューがなければ、の開始方法 AWS Batchのステップに従って作成します。このチュートリアルの最後に、オプションでこのジョブキューにジョブを送信してLambda 関数が正しく設定されていることをテストできます。

ステップ 1: Lambda 関数を作成する

この手順では、AWS Batch イベントストリームメッセージのターゲットとなるシンプルな Lambda 関数を作成します。

ターゲットの Lambda 関数を作成するには

1. AWS Lambda コンソールを (https://console.aws.amazon.com/lambda/) 開きます。
2. 関数の作成 を選択し、一から作成 を選択します。
3. 関数名] に、batch-event-stream-handlerと入力します。
4. ランタイム] では、Python 3.8] を選択します。
5. 関数の作成] を選択します。
6. 関数コード] セクションで、以下の例に合わせてサンプルコードを編集します。

```
import json
```

```
def lambda_handler(event, _context):
    # _context is not used
    del _context
    if event["source"] != "aws.batch":
        raise ValueError("Function only supports input from events with a source
 type of: aws.batch")

    print(json.dumps(event))
```

これは、AWS Batch から送信されたイベントを印刷するシンプルな Python 3.8 関数です。すべてが正しく設定されると、このチュートリアルの最後に、この Lambda 関数に関連付けられている CloudWatch Logs ログストリームにイベントの詳細が表示されます。

7. デプロイ] をクリックします。

ステップ 2: イベントルールを登録する

このセクションでは、AWS Batch リソースから来るジョブイベントをキャプチャする EventBridge イベントルールを作成します。このルールでは、それが定義されているアカウント内の AWS Batch から送信されるすべてのイベントがキャプチャされます。ジョブメッセージ自体に、イベントソースに関する情報 (イベントソースの送信先ジョブキューの情報など) が含まれています。この情報を使用して、プログラムでイベントをフィルタリングおよびソートできます。

> **Note**
>
> AWS Management Console を使用してイベントルールを作成すると、コンソールは自動的に EventBridge が Lambda 関数を呼び出すための IAM 権限を追加します。ただし、AWS CLI を使用してイベントルールを作成する場合は、この権限を明示的に付与する必要があります。詳細については、Amazon EventBridge ユーザーガイドのイベントとイベントパターンを参照してください。

EventBridge ルールを作成するには

1. Amazon EventBridge コンソール (https://console.aws.amazon.com/events/) を開きます。
2. ナビゲーションペインで ルール] を選択します。
3. ルールの作成 を選択します。
4. ルールの名前と説明を入力します。

ルールには、同じリージョン内および同じイベントバス上の別のルールと同じ名前を付けることはできません。

5. イベントバス] では、このルールに関連付けるイベントバスを選択します。このルールをアカウントからのイベントと一致させるには、AWS のデフォルトのイベントバスを選択します。アカウントの AWS サービスがイベントを発行すると、常にアカウントのデフォルトのイベントバスに移動します。
6. ルールタイプ では、イベントパターンを持つルール] を選択します。
7. 次へ を選択します。
8. イベントソース では、その他] を選択します。
9. イベントパターン では、カスタムパターン (JSON エディター) を選択します。
10. 次のイベントパターンをテキストエリアに貼り付けます。

```
{
  "source": [
    "aws.batch"
  ]
}
```

このルールは、すべての AWS Batch グループとすべての AWS Batch イベントに適用されます。または、より具体的なルールを作成し、一部の結果をフィルタで除外することもできます。

11. 次へ を選択します。
12. ターゲットタイプ] では、AWSサービス] を選択します。
13. ターゲットを選択 で、Lambda 関数 を選択します。
14. (オプション) 追加設定] では、以下を実行します。

    a. 最大イベント有効期間 に、1 分 (00:01) から 24 時間 (24:00) の間の値を入力します。

    b. 再試行 で、0 ~ 185 の数値を入力します。

    c. デッドレターキュー で、標準 Amazon SQS キューをデッドレターキューとして使用するかどうかを選択します。EventBridge は、このルールに一致するイベントがターゲットに正常に配信されなかった場合に、そのイベントをデッドレターキューに送信します。次のいずれかを実行します。

    - デッドレターキューを使用しない場合は、なし] を選択します。
    - デッドレターキューとして使用する現在の AWS アカウントの Amazon SQS キューを選択 を選択し、ドロップダウンから使用するキューを選択します。

- 他の AWS アカウントの Amazon SQS キューをデッドレターキューとして選択 を選択し、使用するキューの ARN を入力します。キューにメッセージを送信するための EventBridge 許可を付与するリソースベースのポリシーをそのキューにアタッチする必要があります。詳細については、Amazon EventBridge ユーザーガイドのデッドレターキューへの許可の付与を参照してください。

15. 次へ を選択します。
16. (オプション) ルールに 1 つ以上のタグを入力します。詳細については、Amazon EventBridge ユーザーガイドのAmazon EventBridge のタグを参照してください。
17. 次へ をクリックします。
18. ルールの詳細を確認し、ルールの作成 を選択します。

ステップ 3: 設定をテストする

ジョブキューにジョブを送信して EventBridge 設定をテストできます。すべてが適切に設定されている場合、Lambda 関数がトリガーされ、関数の CloudWatch Logs ログストリームにイベントデータが書き込まれます。

設定をテストするには

1. https://console.aws.amazon.com/batch/ で AWS Batch コンソールを開きます。
2. 新しい AWS Batch ジョブを送信します。詳細については、ジョブの送信を参照してください。
3. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
4. ナビゲーションペインで、ログ] を選択して Lambda 関数 (例えば、/aws/lambda/*my-function##*)] のロググループを選択します。
5. イベントデータを表示するログストリームを選択します。

チュートリアル:ジョブ停止イベントに Amazon シンプル 通知サービス アラートを送信

このチュートリアルでは、ジョブが `FAILED`ステータスに移行したジョブ EventBridge イベントのみをキャプチャするイベントルールを設定します。このチュートリアルの最後に、このジョブキューにジョブを送信することもできます。これは、Amazon SNS アラートが正しく設定されていることをテストするためです。

前提条件

このチュートリアルでは、コンピューティング作業環境と、ジョブを受け付けることができるジョブキューがあることを前提としています。イベントをキャプチャするコンピューティング作業環境とジョブキューがなければ、の開始方法 AWS Batchのステップに従って作成します。

ステップ 1: Amazon SNS トピックを作成してサブスクライブする

このチュートリアルでは、新しいイベントルールのイベントターゲットとして使用する Amazon SNS トピックを設定します。

Amazon SNS トピックを作成するには

1. Amazon SNS コンソール (https://console.aws.amazon.com/sns/v3/home) を開きます。
2. トピック、トピックの作成 の順に選択します。
3. Type] (タイプ) で、Standard] (標準) を選択します。
4. 名前として **`JobFailedAlert`** を入力し、トピックを作成 を選択します。
5. JobFailedAlert 画面で、サブスクリプションの作成を選択します。
6. [Protocol (プロトコル)] として [Email (E メール)] を選択します。
7. エンドポイント では、現在アクセスできるメールアドレスを入力し、サブスクリプションの作成 を選択します。
8. メールアカウントを確認し、サブスクリプションの確認メールメッセージが届くのを待ちます。確認メールが届いたら、Confirm subscription] (サブスクリプションの確認) を選択します。

ステップ 2: イベントルールを登録する

次に、失敗したジョブイベントのみをキャプチャするイベントルールを登録します。

EventBridge ルールを登録するには

1. https://console.aws.amazon.com/events/ で Amazon EventBridge コンソールを開きます。
2. ナビゲーションペインで Rules] (ルール) を選択します。
3. ルールの作成 を選択します。
4. ルールの名前と説明を入力します。

   ルールには、同じリージョン内および同じイベントバス上の別のルールと同じ名前を付けることはできません。

5. Event bus] (イベントバス) では、このルールに関連付けるイベントバスを選択します。このルールをアカウントからのイベントと一致させるには、AWS のデフォルトのイベントバスを選択します。アカウントの AWS サービスがイベントを発行すると、常にアカウントのデフォルトのイベントバスに移動します。
6. ルールタイプ では、イベントパターンを持つルール] を選択します。
7. 次へ をクリックします。
8. Event source] (イベントソース) では、Other] (その他) を選択します。
9. イベントパターン では、カスタムパターン (JSON エディター) を選択します。
10. 次のイベントパターンをテキストエリアに貼り付けます。

```
{
  "detail-type": [
    "Batch Job State Change"
  ],
  "source": [
    "aws.batch"
  ],
  "detail": {
    "status": [
      "FAILED"
    ]
  }
}
```

このコードは、ジョブのステータスが であるすべてのイベントに一致する EventBridge ルールを定義しますFAILED。イベントパターンの詳細については、「Amazon EventBridge ユーザーガイド」の「イベントとイベントパターン」を参照してください。

11. 次へ をクリックします。
12. ターゲットタイプ] では、AWS サービス] を選択します。
13. ターゲット の選択 で SNS トピック を選択し、トピック で を選択しますJobFailedAlert。
14. (オプション) 追加設定では、以下を実行します。

    a. Maximum age of event (最大イベント有効期間) に、1 分 (00:01) から 24 時間 (24:00) の間の値を入力します。

    b. 再試行 で、0 ~ 185 の数値を入力します。

    c. デッドレターキュー では、スタンダード Amazon SQS キューをデッドレターキューとして使用するかどうかを選択します。ターゲットに正常に配信されない場合、 はこのルールに

一致するイベントをデッドレターキュー EventBridge に送信します。次のいずれかを行います。

- デッドレターキューを使用しない場合は、None] (なし) を選択します。
- 現在の AWS アカウントでデッドレターキューとして使用する Amazon SQS キューを選択を選択し、ドロップダウンから使用するキューを選択します。
- デッドレターキューとして他の AWS アカウントの Amazon SQS キューを選択を選択し、使用するキューの ARN を入力します。キューにメッセージを送信する EventBridge アクセス許可を付与するリソースベースのポリシーをアタッチする必要があります。詳細については、「Amazon ユーザーガイド」の「デッドレターキューへのアクセス許可の付与」を参照してください。 EventBridge

15. 次へ をクリックします。
16. (オプション) ルールに 1 つ以上のタグを入力します。詳細については、「Amazon ユーザーガイド」の「Amazon EventBridge タグ」を参照してください。 EventBridge
17. 次へ をクリックします。
18. ルールの詳細を確認し、ルールの作成 を選択します。

ステップ 3: ルールをテストする

ルールをテストするには、ゼロ以外の終了コードで開始直後に終了するジョブを送信します。イベントルールが正しく設定されていれば、数分以内にイベントテキストが記載されたメールメッセージが届きます。

ルールをテストするには

1. https://console.aws.amazon.com/batch/ で AWS Batch コンソールを開きます。
2. 新しい AWS Batch ジョブを送信します。詳細については、「ジョブの送信」を参照してください。ジョブのコマンドでは、このコマンドを置き換え、終了コード 1 でコンテナを終了します。

```
/bin/sh, -c, 'exit 1'
```

3. 失敗したジョブの通知に関するアラートのメールが届いていることを確認します。

代替ルール: バッチジョブキューがブロックされました

バッチジョブキューがブロックされた をモニタリングするイベントルールを作成するには、このチュートリアルの手順を次の変更を加えて繰り返します。

1. ステップ 1 では、トピック名*BlockedJobQueue*として を使用します。
2. ステップ 2 では、JSON エディタで次のパターンを使用します。

```
{
   "detail-type": [
     "Batch Job Queue Blocked"
   ],
   "source": [
     "aws.batch"
   ]
}
```

での CloudWatch ログの使用 AWS Batch

EC2 リソースで AWS Batch ジョブを設定して、詳細なログ情報とメトリクスを CloudWatch ログに送信できます。このようにして、1つの便利な場所でジョブからのさまざまなログを表示できます。CloudWatch ログの詳細については、「Amazon ユーザーガイド」の「Amazon CloudWatch Logs とは」を参照してください。 CloudWatch

> **Note**
>
> デフォルトでは、 AWS Fargate コンテナの CloudWatch ログはオンになっています。

CloudWatch ログのログ記録を有効にしてカスタマイズするには、次の 1 回限りの設定タスクを確認します。

- EC2 リソースに基づく AWS Batch コンピューティング環境の場合は、`ecsInstanceRole`ロールに IAM ポリシーを追加します。詳細については、「the section called “ CloudWatch Logs IAM ポリシーを追加する”」を参照してください。
- 詳細 CloudWatch モニタリングを含む Amazon EC2 起動テンプレートを作成し、 AWS Batch コンピューティング環境を作成するときにテンプレートを指定します。エージェントを CloudWatch 既存のイメージにインストールし、 AWS Batch 初回実行ウィザードでイメージを指定することもできます。
- (オプション) awslogs ドライバーを設定します。EC2 リソースと Fargate リソースの両方で、デフォルトの動作を変更するパラメータを追加できます。詳細については、「the section called “awslogs ログドライバーを使用する”」を参照してください。

CloudWatch Logs IAM ポリシーを追加する

ジョブがログデータと詳細なメトリクスを CloudWatch Logs に送信できるようにするには、Logs APIs を使用する IAM CloudWatch ポリシーを作成する必要があります。IAMポリシーを作成したら、`ecsInstanceRole` ロールにアタッチします。

> **Note**
>
> ECS-CloudWatchLogs ポリシーがecsInstanceRoleロールにアタッチされていない場合でも、基本的なメトリクスを CloudWatch ログに送信できます。ただし、基本メトリックスには、ログデータや空きディスク容量などの詳細なメトリックスは含まれません。

AWS Batch コンピューティング環境は Amazon EC2 リソースを使用します。 AWS Batch 初回実行ウィザードを使用してコンピューティング環境を作成すると、 はecsInstanceRoleロール AWS Batch を作成し、それを使用して環境を設定します。

初回実行ウィザードを使用していない場合は、 AWS Command Line Interface または AWS Batch API でコンピューティング環境を作成するときにecsInstanceRoleロールを指定できます。詳細については、AWS CLI コマンドリファレンスまたはAWS Batch API リファレンスを参照してください。

ECS-CloudWatchLogs IAM ポリシーを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで、ポリシー を選択します。
3. ポリシーの作成を選択します。
4. JSON を選択し、以下のポリシーを入力します：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        }
    ]
```

```
}
```

5. Next: Tags] (次へ: タグ) を選択します。
6. (オプション) タグを追加 で タグを追加 を選択し、ポリシーにタグを追加します。
7. Next: Review] (次へ: レビュー) を選択します。
8. ポリシーの確認 ページで、名前 に**ECS-CloudWatchLogs**と入力し、次にオプションで説明 に入力します。
9. ポリシーの作成を選択します。

ECS-CloudWatchLogs ポリシーを **ecsInstanceRole** にアタッチするには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで Roles] (ロール) を選択します。
3. [ecsInstanceRole] を選択します。ロールが存在しない場合は、Amazon ECS インスタンスロールの手順に従ってロールを作成します。
4. アクセス許可を追加を選択し、次にポリシーをアタッチ を選択します。
5. ECS-CloudWatchLogs ポリシーを選択し、ポリシーのアタッチ を選択します。

CloudWatch エージェントのインストールと設定

CloudWatch モニタリングを含む Amazon EC2 起動テンプレートを作成できます。詳細については、「Amazon EC2 ユーザーガイド」の「起動テンプレートからインスタンスを起動する」および「詳細」を参照してください。 Amazon EC2

エージェントを既存の Amazon EC2 AMI CloudWatch にインストールし、 AWS Batch 初回実行ウィザードでイメージを指定することもできます。詳細については、CloudWatch 「 エージェントのインストール」および「 の開始方法 AWS Batch」を参照してください。

> **Note**
>
> 起動テンプレートは AWS Fargate リソースではサポートされていません。

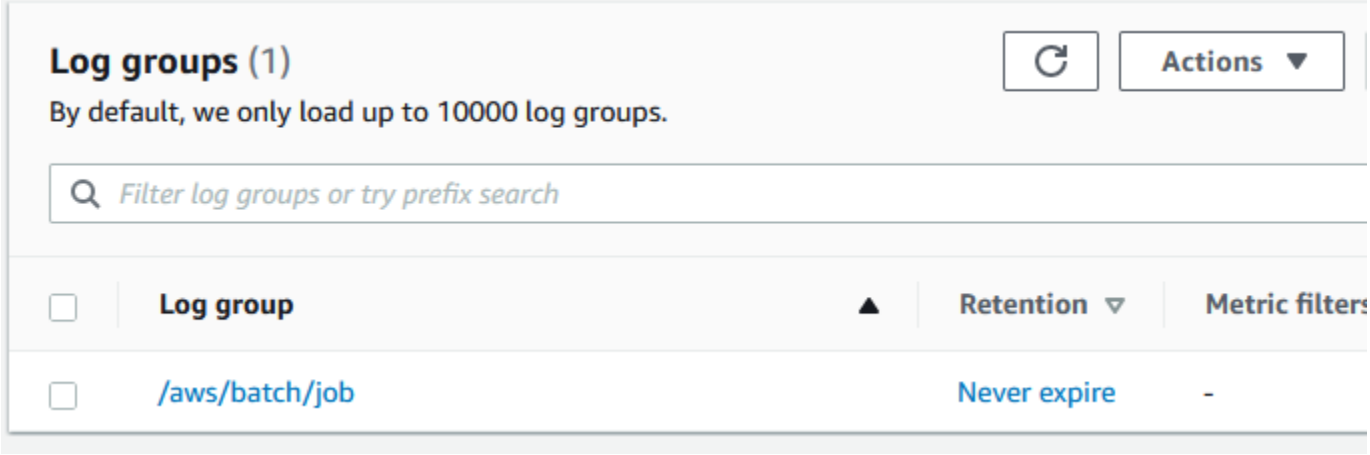
CloudWatch ログの表示

で CloudWatch ログログを表示および検索できます AWS Management Console。

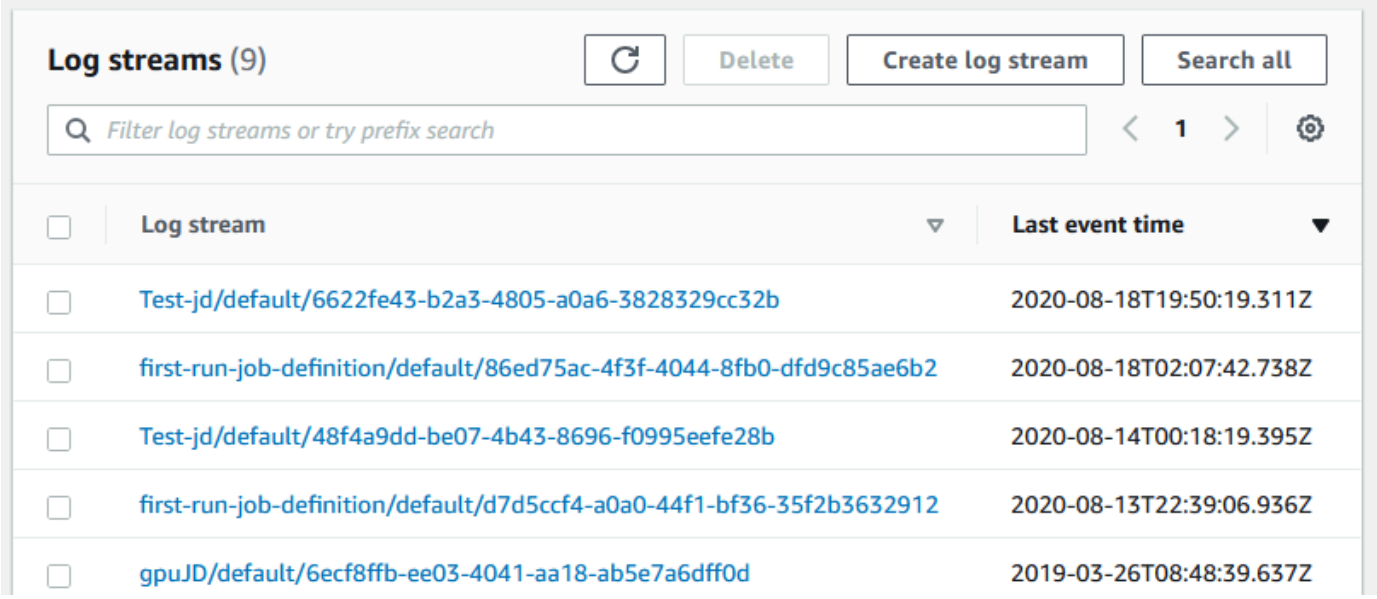
> **Note**
>
> ログにデータが表示されるまでに数分かかる場合があります CloudWatch 。

CloudWatch ログデータを表示するには

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. 左側のナビゲーションペインで、「ログ」を選択し、「ロググループ」を選択します。

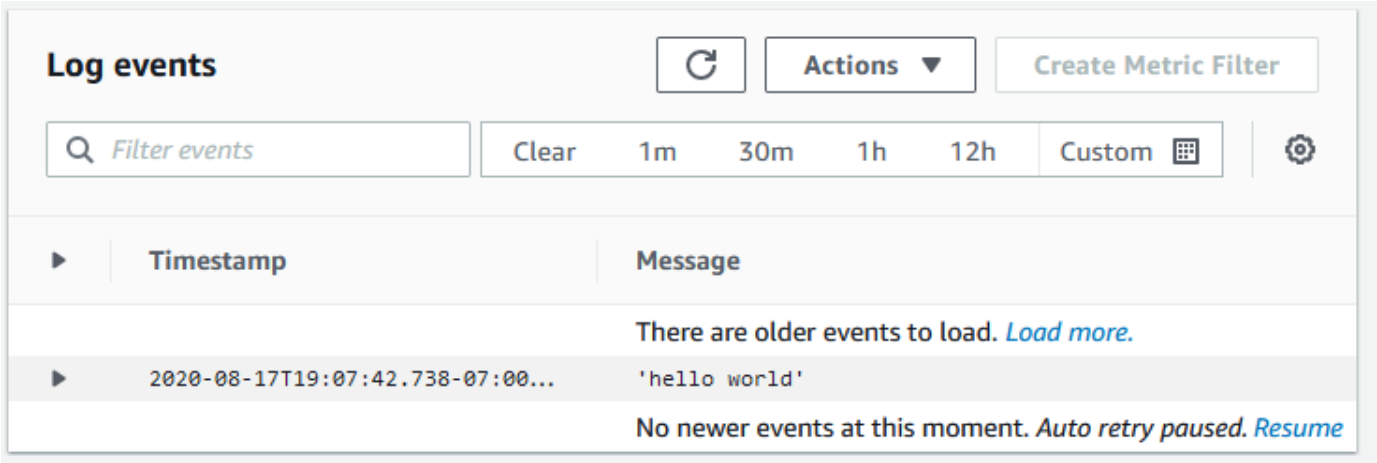


3. 表示するロググループを選択します。



4. 表示するログストリームを選択します。デフォルトでは、ストリームはジョブ名の最初の 200 文字と Amazon ECS タスク ID によって識別されます。

> **Tip**
>
> ログストリームデータをダウンロードするには、アクション を選択します。



CloudWatch Logs を使用して Amazon EKS ジョブにおける AWS Batch をモニタリングする

Amazon CloudWatch Logs を使用して 1 つの場所ですべてのログファイルをモニタリング、保存、表示できます。CloudWatch Logs を使用すると、複数のソースからのログデータを検索、フィルタリング、分析できます。

CloudWatch Logs を使用して Amazon EKS ジョブにおける AWS Batch をモニタリングするためのプラグインを含む AWS for Fluent Bit のイメージをダウンロードできます。Fluent Bit は Docker と Kubernetes 互換性の両方を兼ね備えたオープンソースのログプロセッサおよびフォワーダーです。Fluentd よりもリソースを消費しないため、Fluent Bit をログルーターとして使用することをお勧めします。詳細については、AWS for Fluent Bit イメージの使用を参照してください。

前提条件

ワーカーノードの AWS Identity and Access Management ポリシーに `CloudWatchAgentServerPolicy` ポリシーをアタッチします。詳細については、前提条件を確認を参照してください。

AWS for Fluent Bit をインストールします。

AWS for Fluent Bit をインストールして CloudWatch グループを作成する方法については、Fluent Bit の設定またはCloudWatch エージェントおよび Fluent Bit のクイックスタートを参照してください。

> **Tip**
>
> AWS Batch ノードで、0.5 CPU と 100 MB のメモリが Fluent Bit を使用することに注意してください。これにより、AWS Batch ジョブに使用できるキャパシティの合計が減少します。ジョブの規模を決定する際には、この点を考慮してください。

AWS Batch ノードの Fluent Bit を有効します。

AWS Batch マネージドノードで、Fluent Bit の口グ記録 DaemonSet を実行するには、Fluent Bit の DaemonSet の許容値を変更します。

