AWS Batch

ユーザーガイド

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 はフルマネージド型のサービスであり、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。

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するリージョナルサービスです。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 Amazon EC2 インスタンスで実行されます。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 (p. 14)」を参照してください。

ジョブ定義

ジョブ定義では、ジョブの実行方法を指定します。これはジョブのリソースの設計図であると考えることができます。IAM ロールでジョブを指定して、他の AWS リソースへのプログラムによるアクセスを実行できます。メモリと CPU の要件の両方を指定します。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 (p. 32)」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。ジョブキューには 1 つまたは複数のコンピューティング環境を関連付けます。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。マネージドコンピューティング環境では、複数の詳細レベルで必要なインスタンスタイプを指定できます。コンピューティング環境は、特定のインスタンスタイプや特定のモデル (`c4.2xlarge` や `m4.10xlarge` など) を使用するように設定するか、単に最新のインスタンスタイプを使用するように指定できます。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 EC2 インスタンスの起動、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、お客様自身が 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ル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 (p. 56)」を参照してください。

開始方法

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、「AWS Batch の開始方法 (p. 9)」を参照してください。

AWS Batch でのセットアップ

すでにアマゾン ウェブ サービス (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、すぐに AWS Batch を使用し始めることができます。AWS Batch はコンピューティング環境で Amazon ECS コンテナインスタンスを使用するため、この 3 つのサービスのセットアッププロセスは非常によく似ています。AWS CLI と AWS Batch をともに使用するには、最新の AWS Batch 機能をサポートしているバージョンの AWS CLI を使用する必要があります。AWS CLI で AWS Batch 機能のサポートが表示されない場合は、最新バージョンにアップグレードする必要があります。詳細については、http://aws.amazon.com/cli/ を参照してください。

> Note
>
> AWS Batch では Amazon EC2 のコンポーネントを使用するため、これらの手順の多くで Amazon EC2 コンソールを使用します。

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

1. AWS にサインアップする (p. 3)
2. IAM ユーザーを作成する (p. 3)
3. 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 (p. 5)
4. キーペアの作成 (p. 5)
5. Virtual Private Cloud の作成 (p. 7)
6. セキュリティグループの作成 (p. 7)
7. AWS CLI をインストールする (p. 8)

AWS にサインアップする

AWS にサインアップすると、Amazon EC2 および AWS Batch を含むすべてのサービスに対して AWS アカウントが自動的にサインアップされます。料金が発生するのは、実際に使用したサービスの分のみです。

既に AWS アカウントをお持ちの場合は次のタスクに進んでください。AWS アカウントをお持ちでない場合は、次に説明する手順にしたがってアカウントを作成してください。

AWS アカウントを作成するには

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、電話のキーパッドを用いて確認コードを入力することが求められます。

次のタスクで AWS アカウント番号が必要となるので、メモしておいてください。

IAM ユーザーを作成する

AWS のサービス (Amazon EC2、AWS Batch など) では、サービスにアクセスする際に認証情報を提供する必要があります。このため、サービスのリソースにアクセスする権限があるかどうかがサービスによっ

て判定されます。コンソールを使用するにはパスワードが必要です。AWS アカウントのアクセスキーを作成して、コマンドラインインターフェイスまたは API にアクセスすることができます。ただし、AWS アカウントの認証情報を使って AWS にアクセスすることはお勧めしません。代わりに AWS Identity and Access Management (IAM) を使用することをお勧めします。IAM ユーザーを作成して、管理権限を使ってこのユーザーを IAM グループに追加するか、管理権限を付与します。そして、特別な URL とその IAM ユーザーの認証情報を使用して AWS にアクセスできます。

AWS にサインアップしても、ご自分の IAM ユーザーをまだ作成していない場合は、IAM コンソールを使用して作成できます。

自分用の管理者ユーザーを作成し、そのユーザーを管理者グループに追加するには (コンソール)

1. AWS アカウント E メールアドレスとパスワードを使用して https://console.aws.amazon.com/iam/ で AWS アカウントのルートユーザー として IAM コンソールにサインインします。

   Note

   以下の**Administrator**IAMユーザーの使用に関するベストプラクティスに従い、ルートユーザー認証情報を安全な場所に保管しておくことを強くお勧めします。ルートユーザーとしてサインインして、少数のアカウントおよびサービス管理タスクのみを実行します。
2. ナビゲーションペインで [Users]、[Add user] の順に選択します。
3. [ユーザー名] に「**Administrator**」と入力します。
4. [AWS マネジメントコンソール access (アクセス)] の横にあるチェックボックスをオンにします。[Custom password (カスタムパスワード)] を選択し、その後テキストボックスに新しいパスワードを入力します。
5. (オプション) AWS では、デフォルトで、新しいユーザーに対して初回のサインイン時に新しいパスワードを作成することが必要です。必要に応じて [User must create a new password at next sign-in (ユーザーは次回のサインイン時に新しいパスワードを作成する必要がある)] のチェックボックスをオフにして、新しいユーザーがサインインしてからパスワードをリセットできるようにできます。
6. [Next: Permissions (次へ: アクセス許可)] を選択します。
7. [Set permissions (アクセス許可の設定)] で、[Add user to group (ユーザーをグループに追加)] を選択します。
8. [Create group] を選択します。
9. [グループの作成] ダイアログボックスで、[グループ名] に「**Administrators**」と入力します。
10. [Filter policies (フィルタポリシー)] を選択し、その後 [AWS managed -job function (AWS 管理ジョブの機能] を選択してテーブルのコンテンツをフィルタリングします。
11. ポリシーリストで、[AdministratorAccess] のチェックボックスをオンにします。次に、[Create group] を選択します。

    Note

    `AdministratorAccess` アクセス許可を使用して、AWS Billing and Cost Management コンソールを使用する前に、IAM ユーザーおよびロールの請求へのアクセスをアクティブ化する必要があります。これを行うには、請求コンソールへのアクセスの委任に関するチュートリアルのステップ 1 の手順に従ってください。
12. グループのリストに戻り、新しいグループのチェックボックスをオンにします。必要に応じて [Refresh] を選択し、リスト内のグループを表示します。
13. [次へ: タグ] を選択します。
14. (オプション) タグをキー - 値のペアとしてアタッチして、メタデータをユーザーに追加します。IAM でのタグの使用の詳細については、『IAM ユーザーガイド』の「IAM エンティティのタグ付け」を参照してください。
15. [Next: Review] を選択して、新しいユーザーに追加するグループメンバーシップのリストを表示します。続行する準備ができたら、[Create user] を選択します。

この同じプロセスを繰り返して新しいグループとユーザーを作成し、AWS アカウントのリソースへのアクセス権をユーザーに付与できます。ポリシーを使用して特定の AWS リソースに対するユーザーのアクセス許可を制限する方法については、「アクセス管理」と「ポリシーの例」を参照してください。

新規の IAM ユーザーとしてサインインするには、AWS コンソールからサインアウトし、次の URL を使用します。このとき、your_aws_account_id はハイフンを除いた AWS アカウント番号です (たとえば AWS アカウント番号が `1234-5678-9012` であれば、AWS アカウント ID は `123456789012` となります) 。

```
https://your_aws_account_id.signin.aws.amazon.com/console/
```

作成した IAM ユーザー名とパスワードを入力します。サインインすると、ナビゲーションバーに「your_user_name @ your_aws_account_id」が表示されます。

サインページの URL に AWS アカウント ID を含めない場合は、アカウントのエイリアスを作成します。IAM ダッシュボードから [Create Account Alias (アカウントの別名を作成)] を選択し、エイリアス (会社名など) を入力します。アカウントエイリアスを作成した後、サインインするには、次の URL を使用します。

```
https://your_account_alias.signin.aws.amazon.com/console/
```

アカウントの IAM ユーザーのサインインリンクを確認するには、IAM コンソールを開き、ダッシュボードの [IAM users sign-in link] の下を確認します。

IAM の詳細については、AWS Identity and Access Management ユーザーガイド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 のコンピューティング環境およびコンテナインスタンスでは、ユーザーの代わりに他の AWS API を呼び出すために 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る必要があります。

> Note
>
> AWS Batch の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。代わりに AWS CLI を使用する場合は、最初のコンピューティング環境を作成する前に、「AWS Batch サービス IAM ロール (p. 88)」および「Amazon ECS インスタンスロール (p. 90)」の手順を実行してください。

キーペアの作成

AWS では公開キー暗号化を使用して、お客様のインスタンスのログイン情報の安全性を保護します。AWS Batch の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 インスタンスでは、SSH アクセスにパスワードを使いません。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。

キーペアをまだ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。複数のリージョンでインスタンスを起動する予定がある場合は、各リージョンでキーペアを作成する必要があります。リージョンの詳細については、Linux インスタンス用 Amazon EC2 ユーザーガイド の「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. https://console.aws.amazon.com/ec2/) にある Amazon EC2 コンソールを開きます。
2. ナビゲーションバーで、キーペアを生成するリージョンを選択します。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する必要があります。
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。覚えやすい名前 (IAM ユーザー名など) を選び、その後に `-key-pair` を続け、さらにリージョン名を続けます。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の名前として指定した名前となり、ファイル名の拡張子は `.pem` となります。プライベートキーファイルを安全な場所に保存します。

   Important

   これは、プライベートキーを保存する唯一のチャンスです。インスタンスと対応するプライベートキーの起動時には、毎回インスタンスに接続するたびに、キーペアの名前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すると、お客様以外のユーザーはそれを読み取ることができないようになります。

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

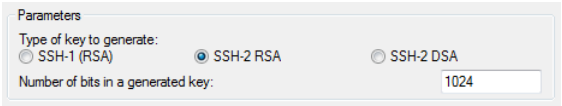
詳細については、Linux インスタンス用 Amazon EC2 ユーザーガイドの「Amazon EC2 のキーペア」を参照してください。

キーペアを使用してインスタンスに接続するには

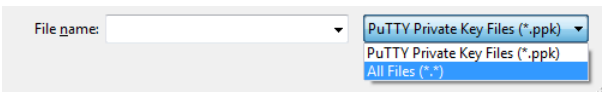
Mac または Linux を実行しているコンピュータから Linux インスタンスに接続するには、`-i` オプションとプライベートキーへのパスを指定して、SSH クライアントに対する `.pem` ファイルを指定します。Windows を実行しているコンピュータから Linux インスタンスに接続する場合は、MindTerm または PuTTY のどちらかを使用できます。PuTTY を使用する予定がある場合は、それをインストールしてから、次のプロシージャを使用して `.pem` ファイルを `.ppk` ファイルに変換します。

(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 を開始します (例: [開始] メニューで [すべてのプログラム] > [PuTTY] > [PuTTYgen] を選択)。
3. [Type of key to generate] で、[SSH-2 RSA] を選択します。



4. [Load] を選択します。デフォルトでは、PuTTYgen には拡張子 `.ppk` 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、[Open] 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 は自動的にファイル拡張子 `.ppk` を加えます。

Virtual Private Cloud の作成

Amazon Virtual Private Cloud (Amazon VPC) を使用すると、定義した仮想ネットワーク内で 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 (p. 7)」に移動できます。デフォルトの VPC があるかどうかを判断するには、Linux インスタンス用 Amazon EC2 ユーザーガイドの「Amazon EC2 コンソールでサポートされるプラットフォーム」を参照してください。また、次の手順を使用して、アカウントにデフォルト以外の VPC を作成することもできます。

> Important
>
> アカウントがリージョン内で EC2-Classic をサポートしている場合、そのリージョンにはデフォルトの VPC はありません。

デフォルト以外の VPC を作成するには

1. https://console.aws.amazon.com/vpc/にある Amazon VPC コンソールを開きます。
2. ナビゲーションバーで、VPC のリージョンを選択します。VPC はリージョンに固有であるため、キーペアを作成したリージョンと同じリージョンを選択してください。
3. VPC ダッシュボードで、[VPC ウィザードの起動] を選択します。
4. [Step 1: Select a VPC Configuration] ページで、[VPC with a Single Public Subnet] が選択されていることを確認したら、[Select] を選択します。
5. [Step 2: VPC with a Single Public Subnet] ページで、[VPC name] にわかりやすい VPC 名を入力します。他のデフォルトの設定はそのままにしておき、[Create VPC] を選択します。確認ページで、[OK] を選択します。

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。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、Linux インスタンス用 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。

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。たとえば、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内側から静的な IP アドレスなしで接続する場合は、クライアントコンピュータで使用されている IP アドレスの範囲を見つける必要があります。

最小限の権限でセキュリティグループを作成するには

1. https://console.aws.amazon.com/ec2/) にある Amazon EC2 コンソールを開きます。
2. ナビゲーションバーで、セキュリティグループのリージョンを選択します。セキュリティグループはリージョンに固有であるため、キーペアを作成したリージョンと同じリージョンを選択してください。
3. ナビゲーションペインで [セキュリティグループ] を選択して、[セキュリティグループの作成] を選択します。
4. 新しいセキュリティグループの名前と説明を入力します。覚えやすい名前 (IAM ユーザー名など) を選び、その後に _SG_ を続け、さらにリージョン名を続けます。たとえば、me_SG_useast1 とします。
5. [VPC] リストで、デフォルトの VPC が選択されていることを確認します。この VPC には、アスタリスク (*) が示されています。

   Note

   アカウントが EC2-Classic をサポートしている場合は、前のタスクで作成した VPC を選択します。
6. AWS Batch コンテナインスタンスでは、インバウンドポートが開いている必要はありません。ただし、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、SSH ルールを追加できます。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。

   - [Add Rule] 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。
   - [Add Rule] を選択します。[タイプ] で [SSH] を選択します。[ソース] では、[カスタム IP] が選択されていることを確認し、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 CIDR 表記で指定します。CIDR 表記で個々の IP アドレスを指定するには、ルーティングプレフィックスを追加します。`/32`たとえば、IP アドレスが `203.0.113.25` の場合は、`203.0.113.25/32` を指定します。会社が特定の範囲からアドレスを割り当てている場合、範囲全体 (`203.0.113.0/24`など) を指定します。

     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。

AWS CLI をインストールする

AWS Batch で AWS CLI を使用するには、最新バージョンの AWS CLI をインストールします。AWS CLI のインストール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。

AWS Batch の開始方法

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。

> Important
>
> 開始する前に、「AWS Batch でのセットアップ (p. 3)」のステップを完了していること、さらに AWS ユーザーに必要なアクセス権限があることを確認してください (管理者ユーザーはアクセス権限の問題について心配する必要はありません)。詳細については、「最初の IAM 管理者のユーザーおよびグループの作成」 (IAM ユーザーガイド) を参照してください。

ステップ 1: ジョブの定義

このセクションでは、ジョブ定義を作成するか、ジョブ定義なしでコンピューティング環境とジョブキューの作成に進むかを選択します。

ジョブのオプションを設定するには

1. AWS Batch コンソールの初回実行ウィザードを、https://console.aws.amazon.com/batch/home#/wizard で開きます。
2. AWS Batch のジョブ定義、コンピューティング環境、およびジョブキューを作成してジョブを送信するには、[Amazon EC2 の使用] を選択します。ジョブを送信せずにコンピュート環境とジョブキューのみを作成するには、[ジョブ送信がありません] を選択します。
3. ジョブ定義を作成する場合、初回起動のウィザードで最初の [ジョブランタイム]、[環境]、[パラメータ]、[環境変数] の 4 つのセクションを完了し、[次] を選択します。ジョブ定義を作成しない場合は、[次] を選択して「ステップ 2: コンピューティング環境とジョブキューの設定 (p. 11)」に進みます。