```
tolerations:
- key: "batch.amazonaws.com/batch-node"
  operator: "Exists"
```

AWS Batch CloudWatchコンテナインサイト

CloudWatchコンテナインサイトは、AWS Batch コンピュート環境とジョブからメトリクスとログを収集、集約、要約します。このメトリクスには、CPU、メモリ、ディスク、ネットワークなどのリソース使用率が含まれます。これらのメトリクスは、 CloudWatch ダッシュボードに追加できます。

運用データは、パフォーマンスログイベントとして収集されます。これらは、高濃度データを取り込み、大規模に保存することが可能な構造化された JSON スキーマを使用するエントリです。CloudWatch はこのデータから、クラスターおよびサービスレベルで、高レベルの集約されたメトリクスを CloudWatch メトリクスとして作成します。詳細については、Amazon CloudWatch ユーザーガイドのAmazon ECS のコンテナインサイト構造化ログを参照してください。

> **Important**
>
> CloudWatchコンテナインサイトによって収集されたメトリクスは、カスタムメトリクスとして課金されます。詳細については、Amazon CloudWatch Events の料金を参照してください。

コンテナインサイトをオンにします。

AWS Batch コンピューティング環境では、コンテナインサイトを有効にできます。

1. AWS Batch コンソールを開きます。
2. コンピュート環境 を選択します。
3. 開く開発環境を選択します。
4. コンテナインサイト については、コンピュート環境のコンテナインサイトをオンにする。

> **Tip**
>
> デフォルトの間隔を選択してメトリクスを集計したり、カスタムの間隔を作成したりできます。。

以下のメトリクスが表示されます。Amazon ECS コンテナインサイトメトリクスの一覧については、Amazon CloudWatch ユーザーガイドのAmazon ECS コンテナインサイトメトリクスの表示を参照してください。

- **JobCount** — コンピューティング環境で実行されるジョブの数。
- **ContainerInstanceCount** — Amazon ECS エージェントを実行し、コンピューティング環境に登録されている Amazon Elastic Compute Cloud インスタンスの数。
- **MemoryReserved** — コンピュート環境のジョブによって予約されるメモリ。このメトリクスは、ジョブ定義にメモリ予約が定義されているタスクに対してのみ収集されます。
- **MemoryUtilized** — コンピュート環境のジョブによって予約されるメモリ。このメトリクスは、ジョブ定義にメモリ予約が定義されているタスクに対してのみ収集されます。
- **CpuReserved** — コンピュート環境ジョブによって予約されている CPU ユニット。このメトリクスは、ジョブ定義にCPU予約が定義されているタスクに対してのみ収集されます。
- **CpuUtilized** — コンピュート環境のジョブが使用する CPU ユニット。このメトリクスは、ジョブ定義にCPU予約が定義されているタスクに対してのみ収集されます。
- **NetworkRxBytes** -受信したバイト数。このメトリクスは、awsvpcまたは ネットワークモードを使用するタスクのコンテナでのみ使用できます。
- **NetworkTxBytes** — 送信されたバイト数。このメトリクスは、awsvpcまたは ネットワークモードを使用するタスクのコンテナでのみ使用できます。
- **StorageReadBytes** — ストレージから読み取られるバイト数。
- **StorageWriteBytes** ストレージに書き込まれるバイト数。

AWS CloudTrail を使用した AWS Batch API コールのログ記録

AWS Batch は AWS CloudTrail という、AWS Batch の ユーザーやロール、または AWS のサービスによって実行されたアクションを記録するサービスと統合しています。CloudTrail は、AWS Batch のすべての API コールをイベントとしてキャプチャします。キャプチャされた呼び出しには、AWS Batch コンソールの呼び出しと、AWS Batch API オペレーションへのコード呼び出しが含まれます。証跡を作成する場合は、AWS Batch のイベントなど、Amazon S3 バケットへの CloudTrail イベントの継続的な配信を有効にすることができます 証跡を設定しない場合でも、CloudTrail コンソールの [Event history (イベント履歴)] で最新のイベントを表示できます。CloudTrail によって収集された情報を使用して、AWS Batch に対して行われた要求、要求が行われた IP アドレス、要求を行った人、要求が行われた日時、および追加の詳細を判別できます。

CloudTrail の詳細については、「AWS CloudTrailユーザーガイド」を参照してください。

AWS BatchCloudTrail での 情報

CloudTrail は、アカウントを作成すると AWS アカウントで有効になります。AWS Batch でアクティビティが発生すると、そのアクティビティは [Event history (イベント履歴)] で AWS のその他のサービスのイベントと共に CloudTrail イベントに記録されます。最近のイベントは、AWS アカウントで表示、検索、ダウンロードできます。詳細については、「CloudTrail イベント履歴でのイベントの表示」を参照してください。

AWS のイベントなど、AWS Batch アカウントのイベントの継続的なレコードについては、追跡を作成します。証跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コンソールで追跡を作成するときに、追跡がすべての AWS リージョンに適用されます。追跡は、AWSパーティションのすべてのリージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、CloudTrail ログで収集したイベントデータをより詳細に分析し、それに基づく対応するためにその他の AWS のサービスを設定できます。詳細については、次を参照してください。

- 追跡を作成するための概要
- CloudTrail のサポート対象サービスと統合
- Amazon SNS の CloudTrail の通知の設定
- 「複数のリージョンから CloudTrail ログファイルを受け取る」および「複数のアカウントから CloudTrail ログファイルを受け取る」

すべて AWS Batch アクションは CloudTrail でログに記録されます。これらのアクションは https://docs.aws.amazon.com/batch/latest/APIReference/ に記載されています。例えば、SubmitJob、ListJobs、および DescribeJobs セクションを呼び出すと、 CloudTrail ログファイルにエントリが生成されます。

各イベントまたはログエントリには、リクエストの生成者に関する情報が含まれます。同一性情報は次の判断に役立ちます。

- リクエストが、ルートと IAM ユーザー認証情報のどちらを使用して送信されたか。
- リクエストがロールまたはフェデレーションユーザーの一時的なセキュリティ認証情報を使用して行われたかどうか。
- リクエストが、別の AWS のサービスによって送信されたかどうか。

詳細については、「CloudTrail userIdentity 要素」を参照してください。

AWS Batch ログファイルエントリの概要

追跡は、指定した Amazon S3 バケットにイベントをログファイルとして配信するように設定できます。CloudTrail のログファイルには、単一か複数のログエントリがあります。イベントはあらゆるソースからの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストのパラメータなどの情報が含まれます。CloudTrail ログファイルは、パブリック API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

次の例は、CreateComputeEnvironmentアクションを示す CloudTrail ログエントリです。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE:admin",
    "arn": "arn:aws:sts::012345678910:assumed-role/Admin/admin",
    "accountId": "012345678910",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2017-12-20T00:48:46Z"
      },
      "sessionIssuer": {
        "type": "Role",
```

```
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::012345678910:role/Admin",
        "accountId": "012345678910",
        "userName": "Admin"
      }
    }
  },
  "eventTime": "2017-12-20T00:48:46Z",
  "eventSource": "batch.amazonaws.com",
  "eventName": "CreateComputeEnvironment",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "203.0.113.1",
  "userAgent": "aws-cli/1.11.167 Python/2.7.10 Darwin/16.7.0 botocore/1.7.25",
  "requestParameters": {
    "computeResources": {
      "subnets": [
        "subnet-5eda8e04"
      ],
      "tags": {
        "testBatchTags": "CLI testing CE"
      },
      "desiredvCpus": 0,
      "minvCpus": 0,
      "instanceTypes": [
        "optimal"
      ],
      "securityGroupIds": [
        "sg-aba9e8db"
      ],
      "instanceRole": "ecsInstanceRole",
      "maxvCpus": 128,
      "type": "EC2"
    },
    "state": "ENABLED",
    "type": "MANAGED",
    "computeEnvironmentName": "Test"
  },
  "responseElements": {
    "computeEnvironmentName": "Test",
    "computeEnvironmentArn": "arn:aws:batch:us-east-1:012345678910:compute-environment/
Test"
  },
  "requestID": "890b8639-e51f-11e7-b038-EXAMPLE",
  "eventID": "874f89fa-70fc-4798-bc00-EXAMPLE",
```

```
  "readOnly": false,
  "eventType": "AwsApiCall",
  "recipientAccountId": "012345678910"
}
```

仮想プライベートクラウド (VPC) の作成

コンピューティング環境のコンピューティングリソースは、AWS Batch および Amazon ECS サービスエンドポイントとの通信に、外部ネットワークアクセスを必要とします。ただし、ユーザーが、プライベートサブネットで実行したいジョブがある場合があります。パブリックサブネットまたはプライベートサブネットのどちらかでジョブを実行する柔軟性を得るには、パブリックサブネットとプライベートサブネットの両方がある VPC を作成します。

Amazon Virtual Private Cloud (Amazon VPC) を使用すると、ユーザーが定義した仮想ネットワーク内で AWSのリソースを起動することができます。このトピックでは、Amazon VPC ウィザードへのリンクと選択できるオプションのリストを提供します。

VPC を作成する

Amazon VPC の作成方法の詳細については、Amazon VPC ユーザーガイドのVPC のみを作成するを参照し、次の表を使用して選択するオプションを決定します。

オプション	値
作成するためのリソース	VPC 専用
名前	オプションで、VPC の名前を指定します。
IPv4 CIDR ブロック	IPv4 CIDR 手動入力 CIDR ブロックサイズは /16 から /28 の間である必要があります。
IPv6 CIDR ブロック	IPv6 CIDR ブロックなし
テナンシー	デフォルト

Amazon VPC の詳細については、Amazon VPC ユーザーガイドのAmazon VPC とはを参照してください。

次のステップ

VPC を作成したら、以下の次のステップを検討します：

- パブリックリソースおよびプライベートリソースでインバウンドネットワークアクセスが必要な場合は、そのセキュリティグループを作成します。詳細については、Amazon VPC ユーザーガイドのセキュリティグループの操作を参照してください。
- コンピューティング環境を新しい VPC に起動する AWS Batch マネージド型コンピューティングリソースを作成します。詳細については、コンピューティング環境の作成を参照してください。AWS Batch コンソールでコンピューティング環境作成ウィザードを使用する場合、先ほど作成した VPC とユースケースに応じて、インスタンスを起動するパブリックサブネットまたはプライベートサブネットを指定できます。
- 新しいコンピューティング環境にマッピングされる AWS Batch ジョブキューを作成します。詳細については、ジョブキューの作成を参照してください。
- ジョブの実行で使用するジョブ定義を作成します。詳細については、シングルノードのジョブ定義を作成する を参照してください。
- ジョブ定義とともに、新しいジョブキューにジョブを送信します。このジョブは、新しい VPC およびサブネットで作成したコンピューティング環境に置かれます。詳細については、ジョブの送信を参照してください。

のセキュリティ AWS Batch

のクラウドセキュリティが最優先事項 AWS です。お客様は AWS 、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャからメリットを得られます。

セキュリティは、 AWS とユーザーの間で共有される責任です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティと説明しています。

- クラウドのセキュリティ — クラウドで AWS サービスを実行するインフラストラクチャを保護する責任 AWS は AWS にあります。 AWS また、 では、安全に使用できるサービスも提供しています。コンプライアンスAWS プログラムコンプライアンスプログラム の一環として、サードパーティーの監査者は定期的にセキュリティの有効性をテストおよび検証。に適用されるコンプライアンスプログラムの詳細については AWS Batch、「コンプライアンスプログラムAWS による対象範囲内のサービスコンプライアンスプログラム」を参照してください。
- クラウド内のセキュリティ – お客様の責任は使用する AWS のサービスによって決まります。また、お客様は、お客様のデータの機密性、企業の要件、および適用可能な法律および規制などの他の要因についても責任を担います。

このドキュメントは、 の使用時に責任共有モデルを適用する方法を理解するのに役立ちます AWS Batch。以下のトピックでは、セキュリティおよびコンプライアンスの目的 AWS Batch を達成するために を設定する方法を示します。また、 AWS Batch リソースのモニタリングや保護に役立つ他の AWS のサービスの使用方法についても説明します。

トピック

- の Identity and Access Management AWS Batch
- インターフェイスエンドポイント AWS Batch を使用したアクセス
- のコンプライアンス検証 AWS Batch
- のインフラストラクチャセキュリティ AWS Batch

の Identity and Access Management AWS Batch

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を認証 (サインイン) し、誰に AWS

Batch リソースの使用を承認する (アクセス許可を付与する) かを制御します。IAM は、追加料金なしで AWS のサービス 使用できる です。

トピック

対象者

AWS Identity and Access Management (IAM) の使用方法は、 で行う作業によって異なります AWS Batch。

サービスユーザー – AWS Batch サービスを使用してジョブを実行する場合、管理者から必要な認証情報とアクセス許可が与えられます。さらに多くの AWS Batch 機能を使用して作業を行う場合は、追加のアクセス許可が必要になることがあります。アクセスの管理方法を理解すると、管理者から適切な権限をリクエストするのに役に立ちます。 AWS Batch機能にアクセスできない場合は、「AWS Batch ID とアクセスのトラブルシューティング」を参照してください。

サービス管理者 – 社内の AWS Batch リソースを担当している場合は、通常、 へのフルアクセスがあります AWS Batch。サービスユーザーがどの AWS Batch 機能やリソースにアクセスするかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。会社で IAM を で使用する方法の詳細については、 AWS Batch「」を参照してくださいが IAM と AWS Batch 連携する方法。

IAM 管理者 - 管理者は、 AWS Batchへのアクセスを管理するポリシーの書き込み方法の詳細について確認する場合があります。IAM で使用できる AWS Batch アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいのアイデンティティベースのポリシーの例 AWS Batch。

アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。として、IAM ユーザーとして AWS アカウントのルートユーザー、または IAM ロールを引き受けて認証（ にサインイン AWS) される必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。 AWS IAM Identity Center（ IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook の認証情報は、フェデレーション ID の例です。フェデレーティッドアイデンティティとしてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーション AWS を使用して にアクセスすると、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、「 ユーザーガイド」の「 にサインインする方法 AWS アカウントAWS サインイン 」を参照してください。

AWS プログラムで にアクセスする場合、 は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、認証情報を使用してリクエストに暗号で署名します。 AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。推奨される方法を使用してリクエストを自分で署名する方法の詳細については、IAM ユーザーガイドの API AWS リクエストの署名を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。例えば、 AWS では、多要素認証 (MFA) を使用してアカウントのセキュリティを向上させることをお勧めします。詳細については、『AWS IAM Identity Center ユーザーガイド』の「Multi-factor authentication」(多要素認証) および『IAM ユーザーガイド』の「AWSにおける多要素認証 (MFA) の使用」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、アカウント内のすべての AWS のサービス およびリソースへの完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることでアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、『IAM ユーザーガイド』の「ルートユーザー認証情報が必要なタスク」を参照してください。

フェデレーティッドアイデンティティ

ベストプラクティスとして、管理者アクセスを必要とするユーザーを含む人間のユーザーに、一時的な認証情報を使用して にアクセスするための ID プロバイダーとのフェデレーションの使用を要求 AWS のサービス します。

フェデレーティッド ID は、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダー、AWS Directory Service、アイデンティティセンターディレクトリのユーザー、または ID ソースを通じて提供された認証情報 AWS のサービス を使用して にアクセスするユーザーです。フェデレーティッド ID が にアクセスすると AWS アカウント、ロールを引き受け、ロールは一時的な認証情報を提供します。

アクセスを一元管理する場合は、 AWS IAM Identity Centerを使用することをお勧めします。IAM Identity Center でユーザーとグループを作成することも、独自の ID ソース内のユーザーとグループのセットに接続して同期して、すべての AWS アカウント とアプリケーションで使用することもできます。IAM Identity Center の詳細については、『AWS IAM Identity Center ユーザーガイド』の「What is IAM Identity Center?」(IAM Identity Center とは) を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、単一のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、IAM ユーザーガイドの「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する権限を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳細については、『IAM ユーザーガイド』の「IAM ユーザー (ロールではなく) の作成が適している場合」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。ロール を切り替える AWS Management Console ことで、 で IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、 または AWS API AWS CLI オペレーションを呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「IAM ロールの使用」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッドアイデンティティに権限を割り当てるには、ロールを作成してそのロールの権限を定義します。フェデレーティッドアイデンティティが認証されると、そのアイデンティティはロールに関連付けられ、ロールで定義されている権限が付与されます。フェデレーションの詳細については、『IAM ユーザーガイド』の「サードパーティーアイデンティティプロバイダー向けロールの作成」 を参照してください。IAM アイデンティティセンターを使用する場合、権限セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。権限セットの詳細については、『AWS IAM Identity Center ユーザーガイド』の「権限セット」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、 (ロールをプロキシとして使用する代わりに) ポリシーをリソースに直接アタッチできます。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、『IAM ユーザーガイド』の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。
- クロスサービスアクセス — 一部の は、他の の機能 AWS のサービス を使用します AWS のサービス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの権限、サービスロール、またはサービスにリンクされたロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用して でアクションを実行する場合 AWS、ユーザーはプリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがありま

す。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストのリクエストと組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

- サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。
- サービスにリンクされたロール – サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールの権限を表示できますが、編集することはできません。

- Amazon EC2 で実行されているアプリケーション – IAM ロールを使用して、EC2 インスタンスで実行され、 AWS CLI または AWS API リクエストを行うアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。 AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、『IAM ユーザーガイド』の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して権限を付与する」を参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、『IAM ユーザーガイド』の「(IAM ユーザーではなく) IAM ロールをいつ作成したら良いのか?」を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、 AWS ID またはリソースにアタッチします。ポリシーは AWS 、アイデンティティまたはリソースに関連付けられているときにアクセス許可を定義する のオブジェクトです。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うときに、これらのポリシー AWS を評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの権限を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。そのポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS API からロール情報を取得できます。

アイデンティティベースのポリシー

アイデンティティベースポリシーは、IAM ユーザー、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 権限ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

アイデンティティベースポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、 内の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。マネージドポリシーまたはインラインポリシーのいずれかを選択する方法については、『IAM ユーザーガイド』の「マネージドポリシーとインラインポリシーの比較」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM の AWS マネージドポリシーを使用できません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための権限を持つかをコントロールします。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、 AWS WAF、および Amazon VPC は、ACLs。ACL の詳細については、『Amazon Simple Storage Service デベロッパーガイド』の「アクセスコントロールリスト (ACL) の概要」を参照してください。

その他のポリシータイプ

AWS は、一般的ではない追加のポリシータイプをサポートします。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースのポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。許可の境界の詳細については、「IAM ユーザーガイド」の「IAM エンティティの許可の境界」を参照してください。
- サービスコントロールポリシー (SCPs) – SCPs は、 の組織または組織単位 (OU) に対する最大アクセス許可を指定する JSON ポリシーです AWS Organizations。 AWS Organizations は、 AWS アカウント ビジネスが所有する複数の をグループ化して一元管理するサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP は、各 を含むメンバーアカウントのエンティティのアクセス許可を制限します AWS アカウントのルートユーザー。Organizations と SCP の詳細については、『AWS Organizations ユーザーガイド』の「SCP の仕組み」を参照してください。
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。複数のポリシータイプが関与する場合にリクエストを許可するかどうか AWS を決定する方法については、IAM ユーザーガイドの「ポリシー評価ロジック」を参照してください。

が IAM と AWS Batch 連携する方法

IAM を使用して へのアクセスを管理する前に AWS Batch、 で使用できる IAM 機能について学びます AWS Batch。

で使用できる IAM の機能 AWS Batch

IAM 機能	AWS Batch サポート
アイデンティティベースのポリシー	Yes
リソースベースのポリシー	No
ポリシーアクション	Yes
ポリシーリソース	Yes
ポリシー条件キー	Yes
ACL	No
ABAC (ポリシー内のタグ)	はい
一時的な認証情報	Yes
プリンシパル権限	Yes
サービスロール	あり
サービスリンクロール	はい

AWS Batch およびその他の AWS のサービスがほとんどの IAM 機能と連携する方法の概要を把握するには、「IAM ユーザーガイド」のAWS 「IAM と連携する のサービス」を参照してください。

のアイデンティティベースのポリシー AWS Batch

アイデンティティベースポリシーをサポートする	Yes

アイデンティティベースポリシーは、IAM ユーザー、ユーザーグループ、ロールなど、アイデンティティにアタッチできる JSON 権限ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、『IAM ユーザーガイド』の「IAM ポリシーの作成」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。プリンシパルは、それが添付されているユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシーで使用できるすべての要素について学ぶには、「IAM ユーザーガイド」の「IAM JSON ポリシーの要素のリファレンス」を参照してください。

AWS Batchのアイデンティティベースのポリシーの例

AWS Batch アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいのアイデンティティベースのポリシーの例 AWS Batch。

のポリシーアクション AWS Batch

ポリシーアクションに対するサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

JSON ポリシーのAction要素には、ポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションの名前は通常、関連付けられた AWS API オペレーションと同じです。一致する API オペレーションのない権限のみのアクションなど、いくつかの例外があります。また、ポリシーに複数アクションが必要なオペレーションもあります。これらの追加アクションは、 依存アクション と呼ばれます。

このアクションは、関連付けられたオペレーションを実行するための権限を付与するポリシーで使用されます。

AWS Batch アクションのリストを確認するには、「サービス認証リファレンス」の「 で定義されるアクション AWS Batch」を参照してください。

のポリシーアクションは、アクションの前に次のプレフィックス AWS Batch を使用します。

```
batch
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [
      "batch:action1",
      "batch:action2"
         ]
```

ワイルドカード (*) を使用して複数アクションを指定できます。例えば、Describe という単語で始まるすべてのアクションを指定するには、次のアクションを含めます。

```
"Action": "batch:Describe*"
```

AWS Batch アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいのアイデンティティベースのポリシーの例 AWS Batch。

のポリシーリソース AWS Batch

ポリシーリソースに対するサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースにどのような条件でアクションを実行できるかということです。