ジョブの実行時間を指定するには

1. 新しいジョブ定義を作成する場合は、[ジョブ定義名] でジョブ定義の名前を指定します。
2. (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。この機能の詳細については、「タスク用の IAM ロール」 (Amazon Elastic Container Service Developer Guide) を参照してください。

   > Note
   >
   > ここには、[Amazon Elastic Container Service タスクロール] の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 ロールの作成の詳細については、「タスク用の IAM ロールとポリシーの作成」 (Amazon Elastic Container Service Developer Guide) を参照してください。
3. [コンテナイメージ] で、ジョブに使用する Docker イメージを選択します。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 コンテナを作成する セクションの `Image` と docker run の `IMAGE` パラメータにマッピングします。

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。たとえば、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 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: *`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`)。

環境のリソースを指定するには

1. [コマンド] で、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 コンテナを作成する セクションの `Cmd` にマッピングし、 `COMMAND` パラメータを docker run にマッピングします。Docker `CMD` パラメーターの詳細については、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 (p. 41)」を参照してください。
2. [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 コンテナを作成する セクションの `CpuShares` にマッピングし、`--cpu-shares` オプションを docker run にマッピングします。各 vCPU は 1,024 個の CPU 配分に相当します。
3. [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、強制終了されます。このパラメータは、Docker Remote API の コンテナを作成する セクションの `Memory` にマッピングし、`--memory` オプションを docker run にマッピングします。
4. [ジョブ試行] で、ジョブが失敗した場合に再試行する最大回数を指定します。詳細については、「ジョブの再試行の自動化 (p. 18)」を参照してください。

Parameters

オプションとして、パラメータ置換のデフォルト値とプレースホルダーをコマンドに指定できます。詳細については、「パラメータ (p. 41)」を参照してください。

1. [キー] で、パラメータのキーを指定します。
2. [値] で、パラメータの値を指定します。

環境変数を指定するには

オプションで、ジョブのコンテナに渡す環境変数を指定できます。このパラメータは、Docker Remote API の コンテナを作成する セクションの `Env` にマッピングし、`--env` オプションを docker run にマッピングします。

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。

1. [キー] で、環境変数のキーを指定します。
2. [値] で、環境変数の値を指定します。

ステップ 2: コンピューティング環境とジョブキューの設定

コンピューティング環境は、コンピューティングリソース (Amazon EC2 インスタンス) を参照する方法であり、インスタンスを設定して自動的に起動する方法を AWS Batch に指示する設定や制約です。ジョブキューに送信したジョブは、AWS Batch スケジューラによってコンピューティング環境内のコンピューティングリソースで実行されるまで、ジョブキューに格納されます。

> Note
>
> 現時点では、初回実行ウィザードで作成できるのはマネージド型のコンピューティング環境のみです。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、「コンピューティング環境の作成 (p. 67)」を参照してください。

コンピューティング環境タイプを設定するには

1. [コンピューティング環境名] では、コンピューティング環境の一意な名前を指定します。
2. [サービスロール] で、新しいロールを作成するか、既存のロールを使用することを選択し、AWS Batch サービスがユーザーに代わって必要な AWS API を呼び出せるようにします。詳細については、「AWS Batch サービス IAM ロール (p. 88)」を参照してください。新しいロールを作成することを選択した場合は、必要なロール (`AWSBatchServiceRole`) が自動的に作成されます。
3. [EC2 インスタンスロール] で、新しいロールを作成するか、既存のロールを使用することを選択し、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がユーザーに代わって必要な AWS API を呼び出せるようにします。詳細については、「Amazon ECS インスタンスロール (p. 90)」を参照してください。新しいロールを作成することを選択した場合は、必要なロール (`ecsInstanceRole`) が自動的に作成されます。

インスタンスを設定するには

1. [プロビジョニングモデル] では、[オンデマンド] を選択して Amazon EC2 オンデマンドインスタンスを起動するか、または [スポット] を選択して Amazon EC2 スポットインスタンスを使用します。
2. Amazon EC2 スポットインスタンスを使用することを選択した場合:

   a. [上限入札価格] 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、入札パーセンテージが 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。

   b. [スポットフリートロール] で、新しいロールを作成するか、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。新しいロールを作成することを選択した場合は、必要なロール (`aws-ec2-spot-fleet-role`) が自動的に作成されます。詳細については、「Amazon EC2 スポットフリートロール (p. 90)」を参照してください。
3. [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 (`c5.8xlarge` など) を指定できます。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 (C、M、および R 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オンザフライで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。たとえば、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。

4. [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。
5. [必要な vCPU] で、コンピューティング環境を起動する EC2 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。
6. [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。

ネットワーキングを設定するには

コンピューティングリソースは、ここで指定した VPC およびサブネット内で起動されます。これにより、AWS Batch コンピューティングリソースのネットワーク隔離を制御できます。

Important

は、Amazon ECS サービスエンドポイントと通信するためのアクセス権限を必要とします。この操作は、インターフェイス VPC エンドポイントを通じて、またはパブリック IP アドレスを持つを通じて実行することができます。
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 Developer Guideの「」を参照してください。
インターフェイス VPC エンドポイントを設定しておらず、にパブリック IP アドレスがない場合、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、の「NAT ゲートウェイ」およびこのガイドの「Amazon VPC ユーザーガイド詳細については、「チュートリアル: のパブリックサブネットとプライベートサブネットを持つ VPC を作成する (p. 105)」を参照してください。

1. [VPC ID] で、インスタンスを起動する先の VPC を選択します。
2. [サブネット] で、選択した VPC のサブネットの中で、インスタンスをホスト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が選択されます。
3. [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。

インスタンスにタグを付けるには

オプションとして、コンピューティング環境で起動するインスタンスに、タグとしてキーと値のペアを適用できます。たとえば、タグとして `"Name": "AWS Batch Instance - C4OnDemand"` を指定し、コンピューティング環境内の各インスタンスにその名前を使用できます (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)。デフォルトでは、インスタンスのタグとしてコンピューティング環境名が使用されます。

1. [キー] で、タグのキーを指定します。
2. [値] で、タグの値を指定します。

ジョブキューを設定するには

ジョブキューに送信したジョブは、AWS Batch スケジューラによってコンピューティング環境内のコンピューティングリソースで実行されるまで、ジョブキューに格納されます。

- [ジョブキュー名] では、ジョブキューに一意の名前を選択します。

確認して作成するには

[このジョブキューに接続されているコンピューティング環境] セクションでは、新規のコンピューティング環境が新しいジョブキューとその順序に関連付けられていることを示しています。後で、ジョブキューに別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。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実行するかを、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` 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。

- コンピューティング環境とジョブキューの設定を確認し、[作成] を選択してコンピューティング環境を作成します。

ジョブ

ジョブは AWS Batch で実行する作業の単位です。ジョブは、ECS クラスターの Amazon ECS コンテナインスタンスで動作するコンテナ化されたアプリケーションとして実行できます。

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、「ジョブ定義のパラメータ (p. 40)」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

ジョブ定義を登録すると、それをジョブとして AWS Batch ジョブキューに送信できます。ジョブ定義に指定されているパラメータの多くは、ランタイム時に上書きできます。

ジョブを送信するには

1. AWS Batch コンソール (https://console.aws.amazon.com/batch/) 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、[ジョブ]、[ジョブの送信] の順に選択します。
4. [ジョブ名] では、ジョブに一意の名前を選択します。
5. [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、「ジョブ定義の作成 (p. 32)」を参照してください。
6. [ジョブキュー] で、作成済み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 (p. 52)」を参照してください。
7. [ジョブタイプ] で、単一ジョブの場合は [単一]、配列ジョブを送信する場合は [配列] を選択します。詳細については、「配列ジョブ (p. 20)」を参照してください。このオプションは、マルチノードの並列ジョブには使用できません。
8. (配列ジョブのみ) [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。
9. (省略可能) 任意のジョブの依存関係を宣言します。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 (p. 19)」を参照してください。

a. [ジョブは以下に依存します] で、このジョブの開始前に終了する必要があるジョブのジョブ ID を入力します。

b. (配列ジョブのみ) [多対多のジョブの依存関係] に、そのジョブの各子ジョブのインデックスが依存関係の対応する子インデックスジョブに依存する任意の配列ジョブのジョブ ID を 1 つ以上指定します。たとえば、`JobB:1` は `JobA:1` に依存するなどです。

c. (配列ジョブのみ) [子をシーケンシャルに実行] を選択して、現在の配列ジョブの `SEQUENTIAL` 依存関係を作成します。これにより、各子インデックスが、それより前の兄弟ジョブの完了を待機します。たとえば、`JobA:1` は `JobA:0` に依存するなどです。

10. [ジョブ試行] で、ジョブが失敗した場合に再試行する最大回数を指定します。詳細については、「ジョブの再試行の自動化 (p. 18)」を参照してください。

11. (オプション) [Execution Timeout (実行のタイムアウト)] で、ジョブの試行で実行を許可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、ジョブは停止し、ステータスは `FAILED` に変わります。詳細については、「ジョブのタイムアウト (p. 19)」を参照してください。

12. (オプション) [パラメータ] セクションで、ジョブのコンテナの起動時に実行されるコマンドにパラメータ置換のデフォルト値とプレースホルダーを指定できます。詳細については、「パラメータ (p. 41)」を参照してください。

    a. [パラメータの追加] を選択します。

    b. [キー] で、パラメータのキーを指定します。

    c. [値] で、パラメータの値を指定します。

13. [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 コンテナを作成する セクションの `CpuShares` にマッピングし、`--cpu-shares` オプションを docker run にマッピングします。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。

14. [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、強制終了されます。このパラメータは、Docker Remote API の コンテナを作成する セクションの `Memory` にマッピングし、`--memory` オプションを docker run にマッピングします。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。

15. (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。

16. [コマンド] で、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。JSON の結果 (実際に Docker デーモンに渡される形式) が正しいことを確認します。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 コンテナを作成する セクションの `Cmd` にマッピングし、`COMMAND` パラメータを docker run にマッピングします。Docker `CMD` パラメーターの詳細については、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 (p. 41)」を参照してください。

17. (オプション) ジョブのコンテナに渡す環境変数を指定できます。このパラメータは、Docker Remote API の コンテナを作成する セクションの `Env` にマッピングし、`--env` オプションを docker run にマッピングします。

    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。

    a. [環境変数の追加] を選択します

b. [キー] で、環境変数のキーを指定します。

Note

環境変数名を AWS_BATCH で始めることはできません。この命名規則は、AWS Batch サービスで設定される変数用に予約されています。

c. [値] で、環境変数の値を指定します。

18. [ジョブの送信] を選択します。

Note

RUNNING、SUCCEEDED、FAILED ジョブのログは CloudWatch Logs で閲覧できます。ロググループは /aws/batch/job、ログストリームの名前の形式は *jobDefinitionName*/default/*ecs_task_id* です (この形式は将来、変更され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その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、「CloudWatch Logs に送信されたログデータの表示」を Amazon CloudWatch Logs User Guideで参照してください。デフォルトでは、このログは永久に失効しませんが、保持期間を変更することもできます。詳細については、Amazon CloudWatch Logs User Guide の「CloudWatch Logs でのログデータ保管期間の変更」を参照してください。

ジョブの状態

AWS Batch ジョブキューに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

ジョブはキューに送信済みで、スケジューラによる評価待ちです。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、トラブルシューティングセクションの「RUNNABLE 状態でジョブが止まる (p. 109)」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。

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 (p. 18)」を参照してください。

Note

RUNNING ジョブのログは CloudWatch Logs で閲覧できます。ロググループは /aws/batch/job、ログストリームの名前の形式は *jobDefinitionName*/default/*ecs_task_id* です (この形式は将来、変更され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その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、「CloudWatch Logs に送信されたログデータの表示」を Amazon CloudWatch Logs User Guideで参照してください。デフォルトでは、このログは永久に失効しませんが、保持期間を変更することもできます。詳細については、Amazon CloudWatch Logs User Guide の「CloudWatch Logs でのログデータ保管期間の変更」を参照してください。

SUCCEEDED

ジョブは、終了コード 0 で正常に完了しました。SUCCEEDED ジョブのジョブ状態は、AWS Batch に 24 時間保持されます。

Note

SUCCEEDED ジョブのログは CloudWatch Logs で閲覧できます。ロググループは /aws/batch/job、ログストリームの名前の形式は *jobDefinitionName*/default/*ecs_task_id* です (この形式は将来、変更され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その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、「CloudWatch Logs に送信されたログデータの表示」を Amazon CloudWatch Logs User Guideで参照してください。デフォルトでは、このログは永久に失効しませんが、保持期間を変更することもできます。詳細については、Amazon CloudWatch Logs User Guide の「CloudWatch Logs でのログデータ保管期間の変更」を参照してください。

FAILED

ジョブのすべての使用可能な試行が失敗しました。FAILED ジョブのジョブ状態は、AWS Batch に 24 時間保持されます。

Note

FAILED ジョブのログは CloudWatch Logs で閲覧できます。ロググループは /aws/batch/job、ログストリームの名前の形式は *jobDefinitionName*/default/*ecs_task_id* です (この形式は将来、変更され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その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、「CloudWatch Logs に送信されたログデータの表示」を Amazon CloudWatch Logs User Guideで参照してください。デフォルトでは、このログは永久に失効しませんが、保持期間を変更することもできます。詳細については、Amazon CloudWatch Logs User Guide の「CloudWatch Logs でのログデータ保管期間の変更」を参照してください。

AWS Batch ジョブ環境変数

AWS Batch は、コンテナジョブ内に特定の環境変数を自動的に設定します。これらの環境変数は、ジョブ内のコンテナの詳細分析を提供し、アプリケーションのロジックでこれらの変数の値を使用できます。AWS Batch で設定されるすべての変数は、プレフィックス AWS_BATCH_ 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あり、このプレフィックスをジョブ定義の独自の変数または上書きで使用することはできません。

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 (p. 24)」を参照してください。

`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 (p. 18)」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_MAIN_NODE_INDEX`

この変数は、マルチノードの並列ジョブ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、マルチノードの並列ジョブの子ノードのみに設定されます (主要なノードには存在しません)。この変数は、ジョブの主要な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、マルチノードの並列ジョブ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、マルチノードの並列ジョブ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。以下のような場合に失敗します。

- 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
- AWS サービスの内部エラーまたは停止

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。詳細については、「再試行戦略 (p. 48)」を参照してください。

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。

何らかの理由でジョブの試行が失敗し、再試行設定で指定した試行数が AWS_BATCH_JOB_ATTEMPT 数より大きい場合、ジョブは RUNNABLE 状態に戻されます。詳細については、「ジョブの状態 (p. 16)」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、「ジョブ定義の作成 (p. 32)」および「ジョブの送信 (p. 14)」を参照してください。

ジョブの依存関係

AWS Batch ジョブを送信する際に、そのジョブが依存するジョブ ID 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 (p. 20)」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、「ジョブの送信 (p. 14)」を参照してください。

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。

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 タイムスタンプからこの秒数が経過すると、ジョブは AWS Batch によって終了します。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。

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。

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 ジョブを送信することについての詳細は、「ジョブの送信 (p. 14)」を参照してください。

配列ジョブ

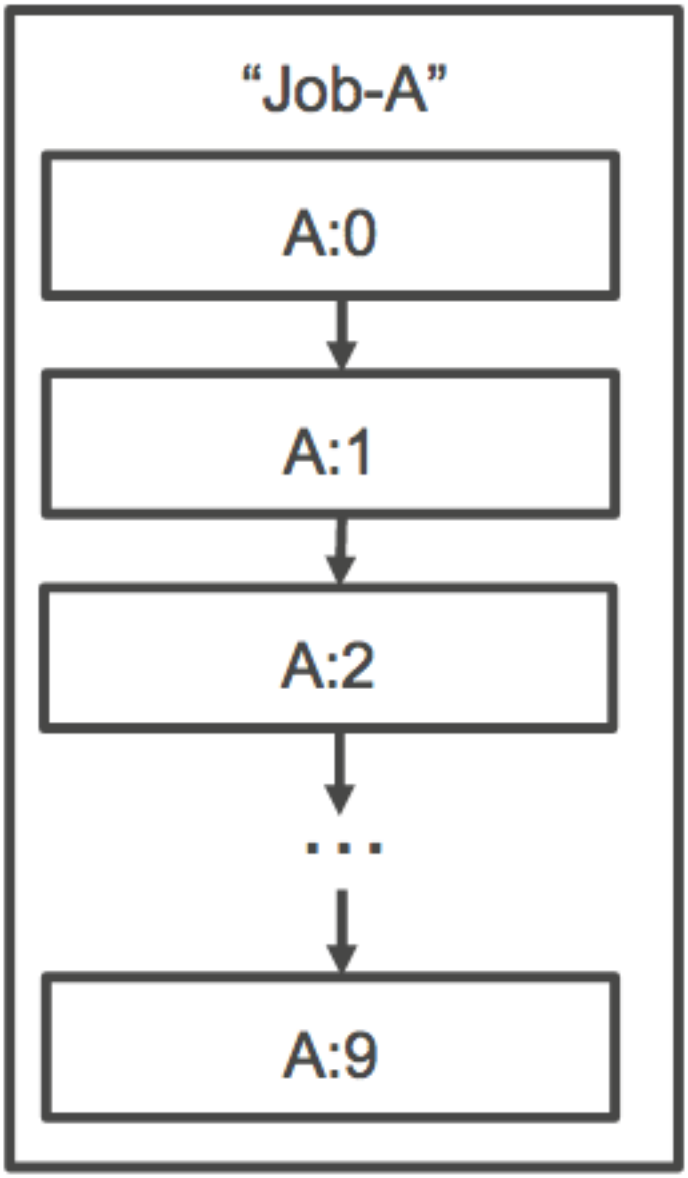
配列ジョブは、ジョブ定義、vCPU、メモリなどの共通パラメータを共有するジョブです。これは、関連しているが個別の基本ジョブのコレクションとして実行されます。複数のホストに分散されたり、同時に実行される場合もあります。配列ジョブは、モンテカルロシミュレーションジョブ、パラメータスイープジョブ、大規模なレンダリングジョブなど、大量の並列ジョブを実行するもっとも効率的な方法です。

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します。

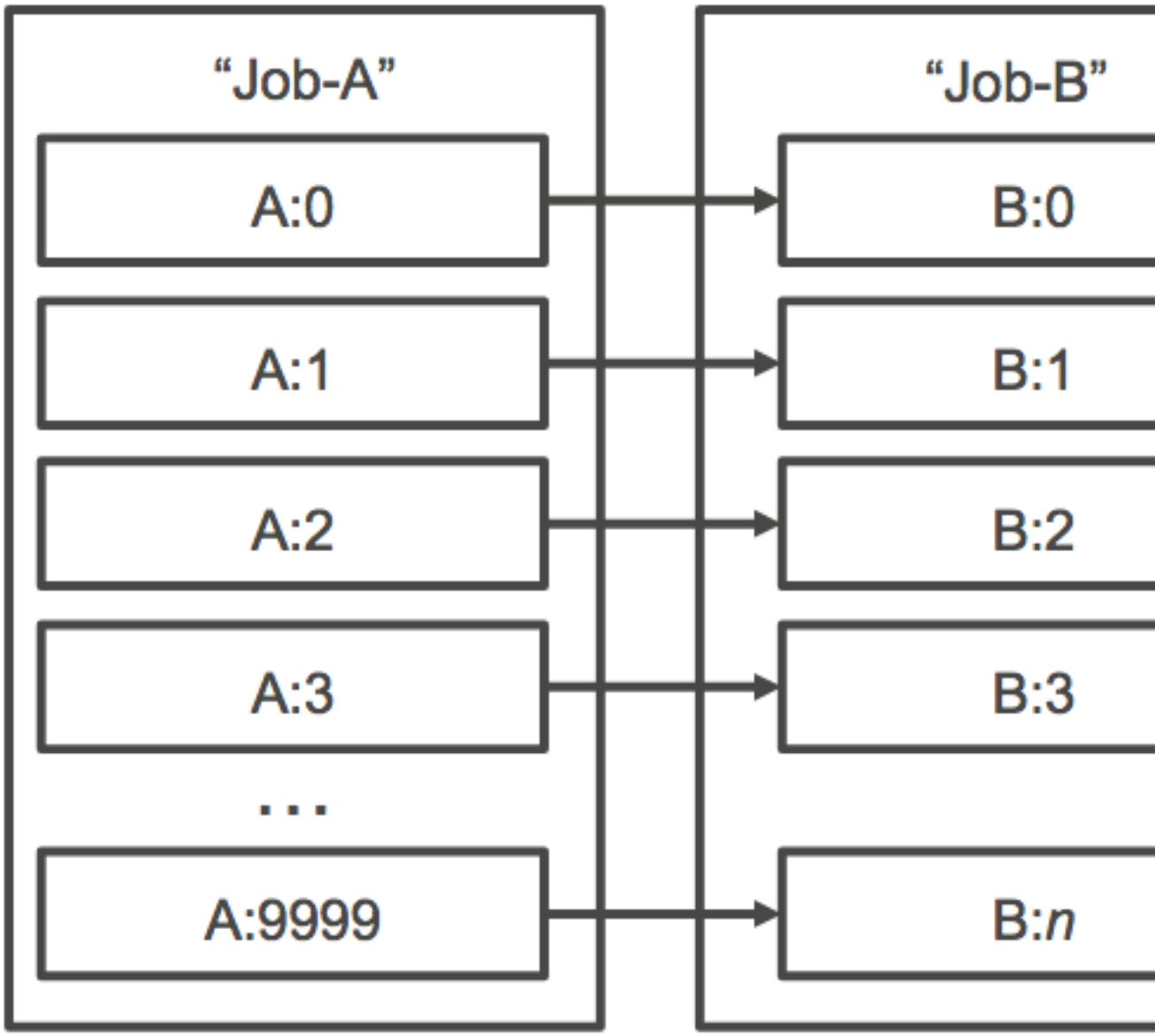
配列ジョブを送信すると、親の配列ジョブが通常の AWS Batch ジョブ ID を取得します。各子ジョブは同じベース ID を持ちますが、子ジョブの配列インデックスが親 ID の末尾に付加されます。たとえば、配列の最初の子ジョブは *example_job_ID*`:0` です。

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 (p. 24)」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、配列ジョブのジョブ 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 4 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 を利用するお客様の一般的なワークフローは、前提条件となるセットアップジョブを実行し、大量の入力タスクに対して一連のコマンドを実行した後、結果を集約し概略データを Amazon S3、DynamoDB、Amazon Redshift、または Aurora に書き込むジョブで完了します。

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。SubmitJob JSON 構文を次に示します。

```
{
    "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。SubmitJob JSON 構文を次に示します。

```
{
    "jobName": "JobB",
    "jobQueue": "ProdQueue",
    "jobDefinition": "JobB-CPU-Intensive-Processing:1",
    "containerOverrides": {
        "vcpus": 32,
        "memory": 4096
   }
    "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。SubmitJob JSON 構文を次に示します。

```
{
    "jobName": "JobC",
    "jobQueue": "ProdQueue",
    "jobDefinition": "JobC-Memory-Intensive-Processing:1",
    "containerOverrides": {
        "vcpus": 1,
        "memory": 32768
   }
    "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。JobD の各ステップは同じコマンドを実行しますが、ジョブのコンテナ内の AWS_BATCH_JOB_ARRAY_INDEX 環境変数の値に基づいて動作は異なります。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)。SubmitJob JSON 構文を次に示します。

```
{
    "jobName": "JobD",
    "jobQueue": "ProdQueue",
    "jobDefinition": "JobD-Sequential-Validation:1",
```

```
    "containerOverrides": {
        "vcpus": 1,
        "memory": 32768
    }
    "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。SubmitJob JSON 構文を次に示します。

```
{
    "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、各子ジョブが割り当てられている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示します。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取り、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している複数の AWS Batch ジョブを持つことができますが、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、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、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 (p. 67)」を参照してください。
- AWS Batch ジョブキューおよび関連付けられた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 (p. 52)」を参照してください。
- AWS CLI 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、「AWS Command Line Interface のインストール」 (AWS Command Line Interface ユーザーガイド) 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で `AWS_BATCH_JOB_ARRAY_INDEX` を使用することはできますが、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ほうがより簡単で強力です。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、次のコードを貼り付けます。

```
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、次のコードを貼り付けます。

   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、次のコードを貼り付けます。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
ENTRYPOINT /tmp/print-color.sh
```

5. Docker イメージの構築:

```
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、AWS_BATCH_JOB_ARRAY_INDEX 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 つの子がある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:

```
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、DockerHub などの別のレジストリを使用することもできます。

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。簡潔にするために、この例では AWS CLI を使用しますが、AWS マネジメントコンソールも使用できます。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 レジストリの login コマンドを取得します。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 --no-include-email
```

4. 前のステップで返された docker login ... コマンドを実行します。
5. イメージを Amazon ECR にプッシュする:

```
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、後で配列ジョブを実行するために使用できます。簡潔にするために、この例では AWS CLI を使用しますが、AWS マネジメントコンソールも使用できます。詳細については、「ジョブ定義の作成 (p. 32)」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.json という名前のファイルを作成し、次のコードを貼り付けます。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",
    "vcpus": 1,
    "memory": 250
  }
}
```

2. AWS Batch でジョブ定義を登録します。

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

ステップ 4: AWS Batch 配列ジョブの送信

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-def.json という名前のファイルを作成し、次のコードを貼り付けます。

   Note

   この例では、AWS Batch 初回実行ウィザードによって作成されるデフォルトのジョブキュー名を想定しています。ジョブキュー名が異なる場合は、first-run-job-queue の名前をジョブキュー名に置き換えます。

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

2. AWS Batch ジョブキューにジョブを送信します。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-def.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。

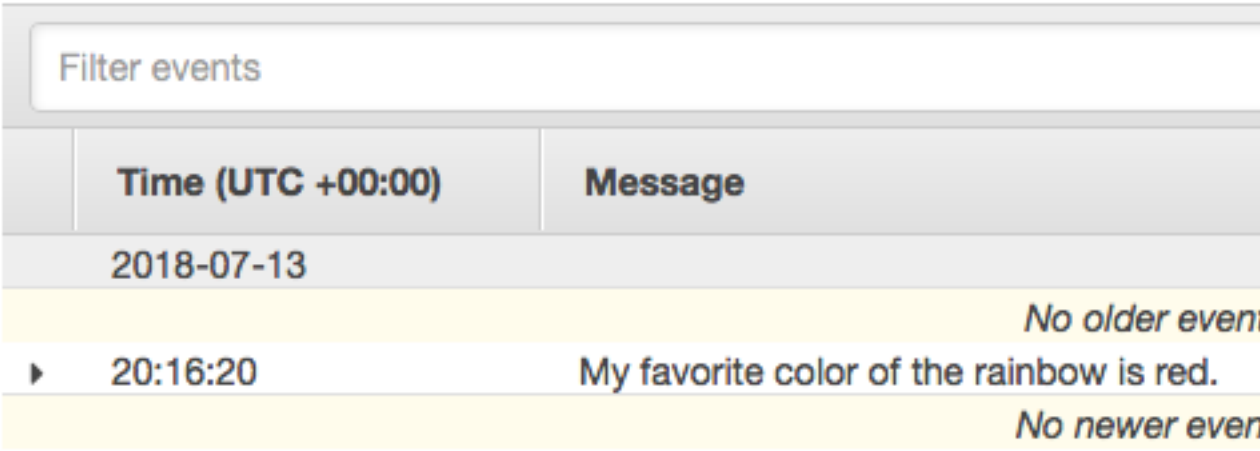
ステップ 5: 配列ジョブログの表示

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 Logs を表示できます。

CloudWatch Logs でジョブの 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. [ジョブキュー] で、キューを選択します。
4. [ステータス] セクションで、[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、[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 マルチパートの並列ジョブでは、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よ分散された 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フレームワークのすべて (Apache MXNet、TensorFlow, Caffe2 や Message Passing Interface (MPI) など) と互換性があります。

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。主要なノードがある場合、ジョブは完了したと見なされ、子ノードは停止します。詳細については、「ノードグループ (p. 29)」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。たとえば、ジョブの 2 番目のノードの詳細にアクセスするには、AWS Batch DescribeJobs API アクションを使用して、*`aws_batch_job_id`*#2 と記述する必要があります。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すると、主要なノードの失敗によって別の試行がトリガーされますが、子ノードの失敗ではトリガーされません。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。

AWS Batch でマルチノードの並列ジョブを実行するには、アプリケーションコードに分散された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

ランタイム時に、すべての AWS Batch ジョブが受信する標準的な環境変数に加えて、各ノードは、マルチノードの並列ジョブに特定の以下の環境変数で設定されます。

`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、マルチノードの並列ジョブの子ノードのみに設定されます (主要なノードには存在しません)。この変数は、ジョブの主要な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。AWS Batch 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。

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、主要なノードに単一の `c4.xlarge` インスタンス、子ノードには 5 つの `c4.xlarge` インスタンスを必要とするジョブを送信できます。それぞれのノードグループでは、各ジョブで別々のコンテナイメージやコマンドを指定することができます。

また、ジョブのすべてのノードで単一のノードグループを使用でき、`AWS_BATCH_JOB_MAIN_NODE_INDEX` 環境変数を `AWS_BATCH_JOB_NODE_INDEX` の独自の値で比較することで、アプリケーションコードではノードロール (主要なノードに対する子ノード) を差別化できます。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。

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き、ジョブは `SUBMITTED` ステータスを入力し、ジョブの依存関係が終了するまで待機します。次に、ジョブは `RUNNABLE` ステータスに移行し、AWS Batch はジョブの実行に必要なインスタンスの容量をプロビジョンして、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。主要なノードとは、送信したマルチノードジョブの結果を決定するために AWS Batch が監視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。

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。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。主要なノードがある場合、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。子ノードが消失した場合、AWS Batch はこのジョブの他のノードで何らかのアクションも実行しません。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。

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、『Linux インスタンス用 Amazon EC2 ユーザーガイド』の「プレースメントグループ」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 の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て、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を付与します。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 Developer Guide』の「タスクネットワーキングと awsvpc ネットワークモード」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。
- 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 インスタンスの制限は、ジョブを実行するために必要なインスタンスの数を満たしている必要があります。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 RUNNABLE ステータスで停止します。
- 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」および「Amazon EC2 P3 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 (GiB)	vCPU	メモリ (GiB)	ネットワーク帯域幅
p2.xlarge	1	12	4	61	高
p2.8xlarge	8	96	32	488	10 Gbps
p2.16xlarge	16	192	64	732	20 Gbps
p3.2xlarge	1	16	8	61	最大 10 Gbps
p3.8xlarge	4	64	32	244	10 Gbps
p3.16xlarge	8	128	64	488	25 Gbps
p3dn.24xlarge	8	256	96	768	100 Gbps