Resource JSON ポリシー要素は、アクションが適用されるオブジェクトを指定します。ステートメントには、Resource または NotResource要素を含める必要があります。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの権限と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

AWS Batch リソースタイプとその ARNs」の「 で定義されるリソース AWS Batch」を参照してください。 どのアクションで各リソースの ARN を指定できるかについては、「AWS Batchで定義されるアクション」を参照してください。

AWS Batch アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいのアイデンティティベースのポリシーの例 AWS Batch。

AWS Batch向けのポリシー条件キー

サービス固有のポリシー条件キーのサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

Condition 要素 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。イコールや未満などの 条件演算子 を使用して条件式を作成することで、ポリシーの条件とリクエスト内の値を一致させることができます。

1 つのステートメントに複数の Condition 要素を指定するか、1 つの Condition 要素に複数のキーを指定すると、 AWS は AND 論理演算子を使用してそれらを評価します。1 つの条件キーに複数の値を指定すると、 は論理ORオペレーションを使用して条件 AWS を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。詳細については、『IAM ユーザーガイド』の「IAM ポリシーの要素: 変数およびタグ」を参照してください。

AWS は、グローバル条件キーとサービス固有の条件キーをサポートします。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイド」のAWS 「 グローバル条件コンテキストキー」を参照してください。

AWS Batch 条件キーのリストを確認するには、「サービス認証リファレンス」の「 の条件キー AWS Batch」を参照してください。条件キーを使用できるアクションとリソースについては、「 で定義されるアクション AWS Batch」を参照してください。

AWS Batch アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいのアイデンティティベースのポリシーの例 AWS Batch。

での属性ベースのアクセスコントロール (ABAC) AWS Batch

ABAC のサポート (ポリシー内のタグ)	はい

属性ベースのアクセス制御 (ABAC) は、属性に基づいてアクセス許可を定義するアクセス許可戦略です。では AWS、これらの属性はタグ と呼ばれます。タグは、IAM エンティティ (ユーザーまたはロール) および多くの AWS リソースにアタッチできます。エンティティとリソースのタグ付けは、ABAC の最初の手順です。その後、プリンシパルのタグがアクセスしようとしているリソースのタグと一致した場合に操作を許可するように ABAC ポリシーを設計します。

ABAC は、急成長する環境やポリシー管理が煩雑になる状況で役立ちます。

タグに基づいてアクセスを管理するには、`aws:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値ははいです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

ABAC の詳細については、『IAM ユーザーガイド』の「ABAC とは?」を参照してください。ABAC をセットアップするステップを説明するチュートリアルについては、「IAM ユーザーガイド」の「属性に基づくアクセスコントロール (ABAC) を使用する」を参照してください。

での一時的な認証情報の使用 AWS Batch

一時的な認証情報のサポート	はい

一部の は、一時的な認証情報を使用してサインインすると機能 AWS のサービス しません。一時的な認証情報 AWS のサービス を使用する などの詳細については、IAM ユーザーガイドのAWS のサービス「IAM と連携する 」を参照してください。

ユーザー名とパスワード以外の AWS Management Console 方法で にサインインする場合、一時的な認証情報を使用します。例えば、会社の Single Sign-On (SSO) リンク AWS を使用して にアクセスすると、そのプロセスによって一時的な認証情報が自動的に作成されます。また、ユーザーとして

コンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えに関する詳細については、「IAM ユーザーガイド」の「ロールへの切り替え (コンソール)」を参照してください。

一時的な認証情報は、 AWS CLI または AWS API を使用して手動で作成できます。その後、これらの一時的な認証情報を使用して . AWS recommends にアクセスできます AWS。これは、長期的なアクセスキーを使用する代わりに、一時的な認証情報を動的に生成することを推奨しています。詳細については、「IAM の一時的セキュリティ認証情報」を参照してください。

AWS Batchのクロスサービスプリンシパル権限

フォワードアクセスセッション (FAS) をサポート	はい

IAM ユーザーまたはロールを使用して でアクションを実行する場合 AWS、ユーザーはプリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストのリクエストと組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

のサービスロール AWS Batch

サービスロールに対するサポート	あり

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。

> **Warning**
>
> サービスロールのアクセス許可を変更すると、 AWS Batch 機能が破損する可能性があります。 AWS Batch が指示する場合以外は、サービスロールを編集しないでください。

のサービスにリンクされたロール AWS Batch

サービスリンクロールのサポート	はい

サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールの権限を表示できますが、編集することはできません。

サービスにリンクされたロールの作成または管理の詳細については、「IAM と提携するAWS のサービス」を参照してください。表の中から、[Service-linked role] (サービスにリンクされたロール) 列に Yes と記載されたサービスを見つけます。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[Yes] リンクを選択します。

AWS Batch 実行 IAM ロール

実行ロールは、Amazon ECS コンテナと AWS Fargate エージェントに、ユーザーに代わって AWS API コールを実行するアクセス許可を付与します。

> **Note**
>
> 実行ロールは Amazon ECS コンテナエージェントバージョン 1.16.0 以降でサポートされています。

実行 IAM ロールは、タスクの要件に応じて必要です。さまざまな目的とサービスの実行ロールを、アカウントに複数関連付けることができます。

> **Note**
>
> 詳細については、 ユーザーガイドのAmazon ECS インスタンスロールを参照してくださいAmazon ECS インスタンスロール。サービスロールの詳細については、が IAM と AWS Batch 連携する方法を参照してください。

Amazon ECS は、`AmazonECSTaskExecutionRolePolicy`マネージドポリシーを提供します。は、という管理ポリシーを提供します。このポリシーには、上記の一般的ユースケースで必要なアク

セス許可が含まれています。以下に示す特殊なユースケースでは、タスク実行ロールにインラインポリシーを追加することが必要な場合があります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*"
    }
  ]
}
```

次の手順を使用してアカウントに既に実行ロールが存在するか確認し、必要に応じてマネージド IAM ポリシーをアタッチすることができます。

IAM コンソールで **ecsTaskExecutionRole** を確認するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで Roles] (ロール) を選択します。
3. ロールのリストで `ecsTaskExecutionRole` を検索します。ロールが検出できない場合は、実行 IAM ロールの作成 を参照してください。ロールが存在する場合は、そのロールを選択して、アタッチされているポリシーを表示します。
4. アクセス許可タブで、AmazonECSTaskExecutionRolePolicy 管理ポリシーがロールにアタッチされていることを確認します。ポリシーがアタッチされている場合、 実行ロールは適切に設定されています。そうでない場合、次のサブステップに従ってポリシーをアタッチします。

   a. アクセス許可を追加、ポリシーをアタッチ の順に選択します。

   b. AmazonECSTaskExecutionRolePolicy を検索します。

   c. AmazonECSTaskExecutionRolePolicy ポリシーの左側にあるチェックボックスをオンにし、ポリシーのアタッチ を選択します。

5. Trust relationships (信頼関係) を選択します。
6. 信頼関係に以下のポリシーが含まれていることを確認します。信頼関係が以下のポリシーと一致する場合、ロールは正しく設定されます。信頼関係の編集 を選択して、次のポリシーを追加し、信頼ポリシーの更新 を選択します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs-tasks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

実行 IAM ロールの作成

アカウントにまだ実行ロールがない場合は、次のステップに従ってロールを作成します。

`ecsTaskExecutionRole` IAM ロールを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで Roles (ロール) を選択します。
3. ロールの作成 を選択します。
4. 信頼できるエンティティタイプ で、を選択します。
5. [ユースケースの選択] で、 [EC2] を選択します。次に、もう一度 EC2 を選択します。
6. Next] (次へ) をクリックします。
7. アクセス許可ポリシー については、AmazonECSTaskExecutionRolePolicy を検索します。
8. AmazonECSTaskExecutionRolePolicy ポリシーの左側にあるチェックボックスを選択し、次へを選択します。
9. ロール名 に`ecsTaskExecutionRole`と入力し、ロールの作成 を選択します。

のアイデンティティベースのポリシーの例 AWS Batch

デフォルトでは、 ユーザーおよびロールには、 AWS Batch リソースを作成または変更する権限はありません。また、、 AWS Command Line Interface （ AWS CLI ） AWS Management Console、または AWS API を使用してタスクを実行することはできません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き受けることができます。

これらサンプルの JSON ポリシードキュメントを使用して、IAM アイデンティティベースのポリシーを作成する方法については、『IAM ユーザーガイド』の「IAM ポリシーの作成」を参照してください。

各リソースタイプの ARN の形式など AWS Batch、 で定義されるアクションとリソースタイプの詳細については、「サービス認証リファレンス」の「 のアクション、リソース、および条件キー AWS Batch」を参照してください。 ARNs

トピック

- ポリシーのベストプラクティス
- AWS Batch コンソールの使用
- 自分の権限の表示をユーザーに許可する

ポリシーのベストプラクティス

ID ベースのポリシーは、ユーザーのアカウントで誰かが AWS Batch リソースを作成、アクセス、または削除できるどうかを決定します。これらのアクションを実行すると、 AWS アカウントに料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーを開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードにアクセス許可を付与するには、多くの一般的なユースケースにアクセス許可を付与する AWS 管理ポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義して、アクセス許可をさらに減らすことをお勧めします。詳細については、IAM ユーザーガイドの「AWS マネージドポリシー」または「AWS ジョブ機能の管理ポリシー」を参照してください。
- 最小特権を適用する – IAM ポリシーで権限を設定するときは、タスクの実行に必要な権限のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定

義します。これは、最小特権権限とも呼ばれています。IAM を使用して権限を適用する方法の詳細については、『IAM ユーザーガイド』の「IAM でのポリシーと権限」を参照してください。

- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。条件を使用して、 などの特定の を介してサービスアクションが使用される場合に AWS のサービス、サービスアクションへのアクセスを許可することもできます AWS CloudFormation。詳細については、IAM ユーザーガイドの [IAM JSON policy elements: Condition] (IAM JSON ポリシー要素：条件) を参照してください。
- IAM Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、「IAM ユーザーガイド」の「IAM Access Analyzer ポリシーの検証」を参照してください。
- 多要素認証 (MFA) を要求する – で IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、セキュリティを強化するために MFA を有効にします。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、「IAM ユーザーガイド」の「MFA 保護 API アクセスの設定」を参照してください。

IAM でのベストプラクティスの詳細については、『IAM ユーザーガイド』の「IAM でのセキュリティのベストプラクティス」を参照してください。

AWS Batch コンソールの使用

AWS Batch コンソールにアクセスするには、最小限のアクセス許可のセットが必要です。これらのアクセス許可により、 の AWS Batch リソースの詳細を一覧表示および表示できます AWS アカウント。最小限必要な許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) に対してコンソールが意図したとおりに機能しません。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスが許可されます。

ユーザーとロールが AWS Batch 引き続きコンソールを使用できるようにするには、エンティティに `ConsoleAccess`または AWS Batch `ReadOnly` AWS 管理ポリシーもアタッチします。詳細については、「IAM ユーザーガイド」の「ユーザーへのアクセス許可の追加」を参照してください。

自分の権限の表示をユーザーに許可する

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWS CLI または AWS API を使用してプログラムでこのアクションを実行するアクセス許可が含まれています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

サービス間での不分別な代理処理の防止

混乱した代理問題は、アクションを実行するためのアクセス許可を持たないエンティティが、より特権のあるエンティティにアクションの実行を強制できてしまう場合に生じる、セキュリティ上の問題です。では AWS、サービス間のなりすましにより、混乱した代理問題が発生する可能性があります。サービス間でのなりすましは、1 つのサービス (呼び出し元サービス) が、別のサービス (呼び出し対象サービス) を呼び出すときに発生する可能性があります。呼び出し元サービスは、本来ならアクセスすることが許可されるべきではない方法でその許可を使用して、別のお客様のリソースに対する処理を実行するように操作される場合があります。これを防ぐために、 AWS には、アカウント内のリソースへのアクセス権が付与されたサービスプリンシパルですべてのサービスのデータを保護するために役立つツールが用意されています。

リソースポリシーで aws:SourceArnおよび aws:SourceAccount グローバル条件コンテキストキーを使用して、 が別のサービスに AWS Batch 付与するアクセス許可をリソースに制限することをお勧めします。aws:SourceArn の値に Amazon S3 バケット ARN などのアカウント ID が含まれていない場合は、両方のグローバル条件コンテキストキーを使用して、アクセス許可を制限する必要があります。同じポリシーステートメントでこれらのグローバル条件コンテキストキーの両方を使用し、アカウント ID にaws:SourceArn の値が含まれていない場合、aws:SourceAccount 値と aws:SourceArn 値の中のアカウントには、同じアカウント ID を使用する必要があります。クロスサービスのアクセスにリソースを 1 つだけ関連付けたい場合は、aws:SourceArn を使用します。そのアカウント内のリソースをクロスサービスの使用に関連付けることを許可する場合は、aws:SourceAccount を使用します。

の値は、 が AWS Batch 保存するリソースaws:SourceArnである必要があります。

混乱した代理問題から保護するための最も効果的な方法は、リソースの完全な ARN を指定して、aws:SourceArn グローバル条件コンテキストキーを使用することです。リソースの完全な ARN が不明な場合や、複数のリソースを指定する場合には、グローバルコンテキスト条件キー aws:SourceArn で、ARN の未知部分を示すためにワイルドカード文字 (*) を使用します。例えば、arn:aws:*servicename*:*:*123456789012*:* です。

次の例は、 で aws:SourceArnおよび aws:SourceAccount グローバル条件コンテキストキーを使用して、混乱した代理問題 AWS Batch を回避する方法を示しています。

例 1:1 つのコンピュート環境にのみアクセスするためのロール

次のロールは、1 つのコンピュート環境へのアクセスにのみ使用できます。* ジョブキューは複数のコンピューティング環境に関連付けることができるため、ジョブ名はとして指定する必要があります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "batch.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ArnLike": {
          "aws:SourceArn": [
            "arn:aws:batch:us-east-1:123456789012:compute-environment/testCE",
            "arn:aws:batch:us-east-1:123456789012:job/*"
          ]
        }
      }
    }
  ]
}
```

例 2:複数のコンピューティング環境にのみアクセスするためのロール

次のロールを使用して複数のコンピュート環境にアクセスできます。* ジョブキューは複数のコンピューティング環境に関連付けることができるため、ジョブ名はとして指定する必要があります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "batch.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ArnLike": {
```

```
          "aws:SourceArn": [
            "arn:aws:batch:us-east-1:123456789012:compute-environment/*",
            "arn:aws:batch:us-east-1:123456789012:job/*"
          ]
        }
      }
    }
  ]
}
```

AWS Batch ID とアクセスのトラブルシューティング

次の情報は、 と IAM の使用時に発生する可能性がある一般的な問題の診断 AWS Batch と修正に役立ちます。

トピック

- AWS Batchでアクションを実行する権限がない
- iam を実行する権限がありません。PassRole
- 自分の AWS アカウント以外のユーザーに自分の AWS Batch リソースへのアクセスを許可したい

AWS Batchでアクションを実行する権限がない

がアクションを実行する権限がないと AWS Management Console 通知した場合は、管理者に連絡してサポートを依頼する必要があります。担当の管理者はお客様のユーザー名とパスワードを発行した人です。

以下のエラー例は、mateojackson ユーザーがコンソールを使用して架空の *my-example-widget* リソースに関する詳細情報を表示しようとしているが、架空の batch:*GetWidget* 許可がないという場合に発生します。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 batch:GetWidget on resource: my-example-widget
```

この場合、Mateo は、batch:*GetWidget* アクションを使用して *my-example-widget* リソースへのアクセスが許可されるように、管理者にポリシーの更新を依頼します。ロールを渡すアクセス許可の付与の詳細については、AWS 「 サービスにロールを渡すアクセス許可をユーザーに付与する」を参照してください。

iam を実行する権限がありません。PassRole

`iam:PassRole` アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して AWS Batchにロールを渡すことができるようにする必要があります。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、そのサービスに既存のロールを渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

以下の例のエラーは、`marymajor` という IAM ユーザーがコンソールを使用して AWS Batchでアクションを実行しようする場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。Mary には、ロールをサービスに渡す権限がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに `iam:PassRole` アクションの実行を許可する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン資格情報を提供した担当者が管理者です。

自分の AWS アカウント以外のユーザーに自分の AWS Batch リソースへのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- がこれらの機能 AWS Batch をサポートしているかどうかを確認するには、「」を参照してくださいが IAM と AWS Batch 連携する方法。
- 所有 AWS アカウント している のリソースへのアクセスを提供する方法については、IAM ユーザーガイドの「所有 AWS アカウント している別の の IAM ユーザーへのアクセスを提供する」を参照してください。

- リソースへのアクセスをサードパーティー に提供する方法については AWS アカウント、IAM ユーザーガイドの「サードパーティー AWS アカウント が所有する へのアクセスを提供する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、『IAM ユーザーガイド』の「外部で認証されたユーザー (ID フェデレーション) へのアクセス権限」を参照してください。
- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いの詳細については、「IAM ユーザーガイド」の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

のサービスにリンクされたロールの使用 AWS Batch

AWS Batch は AWS Identity and Access Management 、 (IAM) サービスにリンクされたロール を使用します。サービスにリンクされたロールは、 に直接リンクされた一意のタイプの IAM ロールです AWS Batch。サービスにリンクされたロールは、 によって事前定義 AWS Batch されており、ユーザーに代わってサービスから他の AWS のサービスを呼び出すために必要なすべてのアクセス許可が含まれています。

サービスにリンクされたロールを使用すると、必要なアクセス許可を手動で追加する必要がなくなるため、 の設定 AWS Batch が簡単になります。 は、サービスにリンクされたロールのアクセス許可 AWS Batch を定義し、特に定義されている場合を除き、 のみが AWS Batch そのロールを引き受けることができます。定義したアクセス許可には、信頼ポリシーと許可ポリシーが含まれます。この許可ポリシーを他の IAM エンティティにアタッチすることはできません。

> **Note**
>
> AWS Batch コンピューティング環境のサービスロールを指定するには、次のいずれかを実行します。
>
> - サービスロールには空の文字列を使用します。これにより、 はサービスロール AWS Batch を作成できます。
> - 以下の形式でサービスロールを指定しますarn:aws:iam::*account_number*:role/aws-service-role/batch.amazonaws.com/AWSServiceRoleForBatch。
>
> 詳細については、「 ユーザーガイドthe section called “正しくないロール名または ARN”」の AWS Batch 「」を参照してください。

サービスリンクロールは、まずその関連リソースを削除しなければ削除できません。これにより、リソースへの意図しないアクセスによるアクセス許可の削除が防止され、 AWS Batch リソースは保護されます。

サービスリンクロールをサポートする他のサービスについては、「IAM と連携するAWS のサービス」を参照して、[サービスリンクロール] 列が [はい] のサービスを探してください。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[はい] リンクを選択します。

のサービスにリンクされたロールのアクセス許可 AWS Batch

AWS Batch は、 という名前のサービスにリンクされたロールを使用しますAWSServiceRoleForBatch。このAWSServiceRoleForBatchロールにより、 AWS Batch はユーザーに代わって AWS リソースを作成および管理できます。

AWSServiceRoleForBatch サービスにリンクされたロールは、`batch.amazonaws.com`サービスプリンシパルを信頼してロールを引き受けます。

という名前の IAM ポリシーBatchServiceRolePolicyは AWS Batch 、 が特定のリソースに対して次のアクションを実行できるようにします。

- `autoscaling` – AWS Batch が Amazon EC2 Auto Scaling resources を作成および管理できるようにします。 は、ほとんどのコンピューティング環境の Amazon EC2 Auto Scaling グループ AWS Batch を作成および管理します。
- `ec2` – AWS Batch が Amazon EC2 インスタンスのライフサイクルを制御し、起動テンプレートとタグを作成および管理できるようにします。 は、一部の EC2 スポットコンピューティング環境の EC2 スポットフリートリクエスト AWS Batch を作成および管理します。
- `ecs` - AWS Batch ジョブ実行用の Amazon ECS クラスター、タスク定義、タスクの作成と管理を許可します。
- `eks` - が検証用の Amazon EKS クラスターリソースを記述 AWS Batch できるようにします。
- `iam` - 所有者によって提供されたロールを検証して Amazon EC2、Amazon EC2 Auto Scaling、および Amazon ECS に渡すことを AWS Batch に許可します。
- `logs` — AWS Batch ジョブのロググループとログストリームの作成と管理を に AWS Batch 許可します。

サービスリンクロールの作成、編集、削除を IAM エンティティ (ユーザー、グループ、ロールなど) に許可するには、アクセス許可を設定する必要があります。詳細については、「IAM ユーザーガイド」の「サービスリンクロール権限」を参照してください。

のサービスにリンクされたロールの作成 AWS Batch

サービスリンクロールを手動で作成する必要はありません。 AWS Management Console、、 AWS CLIまたは AWS API CreateComputeEnvironment で、 `serviceRole`パラメータの値を指定しない場合、 によってサービスにリンクされたロールが自動的に AWS Batch 作成されます。

> **Important**
>
> このサービスリンクロールは、このロールでサポートされている機能を使用する別のサービスでアクションが完了した場合にアカウントに表示されます。また、 AWS Batch 2021 年 3 月 10 日より前にサービスを使用していた場合、サービスにリンクされたロールのサポートが開始されると、 はアカウントに AWSServiceRoleForBatch ロール AWS Batch を作成しました。詳細については、「IAM アカウントに新しいロールが表示される」を参照してください。

このサービスリンクロールを削除した後で再度作成する必要が生じた場合は、同じ手順でアカウントにロールを再作成できます。を使用すると CreateComputeEnvironment、 によってサービスにリンクされたロールが再度 AWS Batch 作成されます。

のサービスにリンクされたロールの編集 AWS Batch

では AWS Batch、 AWSServiceRoleForBatch サービスにリンクされたロールを編集することはできません。サービスリンクロールを作成すると、多くのエンティティによってロールが参照される可能性があるため、ロール名を変更することはできません。ただし、IAM を使用したロール記述の編集はできます。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの編集」を参照してください。

IAM エンティティが AWSServiceRoleForBatch サービスにリンクされたロールの説明を編集できるようにするには

以下のステートメントをアクセス許可ポリシーに追加します。IAM エンティティが サービスにリンクされたロールの説明を編集することを許可します。

```
{
    "Effect": "Allow",
    "Action": [
        "iam:UpdateRoleDescription"
    ],
```

```
    "Resource": "arn:aws:iam::*:role/aws-service-role/batch.amazonaws.com/
AWSServiceRoleForBatch",
    "Condition": {"StringLike": {"iam:AWSServiceName": "batch.amazonaws.com"}}
}
```

のサービスにリンクされたロールの削除 AWS Batch

サービスにリンクされたロールが必要な機能またはサービスが不要になった場合には、そのロールを削除することをお勧めします。そうすることで、使用していないエンティティがアクティブにモニタリングされたり、メンテナンスされたりすることがなくなります。ただし、手動で削除する前に、サービスリンクロールのリソースをクリーンアップする必要があります。

IAM エンティティが AWSServiceRoleForBatch サービスにリンクされたロールを削除できるようにするには

以下のステートメントをアクセス許可ポリシーに追加します。IAM エンティティが サービスにリンクされたロールを削除することを許可します。

```
{
    "Effect": "Allow",
    "Action": [
        "iam:DeleteServiceLinkedRole",
        "iam:GetServiceLinkedRoleDeletionStatus"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/batch.amazonaws.com/
AWSServiceRoleForBatch",
    "Condition": {"StringLike": {"iam:AWSServiceName": "batch.amazonaws.com"}}
}
```