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。これらはそのジョブの期間中は、そのインスタンスで実行される他のジョブで使用できません。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` または `p3` 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を GPU 対応インスタンスで実行することもできますが、GPU 対応インスタンスで実行するのは、類似の GPU 非対応インスタンスで実行するよりも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。

ジョブ定義

AWS Batch のジョブ定義は、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。

目次

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
- ジョブで AWS アクセス権限の取得に使用する IAM ロール (ある場合)

ジョブ定義で使用できるパラメータの詳細については、「ジョブ定義のパラメータ (p. 40)」を参照してください。

ジョブ定義の作成

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、単一ノードと複数ノード並列ジョブでは若干異なります。このトピックでは、複数ノード並列ジョブ以外の AWS Batch ジョブのジョブ定義の作成を対象としています。

複数ノードの並列ジョブ定義を作成するには、「複数ノードの並列ジョブ定義を作成する (p. 35)」を参照してください。マルチノードの並列ジョブの詳細については、「マルチノードの並列ジョブ (p. 28)」を参照してください。

新しいジョブ定義を作成するには

1. AWS Batch コンソール (https://console.aws.amazon.com/batch/) 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 [ジョブ定義]、[作成] を選択します。
4. [ジョブ定義名] では、ジョブ定義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. [ジョブ試行] で、ジョブが失敗した場合に再試行する最大回数を指定します。詳細については、「ジョブの再試行の自動化 (p. 18)」を参照してください。
6. (オプション) [Execution Timeout] (実行のタイムアウト) で、ジョブの試行で実行を許可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、ジョブは停止し、ステータスは `FAILED` に変わります。詳細については、「ジョブのタイムアウト (p. 19)」を参照してください。

7. [Job requires multiple node configurations (複数ノード設定が必要なジョブ)] では、ボックスをオフのままにします。複数ノードの並列ジョブ定義を代わりに作成するには、「複数ノードの並列ジョブ定義を作成する (p. 35)」を参照してください。
8. (オプション) [パラメータ] セクションで、ジョブのコンテナの起動時に実行されるコマンドにパラメータ置換のデフォルト値とプレースホルダーを指定できます。詳細については、「パラメータ (p. 41)」を参照してください。

   a. [パラメータの追加] を選択します。
   b. [キー] で、パラメータのキーを指定します。
   c. [値] で、パラメータの値を指定します。
9. (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。この機能の詳細 (設定の前提条件など) については、「タスク用の IAM ロール」 (Amazon Elastic Container Service Developer Guide) を参照してください。

   Note

   ここには、[Amazon Elastic Container Service タスクロール] の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 ロールの作成の詳細については、「タスク用の IAM ロールとポリシーの作成」 (Amazon Elastic Container Service Developer Guide) を参照してください。
10. [コンテナイメージ] で、ジョブに使用する Docker イメージを選択します。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 コンテナを作成する セクションの `Image` と docker run の `IMAGE` パラメータにマッピングします。

    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。たとえば、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 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: *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`)。
11. [コマンド] で、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。さらに、JSON の結果 (実際に Docker デーモンに渡される形式) が正しいことを確認します。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 コンテナを作成する セクションの `Cmd` にマッピングし、`COMMAND` パラメータを docker run にマッピングします。Docker `CMD` パラメーターの詳細については、https://docs.docker.com/engine/reference/builder/#cmd を参照してください。

    Note

    コマンドでは、パラメータ置換とプレースホルダーのデフォルト値を使用できます。詳細については、「パラメータ (p. 41)」を参照してください。
12. [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 コンテナを作成する セクションの `CpuShares` にマッピングし、`--cpu-shares` オプションを

docker run にマッピングします。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。

13. [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、強制終了されます。このパラメータは、Docker Remote API の コンテナを作成する セクションの `Memory` にマッピングし、`--memory` オプションを docker run にマッピングします。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、「メモリ管理 (p. 75)」を参照してください。

14. (オプション) [リソースの要件] セクションで、ジョブのコンテナに使用するリソースの要件を設定できます。[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。

15. (オプション) [セキュリティ] セクションで、ジョブのコンテナのセキュリティオプションを設定できます。

    a. 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には、[特権] を選択します。このパラメータは、Docker Remote API の コンテナを作成する セクションの `Privileged` と docker run の `--privileged` オプションにマッピングされます。

    b. [ユーザー] に、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 コンテナを作成する セクションの `User` と docker run の `--user` オプションにマッピングされます。

16. (オプション) [マウントポイント] セクションで、アクセスするジョブのコンテナのマウントポイントを設定できます。

    a. [コンテナパス] に、ホストボリュームをマウントするコンテナのパスを入力します。

    b. [ソースボリューム] に、マウントするボリュームの名前を入力します。

    c. コンテナに対してボリュームを読み取り専用にするには、[読み取り専用] を選択します。

17. (オプション) [Volumes (ボリューム)] セクションで、ジョブのコンテナに渡すジョブのデータボリュームを指定できます。

    a. [名前] に、ボリュームの名前を入力します。最大 255 文字の英字 (大文字と小文字)、数字、ハイフン、アンダースコアを使用できます。

    b. (オプション) [Source Path] に、コンテナに渡すホストインスタンスのパスを指定します。このフィールドを空のままにすると、Docker デーモンによってホストパスが割り当てられます。ソースパスを指定した場合、手動で削除するまで、データボリュームはホストコンテナインスタンス上の指定した場所に保持されます。ソースパスがホストコンテナインスタンスに存在しない場合、Docker デーモンによってそのパスが作成されます。その場所が存在する場合は、ソースパスフォルダーの内容がコンテナにエクスポートされます。

18. (オプション) [環境変数] セクションで、ジョブのコンテナに渡す環境変数を指定できます。このパラメータは、Docker Remote API の コンテナを作成する セクションの `Env` と docker run の `--env` オプションにマッピングされます。

    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。

    a. [環境変数の追加] を選択します

    b. [キー] で、環境変数のキーを指定します。

Note

環境変数名を `AWS_BATCH` で始めることはできません。この命名規則は、AWS Batch サービスで設定される変数用に予約されています。

c. [値] で、環境変数の値を指定します。

19. (オプション) [uLimit] セクションで、ジョブのコンテナに使用する `ulimit` 値を設定できます。

a. [制限の追加] を選択します。
b. [制限の名前] で適用する ulimit を選択します。
c. [ソフト制限] で、ulimit タイプに適用するソフト制限を選択します。
d. [ハード制限] で、ulimit タイプに適用するハード制限を選択します。

20. (オプション) [Linux Parameters (Linux パラメータ)] セクションで、ジョブのコンテナに使用するデバイスマッピングを設定して、コンテナがホストインスタンス上のデバイスにアクセスできるようにすることが可能です。

a. [デバイス] セクションで、[デバイスの追加] を選択します。
b. [Host path (ホストパス)] で、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. [コンテナパス] で、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。
d. [Permissions (アクセス許可)] で、コンテナで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。使用可能なアクセス許可は `READ`、`WRITE`、`MKNOD` です。

21. [ジョブ定義の作成] を選択します。

複数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、単一ノードと複数ノード並列ジョブでは若干異なります。このトピックでは、複数ノードの AWS Batch 並列ジョブのジョブ定義の作成を対象としています。詳細については、「マルチノードの並列ジョブ (p. 28)」を参照してください。

単一ノードのジョブ定義を作成するには、「ジョブ定義の作成 (p. 32)」を参照してください。

複数ノードの並列ジョブ定義を作成するには

1. AWS Batch コンソール (https://console.aws.amazon.com/batch/) 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 [ジョブ定義]、[作成] を選択します。
4. [ジョブ定義名] では、ジョブ定義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. [ジョブ試行] で、ジョブが失敗した場合に再試行する最大回数を指定します。詳細については、「ジョブの再試行の自動化 (p. 18)」を参照してください。
6. (オプション) [Execution Timeout] (実行のタイムアウト) で、ジョブの試行で実行を許可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、ジョブは停止し、ステータスは `FAILED` に変わります。詳細については、「ジョブのタイムアウト (p. 19)」を参照してください。
7. [Job requires multiple node configurations (複数ノード設定が必要なジョブ)] を選択し、次のサブステップを実行します。単一ノードの並列ジョブ定義を代わりに作成するには、「ジョブ定義の作成 (p. 32)」を参照してください。

a. [Number of nodes (ノード数)] にmジョブで使用するノードの合計数を入力します。
b. [Main node (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。

c. (オプション) ノードを特定のインスタンスタイプに制限するには、ドロップダウンメニューから 1 つを選択します。インスタンスタイプを指定しない場合、AWS Batch は最も大きいノード (vCPU およびメモリ) の要件を満たす最小のインスタンスタイプをコンピューティング環境で利用可能なインスタンスタイプから選択します。

Important

コンピューティング環境で起動できるインスタンスタイプを選択するようにしてください。それ以外を選択すると、ジョブは `RUNNABLE` ステータスに止まり、後続のジョブがブロックされます。

8. (オプション) [パラメータ] セクションで、ジョブのコンテナの起動時に実行されるコマンドにパラメータ置換のデフォルト値とプレースホルダーを指定できます。詳細については、「パラメータ (p. 41)」を参照してください。

   a. [パラメータの追加] を選択します。

   b. [キー] で、パラメータのキーを指定します。

   c. [値] で、パラメータの値を指定します。

9. [Node properties (ノードのプロパティ)] セクションで、ノードグループを設定します。デフォルトでは、デフォルトのノード数で 1 つのノードグループが自動的に作成されます。

10. [Target nodes (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。

    ジョブに指定するノード数では 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。最後のノードグループの終了インデックス範囲は、Step 7.a (p. 35) で指定したノード数から 1 を引いた数にする必要があります。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場合、終了範囲は 9 にする必要があります。

11. [コンテナイメージ] で、ジョブに使用する Docker イメージを選択します。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 コンテナを作成する セクションの `Image` と docker run の `IMAGE` パラメータにマッピングします。

    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。たとえば、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 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: *`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`)。

12. [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 コンテナを作成する セクションの `CpuShares` にマッピングし、`--cpu-shares` オプションを docker run にマッピングします。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。

13. [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、強制終了されます。このパラメータは、Docker Remote API の コンテナを作成する セクションの `Memory` にマッピングし、`--memory` オプションを

docker run にマッピングします。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、「 メモリ管理 (p. 75)」を参照してください。

14. [コマンド] で、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。さらに、JSON の結果 (実際に Docker デーモンに渡される形式) が正しいことを確認します。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 コンテナを作成する セクションの `Cmd` にマッピングし、`COMMAND` パラメータを docker run にマッピングします。Docker `CMD` パラメーターの詳細については、https://docs.docker.com/engine/reference/builder/#cmd を参照してください。

    Note

    コマンドでは、パラメータ置換とプレースホルダーのデフォルト値を使用できます。詳細については、「パラメータ (p. 41)」を参照してください。

15. (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) をジョブのコンテナに付与するには、[Privileged (特権)] を選択します。このパラメータは、Docker Remote API の コンテナを作成する セクションの `Privileged` と docker run の `--privileged` オプションにマッピングされます。

16. (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。この機能の詳細 (設定の前提条件など) については、「タスク用の IAM ロール」 (Amazon Elastic Container Service Developer Guide) を参照してください。

    Note

    ここには、[Amazon Elastic Container Service タスクロール] の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 ロールの作成の詳細については、「タスク用の IAM ロールとポリシーの作成」 (Amazon Elastic Container Service Developer Guide) を参照してください。

17. [ユーザー] に、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 コンテナを作成する セクションの `User` と docker run の `--user` オプションにマッピングされます。

18. (オプション) ジョブのコンテナがアクセスするマウントポイントを指定します。

    a. [コンテナパス] に、ホストボリュームをマウントするコンテナのパスを入力します。

    b. [ソースボリューム] に、マウントするボリュームの名前を入力します。

    c. コンテナに対してボリュームを読み取り専用にするには、[読み取り専用] を選択します。

19. (オプション) ジョブのコンテナに渡すジョブのデータボリュームを指定できます。

    a. [名前] に、ボリュームの名前を入力します。最大 255 文字の英字 (大文字と小文字)、数字、ハイフン、アンダースコアを使用できます。

    b. (オプション) [Source Path] に、コンテナに渡すホストインスタンスのパスを指定します。このフィールドを空のままにすると、Docker デーモンによってホストパスが割り当てられます。ソースパスを指定した場合、手動で削除するまで、データボリュームはホストコンテナインスタンス上の指定した場所に保持されます。ソースパスがホストコンテナインスタンスに存在しない場合、Docker デーモンによってそのパスが作成されます。その場所が存在する場合は、ソースパスフォルダーの内容がコンテナにエクスポートされます。

20. (オプション) ジョブのコンテナに渡す環境変数を指定できます。このパラメータは、Docker Remote API の コンテナを作成する セクションの `Env` と docker run の `--env` オプションにマッピングされます。

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。

a. [環境変数の追加] を選択します
b. [キー] で、環境変数のキーを指定します。

Note

環境変数名を `AWS_BATCH` で始めることはできません。この命名規則は、AWS Batch サービスで設定される変数用に予約されています。

c. [値] で、環境変数の値を指定します。

21. [Ulimit] で、ジョブのコンテナに使用する ulimit 値を設定します。
    a. [制限の追加] を選択します。
    b. [制限の名前] で適用する ulimit を選択します。
    c. [ソフト制限] で、ulimit タイプに適用するソフト制限を選択します。
    d. [ハード制限] で、ulimit タイプに適用するハード制限を選択します。
22. Step 10 (p. 36) に戻り、各ノードグループごとに繰り返してジョブを設定します。
23. [ジョブ定義の作成] を選択します。

ジョブ定義テンプレート

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ってジョブ定義を作成し、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、ジョブ定義のパラメータ (p. 40)を参照してください。

```
{
    "jobDefinitionName": "",
    "type": "container",
    "parameters": {
        "KeyName": ""
    },
    "containerProperties": {
        "image": "",
        "vcpus": 0,
        "memory": 0,
        "command": [
            ""
        ],
        "jobRoleArn": "",
        "volumes": [
            {
                "host": {
                    "sourcePath": ""
                },
                "name": ""
            }
        ],
        "environment": [
            {
                "name": "",
                "value": ""
            }
        ],
        "mountPoints": [
```

```
            {
                "containerPath": "",
                "readOnly": true,
                "sourceVolume": ""
            }
        ],
        "readonlyRootFilesystem": true,
        "privileged": true,
        "ulimits": [
            {
                "hardLimit": 0,
                "name": "",
                "softLimit": 0
            }
        ],
        "user": "",
        "instanceType": ""
    },
    "nodeProperties": {
        "numNodes": 0,
        "mainNode": 0,
        "nodeRangeProperties": [
            {
                "targetNodes": "",
                "container": {
                    "image": "",
                    "vcpus": 0,
                    "memory": 0,
                    "command": [
                        ""
                    ],
                    "jobRoleArn": "",
                    "volumes": [
                        {
                            "host": {
                                "sourcePath": ""
                            },
                            "name": ""
                        }
                    ],
                    "environment": [
                        {
                            "name": "",
                            "value": ""
                        }
                    ],
                    "mountPoints": [
                        {
                            "containerPath": "",
                            "readOnly": true,
                            "sourceVolume": ""
                        }
                    ],
                    "readonlyRootFilesystem": true,
                    "privileged": true,
                    "ulimits": [
                        {
                            "hardLimit": 0,
                            "name": "",
                            "softLimit": 0
                        }
                    ],
                    "user": "",
                    "instanceType": ""
                }
            }
```

```
        ]
    },
    "retryStrategy": {
        "attempts": 0
    },
    "timeout": {
        "attemptDurationSeconds": 0
    }
}
```

Note

上記のジョブ定義テンプレートは、以下の AWS CLI コマンドで生成できます。

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

ジョブ定義のパラメータ

ジョブ定義は、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、ジョブのコンテナプロパティの 4 つの基本部分に分かれています。

目次

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。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。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。マルチノードの並列ジョブの詳細については、「the section called “複数ノードの並列ジョブ定義を作成する” (p. 35)」を参照してください。

型: 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し、送信時にプログラムでコマンドの値を変更することができます。

型: 文字列から文字列へのマッピング

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。以下に例を示します。

```
"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 (p. 47) レベルで設定されます。

`command`

コンテナに渡すコマンド。このパラメータは、Docker Remote API の コンテナを作成する セクションの `Cmd` にマッピングし、 `COMMAND` パラメータを docker run にマッピングします。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", ...]
```

型: 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 コンテナを作成する セクションの `Env` と docker run の `--env` オプションにマッピングされます。

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。

型: キーと値のペアの配列

必須: いいえ

`name`

環境変数の名前。

型: 文字列

必須: はい (`environment` を使用する場合)

`value`

環境変数の値。

型: 文字列

必須: はい (`environment` を使用する場合)

```
"environment" : [
    { "name" : "string", "value" : "string" },
    { "name" : "string", "value" : "string" }
]
```

`image`

コンテナ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 コンテナを作成する セクションの `Image` と docker run の `IMAGE` パラメータにマッピングします。

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。たとえば、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 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: *`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`)。

型: 文字列

必須: はい

`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 Developer Guideの「タスク用の IAM ロール」を参照してください。

型: 文字列

必須: いいえ

`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 コンテナを作成する セクションの `Memory` にマッピングし、`--memory` オプションを docker run にマッピングします。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、「 メモリ管理 (p. 75)」を参照してください。

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 コンテナを作成する セクションの `Volumes` と docker run の `--volume` オプションにマッピングされます。

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

型: 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。デフォルト値は `false` です。

タイプ: ブール値

必須: いいえ

`privileged`

このパラメーターが 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権限 (`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 コンテナを作成する セクションの `Privileged` と docker run の `--privileged` オプションにマッピングされます。

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

型: ブール値

必須: いいえ

`readonlyRootFilesystem`

このパラメーターが 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 コンテナを作成する セクションの `ReadonlyRootfs` にマッピングし、`--read-only` オプションを docker run にマッピングします。

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

型: ブール値

必須: いいえ

`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 コンテナを作成する セクションの `Ulimits` にマッピングし、`--ulimit` オプションを docker run にマッピングします。

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

型: 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 コンテナを作成する セクションの User と docker run の --user オプションにマッピングされます。

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

型: 文字列

必須: いいえ

resourceRequirements

コンテナ用に予約される GPU の数を示します。

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

型: オブジェクト配列

必須: いいえ

type

GPU はサポートされる唯一の値です。

型: 文字列

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

value

各コンテナが必要とする物理的な GPU の数です。

型: 文字列

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

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"
```

```
                            ]
                        }
                    ]
                }
```

型: 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。

型: Device オブジェクトの配列

必須: いいえ

`hostPath`

ホストでデバイスが使用可能なパス。

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。

型: 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。

型: 文字列の配列

必須: いいえ

有効な値: `READ` | `WRITE` | `MKNOD`

`vcpus`

コンテナ用に予約された vCPU の数。このパラメータは、Docker Remote API の コンテナを作成する セクションの `CpuShares` にマッピングし、`--cpu-shares` オプションを docker run にマッピングします。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。

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。

```
[
  {
    "name": "string",
```

```
    "host": {
      "sourcePath": "string"
    }
  }
]
```

`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 `mountPoints` の `sourceVolume` パラメータで参照されます。

型: 文字列

必須: はい

`host`

`host` パラメーターの内容により、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にデータが保持されるとは限りません。

型: オブジェクト

必須: いいえ

`sourcePath`

コンテナに渡す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` の場所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。

型: 文字列

必須: いいえ

ノードプロパティ

`nodeProperties`

複数ノードの並列ジョブ定義を登録する場合、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ノードプロパティの一覧を指定する必要があります。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 (p. 28)」を参照してください。

型: 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。

タイプ: 整数

必須: はい

`numNodes`

複数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`

複数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。

型: 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 (:n)、開始範囲に 0 が使用されます。終了範囲値が省略されている場合 (:n)、終了範囲に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。ノード範囲をネストする場合 (たとえば、0:10 と 4:5)、4:5 範囲のプロパティが 0:10 プロパティを上書きします。

型: 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 (p. 41)」を参照してください。

型: 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。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 (ゼロ以外の終了コードが返された場合やコンテナインスタンスが終了した場合) に再試行されます。詳細については、「ジョブの再試行の自動化 (p. 18)」を参照してください。

型: 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

タイムアウト

`timeout`

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。詳細については、「ジョブのタイムアウト (p. 19)」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 オペレーション時にタイムアウト設定を指定した場合は、ここで定義されているタイムアウト設定が上書きされます。詳細については、「ジョブのタイムアウト (p. 19)」を参照してください。

型: JobTimeout オブジェクト

必須: いいえ

`attemptDurationSeconds`

AWS Batch 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 `startedAt` 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。

タイプ: 整数

必須: いいえ

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。

環境変数の使用

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して、同定義のジョブを送信することもできます。

```
{
    "jobDefinitionName": "fetch_and_run",
    "type": "container",
    "containerProperties": {
        "image": "012345678910.dkr.ecr.us-east-1.amazonaws.com/fetch_and_run",
        "vcpus": 2,
        "memory": 2000,
        "command": [
            "myjob.sh",
            "60"
        ],
        "jobRoleArn": "arn:aws:iam::012345678910: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 (p. 41)」を参照してください。

```
{
    "jobDefinitionName": "ffmpeg_parameters",
    "type": "container",
    "parameters": {"codec": "mp4"},
    "containerProperties": {
        "image": "my_repo/ffmpeg",
        "vcpus": 2,
        "memory": 2000,
        "command": [
            "ffmpeg",
            "-i",
            "Ref::inputfile",
            "-c",
            "Ref::codec",
            "-o",
            "Ref::outputfile"
        ],
        "jobRoleArn": "arn:aws:iam::012345678910: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 (p. 61) 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",
        "vcpus": 8,
        "command": [
            "sh",
            "-c",
            "cd /tensorflow/tensorflow/examples/tutorials/mnist; python mnist_deep.py"
        ],
        "memory": 32000
    },
    "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
```

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 アカウントは、複数のジョブキューを持つことができます。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。

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時に、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、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、AWS Batch スケジューラが、この順位に基づいてジョブを関連付けられたコンピューティング環境に配置できるようにします。たとえば、1 つのコンピューティング環境が複数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、そのコンピューティング環境にジョブがスケジュールされます。

ジョブキューを作成するには

1. AWS Batch コンソール (https://console.aws.amazon.com/batch/) 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 [ジョブキュー]、[キューの作成] を選択します。
4. [キューの名前] では、一意のジョブキュー名を入力します。
5. [ジョブキューを有効にする] が選択されていて、ジョブキューがジョブの送信を受け入れることができることを確認します。
6. [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。
7. [このキューに接続されているコンピューティング環境] セクションで、ジョブキューに関連付けるコンピューティング環境をジョブキューで配置する順にリストから選択します。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実行するかを、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` 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コンピューティング環境では必ず、同じアーキテクチャを使用する必要があります。AWS Batch では、1 つのジョブキューでコンピューティング環境アーキテクチャを混在させることはできません。

   コンピューティング環境の順番は、テーブルの [順序] 列の横にある上向き矢印と下向き矢印を選択して変更できます。

8. [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、空のジョブキューテンプレートです。このテンプレートを使ってジョブキューを作成し、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、「CreateJobQueue」 (AWS Batch API リファレンス) を参照してください。

```
{
    "jobQueueName": "",
    "state": "",
    "priority": 0,
    "computeEnvironmentOrder": [{
        "order": 0,
        "computeEnvironment": ""
    }]
}
```

Note

上記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-job-queue --generate-cli-skeleton
```

ジョブキューのパラメータ

ジョブキューは、ジョブキューの名前、状態、優先度、およびコンピューティング環境の順番の 4 つの基本コンポーネントに分かれています。

ジョブキュー名

`jobQueueName`

コンピューティング環境の名前です。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

状態

`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。

型: 文字列

有効な値: `ENABLED` | `DISABLED`

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。

タイプ: 整数

必須: はい

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。

Note

ジョブキューに関連付けられているコンピューティング環境では必ず、同じアーキテクチャを使用する必要があります。AWS Batch では、1 つのジョブキューでコンピューティング環境アーキテクチャを混在させることはできません。

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。

ジョブのスケジュール設定

AWS Batch スケジューラは、ジョブキューに送信されたジョブを実行するタイミング、場所、方法を判断します。ジョブの実行順は、他のジョブへの依存関係がすべて満たされている限り、送信順とほぼ同じです。

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。最初のコンピューティング環境に使用可能なリソースがある場合は、そのコンピューティング環境内のコンテナインスタンスにジョブがスケジュールされます。コンピューティング環境に適切なコンピューティングリソースがない場合、スケジューラは次のコンピューティング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

マネージド型のコンピューティング環境では、ビジネス要件を記述できます。マネージド型のコンピューティング環境では、AWS Batch 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、環境内のコンピューティングリソースの容量とインスタンスのタイプを管理します。マネージド型のコンピューティング環境では Amazon EC2 オンデマンドインスタンスを使用するか、スポットインスタンスを使用するかを選択できます。必要に応じて、上限価格を設定し、スポットインスタンスの料金がオンデマンド料金の指定されたパーセンテージを下回る場合にのみスポットインスタンスを起動するようにできます。

マネージド型のコンピューティング環境は、その環境の作成時に指定した VPC およびサブネット内で Amazon ECS コンテナインスタンスを起動します。Amazon ECS コンテナインスタンスは、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。コンテナインスタンスに必要なパブリック IP アドレスが、選択したサブネットではデフォルトで提供されない場合、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 ユーザーガイドの「NAT ゲートウェイ」を参照してください。VPC の作成方法については、「チュートリアル: のパブリックサブネットとプライベートサブネットを持つ VPC を作成する (p. 105)」を参照してください。

デフォルトでは、AWS Batch のマネージド型のコンピューティング環境では、承認されたバージョンの Amazon ECS に最適化された最新の AMI をコンピューティング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 (p. 57)」を参照してください。

> Note
>
> AWS Batch は、作成後のコンピューティング環境 (たとえば、新しいバージョンの Amazon ECS に最適化された AMI が使用可能な場合など) で AMI をアップグレードしません。お客様は、ゲストオペレーティングシステム (更新プログラムやセキュリティパッチを含む) の管理、およびコン

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の責任を負うものとします。AWS Batch ジョブで新しい AMI を使用するには:

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. 古いコンピューティング環境を削除します。

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確認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 (p. 58)」および「コンピューティングリソース AMI の作成 (p. 58)」を参照してください。

アンマネージコンピューティング環境を作成したら、DescribeComputeEnvironments API 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確認します。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。

次の AWS CLI コマンドでは、Amazon ECS クラスターの ARN も提供します。

```
aws batch describe-compute-environments --compute-environments unmanagedCE --query
 computeEnvironments[].ecsClusterArn
```

詳細については、「Amazon ECS コンテナインスタンスの起動」 (Amazon Elastic Container Service Developer Guide) 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* を、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 のマネージド型のコンピューティング環境では、承認されたバージョンの Amazon ECS に最適化された最新の AMI をコンピューティングリソースに使用します。ただし、以下の理由により、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 を独自に作成する場合があります。

- 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
- カスタムオプションを使用して Amazon ECS コンテナエージェントを設定する
- カスタムオプションを使用するように Docker を設定する
- 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するための GPU ワークロードの AMI を設定する

トピック

コンピューティングリソースの AMI 仕様

基本的な AWS Batch コンピューティングリソースの AMI 仕様は、以下で構成されます。

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。

  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 (カスタム AMI を指定せずに)、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。詳細については、「マルチノードの並列ジョブ (p. 28)」を参照してください。
- Amazon ECS コンテナエージェント (最新バージョンを推奨)。詳細については、「Amazon ECS コンテナエージェントのインストール」 (Amazon Elastic Container Service Developer Guide) 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 Developer Guideの「Amazon ECS コンテナエージェントの設定」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 実行時の依存関係。詳細については、Docker ドキュメントの「実行時の依存関係を確認する」を参照してください。

  Note

  最大限のエクスペリエンスを得るには、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。詳細については、Amazon Elastic Container Service Developer Guideの「Amazon ECS コンテナエージェントのバージョン」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 nanny 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。Amazon ECS コンテナエージェントを起動してモニターするために `systemd` を使用するユーザーデータ設定スクリプトの例をいくつか表示するには、「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 Amazon Elastic Container Service Developer Guideで参照してください。`ecs-init` についての詳細は、GitHub の「`ecs-init` プロジェクト」を参照してください。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 エージェントを開始する必要があります。コンピューティングリソースで実行されていない Amazon ECS エージェントでは、AWS Batch からジョブを受け入れることはできません。

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 Developer Guide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。ただし、「コンピューティングリソースの AMI 仕

様 (p. 58)」に従う必要があります。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て、ジョブキューへのジョブの送信を開始できます。

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 を選択して開始します。基本 AMI は、HVM 仮想化を使用する必要があり、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に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している場合は、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 では、Amazon ECS 最適化の Amazon Linux 2 AMI の x86 と ARM のバージョンのいずれも提供しています。詳細については、Amazon Elastic Container Service Developer Guide 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 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 に最適化された AMI は、AWS のエンジニアによって AWS Batch で事前設定済みおよびテスト済みです。最も簡単に開始できる AMI であり、AWS でコンピューティングリソースをすばやく実行できます。詳細については、Amazon Elastic Container Service Developer Guide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 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。

   ```
   sudo yum install -y ecs-init
   ```

   たとえば、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 から開始し、これで AWS Batch のジョブを実行できるように設定することができます。詳細については、「GPU ワークロードの AMI の使用 (p. 61)」を参照してください。

   Important

   `ecs-init` パッケージをサポートしない基本 AMI を選択した場合は、ブート時に Amazon ECS エージェントを開始して実行し続ける方法を設定する必要があります。Amazon ECS コンテナエージェントを起動してモニターするために `systemd` を使用するユーザーデータ設定スクリプトの例をいくつか表示するには、「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 Amazon Elastic Container Service Developer Guideで参照してください。

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、「インスタンスの作成」と「Amazon EC2 インスタンスストア」 (Linux インスタンス用 Amazon EC2 ユーザーガイド) を参照してください。
3. SSH を使用してインスタンスに接続し、必要に応じて、次のような設定タスクを実行します。
   - Amazon ECS コンテナエージェントのインストール詳細については、「Amazon ECS コンテナエージェントのインストール」 (Amazon Elastic Container Service Developer Guide) 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、『Linux インスタンス用 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。

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。これを行わないと、インスタンスは AMI から起動するインスタンスを開始しません。

   a. Amazon ECS コンテナエージェントを停止します。

      - Amazon ECS-optimized Amazon Linux 2 AMI:

      ```
      sudo systemctl stop ecs
      ```

      - Amazon ECS-optimized 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/ecs_agent_data.json` にあります。ファイルを削除するには、次のコマンドを使用します。

   ```
   sudo rm -rf /var/lib/ecs/data/ecs_agent_data.json
   ```

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、Linux インスタンス用 Amazon EC2 ユーザーガイド・ガイドの「Amazon EBS-Backed Linux AMI の作成」を参照してください。

AWS Batch で新しい AMI を使用するには

1. AMI 作成プロセスが完了したら、新しい AMI でコンピューティング環境を作成します ([ユーザー指定の AMI ID を有効にします] を選択し、「Step 5.i (p. 69)」でカスタム AMI ID を指定してください)。詳細については、「コンピューティング環境の作成 (p. 67)」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に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している場合は、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 では、Amazon ECS 最適化の Amazon Linux 2 AMI の x86 と ARM のバージョンのいずれも提供しています。詳細については、Amazon Elastic Container Service Developer Guide 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 (p. 52)」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コンピューティング環境では必ず、同じアーキテクチャを使用する必要があります。AWS Batch では、1 つのジョブキューでコンピューティング環境アーキテクチャを混在させることはできません。

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 (p. 49)」、「ジョブ定義の作成 (p. 32)」、および「ジョブの送信 (p. 14)」を参照してください。

GPU ワークロードの AMI の使用

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 ECS で GPU を操作する」および「Amazon ECS に最適化された AMI」 (Amazon Elastic Container Service Developer Guide) 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` または `p3` 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、AWS Batch では Amazon ECS GPU 最適化 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 最適化 AMI のメタデータを取得できます。

GetParameter の使用例を以下に示します。

AWS CLI