サービスリンクロールのクリーンアップ

IAM を使用してサービスにリンクされたロールを削除する前に、まずロールにアクティブなセッションがないことを確認し、1 つのパーティション内のすべての AWS リージョンでロールを使用するすべての AWS Batch コンピューティング環境を削除する必要があります。

サービスにリンクされたロールがアクティブなセッションを持っているかどうかを確認するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで、ロールを選択し、 AWSServiceRoleForBatch 名前 (チェックボックスではありません) を選択します。

3. Summary] (概要) ページで Access Advisor] (アクセスアドバイザー) を選択し、サービスにリンクされたロールの最新のアクティビティを確認します。

> **Note**
>
> AWS Batch が AWSServiceRoleForBatch ロールを使用しているかどうかわからない場合は、ロールを削除できます。サービスでロールが使用されている場合、ロールは削除されません。ロールが使用されているリージョンが表示されます。ロールが使用されている場合は、ロールを削除する前にセッションが終了するのを待つ必要があります。サービスにリンクされたロールのセッションを取り消すことはできません。

AWSServiceRoleForBatch サービスにリンクされたロールによって使用されている AWS Batch リソースを削除するには

AWSServiceRoleForBatch ロールを削除する前に、すべての AWS リージョンで AWSServiceRoleForBatch ロールを使用するすべての AWS Batch コンピューティング環境を削除する必要があります。

1. https://console.aws.amazon.com/batch/ で AWS Batch コンソールを開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. ナビゲーションペインで、Compute environments] (コンピューティング環境) を選択します。
4. コンピューティング環境を選択します。
5. Disable] (無効化) を選択します。State] (状態) が DISABLED] (無効) になるまで待ちます。
6. コンピューティング環境を選択します。
7. Delete] (削除) を選択します。Delete compute environment] (コンピューティング環境の削除) を選択し、削除したいコンピューティング環境を確認します。
8. すべてのリージョンでサービスにリンクされたロールを使用する、すべてのコンピューティング環境について、手順 1 ~ 7 を繰り返します。

IAM でのサービスにリンクされたロールの削除 (コンソール)

IAM コンソールを使用して、サービスにリンクされたロールを削除できます。

サービスにリンクされたロールを削除するには (コンソール)

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. IAM コンソールのナビゲーションペインで [ロール] を選択します。次に、名前や行自体ではなく AWSServiceRoleForBatch、 の横にあるチェックボックスをオンにします。
3. Delete role] (ロールの削除) を選択します。
4. 確認ダイアログボックスで、サービスの最終アクセス時間データを確認します。これは、選択したそれぞれのロールの AWS のサービスへの最終アクセス時間を示します。これは、そのロールが現在アクティブであるかどうかを確認するのに役立ちます。先に進む場合は、Yes, Delete] (はい、削除する) を選択し、削除するサービスにリンクされたロールを送信します。
5. IAM コンソール通知を見て、サービスにリンクされたロールの削除の進行状況をモニタリングします。IAM サービスにリンクされたロールの削除は非同期であるため、削除するロールを送信すると、削除タスクは成功または失敗する可能性があります。

   - タスクが成功した場合は、ロールがリストから削除され、成功の通知がページの上部に表示されます。
   - タスクが失敗した場合は、通知から View details] (詳細を表示) または View Resources] (リソースを表示) を選択して、削除が失敗した理由を知ることができます。ロールがサービスのリソースを使用しているために削除が失敗したとき、サービスがその情報を返す場合は、通知にはリソースのリストが含まれます。次にリソースをクリーンアップしてから、削除リクエストをもう一度送信できます。

     > **Note**
     >
     > サービスが返す情報に応じて、このプロセスを何度も繰り返す必要があります。たとえば、サービスにリンクされたロールが 6 つのリソースを使用しており、サービスはそのうち 5 つのリソースに関する情報を返すことがあります。5 つのリソースをクリーンアップして削除するロールを再度送信すると、削除は失敗し、残りの 1 つのリソースが報告されます。サービスはすべてのリソースを返しますが、そのうちいくつかはリソースを報告しない場合もあります。

   - タスクが失敗し、通知にリソースのリストが含まれていない場合、サービスはその情報を返さない可能性があります。サービスのリソースをクリーンアップする方法の詳細については、IAM と連携するAWS のサービスを参照してください。使用しているサービスをテーブルで見つけ、はいリンクを選択すると、そのサービスのサービスにリンクされたロールに関するドキュメントが表示されます。

IAM でのサービスにリンクされたロールの削除 (AWS CLI)

から IAM コマンドを使用して AWS Command Line Interface 、サービスにリンクされたロールを削除できます。

サービスにリンクされたロールを削除するには (CLI)

1. サービスにリンクされているロールは、使用されている、または関連するリソースがある場合は削除できないため、削除リクエストを送信する必要があります。これらの条件が満たされない場合、そのリクエストは拒否される可能性があります。レスポンスから `deletion-task-id` を取得して、削除タスクのステータスを確認する必要があります。サービスにリンクされたロールの削除リクエストを送信するには、次のコマンドを入力します：

   ```
   $ aws iam delete-service-linked-role --role-name AWSServiceRoleForBatch
   ```

2. 削除タスクのステータスを確認するには、次のコマンドを入力します：

   ```
   $ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
   ```

   削除タスクのステータスは、`NOT_STARTED`、`IN_PROGRESS`、`SUCCEEDED`、または `FAILED` となります。削除が失敗した場合は、失敗した理由がコールによって返され、トラブルシューティングが可能になります。ロールがサービスのリソースを使用しているために削除が失敗したとき、サービスがその情報を返す場合は、通知にはリソースのリストが含まれます。次にリソースをクリーンアップしてから、削除リクエストをもう一度送信できます。

   > **Note**
   >
   > サービスが返す情報に応じて、このプロセスを何度も繰り返す必要があります。たとえば、サービスにリンクされたロールが 6 つのリソースを使用しており、サービスはそのうち 5 つのリソースに関する情報を返すことがあります。5 つのリソースをクリーンアップして削除するロールを再度送信すると、削除は失敗し、残りの 1 つのリソースが報告されます。サービスはすべてのリソースを返す場合もあれば、一部を返す場合もあります。または、リソースが報告されないこともあります。リソースを報告しないサービスのリソースをクリーンアップする方法については、IAM と連携するAWS サービスを参照してください。使用しているサービスをテーブルで見つけ、はいリンクを選択すると、そのサービスのサービスにリンクされたロールに関するドキュメントが表示されます。

サービスにリンクされた IAM (AWS API) でのロールの削除

IAM API を使用して、サービスにリンクされたロールを削除できます。

サービスにリンクされたロールを削除するには (API)

1. サービスにリンクされたロールの削除リクエストを送信するには、 を呼び出しますDeleteServiceLinkedRole。リクエストで、 AWSServiceRoleForBatch ロール名を指定します。

   サービスにリンクされているロールは、使用されている、または関連するリソースがある場合は削除できないため、削除リクエストを送信する必要があります。これらの条件が満たされない場合、そのリクエストは拒否される可能性があります。レスポンスから `DeletionTaskId` を取得して、削除タスクのステータスを確認する必要があります。

2. 削除のステータスを確認するには、 を呼び出しますGetServiceLinkedRoleDeletionStatus。リクエストで `DeletionTaskId` を指定します。

   削除タスクのステータスは、`NOT_STARTED`、`IN_PROGRESS`、`SUCCEEDED`、または `FAILED` となります。削除が失敗した場合は、失敗した理由がコールによって返され、トラブルシューティングが可能になります。ロールがサービスのリソースを使用しているために削除が失敗したとき、サービスがその情報を返す場合は、通知にはリソースのリストが含まれます。次にリソースをクリーンアップしてから、削除リクエストをもう一度送信できます。

   > **Note**
   >
   > サービスが返す情報に応じて、このプロセスを何度も繰り返す必要があります。たとえば、サービスにリンクされたロールが 6 つのリソースを使用しており、サービスはそのうち 5 つのリソースに関する情報を返すことがあります。5 つのリソースをクリーンアップして削除するロールを再度送信すると、削除は失敗し、残りの 1 つのリソースが報告されます。サービスはすべてのリソースを返しますが、そのうちいくつかはリソースを報告しない場合もあります。リソースを報告しないサービスのリソースをクリーンアップする方法については、IAM と連携するAWS のサービス サービスを参照してください。使用しているサービスをテーブルで見つけ、はいリンクを選択すると、そのサービスのサービスにリンクされたロールに関するドキュメントが表示されます。

AWS Batch サービスにリンクされたロールでサポートされているリージョン

AWS Batch は、サービスが利用可能なすべてのリージョンでサービスにリンクされたロールの使用をサポートします。詳細については、AWS Batch エンドポイントを参照してください。

AWS の マネージドポリシー AWS Batch

AWS 管理ポリシーを使用して、チームやプロビジョニングされた AWS リソースのアイデンティティアクセス管理を簡素化できます。 AWS 管理ポリシーは、さまざまな一般的なユースケースをカバーし、デフォルトで AWS アカウントで利用でき、ユーザーに代わって管理および更新されます。AWS 管理ポリシーのアクセス許可は変更できません。より柔軟性が必要な場合は、IAM カスタマー管理ポリシーを作成することもできます。このようにして、チームのプロビジョニングされたリソースに、必要な権限のみを付与できます。

AWS 管理ポリシーの詳細については、「IAM ユーザーガイド」の「AWS 管理ポリシー」を参照してください。

AWS サービスは、ユーザーに代わって AWS マネージドポリシーを維持および更新します。定期的に、 AWS サービスは AWS マネージドポリシーに許可を追加します。 AWS マネージドポリシーは、新機能の起動またはオペレーションが利用可能になると更新される可能性が最も高くなります。この更新は、ポリシーが添付されているすべてのアイデンティティ (ユーザー、グループ、ロール) に自動的に影響します。ただし、権限を削除したり、既存の権限を破棄することはありません。

さらに、 は、複数の サービスにまたがる職務機能の マネージドポリシー AWS をサポートします。例えば、 `ReadOnlyAccess` AWS 管理ポリシーは、すべての AWS サービスとリソースへの読み取り専用アクセスを提供します。サービスが新機能を起動すると、 は新しいオペレーションとリソースの読み取り専用アクセス許可 AWS を追加します。ジョブ機能のポリシーの一覧および詳細については、「IAM ユーザーガイド」の「AWS のジョブ機能のマネージドポリシー」を参照してください。

AWS マネージドポリシー: BatchServiceRolePolicy

BatchServiceRolePolicy マネージド IAM ポリシーは、`AWSServiceRoleForBatch`サービスにリンクされたロールによって使用されます。これにより、 AWS Batch はユーザーに代わってアクションを実行できます。このポリシーを IAM エンティティにアタッチすることはできません。詳細については、「のサービスにリンクされたロールの使用 AWS Batch」を参照してください。

このポリシーにより AWS Batch 、 は特定のリソースに対して次のアクションを実行できます。

- `autoscaling` – AWS Batch が Amazon EC2 Auto Scaling resources を作成および管理できるようにします。 は、ほとんどのコンピューティング環境の Amazon EC2 Auto Scaling グループ AWS Batch を作成および管理します。
- `ec2` – AWS Batch が Amazon EC2 インスタンスのライフサイクルを制御し、起動テンプレートとタグを作成および管理できるようにします。 は、一部の EC2 スポットコンピューティング環境の EC2 スポットフリートリクエスト AWS Batch を作成および管理します。
- `ecs` - AWS Batch ジョブ実行用の Amazon ECS クラスター、タスク定義、タスクの作成と管理を許可します。
- `eks` - が検証用の Amazon EKS クラスターリソースを記述 AWS Batch できるようにします。
- `iam` - 所有者によって提供されたロールを検証して Amazon EC2、Amazon EC2 Auto Scaling、および Amazon ECS に渡すことを AWS Batch に許可します。
- `logs` — AWS Batch ジョブのロググループとログストリームの作成と管理を に AWS Batch 許可します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AWSBatchPolicyStatement1",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeInstances",
                "ec2:DescribeInstanceStatus",
                "ec2:DescribeInstanceAttribute",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeKeyPairs",
                "ec2:DescribeImages",
                "ec2:DescribeImageAttribute",
                "ec2:DescribeSpotInstanceRequests",
                "ec2:DescribeSpotFleetInstances",
                "ec2:DescribeSpotFleetRequests",
                "ec2:DescribeSpotPriceHistory",
                "ec2:DescribeSpotFleetRequestHistory",
                "ec2:DescribeVpcClassicLink",
```

```
                "ec2:DescribeLaunchTemplateVersions",
                "ec2:RequestSpotFleet",
                "autoscaling:DescribeAccountLimits",
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:DescribeLaunchConfigurations",
                "autoscaling:DescribeAutoScalingInstances",
                "autoscaling:DescribeScalingActivities",
                "eks:DescribeCluster",
                "ecs:DescribeClusters",
                "ecs:DescribeContainerInstances",
                "ecs:DescribeTaskDefinition",
                "ecs:DescribeTasks",
                "ecs:ListClusters",
                "ecs:ListContainerInstances",
                "ecs:ListTaskDefinitionFamilies",
                "ecs:ListTaskDefinitions",
                "ecs:ListTasks",
                "ecs:DeregisterTaskDefinition",
                "ecs:TagResource",
                "ecs:ListAccountSettings",
                "logs:DescribeLogGroups",
                "iam:GetInstanceProfile",
                "iam:GetRole"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AWSBatchPolicyStatement2",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/batch/job*"
        },
        {
            "Sid": "AWSBatchPolicyStatement3",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/batch/job*:log-stream:*"
        },
        {
```

```
            "Sid": "AWSBatchPolicyStatement4",
            "Effect": "Allow",
            "Action": [
                "autoscaling:CreateOrUpdateTags"
            ],
            "Resource": "*",
            "Condition": {
                "Null": {
                    "aws:RequestTag/AWSBatchServiceTag": "false"
                }
            }
        },
        {
            "Sid": "AWSBatchPolicyStatement5",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "ec2.amazonaws.com",
                        "ec2.amazonaws.com.cn",
                        "ecs-tasks.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Sid": "AWSBatchPolicyStatement6",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": [
                        "spot.amazonaws.com",
                        "spotfleet.amazonaws.com",
                        "autoscaling.amazonaws.com",
                        "ecs.amazonaws.com"
                    ]
                }
            }
```

```
        },
        {
            "Sid": "AWSBatchPolicyStatement7",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateLaunchTemplate"
            ],
            "Resource": "*",
            "Condition": {
                "Null": {
                    "aws:RequestTag/AWSBatchServiceTag": "false"
                }
            }
        },
        {
            "Sid": "AWSBatchPolicyStatement8",
            "Effect": "Allow",
            "Action": [
                "ec2:TerminateInstances",
                "ec2:CancelSpotFleetRequests",
                "ec2:ModifySpotFleetRequest",
                "ec2:DeleteLaunchTemplate"
            ],
            "Resource": "*",
            "Condition": {
                "Null": {
                    "aws:ResourceTag/AWSBatchServiceTag": "false"
                }
            }
        },
        {
            "Sid": "AWSBatchPolicyStatement9",
            "Effect": "Allow",
            "Action": [
                "autoscaling:CreateLaunchConfiguration",
                "autoscaling:DeleteLaunchConfiguration"
            ],
            "Resource":
"arn:aws:autoscaling:*:*:launchConfiguration:*:launchConfigurationName/AWSBatch*"
        },
        {
            "Sid": "AWSBatchPolicyStatement10",
            "Effect": "Allow",
            "Action": [
```

```
                "autoscaling:CreateAutoScalingGroup",
                "autoscaling:UpdateAutoScalingGroup",
                "autoscaling:SetDesiredCapacity",
                "autoscaling:DeleteAutoScalingGroup",
                "autoscaling:SuspendProcesses",
                "autoscaling:PutNotificationConfiguration",
                "autoscaling:TerminateInstanceInAutoScalingGroup"
            ],
            "Resource":
"arn:aws:autoscaling:*:*:autoScalingGroup:*:autoScalingGroupName/AWSBatch*"
        },
        {
            "Sid": "AWSBatchPolicyStatement11",
            "Effect": "Allow",
            "Action": [
                "ecs:DeleteCluster",
                "ecs:DeregisterContainerInstance",
                "ecs:RunTask",
                "ecs:StartTask",
                "ecs:StopTask"
            ],
            "Resource": "arn:aws:ecs:*:*:cluster/AWSBatch*"
        },
        {
            "Sid": "AWSBatchPolicyStatement12",
            "Effect": "Allow",
            "Action": [
                "ecs:RunTask",
                "ecs:StartTask",
                "ecs:StopTask"
            ],
            "Resource": "arn:aws:ecs:*:*:task-definition/*"
        },
        {
            "Sid": "AWSBatchPolicyStatement13",
            "Effect": "Allow",
            "Action": [
                "ecs:StopTask"
            ],
            "Resource": "arn:aws:ecs:*:*:task/*/*"
        },
        {
            "Sid": "AWSBatchPolicyStatement14",
            "Effect": "Allow",
```

```
            "Action": [
                "ecs:CreateCluster",
                "ecs:RegisterTaskDefinition"
            ],
            "Resource": "*",
            "Condition": {
                "Null": {
                    "aws:RequestTag/AWSBatchServiceTag": "false"
                }
            }
        },
        {
            "Sid": "AWSBatchPolicyStatement15",
            "Effect": "Allow",
            "Action": "ec2:RunInstances",
            "Resource": [
                "arn:aws:ec2:*::image/*",
                "arn:aws:ec2:*::snapshot/*",
                "arn:aws:ec2:*:*:subnet/*",
                "arn:aws:ec2:*:*:network-interface/*",
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:volume/*",
                "arn:aws:ec2:*:*:key-pair/*",
                "arn:aws:ec2:*:*:launch-template/*",
                "arn:aws:ec2:*:*:placement-group/*",
                "arn:aws:ec2:*:*:capacity-reservation/*",
                "arn:aws:ec2:*:*:elastic-gpu/*",
                "arn:aws:elastic-inference:*:*:elastic-inference-accelerator/*",
                "arn:aws:resource-groups:*:*:group/*"
            ]
        },
        {
            "Sid": "AWSBatchPolicyStatement16",
            "Effect": "Allow",
            "Action": "ec2:RunInstances",
            "Resource": "arn:aws:ec2:*:*:instance/*",
            "Condition": {
                "Null": {
                    "aws:RequestTag/AWSBatchServiceTag": "false"
                }
            }
        },
        {
            "Sid": "AWSBatchPolicyStatement17",
```

```
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "ec2:CreateAction": [
                        "RunInstances",
                        "CreateLaunchTemplate",
                        "RequestSpotFleet"
                    ]
                }
            }
        }
    ]
}
```

AWS 管理ポリシー: AWSBatchServiceRoleポリシー

という名前のロール許可ポリシーAWSBatchServiceRoleでは AWS Batch 、 が特定のリソースに対して次のアクションを実行できます。

AWSBatchServiceRole マネージド IAM ポリシーは、 という名前のロールによってよく使用されAWSBatchServiceRole、次のアクセス許可が含まれます。最小特権を付与するという標準的なセキュリティアドバイスに従って、 AWSBatchServiceRole管理ポリシーをガイドとして使用できます。管理ポリシーで付与されているアクセス許可のいずれかがユースケースに必要でない場合、カスタムポリシーを作成し、必要なアクセス許可のみを追加します。この AWS Batch マネージドポリシーとロールは、ほとんどのコンピューティング環境タイプで使用できますが、サービスにリンクされたロールの使用は less-error-prone、より範囲が広く、管理エクスペリエンスが向上した に推奨されます。

- `autoscaling` – AWS Batch が Amazon EC2 Auto Scaling resources を作成および管理できるようにします。 は、ほとんどのコンピューティング環境の Amazon EC2 Auto Scaling グループ AWS Batch を作成および管理します。
- `ec2` – Amazon EC2 インスタンスのライフサイクルの管理、起動テンプレートとタグの作成と管理 AWS Batch を許可します。 は、一部の EC2 スポットコンピューティング環境の EC2 スポットフリートリクエスト AWS Batch を作成および管理します。

- ecs - AWS Batch ジョブ実行用の Amazon ECS クラスター、タスク定義、タスクの作成と管理を許可します。
- iam - 所有者によって提供されたロールを検証して Amazon EC2、Amazon EC2 Auto Scaling、および Amazon ECS に渡すことを AWS Batch に許可します。
- logs — AWS Batch ジョブのロググループとログストリームの作成と管理を に AWS Batch 許可します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AWSBatchPolicyStatement1",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeInstances",
                "ec2:DescribeInstanceStatus",
                "ec2:DescribeInstanceAttribute",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeKeyPairs",
                "ec2:DescribeImages",
                "ec2:DescribeImageAttribute",
                "ec2:DescribeSpotInstanceRequests",
                "ec2:DescribeSpotFleetInstances",
                "ec2:DescribeSpotFleetRequests",
                "ec2:DescribeSpotPriceHistory",
                "ec2:DescribeSpotFleetRequestHistory",
                "ec2:DescribeVpcClassicLink",
                "ec2:DescribeLaunchTemplateVersions",
                "ec2:CreateLaunchTemplate",
                "ec2:DeleteLaunchTemplate",
                "ec2:RequestSpotFleet",
                "ec2:CancelSpotFleetRequests",
                "ec2:ModifySpotFleetRequest",
                "ec2:TerminateInstances",
                "ec2:RunInstances",
                "autoscaling:DescribeAccountLimits",
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:DescribeLaunchConfigurations",
                "autoscaling:DescribeAutoScalingInstances",
```

```
                "autoscaling:DescribeScalingActivities",
                "autoscaling:CreateLaunchConfiguration",
                "autoscaling:CreateAutoScalingGroup",
                "autoscaling:UpdateAutoScalingGroup",
                "autoscaling:SetDesiredCapacity",
                "autoscaling:DeleteLaunchConfiguration",
                "autoscaling:DeleteAutoScalingGroup",
                "autoscaling:CreateOrUpdateTags",
                "autoscaling:SuspendProcesses",
                "autoscaling:PutNotificationConfiguration",
                "autoscaling:TerminateInstanceInAutoScalingGroup",
                "ecs:DescribeClusters",
                "ecs:DescribeContainerInstances",
                "ecs:DescribeTaskDefinition",
                "ecs:DescribeTasks",
                "ecs:ListAccountSettings",
                "ecs:ListClusters",
                "ecs:ListContainerInstances",
                "ecs:ListTaskDefinitionFamilies",
                "ecs:ListTaskDefinitions",
                "ecs:ListTasks",
                "ecs:CreateCluster",
                "ecs:DeleteCluster",
                "ecs:RegisterTaskDefinition",
                "ecs:DeregisterTaskDefinition",
                "ecs:RunTask",
                "ecs:StartTask",
                "ecs:StopTask",
                "ecs:UpdateContainerAgent",
                "ecs:DeregisterContainerInstance",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogGroups",
                "iam:GetInstanceProfile",
                "iam:GetRole"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AWSBatchPolicyStatement2",
            "Effect": "Allow",
            "Action": "ecs:TagResource",
            "Resource": [
```

```
                "arn:aws:ecs:*:*:task/*_Batch_*"
            ]
        },
        {
            "Sid": "AWSBatchPolicyStatement3",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "ec2.amazonaws.com",
                        "ec2.amazonaws.com.cn",
                        "ecs-tasks.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Sid": "AWSBatchPolicyStatement4",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": [
                        "spot.amazonaws.com",
                        "spotfleet.amazonaws.com",
                        "autoscaling.amazonaws.com",
                        "ecs.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Sid": "AWSBatchPolicyStatement5",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "*"
```

```
            ],
            "Condition": {
                "StringEquals": {
                    "ec2:CreateAction": "RunInstances"
                }
            }
        }
    ]
}
```