```
$ aws ssm get-parameter --name /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/
recommended \
                        --region us-east-2 --output json
```

出力の Value パラメータに AMI の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 名前が含まれており、名前のフルパス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
```

GetParametersByPath の使用例を以下に示します。

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、指定されたパスの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、指定されたパスにあるすべてのパラメータ名の値が、名前のフルパスを使用して含まれています。

```
/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 ECS に最適化された AMI メタデータの取得」 (Amazon Elastic Container Service Developer Guide) を参照してください。

起動テンプレートのサポート

AWS Batch にコンピューティング環境で Amazon EC2 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。起動テンプレートサポートでは、カスタム AMI を作成する必要なく、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。

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 コンソールで作成するか、あるいは AWS CLI または AWS SDK を使用できます。たとえば、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する起動テンプレートを示し、また暗号化するように設定されています。

```
{
    "LaunchTemplateName": "increase-container-volume-encrypt",
    "LaunchTemplateData": {
        "BlockDeviceMappings": [
            {
```

```
                "DeviceName": "/dev/xvdcz",
                "Ebs": {
                    "Encrypted": true,
                    "VolumeSize": 100,
                    "VolumeType": "gp2"
                }
            }
        ]
    }
}
```

上記の起動テンプレートは、JSON ファイルを `lt-data.json` という名前のファイルに保存して、次の AWS CLI コマンドを実行することで作成できます。

```
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、『Linux インスタンス用 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 タグ付き以外) が起動テンプレートおよびコンピューティング環境の両方で指定されている場合、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 キーペア
- Amazon EC2 AMI ID
- セキュリティグループ ID
- Amazon EC2 タグ

以下の起動テンプレートパラメータは、AWS Batch で無視されます。

- 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)
- インスタンスマーケットオプション (AWS Batch でスポットインスタンスの設定を制御する必要があります)
- API 終了の無効化 (AWS Batch でインスタンスのライフサイクルを制御する必要があります)

AWS Batch では、新しい起動テンプレートバージョンを使用したコンピューティング環境の更新をサポートしていません。起動テンプレートを更新する場合、新規のテンプレートで新しいコンピューティング環境を作成して、変更を有効化します。

起動テンプレートの Amazon EC2 ユーザーデータ

インスタンスの起動時に、cloud-init で実行された起動テンプレートに Amazon EC2 ユーザーデータを指定できます。ユーザーデータは一般的な設定シナリオを実行できますが、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 形式にする必要があります。これは、ユーザーデータがコンピューティングリソースを設定するために必要な他の AWS Batch ユーザーデータに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。

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==`
  - ブロックのコンテンツタイプの宣言: `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。
  - ユーザーデータのコンテンツ、たとえば、シェル・コマンドまたは `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==--`

以下に、独自のファイルを作成するために使用できる MIME マルチパートファイルの例をいくつか示します。

> **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、プレーンテキストにユーザーデータを貼り付けるか、またはファイルからアップロードできます。AWS CLI あるいは AWS SDK を使用する場合、まず最初にユーザーデータを `base64` エンコード化し、次に、下記の JSON に示すように、CreateLaunchTemplate の呼び出し時にこの文字列を `UserData` パラメータの値として送信します。

```
{
    "LaunchTemplateName": "base64-user-data",
    "LaunchTemplateData": {
        "UserData":
 "ewogICAgIkxhdW5jaFRlbXBsYXRlTmFtZSI6ICJpbmNyZWFzZS1jb250YWluZXItdm9sdW..."
    }
}
```

Example 既存の Amazon EFS ファイルシステムのマウント

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==--
```

Example デフォルトの Amazon ECS コンテナエージェント設定の上書き

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==--
```

コンピューティング環境の作成

AWS Batch でジョブを実行する前に、コンピューティング環境を作成する必要があります。マネージド型のコンピューティング環境を作成すると、環境内のインスタンスは、ユーザーの仕様に基づいて AWS Batch で管理できます。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と、環境内のインスタンス設定はユーザーが処理します。

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. AWS Batch コンソール (https://console.aws.amazon.com/batch/) 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、[コンピューティング環境]、[環境の作成] 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タイプ] で、[マネージド型] を選択します。
   b. [コンピューティング環境名] では、コンピューティング環境の一意な名前を指定します。最大 128 文字までの英字 (大文字と小文字)、数字、ハイフン、下線を使用できます。
   c. [サービスロール] では、新しいロールを作成するか、または既存のロールを使用するかを選択します。このロールにより、ユーザーの代わりに AWS Batch サービスから必要な AWS API を呼び出すことを許可します。詳細については、「AWS Batch サービス IAM ロール (p. 88)」を参照してください。新しいロールを作成することを選択した場合は、必要なロール (`AWSBatchServiceRole`) が自動的に作成されます。
   d. 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を使用すると、コンピューティング環境用に作成した Amazon ECS コンテナインスタンスにより、必要な AWS API 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 (p. 90)」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。

e. [EC2 キーペア] で、起動時にインスタンスに関連付ける既存の Amazon EC2 キーペアを選択します。このキーペアにより、SSH を使用してインスタンスに接続できます (セキュリティグループがポート 22 への進入を許可することを確認します)。

f. [コンピューティング環境の有効化] が選択されていて、コンピューティング環境で AWS Batch ジョブスケジューラからのジョブを受け入れることができることを確認します。

5. コンピューティングリソースを設定します。

a. [プロビジョニングモデル] では、[オンデマンド] を選択して Amazon EC2 オンデマンドインスタンスを起動するか、または [スポット] を選択して Amazon EC2 スポットインスタンスを使用します。

b. スポットインスタンスを使用することを選択した場合:

i. (省略可能) [上限価格] 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% 未満になる必要があります。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。

ii. [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 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 (p. 90)」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 (「Step 7 (p. 69)」を参照)、Amazon EC2 スポットフリートの IAM ロールは、より新しい 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はありません。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 (p. 110)」を参照してください。

c. [許可されたインスタンスタイプ] で、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 (`c5.8xlarge` など) を指定できます。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 (C、M、および R 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オンザフライで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。たとえば、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、ジョブキューに必要な容量に基づいて GPU をスケールします。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` または `p3` ファミリーのインスタンスタイプが含まれている必要があります。

d. (オプション) [起動テンプレート] で既存の 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入力されます。詳細については、「起動テンプレートのサポート (p. 64)」を参照してください。

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

f. [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。

g. [必要な vCPU] で、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。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。

h. [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。

i. (オプション) [ユーザー指定の AMI ID を有効にします] にチェックを入れて、独自のカスタム AMI を使用します。デフォルトでは、AWS Batch のマネージド型のコンピューティング環境では、承認されたバージョンの Amazon ECS に最適化された最新の AMI をコンピューティングリソースに使用します。コンピューティング環境で独自の AMI を作成して使用する場合は、コンピューティングリソースの AMI 仕様に従う必要があります。詳細については、「コンピューティングリソースの AMI (p. 57)」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に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している場合は、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 では、Amazon ECS 最適化の Amazon Linux 2 AMI の x86 と ARM のバージョンのいずれも提供しています。詳細については、Amazon Elastic Container Service Developer Guide の「Amazon ECS に最適化された Amazon Linux 2 AMI」を参照してください。

   - [AMI ID] にはカスタム AMI ID を貼り付け、[AMI の検証] を選択します。

6. ネットワーキングを設定します。

   Important

   は、Amazon ECS サービスエンドポイントと通信するためのアクセス権限を必要とします。この操作は、インターフェイス VPC エンドポイントを通じて、またはパブリック IP アドレスを持つを通じて実行することができます。
   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 Developer Guideの「」を参照してください。
   インターフェイス VPC エンドポイントを設定しておらず、にパブリック IP アドレスがない場合、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、の「NAT ゲートウェイ」およびこのガイドの「Amazon VPC ユーザーガイド詳細については、「チュートリアル: のパブリックサブネットとプライベートサブネットを持つ VPC を作成する (p. 105)」を参照してください。

   a. [VPC ID] で、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、選択した VPC のサブネットの中で、インスタンスをホスト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が選択されます。

   c. [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。

7. (オプション) インスタンスにタグを付けます。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。

8. [Create] を選択して終了します。

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. AWS Batch コンソール (https://console.aws.amazon.com/batch/) 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、[コンピューティング環境]、[環境の作成] を選択します。
4. [コンピューティング環境タイプ] で、[アンマネージド型] を選択します。
5. [コンピューティング環境名] では、コンピューティング環境の一意な名前を指定します。最大 128 文字までの英字 (大文字と小文字)、数字、ハイフン、下線を使用できます。
6. [サービスロール] で、新しいロールを作成するか、既存のロールを使用することを選択し、AWS Batch サービスがユーザーに代わって必要な AWS API を呼び出せるようにします。詳細については、「AWS Batch サービス IAM ロール (p. 88)」を参照してください。新しいロールを作成することを選択した場合は、必要なロール (`AWSBatchServiceRole`) が自動的に作成されます。
7. [コンピューティング環境の有効化] が選択されていて、コンピューティング環境で AWS Batch ジョブスケジューラからのジョブを受け入れることができることを確認します。
8. [Create] を選択して終了します。
9. (オプション) 関連付けられたクラスターの Amazon ECS クラスター ARN を取得します。次の AWS CLI コマンドは、コンピューティング環境の Amazon ECS クラスター ARN を提供します。

```
aws batch describe-compute-environments --compute-environments unmanagedCE --query
 computeEnvironments[].ecsClusterArn
```

10. (オプション) 関連付けられた Amazon ECS クラスター内でコンテナインスタンスを起動します。詳細については、「Amazon ECS コンテナインスタンスの起動」 (Amazon Elastic Container Service Developer Guide) 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* を、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
```

> **Note**
>
> アン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リソースを手動で起動するまで、コンピューティングリソースは利用できません。

コンピューティング環境テンプレート

以下に示すのは、空のコンピューティング環境テンプレートです。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 Batch API リファレンス) を参照してください。

```
{
    "computeEnvironmentName": "",
    "type": "UNMANAGED",
    "state": "ENABLED",
    "computeResources": {
        "type": "EC2",
        "minvCpus": 0,
        "maxvCpus": 0,
        "desiredvCpus": 0,
        "instanceTypes": [
            ""
        ],
        "imageId": "",
        "subnets": [
            ""
        ],
        "securityGroupIds": [
            ""
```

```
        ],
        "ec2KeyPair": "",
        "instanceRole": "",
        "tags": {
            "KeyName": ""
        },
        "bidPercentage": 0,
        "spotIamFleetRole": "",
        "launchTemplate": {
            "launchTemplateId": "",
            "launchTemplateName": "",
            "version": ""
        }
    },
    "serviceRole": ""
}
```

Note

上記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 5 つの基本コンポーネントに分かれています。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 (マネージド型のコンピューティング環境の場合)、および AWS Batch に IAM アクセス権限を提供するためのサービスロールです。

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。最大 128 文字までの英字 (大文字と小文字)、数字、ハイフン、下線を使用できます。

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。定義したコンピューティングリソースを AWS Batch で管理する場合は、`MANAGED` 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 (p. 72)」を

参照してください。独自の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。

型: 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 ENABLED である場合、AWS Batch スケジューラは環境内のコンピューティングリソースに関連付けられたジョブキューからジョブの配置を試みます。コンピューティング環境がマネージド型の場合は、ジョブキューの需要に基づいてそのインスタンスを自動的にスケールアウトまたはスケールインできます。

状態が DISABLED である場合、AWS Batch スケジューラは環境内でのジョブの配置を試行しません。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。マネージド型のコンピューティング環境は、状態が DISABLED である間はスケールアウトしません。ただし、インスタンスがアイドル状態になると、minvCpus 値にスケールインします。

型: 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。

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。このパラメータを使用して、コンピューティング環境で Amazon EC2 オンデマンドインスタンスを使用するか、Amazon EC2 スポットインスタンスを使用するかを指定します。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 (p. 90)」を参照してください。

有効な値: EC2 | SPOT

必須: はい

minvCpus

コンピューティング環境で維持する EC2 vCPU の最小数 (コンピューティング環境が DISABLED であっても)。

タイプ: 整数

必須: はい

`maxvCpus`

環境で到達できる EC2 vCPU の最大数。

タイプ: 整数

必須: はい

`desiredvCpus`

コンピューティング環境で必要な EC2 vCPUS の数。AWS Batch では、この値をジョブキューの需要に応じて最小値と最大値の間で調整します。

タイプ: 整数

必須: いいえ

`instanceTypes`

起動対象のインスタンスタイプ。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 (`c5.8xlarge` など) を指定できます。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 (C、M、および R 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オンザフライで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。たとえば、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。

タイプ: 文字列の配列

必須: はい

`imageId`

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 Machine Image (AMI) の ID。

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に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している場合は、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 では、Amazon ECS 最適化の Amazon Linux 2 AMI の x86 と ARM のバージョンのいずれも提供しています。詳細については、Amazon Elastic Container Service Developer Guide の「Amazon ECS に最適化された Amazon Linux 2 AMI」を参照してください。

型: 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。

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る EC2 セキュリティグループ。

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。

型: 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する Amazon ECS インスタンスプロファイル。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 (p. 90)」を参照してください。

型: 文字列

必須: はい

`tags`

コンピューティング環境で起動されるインスタンスにタグとして適用するキーと値のペア。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。

型: 文字列から文字列へのマッピング

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、最大パーセンテージが 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。

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)。詳細については、「Amazon EC2 スポットフリートロール (p. 90)」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、ここで指定したスポットフリートの IAM ロールは、より新しい 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はありません。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 (p. 110)」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。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 (p. 64)」を参照してください。

次を入力します。LaunchTemplateSpecification

オブジェクト

必須: いいえ

`launchTemplateId`

起動テンプレートの ID。

型: 文字列

必須: いいえ

`launchTemplateName`

起動テンプレートの名前。

型: 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。

型: 文字列

必須: いいえ

サービスロール

`serviceRole`

ユーザーに代わって AWS Batch が AWS の他のサービスを呼び出すことを許可する IAM ロールの完全な Amazon リソースネーム (ARN)。詳細については、「AWS Batch サービス IAM ロール (p. 88)」を参照してください。

型: 文字列

必須: はい

メモリ管理

Amazon ECS コンテナエージェントが を に登録するとき、 が に予約できるメモリの量をエージェントが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 が登録されたときに、 用に 8192 MiB として厳密に変換されるとは限りません。

に 8192 MiB を指定し、使用可能なメモリが 8192 MiB 以上の がなくこの要件を満たせない場合、その は 。

詳細については、「システムメモリの予約 (p. 76)」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。 と Windows の 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返します。