AWS マネージドポリシー: AWSBatchFullAccess

このAWSBatchFullAccessポリシーは、 AWS Batch リソースへのフルアクセスを AWS Batch アクションに付与します。また、Amazon EC2、Amazon ECS、Amazon EKS、および IAM サービスのアクションアクセスを記述 CloudWatchおよび一覧表示します。これは、ユーザーまたはロールのいずれかの IAM ID が、ユーザーに代わって作成された AWS Batch マネージドリソースを表示できるようにするためです。最後に、このポリシーは、選択した IAM ロールをこれらのサービスに渡すことも許可します。

を IAM エンティティAWSBatchFullAccessにアタッチできます。 AWS Batch は、ユーザーに代わって がアクションを実行できるようにするサービスロールにもこのポリシー AWS Batch をアタッチします。

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect":"Allow",
      "Action":[
        "batch:*",
        "cloudwatch:GetMetricStatistics",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeKeyPairs",
        "ec2:DescribeVpcs",
        "ec2:DescribeImages",
        "ec2:DescribeLaunchTemplates",
        "ec2:DescribeLaunchTemplateVersions",
        "ecs:DescribeClusters",
        "ecs:Describe*",
        "ecs:List*",
        "eks:DescribeCluster",
```

```
        "eks:ListClusters",
        "logs:Describe*",
        "logs:Get*",
        "logs:TestMetricFilter",
        "logs:FilterLogEvents",
        "iam:ListInstanceProfiles",
        "iam:ListRoles"
      ],
      "Resource":"*"
    },
    {
      "Effect":"Allow",
      "Action":[
        "iam:PassRole"
      ],
      "Resource":[
        "arn:aws:iam::*:role/AWSBatchServiceRole",
        "arn:aws:iam::*:role/service-role/AWSBatchServiceRole",
        "arn:aws:iam::*:role/ecsInstanceRole",
        "arn:aws:iam::*:instance-profile/ecsInstanceRole",
        "arn:aws:iam::*:role/iaws-ec2-spot-fleet-role",
        "arn:aws:iam::*:role/aws-ec2-spot-fleet-role",
        "arn:aws:iam::*:role/AWSBatchJobRole*"
      ]
    },
    {
      "Effect":"Allow",
      "Action":[
        "iam:CreateServiceLinkedRole"
      ],
      "Resource":"arn:aws:iam::*:role/*Batch*",
      "Condition": {
        "StringEquals": {
          "iam:AWSServiceName": "batch.amazonaws.com"
          }
      }
    }
  ]
}
```

AWS BatchAWS 管理ポリシーの更新

このサービスがこれらの変更の追跡を開始した AWS Batch 以降の の AWS マネージドポリシーの更新に関する詳細を表示します。このページの変更に関する自動通知については、 AWS Batch ドキュメント履歴ページの RSS フィードをサブスクライブしてください。

変更	説明	日付
BatchServiceRolePolicy ポリシーが更新されました	スポットフリートのリクエスト履歴と Amazon EC2 Auto Scaling アクティビティを記述するためのサポートを追加するために更新されました。	2023 年 12 月 5 日
AWSBatchServiceRole ポリシーが追加されました	ステートメント IDsec2:DescribeSpotFleetRequestHistory および にアクセス AWS Batch 許可を付与するように更新されましたautoscaling:DescribeScalingActivities 。	2023 年 12 月 5 日
BatchServiceRolePolicy ポリシーが更新されました	Amazon EKS クラスターを記述するためのサポートを追加するために更新されました。	2022 年 10 月 20 日
AWSBatchFullAccess ポリシーが更新されました	Amazon EKS クラスターを記述するためのサポートを追加するために更新されました。	2022 年 10 月 20 日
BatchServiceRolePolicy ポリシーが更新されました	AWS Resource Groupsによって管理される Amazon EC2 キャパシティ予約グループのサポートを追加するように更新されました。詳細については、Amazon EC2 ユーザーガイド」の「キャパシティ予約グループの使用」を参照してください。	2022 年 5 月 18 日
BatchServiceRolePolicy および AWSBatchServiceRoleポリシーが更新されました	異常なインスタンスが置き換えられるように、Amazon EC2 の AWS Batch マネージドインスタンスのステータスを記述するサポートを追加しました。	2021 年 12 月 6 日

変更	説明	日付
BatchServiceRolePolicy ポリシーが更新されました	Amazon EC2 で配置グループ、キャパシティの予約、Elastic GPU、および Elastic Inference リソースのサポートを追加するように更新されました。	2021 年 3 月 26 日
BatchServiceRolePolicy ポリシーが追加されました	AWSServiceRoleForBatch サービスにリンクされたロールの BatchServiceRolePolicyマネージドポリシーを使用すると、によって管理されるサービスにリンクされたロールを使用できます AWS Batch。このポリシーを使用すると、コンピュート環境で使用するために独自のロールを維持する必要がありません。	2021 年 3 月 10 日
AWSBatchFullAccess - サービスにリンクされたロールを追加するアクセス許可を追加する	IAM アクセス許可を追加して、AWSServiceRoleForBatchサービスにリンクされたロールをアカウントに追加できるようにします。	2021 年 3 月 10 日
AWS Batch が変更の追跡を開始しました	AWS Batch が AWS マネージドポリシーの変更の追跡を開始しました。	2021 年 3 月 10 日

インターフェイスエンドポイント AWS Batch を使用したアクセス

を使用して AWS PrivateLink 、VPC と の間にプライベート接続を作成できます AWS Batch。インターネットゲートウェイ、NAT デバイス、VPN 接続、または AWS Direct Connect 接続を使用せずに、VPC 内にあるかのように AWS Batch にアクセスできます。VPC のインスタンスは、パブリック IP アドレスがなくても AWS Batchにアクセスできます。

このプライベート接続を確立するには、 AWS PrivateLinkを利用したインターフェイスエンドポイントを作成します。インターフェイスエンドポイントに対して有効にする各サブネットにエンドポイントネットワークインターフェイスを作成します。これらは、 AWS Batch宛てのトラフィックのエントリポイントとして機能するリクエスタ管理型ネットワークインターフェイスです。

詳細については、 ガイドのインターフェイス VPC エンドポイント ()を参照してください。

に関する考慮事項 AWS Batch

のインターフェイスエンドポイントを設定する前に AWS Batch、「 AWS PrivateLink ガイド」の「インターフェイスエンドポイントのプロパティと制限」を参照してください。

AWS Batch は、インターフェイスエンドポイントを介したすべての API アクションの呼び出しをサポートします。

のインターフェイス VPC エンドポイントを設定する前に AWS Batch、次の考慮事項に注意してください。

- Fargate リソース起動タイプを使用するジョブでは、Amazon ECS のインターフェイス VPC エンドポイントは必要ありませんが、以下のポイントで説明する AWS Batch、Amazon ECR、Secrets Manager、または Amazon CloudWatch Logs のインターフェイス VPC エンドポイントが必要になる場合があります。
  - ジョブを実行するには、Amazon ECS 用に、インターフェイス VPC エンドポイントを作成する必要があります。詳細については、Amazon Elastic Container Service デベロッパーガイドのインターフェイス VPC エンドポイント (AWS PrivateLink PrivateLink)を参照してください。
  - Amazon ECR からプライベートイメージをプルできるようにするには、Amazon ECR 用のインターフェイス VPC エンドポイントを 作成する必要があります。詳細については、Amazon Elastic Container Registry ユーザーガイドのインターフェイス VPC エンドポイント (AWS PrivateLink)を参照してください。
  - タスクで Secrets Manager から機密データをプルできるようにするには、Secrets Manager 用のインターフェイス VPC エンドポイントを作成する必要があります。詳細については、AWS Secrets Manager ユーザーガイドのVPC EndpointでSecrets Managerを使用するを参照してください。
  - VPC にインターネットゲートウェイがなく、ジョブが`awslogs`ログドライバーを使用してログ情報を CloudWatch ログに送信する場合は、ログのインターフェイス VPC CloudWatch エンドポイントを作成する必要があります。詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「インターフェイス VPC エンドポイントでのログの使用」を参照してください。CloudWatch
- EC2 起動タイプを使用するタスクでは、起動されたコンテナインスタンスが Amazon ECS コンテナエージェントのバージョン `1.25.1` 以降を実行する必要があります。詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS コンテナエージェントバージョンを参照してください。

- 現在、VPC エンドポイントはクロスリージョンリクエストをサポートしていません。 AWS Batch に対して API コールを発行するリージョンと同じリージョンにエンドポイントを作成してください。
- VPC エンドポイントでは、Amazon Route 53 を介して Amazon 提供の DNS のみがサポートされています。独自の DNS を使用したい場合は、条件付き DNS 転送を使用できます。詳細については、Amazon VPC ユーザーガイドのDHCP Options Setsを参照してください。
- VPCエンドポイントにアタッチされたセキュリティグループでは、VPCのプライベートサブネットから、ポート 443 で着信接続を許可する必要があります。
- AWS Batch は、次の で VPC インターフェイスエンドポイントをサポートしていません AWS リージョン。
  - アジアパシフィック (大阪) (ap-northeast-3)
  - アジアパシフィック (ジャカルタ) (ap-southeast-3)

のインターフェイスエンドポイントを作成する AWS Batch

Amazon VPC コンソールまたは AWS Command Line Interface () AWS Batch を使用して、 用のインターフェイスエンドポイントを作成できますAWS CLI。詳細については、「AWS PrivateLink ガイド」の「インターフェイスエンドポイントを作成」を参照してください。

次のサービス名 AWS Batch を使用して、 用のインターフェイスエンドポイントを作成します。

```
com.amazonaws.region.batch
```

例:

```
com.amazonaws.us-east-2.batch
```

aws-cn パーティションでは形式が異なります。

```
cn.com.amazonaws.region.batch
```

例:

```
cn.com.amazonaws.cn-northwest-1.batch
```

インターフェイスエンドポイントのプライベート DNS を有効にすると、デフォルトのリージョン DNS 名 AWS Batch を使用して に API リクエストを実行できます。例えば `batch.us-east-1.amazonaws.com` です。

詳細については、AWS PrivateLink ガイドのインターフェイスエンドポイントを介したサービスへのアクセスを参照してください。

インターフェイスエンドポイントのエンドポイントポリシーを作成する

エンドポイントポリシーは、インターフェイスエンドポイントにアタッチできる IAM リソースです。デフォルトのエンドポイントポリシーでは、インターフェイスエンドポイント AWS Batch を介した へのフルアクセスが許可されます。VPC から AWS Batch API への許可されたアクセスをコントロールするには、カスタムエンドポイントポリシーをインターフェイスエンドポイントにアタッチします。

エンドポイントポリシーは、以下の情報を指定します。

- アクションを実行できるプリンシパル (AWS アカウント、ユーザー、IAM ロール)。
- 実行可能なアクション。
- このアクションを実行できるリソース。

詳細については、AWS PrivateLink ガイドのControl access to services using endpoint policies (エンドポイントポリシーを使用してサービスへのアクセスをコントロールする)を参照してください。

例: AWS Batch アクションの VPC エンドポイントポリシー

以下は、カスタムエンドポイントポリシーの例です。このポリシーをインターフェイスエンドポイントにアタッチすると、すべてのリソースのすべてのプリンシパルに対して、リストされている AWS Batch アクションへのアクセスが許可されます。

```
{
   "Statement": [
      {
         "Principal": "*",
         "Effect": "Allow",
         "Action": [
            "batch:SubmitJob",
            "batch:ListJobs",
            "batch:DescribeJobs"
         ],
```

```
            "Resource":"*"
        }
    ]
}
```

のコンプライアンス検証 AWS Batch

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、コンプライアンスプログラムAWS のサービス による対象範囲内のコンプライアンスプログラムを参照し、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS 「コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「 でのレポートのダウンロード AWS Artifact」の」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。 は、コンプライアンスに役立つ以下のリソース AWS を提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド – これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境 AWS を にデプロイする手順について説明します。
- アマゾン ウェブ サービスにおける HIPAA セキュリティとコンプライアンスのアーキテクチャ — このホワイトペーパーでは、企業が AWS を使用して HIPAA 対象アプリケーションを作成する方法について説明します。

> **Note**
>
> すべて AWS のサービス HIPAA の対象となるわけではありません。詳細については、「HIPAA 対応サービスのリファレンス」を参照してください。

- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- AWS カスタマーコンプライアンスガイド – コンプライアンスの観点から責任共有モデルを理解します。このガイドでは、ガイダンスを保護し AWS のサービス 、複数のフレームワーク (米国国立標準技術研究所 (NIST)、Payment Card Industry Security Standards Council (PCI)、国際標準化機構 (ISO) を含む) のセキュリティコントロールにマッピングするためのベストプラクティスをまとめています。

- 「 デベロッパーガイド」の「 ルールによるリソースの評価」 – この AWS Config サービスは、リソース設定が社内プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。 AWS Config
- AWS Security Hub – これにより AWS のサービス 、 内のセキュリティ状態を包括的に確認できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールのリストについては、「Security Hub のコントロールリファレンス」を参照してください。
- Amazon GuardDuty — これにより AWS アカウント、疑わしいアクティビティや悪意のあるアクティビティがないか環境を監視することで、、、ワークロード、コンテナ、データに対する潜在的な脅威 AWS のサービス を検出します。 GuardDuty は、特定のコンプライアンスフレームワークで義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件への対応に役立ちます。
- AWS Audit Manager – これにより AWS のサービス 、 AWS 使用状況を継続的に監査し、リスクの管理方法と規制や業界標準への準拠を簡素化できます。

のインフラストラクチャセキュリティ AWS Batch

マネージドサービスである AWS Batch は、 AWS グローバルネットワークセキュリティで保護されています。 AWS セキュリティサービスと がインフラストラクチャ AWS を保護する方法については、AWS 「 クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには、「 Security Pillar AWS Well-Architected Framework」の「Infrastructure Protection」を参照してください。

が AWS 公開した API コールを使用して、ネットワーク AWS Batch 経由で にアクセスします。クライアントは以下をサポートする必要があります:

- Transport Layer Security (TLS)。TLS 1.2 は必須で TLS 1.3 がお勧めです。
- DHE (楕円ディフィー・ヘルマン鍵共有) や ECDHE (楕円曲線ディフィー・ヘルマン鍵共有) などの完全前方秘匿性 (PFS) による暗号スイート。これらのモードは、Java 7 以降など、ほとんどの最新システムでサポートされています。

また、リクエストには、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

これらの API オペレーションは任意のネットワークの場所から呼び出すことができますが、 AWS Batch はリソースベースのアクセスポリシーをサポートしています。このポリシーには、送信元 IP アドレスに基づく制限を含めることができます。 AWS Batch ポリシーを使用して、特定の Amazon Virtual Private Cloud (Amazon VPC) エンドポイントまたは特定の VPCsからのアクセスを制御することもできます。これにより、実質的にネットワーク内の特定の VPC からのみ、特定の AWS Batch リソースへの AWS ネットワークアクセスが分離されます。

AWS Batch リソースのタグ付け

AWS Batch リソースを管理しやすくするために、タグ形式で各リソースに独自のメタデータを割り当てることができます。このトピックでは、タグとその作成方法について説明します。

目次

タグの基本

タグとは、AWS リソースに付けるラベルです。タグはそれぞれ、1 つのキーとオプションの 1 つの値で構成されており、どちらもお客様側が定義します。

タグを使用すると、 AWS リソースを目的、所有者、環境などで分類できます。同じ型のリソースが多い場合に、割り当てたタグに基づいて特定のリソースをすばやく識別できます。たとえば、AWS Batch サービスに一連のタグを定義して、各サービスの所有者とスタックレベルを追跡できます。リソースタイプごとに一貫した一連のタグキーを考案することをお勧めします。

タグは自動的にはリソースに割り当てられません。タグを追加したら、いつでもタグキーと値は編集でき、タグはリソースからいつでも削除できます。リソースを削除すると、リソースのタグも削除されます。

タグには、AWS Batch に関連する意味はなく、完全に文字列として解釈されます。タグの値を空の文字列に設定することはできますが、タグの値を null に設定することはできません。特定のリソースについて既存のタグと同じキーを持つタグを追加した場合、以前の値は新しい値によって上書きされます。

AWS Management Console、AWS CLI、および AWS Batch API を使用してタグを操作できます。

AWS Identity and Access Management (IAM) を使用している場合は、タグを作成、編集、削除する許可を持つ AWS アカウントのユーザーを制御できます。

リソースのタグ付け

新規または既存のAWS Batch コンピューティング環境、ジョブ、ジョブ定義、ジョブキュー、スケジューリングポリシーのタグをつけることができます。

AWS Batch コンソールを使用している場合、新規リソースには作成時にタグを適用でき、既存のリソースには関連するリソースページの Tags] (タグ) タブを使用していつでもタグを適用できます。

AWS Batch API、AWS CLI、または AWS SDK を使用している場合、関連する API アクションの `tags` パラメータを使用して新規リソースにタグを適用でき、`TagResource` API アクションを使用して既存のリソースにタグを適用できます。詳細については、TagResourceを参照してください。

リソース作成アクションによっては、リソースの作成時にリソースのタグを指定できます。リソースの作成時にタグを適用できない場合、リソースの作成プロセスは失敗します。これにより、作成時にタグ付けするリソースが、指定したタグで作成されるか、まったく作成されないことが確認されます。作成時にリソースにタグを付ける場合、リソースの作成後にカスタムのタグ付けスクリプトを実行する必要はありません。

次の表では、タグ付け可能な AWS Batch リソースと、作成時にタグ付け可能なリソースについて説明します。

AWS Batch リソースのタグ付けのサポート

リソース	タグをサポート	タグの伝播をサポート	作成時のタグ付けをサポート (AWS Batch API、AWS CLI、AWS SDK)
AWS Batch コンピューティング環境	Yes	いいえ。 コンピューティング環境タグは、他のリソースには伝播されません。リソースのタグは、CreateComputeEnvironment API オペレーションで渡される computeResources オブジェクト	Yes

リソース	タグをサポート	タグの伝播をサポート	作成時のタグ付けをサポート (AWS Batch API、AWS CLI、AWS SDK)
		の tags メンバーで指定されます。	
AWS Batch ジョブ	はい	Yes	Yes
AWS Batchジョブ定義	Yes	いいえ	Yes
AWS Batchジョブキュー	Yes	いいえ	Yes
AWS Batchスケジューリングポリシー	Yes	いいえ	Yes

タグの制限

タグには以下のような基本制限があります。

- リソースあたりのタグの最大数 - 50 件
- タグキーは、リソースごとにそれぞれ一意である必要があります。また、各タグキーに設定できる値は 1 つのみです。
- キーの最大長 - UTF-8 の 128 Unicode 文字
- 値の最大長 - UTF-8 の 256 Unicode 文字
- 複数の AWS サービス間およびリソース間でタグ付けスキーマを使用する場合、他のサービスでも許可される文字に制限が適用されることがあるのでご注意ください。一般的に使用が許可される文字は、UTF-8 で表現できる文字、数字、スペース、および +、-、=、.、_、:、/、@。
- タグのキーと値は大文字と小文字が区別されます。
- `aws:`、`AWS:`、またはその大文字または小文字の組み合わせを、キーまたは値のプレフィックスとして使用しないでください。これらの文字列は AWS による使用のために予約されています。このプレフィックスが含まれるタグのキーや値を編集したり削除することはできません。このプレフィックスを持つタグは、リソースあたりのタグ数の制限時には計算されません。

コンソールでのタグの処理

AWS Batch コンソールを使って、新規または既存のコンピューティング環境、ジョブ、ジョブ定義、ジョブキューに関連するタグを管理することができます。

作成時に個々のリソースにタグを追加する

タグは、AWS Batchコンピューティング環境、ジョブ、ジョブ定義、ジョブキュー、およびスケジューリングポリシーの作成時に追加することができます。

個々のリソースでのタグの追加と削除

AWS Batch では、クラスターに関連付けられたタグをリソースのページから直接追加または削除できます。

個々のリソースのタグを追加または削除するには

1. https://console.aws.amazon.com/batch/ で AWS Batch コンソールを開きます。
2. ナビゲーションバーから、使用するリージョンを選択します。
3. ナビゲーションペインでリソースタイプ (例: ジョブキュー) を選択します。
4. 特定のリソースを選択し、Edit tags] (タグの編集) を選択します。
5. 必要に応じてタグを追加または削除します。
   - タグを追加するには — リストの末尾にある空のテキストボックスにキーと値を指定します。
   - タグを削除するには — タグの横にある

     Delete icon
     ボタンを選択します。
6. 追加または削除するタグごとにこのプロセスを繰り返し、Edit tags] (タグの編集) を選択して終了します。

CLI または API でのタグの操作

リソースのタグの追加、更新、リスト表示、および削除には、次の AWS CLI コマンドまたは AWS Batch API オペレーションを使用します。

AWS Batch リソースのタグ付けのサポート

タスク	API アクション	AWS CLI	AWS Tools for Windows PowerShell
1 つ以上のタグを追加、または上書きします。	TagResource	タグリソース	追加-BATResourceTag
1 つ以上のタグを削除します。	UntagResource	タグなしリソース	削除-BATResourceTag
リソースのタグを一覧表示します	ListTagsForResource	list-tags-for-resource	Get-BATResourceTag

以下の例では、AWS CLIを使用して、リソースに対してタグ付けまたはタグ削除する方法を示しています。

例 1: 既存のリソースにタグ付けする

次のコマンドでは、既存のリソースにタグ付けします。

```
aws batch tag-resource --resource-arn resource_ARN --tags team=devs
```

例 2: 既存のリソースからタグを削除する

次のコマンドでは、既存のリソースからタグを削除します。

```
aws batch untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

例 3: リソースのタグのリスト取得

次のコマンドは、既存のリソースに関連付けられているタグのリストを取得します。

```
aws batch list-tags-for-resource --resource-arn resource_ARN
```

一部のリソース作成アクションでは、リソースの作成時にタグを指定できます。以下のアクションでは、作成時のタグ付けがサポートされます。

タスク	API アクション	AWS CLI	AWS Tools for Windows PowerShell
コンピューティング環境の作成	CreateComputeEnvironment	create-compute-environment	New-BATComputeEnvironment
ジョブキューの作成	CreateJobQueue	create-job-queue	New-BATJobQueue
スケジューリングポリシーの作成	CreateSchedulingPolicy	create-scheduling-policy	New-BATSchedulingPolicy
ジョブ定義を登録する	RegisterJobDefinition	register-job-definition	Register-BATJobDefinition
ジョブを送信する	SubmitJob	submit-job	Submit-BATJob