```
$ free -b
```

Amazon ECS-optimized 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ーがあり、タスクに使用できる 7985 MiB に変換されます。

システムメモリの予約

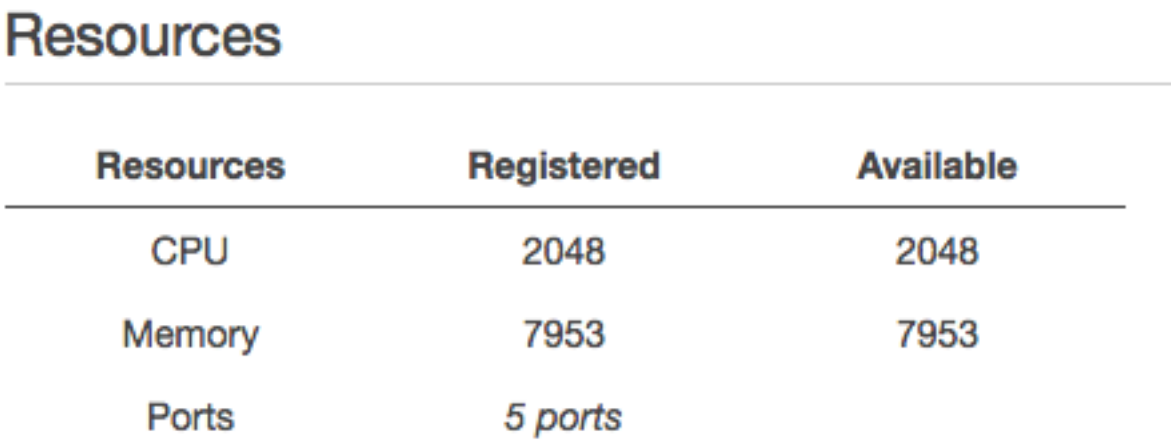
で 上のすべてのメモリを占有している場合、 がメモリが必要な重要なシステムプロセスと競合し、システム障害の引き金となる可能性があります。Amazon ECS コンテナエージェントは `ECS_RESERVED_MEMORY` という構成変数を提供します。この変数を使用して、 に割り当てられたプールから指定されただけのメモリ (MiB) を削減できます。これにより、重要なシステムプロセスのメモリを効果的に確保することができます。

メモリの表示

Amazon ECS コンソール (または DescribeContainerInstances API 操作) で、 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、可能な限り多くのメモリを に割り当て、リソース使用率を最大化しようとしている場合は、その のために使用可能なメモリを観察してから、 に可能な限り多くのメモリを割り当ててください。

メモリを表示するには

1. https://console.aws.amazon.com/ecs/ にある Amazon ECS コンソールを開きます。
2. 表示する をホストするクラスタを選択します。
3. [ECS Instances (ECS インスタンス)] を選択し、[Container Instance (コンテナインスタンス)] 列から表示を選択します。
4. [Resources (リソース)] セクションに、 用に登録され使用できるメモリが表示されます。



[Registered (登録済み)] メモリの値は Amazon ECS の初回起動時に登録された時の のメモリの値です。[Available (使用可能)] メモリの値はまだ に割り当てられていないメモリの値です。

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。

- コンピューティング環境には、サポートされているインスタンスタイプ (`c5n.18xlarge`、`c5n.metal`、`i3en.24xlarge`、`p3dn.24xlarge`) のみが含まれています。
- AMI 内の Amazon Linux、Amazon Linux 2、Red Hat Enterprise Linux 7.6、CentOS 7.6、Ubuntu 16.04、Ubuntu 18.04 の OS は EFA をサポートしています。
- 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る必要があります。
- 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` を設定する場合は、このパスも `/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とでは異なります。以下の例は、その違いを示すものであり、必須の値は含まれていません。

  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、Linux インスタンス用 Amazon EC2 ユーザーガイド の「Elastic Fabric Adapter」を参照してください。

AWS BatchIAM ポリシー、ロール、アクセス権限

デフォルトでは、IAM ユーザーには AWS Batch リソースを作成または変更、または AWS Batch API を使用するタスクを実行する権限がありません。つまり、AWS Batch コンソールまたは AWS CLI を使用して実行することもできません。IAM ユーザーがリソースを作成または変更、およびジョブを送信できるようにするには、IAM ポリシーを作成し、必要な特定のリソースおよび API アクションを使用するためのアクセス許可を IAM ユーザーに付与する必要があります。続いて、上記のアクセス権限が必要な IAM 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の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「アクセス権限とポリシー」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、「 ポリシーの管理IAM」を参照してください。

同様に、AWS Batch はユーザーの代わりに他の AWS サービスを呼び出すため、サービスで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。この認証を行うためには、これらのアクセス権限を付与できる IAM ロールおよびポリシーを作成し、そのロールをコンピューティング環境の作成時に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 (p. 90)」、「IAM ロール」、「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。

はじめに

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

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

- AWS Batch 用の Amazon リソースネーム (p. 82)
- 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 (p. 82)

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、IAM 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ないため、リクエストはすべて拒否されます。明示的な許可はデフォルトに優先します。明示的な拒否はすべての許可に優先します。
- [Action]: 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。action の指定については、AWS Batch のアクション (p. 81) を参照してください。
- [Resource]: 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 (p. 83)」および「AWS Batch 用の Amazon リソースネーム (p. 82)」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 (*) を使用して、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]: 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 (p. 87)」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 `batch:` の名前で次のプレフィックスを使用します。例: `batch:SubmitJob` および `batch:CreateComputeEnvironment`。

単一のステートメントに複数のアクションを指定するには、次のようにコンマで区切ります。

```
"Action": ["batch:action1", "batch:action2"]
```

ワイルドカードを使用して複数のアクションを指定することもできます。たとえば、以下のように「Describe」という単語で始まる名前のすべてのアクションを指定できます。

```
"Action": "batch:Describe*"
```

AWS Batch API アクションをすべて指定するには、* ワイルドカードを以下のように使用します。

```
"Action": "batch:*"
```

AWS Batch アクションの一覧については、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、ARN を使用して指定したリソースに適用されます。

ARN には以下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`)。

リージョン

リソースのリージョン (例: `us-east-1`)。

アカウント

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 オペレーションでは、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がサポートされるようになりました。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 (p. 83)」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、以下の要領で、`Resource` エレメント内で * ワイルドカードを使用します。

```
"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特定の API アクションおよび必要なリソースを使用する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。

まずテスト目的の IAM ユーザーを作成し、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の使用」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には 5 分かかると見ておいてください。

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 Command Reference」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ます。特定の AWS Batch アクションでは、満たす必要がある条件、またはユーザーが使用できる特定のリソースに基づいて、ユーザーがそれらのアクションをいつ使用できるかを制御できます。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。

次の表では、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 API アクションと、サポートされるリソース、リソース ARN、および各アクションの条件キーについて説明しています。

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 アクションでリソースレベルのアクセス許可がサポートされない場合、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が、ポリシーステートメントのリソース要素としてワイルドカード (*) を指定する必要があります。

API アクション	リソース	条件キー
RegisterJobDefinition	ジョブ定義 arn:aws:batch:region:account:job-definition/* arn:aws:batch:region:account:job-definition/definition-name:revision	batch:User batch:Privileged batch:Image
DeregisterJobDefinition	ジョブ定義 arn:aws:batch:region:account:job-definition/* arn:aws:batch:region:account:job-definition/definition-name:revision	該当なし
SubmitJob	ジョブ定義	該当なし

API アクション	リソース	条件キー
	arn:aws:batch:region:account:job-definition/* arn:aws:batch:region:account:job-definition/definition-name:revision ジョブキュー arn:aws:batch:region:account:job-queue/* arn:aws:batch:region:account:job-queue/queue-name	

ポリシーの例

以下の例では、AWS Batch に対して IAM ユーザーが所有するアクセス許可を制御するために使用できるポリシーステートメントを示しています。

例

例: 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許可をユーザーに与えます。

デフォルトで API アクションを使用するアクセス許可が拒否されているため、ユーザーには (別のステートメントでアクセス許可が与えられない限り) そのリソースに対してアクションを実行するアクセス許可がありません。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "batch:Describe*",
                "batch:List*"
            ],
            "Resource": "*"
        }
    ]
}
```

例 : 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

次のポリシーでは、ユーザーに独自の制限されたジョブ定義の管理を許可します。

最初と 2 番目のステートメントでは、ユーザーに *JobDefA_* というプレフィックスがあるジョブ定義名の登録と登録解除を許可します。

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、Amazon ECR リポジトリで POSIX ユーザーが `nobody`、特権フラグが `false`、イメージが `myImage` 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。

> **Important**
>
> Docker は、`user` パラメータをコンテナイメージ内からユーザーの `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 `uid` ポリシーの両方で直接 IAM 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 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では、ユーザーが特定のロールのみをジョブ定義に渡すように制限します。

```
{
    "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"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
```

```
                    "batch.amazonaws.com"
                ]
            }
        }
    }
]
}
```

例: ジョブ送信でジョブ定義プレフィックスに制限する

次のポリシーでは、*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すること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/JobDefA_*",
                "arn:aws:batch:<aws_region>:<aws_account_id>:job-queue/*"
            ]
        }
    ]
}
```

例: ジョブキューの制限

次のポリシーでは、任意のジョブ定義名で queue1 と名付けられた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"
            ]
        }
```

```
    ]
}
```

AWS Batch 管理ポリシー

AWS Batch は、IAM 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",
                "ecs:DescribeClusters",
                "ecs:Describe*",
                "ecs:List*",
                "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/ecsInstanceRole",
                "arn:aws:iam::*:role/iaws-ec2-spot-fleet-role",
                "arn:aws:iam::*:role/aws-ec2-spot-fleet-role",
                "arn:aws:iam::*:role/AWSBatchJobRole*"
            ]
        }
    ]
}
```

AWS Batch IAM ポリシーの作成

お客様のアカウントのユーザーがアクセスを許可される呼び出しとリソースを制限する IAM ポリシーを作成し、IAM ユーザーにそれらのポリシーをアタッチできます。

ポリシーをユーザーまたはユーザーの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「アクセス権限とポリシー」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 サービス IAM ロール

AWS Batch はユーザーに代わって他の AWS サービスを呼び出してサービスで使用するリソースを管理します。このサービスを使用するには、AWS Batch に必要なアクセス権限のある IAM ポリシーおよびロールが必要です。

ほとんどの場合、AWS Batch サービスロールは、コンソールの最初の実行時に自動的に作成されます。すでに AWS Batch サービスロールにアカウントがあるかどうかを確認するには、次の手順を使用できます。

`AWSBatchServiceRole` のポリシーを次に示します。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "ec2:DescribeAccountAttributes",
            "ec2:DescribeInstances",
            "ec2:DescribeSubnets",
            "ec2:DescribeSecurityGroups",
            "ec2:DescribeKeyPairs",
            "ec2:DescribeImages",
            "ec2:DescribeImageAttribute",
            "ec2:DescribeSpotFleetInstances",
            "ec2:DescribeSpotFleetRequests",
            "ec2:DescribeSpotPriceHistory",
            "ec2:RequestSpotFleet",
            "ec2:CancelSpotFleetRequests",
            "ec2:ModifySpotFleetRequest",
            "ec2:TerminateInstances",
            "autoscaling:DescribeAccountLimits",
            "autoscaling:DescribeAutoScalingGroups",
            "autoscaling:DescribeLaunchConfigurations",
            "autoscaling:DescribeAutoScalingInstances",
            "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:DescribeTaskDefinitions",
            "ecs:DescribeTasks",
            "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:PassRole"
        ],
        "Resource": "*"
    }]
}
```

次の手順を使用してアカウントに既に AWS Batch サービスのロールが存在するか確認して、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 **`AWSBatchServiceRole`** を確認するには

1. https://console.aws.amazon.com/iam/ にある IAM コンソールを開きます。
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 `AWSBatchServiceRole` を検索します。ロールが見つからない場合、次の手順を使用してロールを作成します。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. [Permissions] を選択します。
5. [AWSBatchServiceRole] 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、AWS Batch サービスのロールは適切に構成されています。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. [Attach Policy] を選択します。
   b. 利用可能なポリシーの一覧を絞り込むには、[フィルタ] で [AWSBatchServiceRole] と入力します。
   c. [AWSBatchServiceRole] ポリシーを選択し、[ポリシーのアタッチ] を選択します。
6. [Trust Relationship]、[Edit Trust Relationship] を選択します。
7. 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、[Cancel] を選択します。信頼関係が一致しない場合、ポリシーを [Policy Document] ウィンドウにコピーし、[Update Trust Policy] を選択します。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Principal": {"Service": "batch.amazonaws.com"},
        "Action": "sts:AssumeRole"
    }]
}
```

`AWSBatchServiceRole` IAM ロールを作成するには

1. https://console.aws.amazon.com/iam/ にある IAM コンソールを開きます。
2. ナビゲーションペインで [Roles]、[Create New Role] の順に選択します。
3. [Select type of trusted entity (信頼されたエンティティのタイプの選択)] で、[AWS サービス] を選択します。[このロールを使用するサービスを選択] で、[Batch] を選択します。

4. [Next: Permissions (次へ: アクセス許可)]、[Next: Tags (次へ: タグ)]、[Next: Review (次へ: レビュー)] の順に選択します。
5. [ロール名] に `AWSBatchServiceRole` を入力し、[Create Role (ロールの作成)] を選択します。

Amazon ECS インスタンスロール

AWS Batch コンピューティング環境では、Amazon ECS コンテナインスタンスが自動入力され、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ってさまざまな AWS API を呼び出すため、エージェントを実行するコンテナインスタンスでは、エージェントがユーザーに属していることをサービスに伝えるため、IAM ポリシーおよびロールが必要です。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前に、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。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。

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 **`ecsInstanceRole`** を確認するには