AWS Batch サービスクォータ

次の表に、変更できない AWS Batch のサービスクォータを示します。各クォータはリージョンごとにあります。

リソース	クォータ
ジョブキューの最大数。詳細については、ジョブキューを参照してください。	50
Amazon ECS と Amazon EKS 全体のコンピューティング環境の最大数。詳細については、コンピューティング環境を参照してください。	50
Amazon EKS クラスターごとのコンピューティング環境の最大数。	5
ジョブキューあたりのコンピューティング環境の最大数	3
ジョブの依存関係の最大数	20
ジョブ定義の最大サイズ (`RegisterJobDefinition` API オペレーションの場合)	24 KiB
ジョブの最大ペイロードサイズ (`SubmitJob` API オペレーションの場合)	30 KiB
配列ジョブの最大配列サイズ	10000
`SUBMITTED` 状態のジョブの最大数	1000000
`SubmitJob` オペレーションに必要な各アカウントの最大秒間トランザクション数 (TPS)	50

AWS Batch の使用方法によっては、追加のクォータが適用される場合があります。Amazon EC2 クォータについて知るには、AWS 全般のリファレンスの Amazon EC2 Service Quotasを参照してください。Amazon ECS クォータの詳細については、AWS 全般のリファレンスのAmazon ECS Service Quotasを参照してください。Amazon EKS クォータの詳細については、AWS 全般のリファレンスのAmazon EKS Service Quotas を参照してください。

トラブルシューティング AWS Batch

コンピューティング環境、ジョブキュー、ジョブ定義、またはジョブに関する問題のトラブルシューティングが必要となる場合があります。この章では、 AWS Batch 環境でこのような問題をトラブルシューティングして解決する方法について説明します。

AWS Batch は IAM ポリシー、ロール、アクセス許可を使用し、Amazon EC2、Amazon ECS AWS Fargate、および Amazon Elastic Kubernetes Service インフラストラクチャで実行されます。これらのサービスに関連する問題のトラブルシューティングを行うには、以下を参照してください:

- IAM ユーザーガイドのIAM のトラブルシューティング
- Amazon Elastic Container Service 開発者ガイドのAmazon ECS のトラブルシューティング
- Amazon EKS ユーザーガイドのAmazon EKS のトラブルシューティング
- Amazon EC2 ユーザーガイドの「EC2 インスタンスのトラブルシューティングAmazon EC2」

目次

    - インスタンスプロファイルが存在しません
    - 無効な Kubernetes 名前空間が
    - 削除されたコンピューティング環境
    - ノードは Amazon EKS クラスターに加わりません
- AWS Batch Amazon EKS ジョブの が RUNNABLEステータスのまま
- aws-auth ConfigMapのフィールドが正しく設定されていることを確認します
- RBAC の権限またはバインディングが適切に設定されていない

AWS Batch

INVALIDコンピューティング環境

マネージド型コンピューティング環境を誤って設定した可能性があります。設定した場合、コンピューティング環境は INVALID の状態になり、配置するジョブを受け入れられなくなります。以下のセクションでは、考えられる原因と、その原因に基づいトラブルシューティング方法について説明します。

正しくないロール名または ARN

コンピューティング環境が INVALID状態になる最も一般的な原因は、 AWS Batch サービスロールまたは Amazon EC2 スポットフリートロールの名前または Amazon リソースネーム (ARN) が正しくないことです。これは、 AWS CLI または AWS SDKs。でコンピューティング環境を作成する AWS Batch と AWS Management Console、正しいサービスまたはスポットフリートロールを選択できます。ただし、名前または ARN を手動で入力し、またはそれを間違って入力したとします。そうすると、生成されるコンピューティング環境も INVALID になります。

しかし、IAM リソースの名前または ARN を AWS CLI コマンドまたは SDK コードに手動で入力すると仮定します。この場合、 AWS Batch は文字列を検証できません。代わりに、 AWS Batch は不正な値を受け入れて環境作成の試みをする必要があります。が環境の作成に AWS Batch 失敗すると、環境は INVALID状態に移行し、次のエラーが表示されます。

無効なサービスロールの場合:

```
CLIENT_ERROR - Not authorized to perform sts:AssumeRole (Service:
AWSSecurityTokenService; Status Code: 403; Error Code: AccessDenied;
Request ID: dc0e2d28-2e99-11e7-b372-7fcc6fb65fe7)
```

無効なスポットフリートロールの場合:

```
CLIENT_ERROR - Parameter: SpotFleetRequestConfig.IamFleetRole
is invalid. (Service: AmazonEC2; Status Code: 400; Error Code:
InvalidSpotFleetRequestConfig; Request ID: 331205f0-5ae3-4cea-
bac4-897769639f8d) Parameter: SpotFleetRequestConfig.IamFleetRole is
invalid
```

この問題の一般的な原因の 1 つには、次のシナリオがあります。 AWS CLI または AWS SDKs を使用する場合にのみ、完全な Amazon リソースネーム (ARN) ではなく、IAM ロールの名前を指定します。ロールを作成した方法によって、ARN に aws-service-role パスプレフィックスが含まれることがあるからです。たとえば、 AWS Batch サービスロールを、のサービスにリンクされたロールの使用 AWS Batchの手順を使用して手動で作成すると、サービスロール ARN は次のようになります。

```
arn:aws:iam::123456789012:role/AWSBatchServiceRole
```

ただし、コンソールの最初の実行ウィザードの一環としてサービスロールを作成した場合、サービスロール ARN は次のようになります。

```
arn:aws:iam::123456789012:role/aws-service-role/AWSBatchServiceRole
```

この問題は、 AWS Batch サービスレベルポリシー (AWSBatchServiceRole) を非サービスロールにアタッチする場合にも発生する可能性があります。たとえば、このシナリオでは、次のような エラーメッセージが表示される場合があります:

```
CLIENT_ERROR - User: arn:aws:sts::account_number:assumed-role/batch-replacement-role/
aws-batch is not
     authorized to perform: action on resource ...
```

問題を解決するには、次のいずれかを実行します。

- AWS Batch コンピューティング環境を作成するときは、サービスロールに空の文字列を使用します。
- 以下の形式でサービスロールを指定しますarn:aws:iam::*account_number*:role/aws-service-role/batch.amazonaws.com/AWSServiceRoleForBatch。

AWS CLI または AWS SDKs、 は ARN がaws-service-roleパスプレフィックスを使用しないことを AWS Batch 前提としています。このため、コンピューティング環境を作成するときには、IAM ロールに完全 ARN を指定することが推奨されます。

この設定の誤りがあるコンピューティング環境を修復するには、INVALID コンピューティング環境を修復するを参照してください。

INVALID コンピューティング環境を修復する

コンピューティング環境が INVALIDの状態にある場合、無効なパラメータを修復して更新する必要があります。正しくないロール名または ARN については、正しいサービスロールを使ってコンピューティング環境を更新できます。

誤って設定されたコンピューティング環境を修復するには

1. https://console.aws.amazon.com/batch/ で AWS Batch コンソールを開きます。
2. ナビゲーションバーから、 AWS リージョン 使用する を選択します。
3. ナビゲーションペインで、Compute environments] (コンピューティング環境) を選択します。
4. Compute environments] (コンピューティング環境) ページで、編集するコンピューティング環境の横にあるラジオボタンを選択し、Edit] (編集) を選択します。
5. Update compute environment] (コンピューティング環境の更新) ページの Service role] (サービスロール) で、コンピューティング環境に使用する IAM ロールを選択します。 AWS Batch コンソールには、コンピューティング環境と正しい信頼関係があるロールのみが表示されます。
6. Save] (保存) を選択して、コンピューティング環境を更新します。

RUNNABLE 状態でジョブが止まる

コンピューティング環境にコンピューティングリソースがあるのに、ジョブがRUNNABLE の状態よりも先に進まないとします。次に、何らかの原因によってジョブがコンピューティングリソースに配置され、ジョブキューがブロックされている可能性があります。ジョブがターンを待っているか、停止してキューをブロックしているかを確認する方法は次のとおりです。

がRUNNABLEジョブが先頭にあり、キューをブロックしていること AWS Batch を検出すると、Amazon CloudWatch Events からブロックされたジョブキューイベントが理由とともに送信されます。同じ理由も、 ListJobsおよび DescribeJobs API コールの一部としてstatusReasonフィールドに更新されます。

必要に応じて、 CreateJobQueueおよび UpdateJobQueue API アクションを使用して jobStateTimeLimitActionsパラメータを設定できます。

> **Note**
>
> 現在、 で使用できる唯一のアクションjobStateLimitActions.actionはジョブをキャンセルすることです。

jobStateTimeLimitActions パラメータは、特定の状態のジョブに対して が AWS Batch 実行する一連のアクションを指定するために使用されます。maxTimeSeconds フィールドを使用して、時間しきい値を秒単位で設定できます。

ジョブが が定義されたRUNNABLE状態になるとstatusReason、 maxTimeSecondsは経過後に指定されたアクションを実行 AWS Batch します。

例えば、 jobStateTimeLimitActionsパラメータを設定して、十分な容量が利用可能になるまで待機している RUNNABLE 状態のジョブを最大 4 時間待つことができます。これを行うには、ジョブをキャンセルCAPACITY:INSUFFICIENT_INSTANCE_CAPACITYしstatusReason、次のジョブがジョブキューの先頭maxTimeSecondsに進むのを許可する前に、 を に設定し、 を 144000 に設定します。

ジョブキューがブロックされていることを検出すると AWS Batch 、 が に提供する理由は、次のとおりです。このリストは、 ListJobsおよび DescribeJobs API アクションから返されるメッセージを提供します。これらは、 jobStateLimitActions.statusReasonパラメータに定義できる値と同じです。

1. 理由： 接続されているすべてのコンピューティング環境で容量不足エラーが発生しています。リクエストされると、容量不足エラーが発生した Amazon EC2 インスタンス AWS Batch が検出されます。ジョブを手動でキャンセルするか、 に jobStateTimeLimitActionsパラメータを設定することでstatusReason、後続のジョブをキューの先頭に移動できます。
   - **statusReason** ジョブがスタックしている間の メッセージ： `CAPACITY:INSUFFICIENT_INSTANCE_CAPACITY - Service cannot fulfill the capacity requested for instance type [instanceTypeName]`
   - **reason** に使用される **jobStateTimeLimitActions**： `CAPACITY:INSUFFICIENT_INSTANCE_CAPACITY`

- **statusReason** ジョブがキャンセルされた後の メッセージ：
  `Canceled by JobStateTimeLimit action due to reason: CAPACITY:INSUFFICIENT_INSTANCE_CAPACITY`

  [Note:] (メモ:)

  a. AWS Batch サービスロールには、この検出が機能するため のautoscaling:DescribeScalingActivitiesアクセス許可が必要です。AWSServiceRoleForBatch サービスにリンクされたロール (SLR) または AWSBatchServiceRolePolicyマネージドポリシーを使用する場合、アクセス許可ポリシーが更新されるため、何もする必要はありません。

  b. SLR または 管理ポリシーを使用する場合は、autoscaling:DescribeScalingActivitiesでブロックされたジョブキューイベントと更新されたジョブステータスを受信できるように、 および アクセスec2:DescribeSpotFleetRequestHistory許可を追加する必要がありますRUNNABLE。さらに、 では、ジョブキューで設定されていても、 jobStateTimeLimitActionsパラメータを使用してcancellationアクションを実行するためにこれらのアクセス許可 AWS Batch が必要です。

  c. マルチノード並列 (MNP) ジョブの場合、アタッチされた優先度の高い Amazon EC2 コンピューティング環境でinsufficient capacityエラーが発生すると、優先度の低いコンピューティング環境でこのエラーが発生した場合でもキューがブロックされます。

2. 理由： すべてのコンピューティング環境には、ジョブ要件よりも小さいmaxvCpusパラメータがあります。ジョブを手動でキャンセルするか、 に jobStateTimeLimitActionsパラメータを設定することでstatusReason、後続のジョブをキューの先頭に移動できます。オプションで、ブロックされたジョブのニーズを満たすために、プライマリコンピューティング環境のmaxvCpusパラメータを増やすことができます。

   - **statusReason** ジョブがスタックしている間の メッセージ：
     `MISCONFIGURATION:COMPUTE_ENVIRONMENT_MAX_RESOURCE - CE(s) associated with the job queue cannot meet the CPU requirement of the job.`
   - **reason** に使用される **jobStateTimeLimitActions**：
     `MISCONFIGURATION:COMPUTE_ENVIRONMENT_MAX_RESOURCE`
   - **statusReason** ジョブがキャンセルされた後の メッセージ：
     `Canceled by JobStateTimeLimit action due to reason: MISCONFIGURATION:COMPUTE_ENVIRONMENT_MAX_RESOURCE`

3. 理由： どのコンピューティング環境にも、ジョブ要件を満たすインスタンスがありません。ジョブがリソースをリクエストすると、 はアタッチされたコンピューティング環境が受信

ジョブに対応できないことを AWS Batch 検出します。ジョブを手動でキャンセルするか、 に jobStateTimeLimitActionsパラメータを設定することでstatusReason、後続のジョブをキューの先頭に移動できます。オプションで、コンピューティング環境で許可されているインスタンスタイプを再定義して、必要なジョブリソースを追加できます。

- **statusReason** ジョブがスタックしている間の メッセージ : `MISCONFIGURATION:JOB_RESOURCE_REQUIREMENT - The job resource requirement (vCPU/memory/GPU) is higher than that can be met by the CE(s) attached to the job queue.`
- **reason** に使用される **jobStateTimeLimitActions** : `MISCONFIGURATION:JOB_RESOURCE_REQUIREMENT`
- **statusReason** ジョブがキャンセルされた後の メッセージ : `Canceled by JobStateTimeLimit action due to reason: MISCONFIGURATION:JOB_RESOURCE_REQUIREMENT`

4. 理由： すべてのコンピューティング環境にサービスロールの問題があります。これを解決するには、サービスロールのアクセス許可をAWS Batch マネージドサービスロールのアクセス許可と比較し、ギャップがあれば対処します。

   同様のエラーを避けるために、AWS Batch コンピューティング環境に SLR を使用するのがベストプラクティスです。

   ジョブを手動でキャンセルするか、 に jobStateTimeLimitActionsパラメータを設定することでstatusReason、後続のジョブをキューの先頭に移動できます。サービスロールの問題を解決しないと、次のジョブもブロックされる可能性があります (複数可) 。この問題については、手動で調査して解決することをお勧めします。

   - **statusReason** ジョブがスタックしている間の メッセージ : `MISCONFIGURATION:SERVICE_ROLE_PERMISSIONS – Batch service role has a permission issue.`
   - **reason** に使用される **jobStateTimeLimitActions** : `MISCONFIGURATION:SERVICE_ROLE_PERMISSIONS`
   - **statusReason** ジョブがキャンセルされた後の メッセージ : `Canceled by JobStateTimeLimit action due to reason: MISCONFIGURATION:SERVICE_ROLE_PERMISSIONS`

5. 理由： すべてのコンピューティング環境が無効です。詳細については、「INVALIDコンピューティング環境」を参照してください。注: このエラーを解決するために、

jobStateTimeLimitActionsパラメータを使用してプログラム可能なアクションを設定することはできません。

- **statusReason** ジョブがスタックしている間の メッセージ： `ACTION_REQUIRED - CE(s) associated with the job queue are invalid.`

6. 理由： AWS Batch がブロックされたキューを検出しましたが、理由を特定できません。注: このエラーを解決するために、 jobStateTimeLimitActionsパラメータを使用してプログラム可能なアクションを設定することはできません。トラブルシューティングの詳細については、 re:Post の「 で RUNNABLE で AWS Batch ジョブが停止する AWS理由」を参照してください。

   - **statusReason** ジョブがスタックしている間の メッセージ： `UNDETERMINED - Batch job is blocked, root cause is undetermined.`

CloudWatch イベントからイベントを受信しなかった場合や、不明な理由イベントを受信した場合、この問題の一般的な原因をいくつか次に示します。

awslogsログドライバーがコンピューティングリソースに設定されていない

AWS Batch ジョブはログ情報を Logs CloudWatch に送信します。この機能を有効にするには、awslogsのログドライバーを使用するようにコンピューティングリソースを設定する必要があります。コンピューティングリソース AMI を、Amazon ECSに最適化されたAMI (またはAmazon Linux) に基づいて作成すると仮定します。次に、このドライバーはデフォルトにより ecs-initのパッケージに登録されます。ここで、別のベース AMI を使用すると仮定します。別の基本 AMI を使用する場合、Amazon ECS コンテナエージェントが開始するときに、awslogs ログドライバーが、 ECS_AVAILABLE_LOGGING_DRIVERS 環境変数で使用可能なログドライバーとして指定されていることを検証する必要があります。詳細については、コンピューティングリソースの AMI 仕様 およびコンピューティングリソース AMI の作成を参照してください。

リソースの不足

コンピューティングリソースが割り当てることができるリソースを超えた CPU、またはメモリリソースがジョブ定義に指定されている場合、ジョブは配置されません。たとえば、ジョブが 4 GiB のメモリを指定し、コンピューティングリソースで使用できるのがそれ以下の場合を考えます。そうなると、そのジョブをそれらのコンピューティングリソースに配置できない場合があります。この場合、ジョブ定義に指定するメモリを減らすか、あるいは環境のコンピューティングリソースを追加する必要があります。一部のメモリは、Amazon ECS コンテナエージェントやその他の重要なシステムプロセス用に予約されています。詳細については、コンピューティングリソースメモリ管理を参照してください。

コンピューティングリソースへのインターネットアクセスがない

コンピューティングリソースには、Amazon ECS サービスエンドポイントと通信するために外部ネットワークアクセスが必要です。これは、インターフェイス VPC エンドポイントを介して、またはパブリック IP アドレスを持つコンピュートリソースを通じて可能になります。

インターフェイス VPC エンドポイントの詳細については、Amazon Elastic Container Service 開発者ガイドのAmazon ECS インターフェイス VPC エンドポイント (AWS PrivateLink)を参照してください。

インターフェイス VPC エンドポイントが設定されておらず、コンピューティングリソースがパブリック IP アドレスを持たない場合は、ネットワークアドレス変換 (NAT) を使用してこのアクセスを提供する必要があります。詳細については、Amazon VPC ユーザーガイドに含まれるNAT ゲートウェイを参照してください。詳細については、the section called “「VPC を作成する」”を参照してください。

Amazon EC2 インスタンスの制限に達しました

アカウントが で起動できる Amazon EC2 インスタンスの数 AWS リージョン は、EC2 インスタンスのクォータによって決まります。特定のインスタンスタイプには per-instance-typeクォータもあります。制限の引き上げをリクエストする方法など、アカウントの Amazon EC2 インスタンスクォータの詳細については、Amazon EC2 ユーザーガイド」の「Amazon EC2 サービスの制限」を参照してください。 Amazon EC2

Amazon ECS コンテナエージェントがインストールされていない

AWS Batch ジョブを実行するには、Amazon ECS コンテナエージェントは Amazon マシンイメージ (AMI) にインストールされる必要があります。Amazon ECS コンテナエージェントが Amazon ECSに最適化 AMI にデフォルトでインストールされます。詳細については、Amazon ECS コンテナエージェントの構成 を参照してくださいAmazon Elastic Container Service デベロッパーガイドにあります。

詳細については、 re:Post の AWS Batch 「ジョブが RUNNABLE ステータスのままになるのはなぜですか？」を参照してください。

作成時にタグが付けられていないスポットインスタンス

AWS Batch コンピューティングリソースのスポットインスタンスのタグ付けは、2017 年 10 月 25 日にサポートされています。以前には、Amazon EC2 スポットフリートロール用の推奨 IAM 管理ポリシー (`AmazonEC2SpotFleetRole`) には、起動時にスポットインスタンス

にタグを付けるアクセス権限が含まれていませんでした。新しい推奨の IAM 管理ポリシーは、AmazonEC2SpotFleetTaggingRole と呼ばれます。起動時のスポットインスタンスへのタグ付けをサポートします。

作成時にスポットインスタンスのタグ付けを修正するには、以下の手順に従って、現在推奨の IAM 管理ポリシーを Amazon EC2 スポットフリートロールに割り当てます。その方法で、そのロールで今後作成されるスポットインスタンスには、作成時にインスタンスタグを適用する権限が付与されます。

現在の IAM 管理ポリシーを Amazon EC2 スポットフリートロールに割り当てるには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. Roles] (ロール) を選択し、Amazon EC2 スポットフリートロールを選択します。
3. Attach policy] (ポリシーのアタッチ) を選択します。
4. AmazonEC2SpotFleetTaggingRole を選択し、ポリシーのアタッチ を選択します。
5. Amazon EC2 スポットフリートロールを再度選択し、前のポリシーを削除します。
6. AmazonEC2SpotFleetRole ポリシーの右側にある x を選択し、デタッチ を選択します。

スポットインスタンスがスケールダウンしない

AWS Batch は、2021 年 3 月 10 日に AWSServiceRoleForBatchサービスにリンクされたロールを導入しました。コンピューティング環境の `serviceRole` パラメータにロールが指定されていない場合、このサービスにリンクされたロールが、サービスロールとして使用されます。ただし、サービスにリンクされたロールが EC2 スポットコンピューティング環境で使用されているが、使用されるスポットロールに AmazonEC2SpotFleetTaggingRole 管理ポリシーが含まれていないとします。そうであれば、スポットインスタンスはスケールダウンしません。その結果、このオペレーションを実行する権限がありませんというエラーメッセージを受け取ります。以下の手順で、`spotIamFleetRole` パラメータで使用するスポットフリートロールを更新してください。詳細については、「IAM ユーザーガイド」の「サービスにリンクされたロールの使用」および AWS「 のサービスにアクセス許可を委任するロールの作成」を参照してください。

トピック

- のスポットフリートロールに AmazonEC2SpotFleetTaggingRole 管理ポリシーをアタッチする AWS Management Console
- を使用して AmazonEC2SpotFleetTaggingRole 管理ポリシーをスポットフリートロールにアタッチする AWS CLI

のスポットフリートロールに AmazonEC2SpotFleetTaggingRole 管理ポリシーをアタッチする AWS Management Console

現在の IAM 管理ポリシーを Amazon EC2 スポットフリートロールに割り当てるには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. Roles] (ロール) を選択し、Amazon EC2 スポットフリートロールを選択します。
3. Attach policy] (ポリシーのアタッチ) を選択します。
4. AmazonEC2SpotFleetTaggingRole を選択し、ポリシーのアタッチ を選択します。
5. Amazon EC2 スポットフリートロールを再度選択し、前のポリシーを削除します。
6. AmazonEC2SpotFleetRole ポリシーの右側にある x を選択し、デタッチ を選択します。

を使用して AmazonEC2SpotFleetTaggingRole 管理ポリシーをスポットフリートロールにアタッチする AWS CLI

コマンド例では、Amazon EC2 スポットフリートロールの名前が *AmazonEC2SpotFleetRole* であることを前提としています。ロールで別の名前が使用されている場合は、一致するようにコマンドを調整します。

AmazonEC2SpotFleetTaggingRole 管理ポリシーをスポットフリートロールにアタッチするには

1. AmazonEC2SpotFleetTaggingRole マネージド IAM ポリシーを *AmazonEC2SpotFleetRole* ロールにアタッチするには、 を使用して次のコマンドを実行します AWS CLI。

```
$ aws iam attach-role-policy \
    --policy-arn arn:aws:iam::aws:policy/service-role/AmazonEC2SpotFleetTaggingRole
 \
    --role-name AmazonEC2SpotFleetRole
```

2. AmazonEC2SpotFleetRole マネージド IAM ポリシーを *AmazonEC2SpotFleetRole* ロールからデタッチするには、 を使用して次のコマンドを実行します AWS CLI。

```
$ aws iam detach-role-policy \
    --policy-arn arn:aws:iam::aws:policy/service-role/AmazonEC2SpotFleetRole \
    --role-name AmazonEC2SpotFleetRole
```

Secrets Manager のシークレットを取得できない

バージョン 1.16.0-1 より以前の Amazon ECS エージェントで AMI を使用している場合は、この機能を使用するため、Amazon ECS エージェント設定変数 `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE=true` を使用する必要があります。そのインスタンスを作成するときに、新しいコンテナインスタンスの `./etc/ecs/ecs.config` ファイルに追加します。または、既存のインスタンスに追加することもできます。既存のインスタンスに追加する場合は、追加後に ECS Agent を再起動する必要があります。詳細については、Amazon Elastic Container Service デベロッパーガイドのAmazon ECS コンテナエージェントの構成を参照してください。

ジョブ定義リソース要件を上書きできない

に渡される containerOverrides 構造および `vcpus`メンバーで指定されたメモリ`memory`と vCPU のオーバーライドはSubmitJob、ジョブ定義の resourceRequirements 構造で指定されたメモリと vCPU の要件をオーバーライドできません。

これらのリソース要件を上書きしようとすると、次のエラーメッセージが表示されることがあります:

この値は非推奨のキーで送信され、ジョブ定義のリソース要件で提供される値と競合する可能性があります。

これを修正するには、containerOverrides] の resourceRequirements] メンバーにメモリと vCPU の要件を指定します。たとえば、メモリと vCPU の上書きが次の行で指定されているとします。

```
"containerOverrides": {
   "memory": 8192,
   "vcpus": 4
}
```

それらを次に変更します:

```
"containerOverrides": {
   "resourceRequirements": [
      {
         "type": "MEMORY",
         "value": "8192"
      },
      {
```

```
            "type": "VCPU",
            "value": "4"
        }
    ],
}
```

ジョブ定義の containerProperties] オブジェクトで指定されているメモリと vCPU の要件と同じ変更を行います。たとえば、メモリと vCPU の要件が次の行で指定されているとします。

```
{
    "containerProperties": {
        "memory": 4096,
        "vcpus": 2,
}
```

それらを次に変更します:

```
"containerProperties": {
    "resourceRequirements": [
        {
            "type": "MEMORY",
            "value": "4096"
        },
        {
            "type": "VCPU",
            "value": "2"
        }
    ],
}
```

desiredvCpus 設定を更新すると、エラーメッセージが表示されます

AWS Batch API を使用して目的の vCPUs (desiredvCpus) 設定を更新すると、次のエラーメッセージが表示されます。

```
Manually scaling down compute environment is not supported. Disconnecting job queues from compute environment will cause it to scale-down to minvCpus.
```

この問題は、desiredvCpus 更新された desiredvCpus 値が現在の値より小さい場合に発生します。desiredvCpus 値を更新するには、次のいずれかが当てはまる必要があります :

- desiredvCpus 値は minvCpus と maxvCpus 値の間になければなりません。
- 値は、desiredvCpusStart Frame (スタートフレーム) desiredvCpus 値と同等かそれ以上である必要があります。

AWS Batch Amazon EKS での

トピック

INVALIDコンピューティング環境

マネージド型コンピューティング環境を誤って設定した可能性があります。設定した場合、コンピューティング環境は INVALID の状態になり、配置するジョブを受け入れられなくなります。以下のセクションでは、考えられる原因と、その原因に基づいトラブルシューティング方法について説明します。

サポート対象外の Kubernetes バージョン

CreateComputeEnvironment API オペレーションまたは API オペレーション UpdateComputeEnvironment を使用してコンピュート環境を作成または更新すると、次のようなエラーメッセージが表示されることがあります。この問題は、Kubernetes でサポートされていないバージョンを EC2Configuration で指定した場合に発生します。

```
At least one imageKubernetesVersion in EC2Configuration is not supported.
```

この問題を解決するには、コンピュート環境を削除し、サポートされている Kubernetes バージョンで再作成してください。

Amazon EKS クラスターでマイナーバージョンアップグレードを実行できます。たとえば、イナーバージョンがサポートされていない場合でも、クラスターを 1.xx から 1.yy マにアップグレードできます。

ただし、INVALID メジャーバージョンを更新すると、コンピュート環境のステータスがに変わることがあります。たとえば、1.xx から 2.yy へのメジャーバージョンアップグレードを実行する場合

などです。メジャーバージョンが でサポートされていない場合は AWS Batch、次のようなエラーメッセージが表示されます。

```
reason=CLIENT_ERROR - ... EKS Cluster version [2.yy] is unsupported
```

この問題を解決するには、API オペレーションを使用してコンピューティング環境を作成または更新するときに、Kubernetes サポートされているバージョンを指定してください。

AWS Batch Amazon EKS の は現在、次のKubernetesバージョンをサポートしています。

- 1.29
- 1.28
- 1.27
- 1.26
- 1.25
- 1.24
- 1.23

インスタンスプロファイルが存在しません

指定されたインスタンスプロファイルが存在しない場合、Amazon EKS コンピューティング環境 AWS Batch のステータスは に変更されますINVALID。statusReason パラメータに次のようなエラーセットが表示されます。

```
CLIENT_ERROR - Instance profile arn:aws:iam::...:instance-profile/<name> does not exist
```

この問題を解決するには、ワーキングインスタンスプロファイルを指定または作成します。詳細については、Amazon EKS ユーザーガイドのAmazon EKS ノードの IAM ロールを参照してくださない。

無効な Kubernetes 名前空間が

Amazon EKS AWS Batch でコンピューティング環境の名前空間を検証できない場合、コンピューティング環境のステータスは に変更されますINVALID。たとえば、この問題は名前空間が存在しない場合に発生する可能性があります。

statusReason パラメータには、次のようなエラーメッセージが設定されています。

```
CLIENT_ERROR - Unable to validate Kubernetes Namespace
```

この問題は、次のいずれかに該当する場合に発生する可能性があります:

- `CreateComputeEnvironment` 呼び出し中のKubernetes 名前空間文字列は存在しません。詳細については、CreateCompute「環境」を参照してください。
- 名前空間の管理に、必要なロールベースのアクセス制御 (RBAC) 権限が正しく設定されていません。
- AWS Batch は Amazon EKS Kubernetes API サーバーエンドポイントにアクセスできません。

この問題を解決するには、`aws-auth ConfigMap`のフィールドが正しく設定されていることを確認しますをご参照ください。詳細については、Amazon EKS AWS Batch での の開始方法を参照してください。

削除されたコンピューティング環境

Amazon EKS コンピューティング環境でアタッチされている を削除する前に AWS Batch 、Amazon EKS クラスターを削除するとします。そうすると、コンピューティング環境のステータスは `INVALID` に変わります。このシナリオでは、Amazon EKS クラスターを同じ名前で再作成すると、コンピューティング環境は正しく機能しません。

この問題を解決するには、Amazon EKS コンピューティング環境で を削除してから再作成 AWS Batch します。

ノードは Amazon EKS クラスターに加わりません

AWS Batch Amazon EKS の は、すべてのノードが Amazon EKS クラスターに参加していないと判断した場合、コンピューティング環境をスケールダウンします。Amazon EKS AWS Batch でコンピューティング環境をスケールダウンすると、コンピューティング環境のステータスは に変更されます`INVALID`。

> **Note**
>
> AWS Batch は、問題をデバッグできるように、コンピューティング環境のステータスをすぐに変更しません。

`statusReason` パラメータには、次のようなエラーメッセージが設定されています。

```
Your compute environment has been INVALIDATED and scaled down because
none of the instances joined the underlying ECS Cluster. Common issues
```

```
preventing instances joining are the following: VPC/Subnet configuration
preventing communication to ECS, incorrect Instance Profile policy
preventing authorization to ECS, or customized AMI or LaunchTemplate
configurations affecting ECS agent.

Your compute environment has been INVALIDATED and scaled down because
none of the nodes joined the underlying Amazon EKS Cluster. Common issues
preventing nodes joining are the following: networking configuration
preventing communication to Amazon EKS Cluster, incorrect Amazon EKS
Instance Profile or Kubernetes RBAC policy preventing authorization
to Amazon EKS Cluster, customized AMI or LaunchTemplate configurations
affecting Amazon EKS/Kubernetes node bootstrap.
```

デフォルトの Amazon EKS AMI を使用する場合、この問題の最も一般的な原因は次のとおりです：

- インスタンスロールが正しく設定されていません。詳細については、Amazon EKS ユーザーガイドのAmazon EKS ノードの IAM ロールを参照してください。
- サブネットが正しく設定されていません。詳細については、Amazon EKS ユーザーガイドのAmazon EKS VPC とサブネットの要件と考慮事項 を参照してください。
- セキュリティグループが正しく設定されていません。詳細については、Amazon EKS ユーザーガイドのAmazon EKS セキュリティグループに関する考慮事項を参照してください。

> **Note**
>
> また、Personal Health Dashboard (PHD) にエラー通知が表示される場合があります。

AWS Batch Amazon EKS ジョブの が **RUNNABLE**ステータスのまま

マネージド型ノードグループを作成する場合、または `aws-auth` を使用してノードグループを作成する場合、`ConfigMap` `eksctl` が自動的に作成され、クラスターに適用されます。`aws-auth` `ConfigMap` は最初に、ノードがクラスターに参加できるようにするために作成されました。ただし、`aws-auth` `ConfigMap` を使用して、ユーザーとロールにロールベースのアクセス制御 (RBAC) アクセスを追加することもできます。

`aws-auth` `ConfigMap` が正しく設定されていることを確認するには：

1. マップされたロールを以下の `aws-auth` `ConfigMap` から取得します：

```
$ kubectl get configmap -n kube-system aws-auth -o yaml
```

2. roleARN が次のように設定されていることを確認します。

   ```
   rolearn: arn:aws:iam::aws_account_number:role/AWSServiceRoleForBatch
   ```

> **Note**
>
> また、Amazon EKS コントロールプレーンログを確認することもできます。詳細については、Amazon EKS ユーザーガイドの Amazon EKS クラスターコントロールプレーンのログを参照してください。

ジョブが RUNNABLEのステータスのままになる問題を解決するには、kubectl を使用してマニフェストを再適用することをお勧めします。詳細については、ステップ 1: 用の Amazon EKS クラスターを準備する AWS Batchを参照してください。または、kubectl を使用して aws-auth ConfigMap を手動で編集することもできます。詳細については、Amazon EKS ユーザーガイドのIAMユーザーとロールのAmazon EKS クラスターへのアクセスを有効にするを参照してください。

aws-auth ConfigMapのフィールドが正しく設定されていることを確認します

aws-auth ConfigMap が正しく設定されていることを確認するには：

1. aws-auth ConfigMap にマップされたロールを取得します。

   ```
   $ kubectl get configmap -n kube-system aws-auth -o yaml
   ```

2. roleARN が次のように設定されていることを確認します。

   ```
   rolearn: arn:aws:iam::aws_account_number:role/AWSServiceRoleForBatch
   ```

> **Note**
>
> パス aws-service-role/batch.amazonaws.com/が、サービスにリンクされたロールの ARN から削除されました。これは aws-auth 設定マップに問題があるためです。詳細については、パスを持つロールは、aws-authconfigmap の ARN にパスが含まれていると機能しない を参照してください。

> **Note**
>
> また、Amazon EKS コントロールプレーンログを確認することもできます。詳細については、Amazon EKS ユーザーガイドの Amazon EKS クラスターコントロールプレーンのログを参照してください。

ジョブが RUNNABLEのステータスのままになる問題を解決するには、kubectl を使用してマニフェストを再適用することをお勧めします。詳細については、ステップ 1: 用の Amazon EKS クラスターを準備する AWS Batchを参照してください。または、kubectl を使用して aws-auth ConfigMap を手動で編集することもできます。詳細については、Amazon EKS ユーザーガイドのIAMユーザーとロールのAmazon EKS クラスターへのアクセスを有効にするを参照してください。

RBAC の権限またはバインディングが適切に設定されていない

RBAC 権限またはバインディングの問題が発生した場合は、aws-batch Kubernetesのロールが Kubernetes 名前空間にアクセスできることを確認してください：

```
$ kubectl get namespace namespace --as=aws-batch
```

```
$ kubectl auth can-i get ns --as=aws-batch
```

kubectl describe コマンドを使用して、クラスターロールまたは Kubernetesの名前空間の権限を確認することもできます。

```
$ kubectl describe clusterrole aws-batch-cluster-role
```

以下は出力例です。

```
Name:         aws-batch-cluster-role
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources                                  Non-Resource URLs  Resource Names
 Verbs
  ---------                                  -----------------  --------------
 -----
```

```
  configmaps                                          []                  []
 [get list watch]
  nodes                                               []                  []
 [get list watch]
  pods                                                []                  []
 [get list watch]
  daemonsets.apps                                     []                  []
 [get list watch]
  deployments.apps                                    []                  []
 [get list watch]
  replicasets.apps                                    []                  []
 [get list watch]
  statefulsets.apps                                   []                  []
 [get list watch]
  clusterrolebindings.rbac.authorization.k8s.io  []                  []
 [get list]
  clusterroles.rbac.authorization.k8s.io         []                  []
 [get list]
  namespaces                                          []                  []
 [get]
```

```
$ kubectl describe role aws-batch-compute-environment-role -n my-aws-batch-namespace
```

以下は出力例です。

```
Name:         aws-batch-compute-environment-role
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources                                  Non-Resource URLs  Resource Names  Verbs
  ---------                                  -----------------  --------------  -----
  pods                                       []                 []              [create
 get list watch delete patch]
  serviceaccounts                            []                 []              [get list]
  rolebindings.rbac.authorization.k8s.io  []                 []              [get list]
  roles.rbac.authorization.k8s.io         []                 []              [get list]
```

この問題を解決するには、RBAC 権限と `rolebinding` コマンドを再度適用します。詳細については、ステップ 1: 用の Amazon EKS クラスターを準備する AWS Batchを参照してください。

AWS Batch のベストプラクティス

AWS Batch を使用して、複雑なアーキテクチャを管理する必要がなく、要求の厳しいさまざまな計算ワークロードを大規模に実行できます。AWS Batch ジョブは、疫学、ゲーミング、機械学習などの幅広いユースケースで使用が可能です。

このトピックでは、AWS Batchの使用時に考慮すべきベストプラクティスと、AWS Batchの使用時にワークロードを実行および最適化する方法に関するガイダンスについて説明します。

トピック

AWS Batch を使用する場合

AWS Batchは、ジョブを大規模かつ低コストで実行し、キューイングサービスとコストが最適化されたスケーリングを提供します。ただし、すべてのワークロードが AWS Batch を使用して実行するのに適しているわけではありません。

- ショートジョブ — ジョブが数秒しか実行されない場合、バッチジョブをスケジュールするためのオーバーヘッドは、ジョブ自体の実行時間よりも長くかかる可能性があります。回避策として、binpackはタスクをまとめてから、それらをAWS Batchで送信します 。次にタスクを繰り返し処理するように、ユーザーのAWS Batchのジョブを設定します。たとえば、個々のタスク引数を、Amazon DynamoDB テーブルにステージするか、Amazon S3 バケット内のファイルとしてステージします。各ジョブが3～5分実行されるように、タスクをグループ化することを検討します。ジョブのbinpack の後、AWS Batch ジョブ内のタスクグループをループスルーします。
- すぐに実行する必要のあるジョブ — AWS Batchはジョブを迅速に処理できます。ただし、AWS Batch はスケジューラーであり、コストパフォーマンス、ジョブの優先度、スループットを最適化します。AWS Batch は、リクエストの処理に時間がかかる場合があります。数秒以内に応答が必

要な場合は、Amazon ECS または Amazon EKS を使用するサービスベースのアプローチの方がより適しています。

大規模な実行のチェックリスト

5万以上の vCPUs で、大きなワークロードを実行する前に、次のチェックリストを検討してください。

> **Note**
>
> 100 万台以上の vCPUs で大規模なワークロードを実行する予定がある場合、または大規模な実行に関するガイダンスが必要な場合は、AWSの担当チームにお問い合わせください。

- Amazon EC2 クォータの確認 — AWS Management Console の Service Quotas パネルで、Amazon EC2 のクォータ (制限ともいいます) を確認してください。必要に応じて、Amazon EC2 インスタンスのピーク数に合わせてクォータ引き上げをリクエストしてください。Amazon EC2 スポットインスタンスと Amazon オンデマンドインスタンスには、別々のクォータがあることを憶えていてください。詳細については、Service Quotas を開始させる を参照してください。
- リージョンごとに Amazon Elastic Block Store の割り当てを確認する — 各インスタンスはオペレーティングシステムに GP2 または GP3 ボリュームを使用します。デフォルトでは、各AWS リージョンのクォータは300 TiB です。ただし、各インスタンスはこのクォータの一部としてカウントを使用します。そのため、各リージョンの Amazon Elastic Block Store の割り当てを確認するときは、この点を必ず考慮に入れてください。クォータに達すると、それ以上インスタンスを作成することはできません。詳細については、 Amazon Elastic Block Store エンドポイントとクォータを参照してください。
- ストレージに Amazon S3 を使用する — Amazon S3 はハイスループットを提供し、各アベイラビリティーゾーンのジョブとインスタンスの数に基づいて、プロビジョニングするストレージの量を推測する必要がなくなります。詳細については、設計パターンのベストプラクティス: Simple Storage Service (Amazon S3) のパフォーマンスの最適化 を参照してください。
- 段階的にスケールしてボトルネックを早期に特定する — 100 万個以上の vCPUs で実行されるジョブでは、ボトルネックを早期に特定できるように、低レベルから始めて徐々に増やしていきます。たとえば、5万個のvCPUs で実行することから始めます。次にその数を20万個のvCPUsに、次に 50万個のvCPUs にと増やします。つまり、望ましい vCPU 数に達するまで vCPUs 数を徐々に増やし続けます。

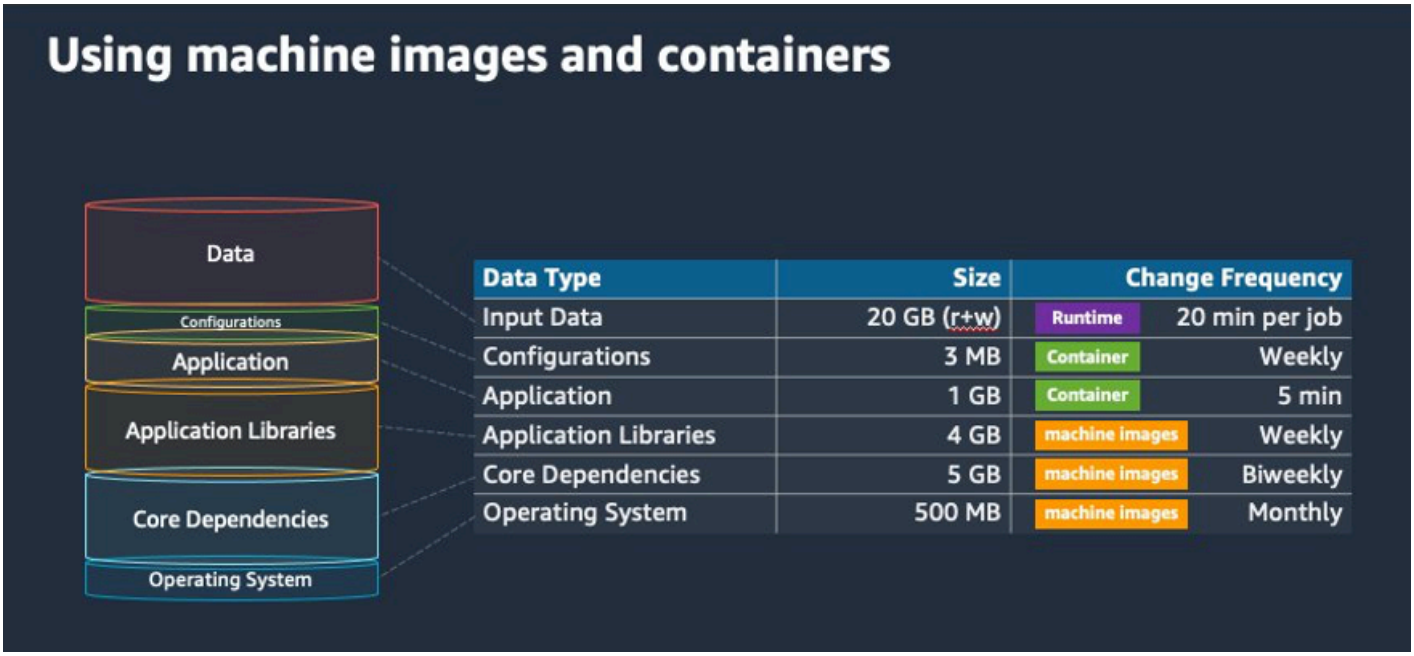
- 監視して潜在的な問題を早期に特定する — 大規模に実行する際に発生する可能性のある中断や問題を避けるために、アプリケーションとアーキテクチャの両方を必ず監視してください。仮 vCPUs を 1,000から5,000にスケーリングしても、中断が発生する可能性があります。Amazon CloudWatch Logs を使用して、ログデータを確認したり、クライアントライブラリを使い CloudWatch 埋め込みメトリックスを使用したりできます。詳細については、CloudWatch Logs エージェントリファレンス と`aws-embedded-metrics` を参照してください。

コンテナと AMI の最適化

最初に実行するジョブセットでは、コンテナのサイズと構造が重要です。コンテナが 4 GBを超える場合は、特に当てはまります。コンテナイメージは、レイヤーで構築されます。レイヤーは、3つの並行スレッドを使用して Docker により並行して取得されます。`max-concurrent-downloads` パラメータを使用して、同時にスレッド数を増やすことができます。詳細については、Dockerドキュメント を参照してください。

より大きなコンテナを使用することもできますが、起動時間を短縮するために、コンテナの構造とサイズを最適化することをお勧めします。

- コンテナが小さいほどフェッチが速い — コンテナが小さいほど、アプリケーションの起動時間の高速化につながります。コンテナのサイズを小さくするには、あまり更新されないライブラリやファイルをAmazon マシンイメージ (AMI) にオフロードします。またバインドマウントを使用して、コンテナへのアクセスを可能にすることもできます。詳細については、バインドマウントを参照してください。
- サイズが同じレイヤーを作成し、大きなレイヤーを分割する — 各レイヤーは 1つのスレッドで検索されます。ですから、大きなレイヤーは、ジョブの起動時間に大きく影響する可能性があります。コンテナのサイズを大きくすることと、起動時間を短縮することの望ましいトレードオフとして、レイヤーの最大サイズを 2 GB にすることをお勧めします。`docker history your_image_id` コマンドを実行して、コンテナイメージの構造とレイヤーサイズを確認できます。詳細については、Docker ドキュメントを参照してください。
- Amazon Elastic Container Registry をコンテナリポジトリとして使用する — 何千ものジョブをパラレル実行すると、自己管理型リポジトリに障害が発生したり、スロットルされたスループットが抑制されたりする可能性があります。Amazon ECR は大規模に動作し、最大 100 万個以上の vCPUs を使用するワークロードを処理できます。



適切なコンピューティング環境リソースを選択する

AWS Fargate Amazon EC2 よりも初期設定や設定が少なくて済み、特に、初めて使用する場合は使いやすいでしょう。Fargate を使用すると、サーバーを管理したり、容量計画に対処したり、セキュリティのためにコンテナワークロードを分離したりする必要がなくなります。

以下の要件がある場合は、Fargate インスタンスを使用することをお勧めします：

- ジョブは、すぐに、具体的には 30 秒未満で開始する必要があります。
- ジョブの要件は、16 vCPUs 以下、GPU なし、120 GiB 以下のメモリです。

詳細については、Fargateをいつ使うべきかを参照してください。

以下の要件がある場合は、Amazon EC2 インスタンスを使用することをお勧めします。

- インスタンスの選択をより細かく制御するか、または特定のインスタンスタイプを使用する必要があります。
- ジョブには、GPU、より多くのメモリ、カスタム AMI、Amazon Elastic Fabric Adapter など、AWS Fargate が提供できないリソースが必要です。
- ハイレベルのスループット、または同時実行性が必要です。
- AMI、Amazon EC2 起動テンプレート、または特別な Linux パラメータへのアクセスをカスタマイズする必要があります。

Amazon EC2 を使用すると、特定の要件に対してワークロードをより細かく調整し、必要に応じて大規模に実行できます。

Amazon EC2 オンデマンドまたはAmazon EC2 スポット

ほとんどの AWS Batch のお客様は、Amazon EC2 スポットインスタンスを使用しています。これは、オンデマンドインスタンスよりもコストが低いからです。ただし、ワークロードが数時間にわたって実行され、中断できない場合は、オンデマンドインスタンスの方が適している可能性があります。いつでも最初にスポットインスタンスを試して、必要に応じてオンデマンドに切り替えることができます。

以下の要件と期待がある場合は、Amazon EC2 オンデマンドインスタンスを使用してください：

- ジョブの実行時間が1時間以上の場合、ワークロードの中断は許されません。
- ワークロード全体に対して厳格な SLO (サービスレベル目標) が設定されており、計算時間を増やすことはできません。
- 必要なインスタンスは、おそらく中断される可能性があります。

以下の要件と期待がある場合には、Amazon EC2 スポットインスタンスを使用してください:

- ジョブの実行時間は、通常 30 分以下です。
- ワークロードの一部として、潜在的な中断やジョブの再スケジュールを許容できます。詳細については、スポットインスタンスアドバイザー を参照してください。
- 実行時間の長いジョブが中断された場合、チェックポイントから再開できます。

最初にスポットインスタンスで申請し、次にオンデマンドインスタンスをフォールバックオプションとして使用することで、両方の購入モデルを混在させることができます。たとえば、Amazon EC2 スポットインスタンスで実行されているコンピューティング環境に接続されているキューにジョブを送信します。ジョブが中断された場合は、Amazon EventBridge からのイベントをキャッチし、スポットインスタンスの再利用と関連付けます。次に、AWS Lambda 関数または AWS Step Functions を使用して、ジョブをオンデマンドキューに再送信します。詳細については チュートリアル:ジョブ停止イベントに Amazon シンプル 通知サービス アラートを送信、Amazon EC2 スポットインスタンスの中断を処理するためのベストプラクティス とStep Functions AWS Batch による管理 を参照してください。

> **Important**
>
> Amazon EC2 スポットインスタンスプールの可用性を維持し、中断率を下げるには、さまざまなインスタンスタイプ、サイズ、アベイラビリティーゾーンをオンデマンドコンピューティング環境のために使用します。

AWS Batchに関するAmazon EC2 スポット利用のベストプラクティスを利用する

Amazon Elastic Compute Cloud (EC2) スポットインスタンスを選択すると、ワークフローを最適化してコスト低減化（場合によっては大幅な節約実現）の可能性が高くなります。詳しくは、Amazon S3 のセキュリティベストプラクティス を参照してください。

ワークフローを最適化してコストを低減するには、以下の Amazon EC2 スポットベストプラクティスを検討してください AWS Batch：

- **SPOT_CAPACITY_OPTIMIZED** 割り当て戦略 を選択 — AWS Batch 最も深い Amazon EC2 スポット容量プールからAmazon EC2 インスタンスを選択します。中断が心配な場合は、この方法が適切な選択です。詳細については、配分戦略を参照してください。
- インスタンスタイプの多様化 — インスタンスタイプを多様化するには、互換性のあるサイズとファミリーを検討し、AWS Batch 価格や在庫状況に基づいて選択してください。たとえば、`c5.24xlarge`を、`c5.12xlarge`、`c5a`、`c5n`、`c5d`、`m5` 及び `m5d` のファミリーの代替として検討してください。詳細については、インスタンスタイプとアベイラビリティーゾーンへの柔軟な対応 を参照してください。
- ジョブの実行時間またはチェックポイントの短縮 — Amazon EC2 スポットインスタンスを使用する場合は、中断を避けるために、1時間以上かかるジョブを実行しないことをお勧めします。ジョブを30分以下のより小さな部分に分割したり、チェックポイントを設定したりすることで、中断の可能性を大幅に減らすことができます。
- 自動リトライを使用する — AWS Batchのジョブの中断を避けるには、ジョブには自動リトライを設定します。Batch ジョブは、ゼロ以外の終了コードが返された、サービスエラーが発生した、またはインスタンスが再利用されたなどの理由で中断される可能性があります。最大10個まで自動再試行を設定できます。先ず、少なくとも1～3回の自動再試行を設定することをお勧めします。Amazon EC2 スポット中断の追跡については、スポット中断ダッシュボードを参照してください。

AWS Batchについて、再試行パラメータを設定すると、ジョブはジョブキューの先頭に置かれるからです。つまり、ジョブが優先されます。ジョブ定義を作成したり、またはAWS CLI でジョブを送信したりするときに、再試行方法を設定できます。詳細については、ジョブの送信 を参照してください。

```
$ aws batch submit-job --job-name MyJob \
    --job-queue MyJQ \
    --job-definition MyJD \
    --retry-strategy attempts=2
```

- カスタムリトライを使用する — 特定のアプリケーション終了コードまたはインスタンスの再利用に合わせて、ジョブの再試行方法を設定できます。次の例では、ホストが原因で障害が発生した場合、ジョブは最大5回まで再試行できます。ただし別の理由によりジョブが失敗すると、ジョブは終了し、ステータスは FAILED に設定されます。

```
"retryStrategy": {
    "attempts": 5,
    "evaluateOnExit":
    [{
        "onStatusReason" :"Host EC2*",
        "action": "RETRY"
    },{
      "onReason" : "*"
        "action": "EXIT"
    }]
}
```

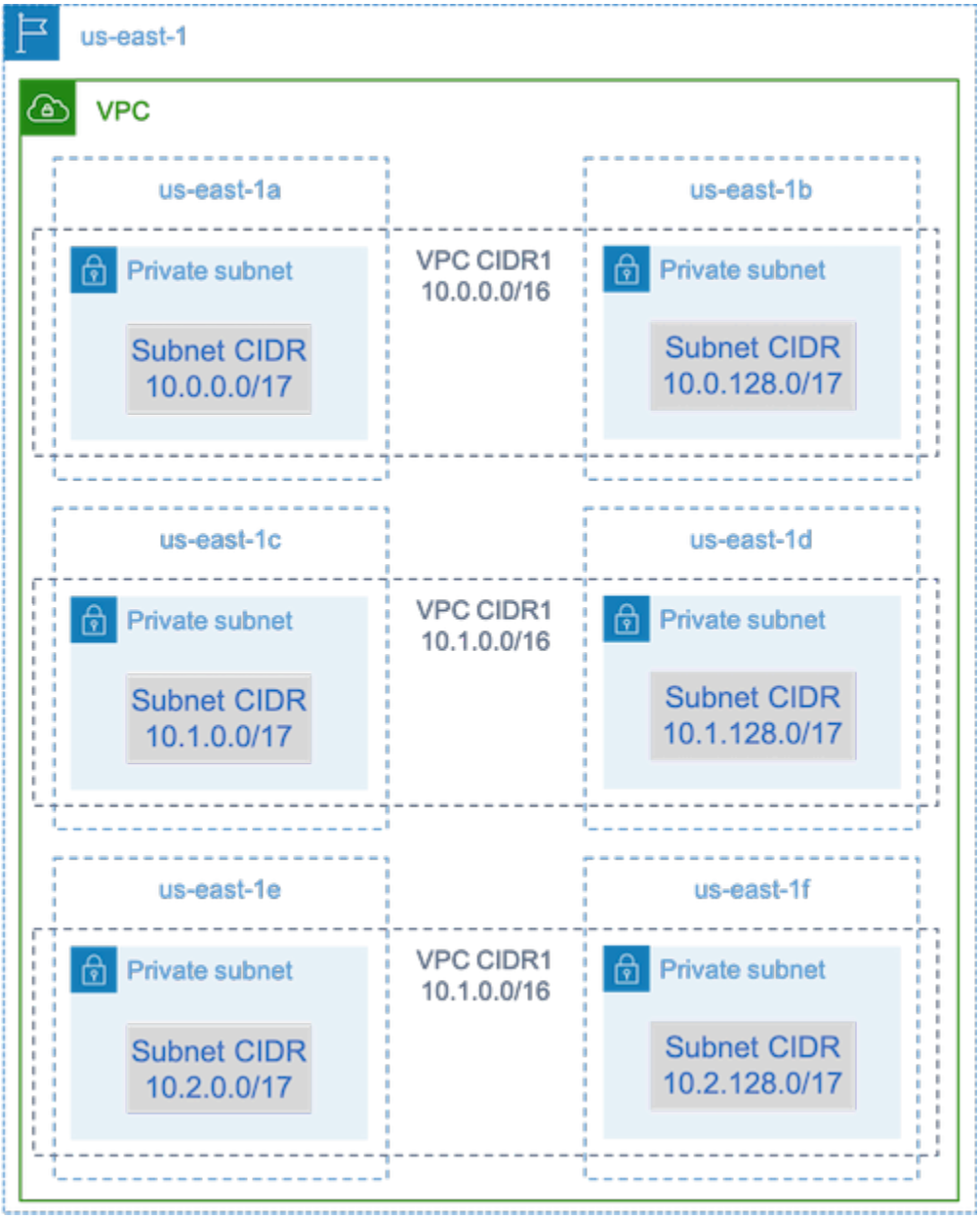
- スポット中断ダッシュボードを使用する — スポット中断ダッシュボードを使用して、スポット中断を追跡できます。このアプリケーションは、再利用された Amazon EC2 スポットインスタンスと、そのスポットインスタンスが属するアベイラビリティーゾーンに関するメトリクスを提供します。詳細については、スポット中断ダッシュボード を参照してください

一般的なエラーとトラブルシューティング

AWS Batchにおけるエラーは、多くの場合、アプリケーションレベルで発生したり、ユーザーの特定のジョブ要件を満たさないインスタンス構成が原因で発生したりします。その他の問題としては、ジョブがRUNNABLE のステータス内で停止したり、INVALID の状態でコンピューティング環境が停止したりすることが挙げられます。RUNNABLE のステータスのまま動かなくなるジョブのトラブルシューティングの詳細については、RUNNABLE 状態でジョブが止まる を参照してください。ある

INVALIDの状態にあるコンピューティング環境のトラブルシューティングについては、INVALIDコンピューティング環境 を参照してください。

- Amazon EC2 スポット vCPU クォータの確認 — 現在のService Quotasが、ジョブの要件を満たしていることを確認します。たとえば、現在のサービスクォータが 256 vCPUs で、ジョブに10,000 vCPUs が必要だとします。その場合、サービスクォータはジョブの要件を満たしていません。詳細とトラブルシューティングの手順の詳細については、Amazon EC2 Service Quotas と Amazon EC2 リソースのService Quotaを増やすには?を参照してください。
- アプリケーション実行前のジョブが失敗する — ジョブの一部には、DockerTimeoutErrorのエラーまたはCannotPullContainerErrorのエラーが原因で失敗するものがあります。トラブルシューティング情報については、AWS Batch の"DockerTimeoutError"エラーを解決する方法を参照してください。
- 不十分なIP アドレス — VPC とサブネットの IP アドレスの数によって、作成できるインスタンスの数が制限されることがあります。Classless Inter-Domain Routings (CIDR) を使用して、ワークロード実行に必要なIP アドレスよりも多くの IP アドレスを指定します。必要な場合、大きなアドレス空間を持つ専用VPCを構築することもできます。たとえば、10.x.0.0/16において複数の CIDR を持つ VPC を作成し、すべてのアベイラビリティーゾーンに 10.x.y.0/17の CIDR を使い サブネットを作成できます。この例では、xは1～4で、yは0または128です。この設定では、すべてのサブネットに36,000個のIP アドレスが割り当てられます。



- インスタンスが Amazon EC2 に登録されていることを確認する — Amazon EC2 コンソールにおいてインスタンスが表示されるが、Amazon ECS クラスターに Amazon Elastic Container Service コンテナインスタンスが何も表示されない場合は、Amazon ECS エージェントが Amazon マシンイメージ (AMI)にインストールされていない可能性があります。AMI内のAmazon ECS エージェント、AMI Amazon EC2 データ、または起動テンプレートも、正しく設定されていない可能性があります。根本原因を探し出すには、別のAmazon EC2 インスタンスを作成するか、または SSH を使用して既存のインスタンスに接続します。詳細については、Amazon ECS コンテナエージェントの設定、Amazon ECS ログファイルの場所、およびコンピューティングリソースの AMI を参照してください。

- AWSダッシュボードを確認する — AWSダッシュボードを確認して、予想されるジョブの状態とコンピューティング環境が予想どおりにスケーリングされていることを確認します。CloudWatch でジョブログを確認することもできます。
- インスタンスが作成されたことを確認する — インスタンスが作成されたら、コンピューティング環境が想定どおりにスケーリングされたことを意味します。インスタンスが作成されていない場合は、コンピューティング環境内の関連するサブネットを探して変更します。詳細については、Auto Scaling グループのスケーリングアクティビティを検証するを参照してください。

  また、インスタンスが関連するジョブ要件を満たしていることを確認するようお勧めします。たとえば、ジョブに 1 TiB のメモリが必要でも、コンピューティング環境では192 GB のメモリに制限されている C5 インスタンスタイプが使用されているとします。
- AWS Batch により、インスタンスがからリクエストされていることを確認する — Auto Scaling グループの履歴をチェックして、インスタンスが AWS Batch によってリクエストされていることを確認します。これは、Amazon EC2 がどのようにインスタンスを取得しようとしているかを示しています。Amazon EC2 スポットが特定のアベイラビリティーゾー、のインスタンスを取得できない、というエラーが表示される場合は、アベイラビリティーゾーンが特定のインスタンスファミリーを提供していないことが原因である可能性があります。
- インスタンスが Amazon ECS に登録されていることを確認する — Amazon EC2 コンソールにインスタンスが表示されるが、Amazon ECS クラスターに Amazon ECS コンテナインスタンスが表示されない場合は、Amazon ECS エージェントが Amazon マシンイメージ (AMI) にインストールされていない可能性があります。さらに、Amazon ECS エージェント、AMI の Amazon EC2 データ、または起動テンプレートが正しく設定されていない可能性があります。根本原因を探し出すには、別のAmazon EC2 インスタンスを作成するか、または SSH を使用して既存のインスタンスに接続します。詳細については、CloudWatch エージェント構成ファイル: ログセクション、Amazon ECS Log File Locations、およびコンピューティングリソースの AMIを参照してください。
- サポートチケットを開く — トラブルシューティングを行っても問題が解決せず、サポートプランがある場合は、サポートチケットを開いてください。サポートチケットには、問題、ワークロードの仕様、構成、およびテスト結果に関する情報を必ず含めてください。詳細については、AWS Support プランの比較を参照してください。
- AWS Batch および HPC フォーラムを確認する — 詳細については、AWS BatchおよびHPCフォーラムを参照してください。
- AWS Batch ランタイムモニタリングダッシュボードを確認する — このダッシュボードは、サーバーレスアーキテクチャを使用して Amazon ECS からのイベントをキャプチャし AWS Batch、そしてAmazon EC2 を使用してジョブとインスタンスに関する洞察を提供します。詳細については、AWS Batch ランタイム監視ダッシュボードソリューションを参照してください。

ドキュメント履歴

次の表は、 の初回リリース以降のドキュメントの重要な変更点をまとめたものです AWS Batch。また、お客様からいただいたフィードバックに対応するために、ドキュメントを頻繁に更新しています。

変更	説明	日付
AWS Batch サポートされている Amazon EKS バージョンの更新	が AWS Batch サポートする Amazon EKS バージョンを更新し、バージョン 1.22 を削除しました。	2024 年 3 月 11 日
AWS Batch サポートされている Amazon EKS バージョンの更新	が AWS Batch サポートする Amazon EKS バージョンを更新して、バージョン 1.29 を含めました。	2024 年 2 月 29 日
ジョブの再試行の自動化	コードサンプルを修正しました。	2024 年 2 月 29 日
のマルチコンテナジョブのサポートを追加 AWS Batch	Amazon Elastic Container Service、Amazon Elastic Kubernetes Service、および AWS Batch のマルチコンテナジョブのサポートを追加しました AWS Fargate。	2024 年 2 月 28 日
AWS Batch サポートされている Amazon EKS バージョンの更新	が AWS Batch サポートする Amazon EKS バージョンを更新し、バージョン 1.28 を含むようになりました。	2024 年 1 月 27 日
と を更新BatchServiceRolePolicyしました AWSBatchServiceRole	BatchServiceRolePolicy スポットフリートのリクエスト履歴と Amazon EC2	2023 年 12 月 5 日

	Auto Scaling アクティビティを記述するためのサポートを追加するために更新されました。 AWSBatchServiceRole ステートメント IDs、 `ec2:DescribeSpotFleetRequestHistory` および にアクセス AWS Batch 許可を付与するように更新されました`autoscaling:DescribeScalingActivities` 。	
AWS Batch Amazon EKS での	AWS Batch では、Amazon EKS クラスターでジョブを実行するためのサポートが追加されました。	2022 年 10 月 25 日
のサービス間の混乱した代理の防止 AWS Batch	AWS Batch は、エンティティ (サービスまたはアカウント) が別のエンティティによってアクションを実行するよう強制された場合に発生する混乱した代理のセキュリティ問題の回避策を提供するようになりました。	2022 年 6 月 6 日

インターフェイス VPC エンドポイント ()	によるインターフェイス VPC エンドポイントの設定のサポートが追加されました AWS PrivateLink。つまり、NAT インスタンス、VPN 接続、または を介したアクセスを必要と AWS Batch せずに、VPC と の間にプライベート接続を作成できます AWS Direct Connect。	2022 年 4 月 15 日
コンピューティング環境の更新機能の強化	AWS Batch コンピューティング環境に対する の拡張サポートの更新。	2022 年 4 月 14 日
AWS マネージドポリシーの更新 - 既存のポリシーの更新	AWS Batch が既存の マネージドポリシーを更新しました。	2021 年 12 月 6 日
公平配分のスケジューリング	AWS Batch は、ジョブキューにスケジューリングポリシーを追加するためのサポートを追加します。	2021 年 11 月 9 日
Amazon EFS	AWS Batch では、ジョブ定義に Amazon EFS ファイルシステムを追加するためのサポートが追加されました。	2021 年 4 月 1 日
サービスにリンクされたロールが追加されました	AWS Batch は、AWSServiceRoleForBatchサービスにリンクされたロールを追加します。	2021 年 3 月 10 日
AWS Fargate のサポート	AWS Batch では、Fargate リソースでジョブを実行するためのサポートが追加されました。	2020年12月3日

Amazon Linux 2 サポート	AWS Batch では、EC2 設定パラメータを使用してコンピューティング環境で Amazon Linux 2 AMIs を自動選択できるようになりました。	2020 年 11 月 24 日
強化された再試行戦略	AWS Batch は、ジョブの再試行戦略を強化します。これで、ジョブを再試行したり、ジョブの`ExitCode`、`Reason`、または`StatusReason` をパターンとマッチングし、それ以上の再試行を停止したりすることができます。	2020 年 10 月 20 日
リソースへのタグ付け	AWS Batch は、コンピューティング環境、ジョブ定義、ジョブキュー、ジョブにメタデータタグを追加するためのサポートを追加します。	2020 年 10 月 7 日
Secrets	AWS Batch では、ジョブにシークレットを渡すためのサポートが追加されました。	2020 年 10 月 1 日
ログ記録	AWS Batch では、ジョブに追加のログドライバーを指定するためのサポートが追加されました。	2020 年 10 月 1 日
配分戦略	AWS Batch では、インスタンスタイプを選択するための複数の戦略のサポートが追加されました。	2019 年 10 月 16 日

EFA のサポート	AWS Batch では、Elastic Fabric Adapter (EFA) デバイスのサポートが追加されました。	2019 年 8 月 2 日
GPU スケジューリング	AWS Batch は GPU スケジューリングを追加します。この機能を使用すると、各ジョブが必要とする GPU の数を指定できます。 AWS Batch は、それに応じてインスタンスをスケールアップします。	2019 年 4 月 4 日
マルチノードの並列ジョブ	AWS Batch では、マルチノードの並列ジョブのサポートが追加されました。この機能を使用すると、複数の Amazon EC2 インスタンスにまたがる単一のジョブを実行できます。	2018 年 11 月 19 日
リソースレベルのアクセス許可	AWS Batch では、複数の API オペレーションに対するリソースレベルのアクセス許可がサポートされています。	2018 年 11 月12 日
Amazon EC2 起動テンプレートのサポート	AWS Batch では、コンピューティング環境で起動テンプレートを使用するためのサポートが追加されました。	2018 年 11 月12 日

AWS Batch ジョブタイムアウト	AWS Batch では、ジョブタイムアウトのサポートが追加されました。このサポートにより、ジョブの特定のタイムアウト期間を設定して、ジョブがそれよりも長く実行された場合に がジョブ AWS Batch を終了するようにできます。	2018 年 4 月 5 日
AWS Batch EventBridge ターゲットとしての ジョブ	AWS Batch ジョブは EventBridge ターゲットとして利用可能になります。簡単なルールを作成することで、イベントをマッチングさせ、それに応じて AWS Batch ジョブを送信できます。	2018 年 3 月 1 日
CloudTrail の監査 AWS Batch	CloudTrail は、 AWS Batch API アクションに対して行われた呼び出しを監査できます。	2018 年 1 月 10 日
配列ジョブ	AWS Batch は配列ジョブのサポートを追加します。パラメータスイープおよび Monte Carlo ワークロードのパラメータに配列ジョブを使用できます。	2017 年 11 月 28 日
AWS Batch タグ付けの拡張	AWS Batch は、 タグ付け関数のサポートを拡張しました。この機能を使用して、マネージド型のコンピューティング環境内で起動された Amazon EC2 スポットインスタンスのタグを指定できます。	2017 年 10 月 26 日

AWS Batch の イベントストリーム EventBridge	AWS Batch は、 のイベントストリームを追加します EventBridge。 AWS Batch イベントストリームを使用して、ジョブキューに送信されたジョブの状態に関するほぼリアルタイムの通知を受け取ることができます。	2017 年 10 月 24 日
ジョブの再試行の自動化	AWS Batch では、ジョブの再試行のサポートが追加されました。この更新により、ジョブおよびジョブ定義に再試行戦略を適用し、ジョブが失敗した場合に自動的に再試行できます。	2017 年 3 月 28 日
AWS Batch 一般提供	AWS Batch は、 でバッチコンピューティングワークロードを実行するための手段として設計されています AWS クラウド。	2017 年 1 月 5 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。