1. https://console.aws.amazon.com/iam/ にある IAM コンソールを開きます。
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. [Create Role (ロールの作成)] を選択します。
   b. [Select type of trusted entity (信頼されたエンティティのタイプの選択)] で、[AWS サービス] を選択します。[このロールを使用するサービスを選択] で、[Elastic Container Service] を選択します。[Select your use case (ユースケースを選択)] で、[EC2 Role for Elastic Container Service (Elastic Container Service の EC2 ロール)] を選択します。
   c. [Next: Permissions (次へ: アクセス許可)]、[Next: Tags (次へ: タグ)]、[Next: Review (次へ: レビュー)] の順に選択します。
   d. [ロール名] に `ecsInstanceRole` を入力し、[Create Role (ロールの作成)] を選択します。

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を作成する場合は、ユーザーの代わりにインスタンスの入札、起動、タグ付け、終了を行う権限をスポットフリートに付与するためのロールを作成する必要があります。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の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。これらのロールをすべて作成するには、以下の手順を使用します。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 のサービスにアクセス権限を委任するロールの作成」を参照してください。

トピック

- AWS マネジメントコンソールで Amazon EC2 スポットフリートロールを作成する (p. 91)
- 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 (p. 92)

AWS マネジメントコンソールで Amazon EC2 スポットフリートロールを作成する

スポットフリートのコンピューティング環境用の **`AmazonEC2SpotFleetRole`** IAM ロールを作成するには

1. https://console.aws.amazon.com/iam/ にある IAM コンソールを開きます。
2. ナビゲーションペインで [Roles]、[Create role] の順に選択します。
3. [Select type of trusted entity (信頼されたエンティティのタイプの選択)] で、[AWS サービス] を選択します。[このロールを使用するサービスを選択] で、[EC2] を選択します。
4. [Select your use case (ユースケースの選択)] セクションで、[EC2 Spot Fleet Role (EC2 スポットフリートロール)] 選択します。
5. [Next: Permissions (次へ: アクセス許可)]、[Next: Tags (次へ: タグ)]、[Next: Review (次へ: レビュー)] の順に選択します。

   Note

   これまでは、Amazon EC2 スポットフリートロールに対して 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元の管理ポリシーです。より厳密な IAM アクセス権限が含まれますが、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は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、「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 (p. 110)」を参照して、新しい推奨のポリシーをそのロールに適用してください。
   - 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。
6. [ロール名] に `AmazonEC2SpotFleetRole` を入力します。[Create Role (ロールの作成)] を選択します。

Amazon EC2 スポットの **`AWSServiceRoleForEC2Spot`** IAM サービスにリンクされたロールを作成するには

1. https://console.aws.amazon.com/iam/ にある IAM コンソールを開きます。
2. ナビゲーションペインで [Roles]、[Create role] の順に選択します。
3. [Select type of trusted entity (信頼されたエンティティのタイプの選択)] で、[AWS サービス] を選択します。[このロールを使用するサービスを選択] で、[EC2] を選択します。
4. [Select your use case (ユースケースの選択)] セクションで、[EC2 - Spot Instances (EC2 - スポットインスタンス)] 選択します。
5. [Next: Permissions (次へ: アクセス許可)]、[Next: Tags (次へ: タグ)]、[Next: Review (次へ: レビュー)] の順に選択します。
6. [ロール名] に `AmazonEC2SpotFleetRole` を入力します。[Create Role (ロールの作成)] を選択します。

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

1. https://console.aws.amazon.com/iam/ にある IAM コンソールを開きます。
2. ナビゲーションペインで [Roles]、[Create role] の順に選択します。

3. [Select type of trusted entity (信頼されたエンティティのタイプの選択)] で、[AWS サービス] を選択します。[このロールを使用するサービスを選択] で、[EC2] を選択します。
4. [Select your use case (ユースケースの選択)] セクションで、[EC2 - Spot Fleet (EC2 - スポットフリート)] 選択します。
5. [Next: Permissions (次へ: アクセス許可)]、[Next: Tags (次へ: タグ)]、[Next: Review (次へ: レビュー)] の順に選択します。
6. [ロール名] に `AWSServiceRoleForEC2SpotFleet` を入力します。[Create Role (ロールの作成)] を選択します。

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用の **`AmazonEC2SpotFleetRole`** 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 AmazonEC2SpotFleetRole \
    --assume-role-policy-document '{"Version":"2012-10-17","Statement":
[{"Sid":"","Effect":"Allow","Principal":
{"Service":"spotfleet.amazonaws.com"},"Action":"sts:AssumeRole"}]}'
```

2. IAM の `AmazonEC2SpotFleetTaggingRole` マネージドポリシーを `AmazonEC2SpotFleetRole` ロールにアタッチするには、AWS CLI で次のコマンドを実行します。

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::aws:policy/service-role/AmazonEC2SpotFleetTaggingRole \
    --role-name AmazonEC2SpotFleetRole
```

Amazon EC2 スポットの **`AWSServiceRoleForEC2Spot`** IAM サービスにリンクされたロールを作成するには

- 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

- 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
```

CloudWatch イベント IAM ロール

Amazon CloudWatch Events は、さまざまなターゲットに対する Amazon Web Services リソースの変更を示すシステムイベントのほぼリアルタイムのストリームを提供します。AWS Batch ジョブは CloudWatch イベント 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、ルールに一致したイベントに応じて AWS Batch ジョブを送信できます。CloudWatch イベント ルールおよび

ターゲットを使用して AWS Batch ジョブを送信するには、CloudWatch イベント にユーザーに代わって AWS Batch ジョブを実行するアクセス権限が必要です。

> Note
>
> AWS Batch キューをターゲットとして指定するルールを CloudWatch イベント コンソールで作成すると、このロールを作成する機会が提供されます。チュートリアル例については、「CloudWatch イベント ターゲットとしての AWS Batch ジョブ (p. 95)」を参照してください。

CloudWatch イベント の IAM ロールの信頼関係では、以下に示すように、`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"
    }
  ]
}
```

CloudWatch イベント の IAM ロールにアタッチされたポリシーでは、リソースに対する `batch:SubmitJob` アクセス権限を許可する必要があります。AWS Batch が提供する `AWSBatchServiceEventTargetRole` マネージドポリシーでは、以下に示すように、これらのアクセス権限が付与されます。

```
{
       "Version": "2012-10-17",
       "Statement": [
             {
                "Effect": "Allow",
                "Action": [
                     "batch:SubmitJob"
                 ],
                 "Resource": "*"
              }
        ]
}
```

CloudWatch イベント の AWS Batch イベントストリーム

CloudWatch イベント の 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現在の状態に関する通知を、ほぼリアルタイムで受け取ることができます。

CloudWatch イベント を使用すると、ジョブの進行状況のモニタリング、複雑な依存関係を持つ AWS Batch カスタムワークフローの構築、ジョブの実行に関連する使用状況レポートやメトリクスの生成、または独自のカスタムダッシュボードの構築ができます。AWS Batch および CloudWatch イベント では、AWS Batch を継続的にポーリングしてジョブのステータスの変更を確認するコードのスケジュールとモニタリングを排除できます。代わりに、CloudWatch イベント ターゲット (AWS Lambda、Amazon Simple Queue Service、Amazon Simple Notification Service、Amazon Kinesis Data Streams など) を使用して、AWS Batch ジョブの状態の変更を非同期的に処理できます。

AWS Batch イベントストリームのイベントは、少なくとも 1 回必ず送信されます。重複したイベントが送信された場合、イベントは重複を識別するために十分な情報を提供します (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 CloudWatch イベント 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、ルールに一致したイベントに応じて AWS Batch ジョブを送信できます。詳細については、Amazon CloudWatch Events ユーザーガイド の「Amazon CloudWatch Events とは」を参照してください。CloudWatch イベント を使用して、cron 式や rate 式により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 CloudWatch Events ユーザーガイドの「ルールのスケジュール式」を参照してください。チュートリアル例については、「CloudWatch イベント ターゲットとしての AWS Batch ジョブ (p. 95)」を参照してください。

トピック

AWS Batch イベント

AWS Batch は、ジョブのステータス変更イベントを CloudWatch イベント に送信します。AWS Batch はジョブの状態を追跡します。以前に提出されたジョブの状態が変わった場合、イベントがトリガーされます。たとえば、状態が `RUNNING` のジョブは `FAILED` に変わります。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
>
> 将来、AWS Batch には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。プログラムを使用してイベントの JSON データを逆シリアル化する場合は、不明なプロパティが追加されたときに問題が発生しないようにアプリケーションで対応する準備を整えます。

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 ジョブ状態の詳細については、「ジョブの状態 (p. 16)」を参照してください。

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

ジョブ状態変更イベントは次の形式で送信されます (以下の `detail` セクションは、AWS Batch API リファレンスの DescribeJobs API オペレーションから返される Job オブジェクトに似ています)。CloudWatch イベント パラメータの詳細については、Amazon CloudWatch Events 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": "aws_account_id",
  "time": "2017-10-23T17:56:03Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:batch:us-east-1:aws_account_id:job/4c7599ae-0a82-49aa-ba5a-4727fcce14a8"
  ],
  "detail": {
    "jobName": "event-test",
    "jobId": "4c7599ae-0a82-49aa-ba5a-4727fcce14a8",
    "jobQueue": "arn:aws:batch:us-east-1:aws_account_id:job-queue/HighPriority",
    "status": "RUNNABLE",
    "attempts": [],
    "createdAt": 1508781340401,
    "retryStrategy": {
      "attempts": 1
    },
    "dependsOn": [],
    "jobDefinition": "arn:aws:batch:us-east-1:aws_account_id:job-definition/first-run-job-
definition:1",
    "parameters": {},
    "container": {
      "image": "busybox",
      "vcpus": 2,
      "memory": 2000,
      "command": [
        "echo",
        "'hello world'"
      ],
      "volumes": [],
      "environment": [],
      "mountPoints": [],
      "ulimits": []
    }
  }
}
```

CloudWatch イベント ターゲットとしての AWS Batch ジョブ

Amazon CloudWatch Events は、さまざまなターゲットに対する Amazon Web Services リソースの変更を示すシステムイベントのほぼリアルタイムのストリームを提供します。AWS Batch ジョブは CloudWatch イベント 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、

ルールに一致したイベントに応じて AWS Batch ジョブを送信できます。詳細については、Amazon CloudWatch Events ユーザーガイド の「Amazon CloudWatch Events とは」を参照してください。

CloudWatch イベント を使用して、cron 式や rate 式により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 CloudWatch Events ユーザーガイドの「ルールのスケジュール式」を参照してください。

CloudWatch イベント ターゲットとしての AWS Batch ジョブの一般的ユースケースは以下のとおりです。

- 定期的な間隔で発生するスケジュールされたジョブを作成する。たとえば、使用量が低下して Amazon EC2 スポットインスタンスの料金が低くなる時間に発生する cron ジョブなど。
- CloudTrail に記録された API オペレーションに応じて AWS Batch ジョブを実行します。たとえば、指定された Amazon S3 バケットにオブジェクトがアップロードされたときにジョブを自動的に送信し、CloudWatch イベント のインプットトランスフォーマーを使用してオブジェクトのバケットとキー名を AWS Batch パラメータに渡します。

  Note

  このシナリオでは、すべての AWS リソース (Amazon S3 バケット、CloudWatch イベント ルール、CloudTrail ログなど) は同じリージョンに存在する必要があります。

CloudWatch イベント ルールおよびターゲットを使用して AWS Batch ジョブを送信するには、ユーザーに代わって AWS Batch ジョブを実行するアクセス権限が CloudWatch イベント サービスに必要です。AWS Batch ジョブをターゲットとして指定するルールを CloudWatch イベント コンソールで作成すると、このロールを作成する機会が提供されます。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、「CloudWatch イベント IAM ロール (p. 92)」を参照してください。

スケジュールされた AWS Batch ジョブを作成する

次の手順では、スケジュールされた AWS Batch ジョブと必要な CloudWatch イベント の IAM ロールを作成する方法を示します。

CloudWatch イベント でスケジュールされた AWS Batch ジョブを作成するには

1. https://console.aws.amazon.com/cloudwatch/にある CloudWatch コンソールを開きます。
2. 左のナビゲーションペインで、[イベント]、[ルールの作成] の順に選択します。
3. [イベントソース] で、[スケジュール] を選択します。次に、固定間隔スケジュールを使用するか `cron` 式を使用するかを選択します。詳細については、Amazon CloudWatch Events ユーザーガイドの「ルールのスケジュール式」を参照してください。
   - [一定の速度] に、スケジュールの間隔と単位を入力します。
   - [Cron expression] に、タスクスケジュールの `cron` 式を入力します。これらの式には 6 つの必須フィールドがあり、フィールドは空白で区切られます。詳細と `cron` 式の例については、Amazon CloudWatch Events ユーザーガイドの「Cron 式」を参照してください。
4. [Targets] で、[Add target] を選択します。
5. [Batch job queue] (バッチジョブのキュー) を選択し、以下のフィールドに適切な内容を入力します。
   - [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 (p. 20)」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 (p. 18)」を参照してください。

6. ジョブに使用する既存の CloudWatch イベント の IAM ロールを選択するか、[この特定のリソースに対して新しいロールを作成する] を選択して新しいロールを作成します。詳細については、「CloudWatch イベント IAM ロール (p. 92)」を参照してください。
7. [ルールの定義] で、以下のフィールドに適切に入力し、[ルールの作成] を選択します。

   - [名前]: ルールの名前を入力します。
   - [説明]: (オプション) ルールの説明を入力します。
   - [状態]: ルールを有効にするかどうかを選択し、次の間隔でスケジューリングを開始するか、後日まで無効にするかを指定します。

CloudWatch イベント インプットトランスフォーマーを使用してイベント情報を AWS Batch ターゲットに渡す

CloudWatch イベント のインプットトランスフォーマーを使用して、ジョブ送信で AWS Batch にイベント情報を渡します。これは、Amazon S3 バケットへのオブジェクトのアップロードなど、他の AWS イベント情報の結果としてジョブをトリガーする場合は、特に重要です。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。CloudWatch イベント のインプットトランスフォーマーによって、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。たとえば、次のジョブ定義では、*S3bucket* および *S3key* というパラメータ値を参照します。

```
{
  "jobDefinitionName": "echo-parameters",
  "containerProperties": {
    "image": "busybox",
    "vcpus": 2,
    "memory": 2000,
    "command": [
      "echo",
      "Ref::S3bucket",
      "Ref::S3key"
    ]
  }
}
```

こうすることで、トリガーとなる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 AWS Batch イベントターゲットを作成するだけです。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、CloudWatch イベント ルール、CloudTrail ログなど) は同じリージョンに存在する必要があります。

インプットトランスフォーマーを使用して AWS Batch ターゲットを作成するには

1. https://console.aws.amazon.com/cloudwatch/にある CloudWatch コンソールを開きます。
2. 左のナビゲーションペインで、[イベント]、[ルールの作成] の順に選択します。
3. [イベントソース] で、[イベントパターン] を選択してから、アプリケーションのニーズに合わせてルールを作成します。
4. [ターゲット] で [Batch job queue (バッチジョブのキュー)] を選択し、このルールによってトリガーされるジョブのジョブキュー、ジョブ定義、およびジョブ名を指定します。

5. [入力の設定] で、[インプットトランスフォーマー] を選択します。
6. インプットトランスフォーマーの上部のテキストボックスで、トリガーされるイベントから解析する値を指定します。たとえば、Amazon S3 イベントからのバケットおよびキー名を解析するには、次の JSON を使用します。

```
{"S3BucketValue":"$.detail.requestParameters.bucketName","S3KeyValue":"$.detail.requestParameters.k
```

7. インプットトランスフォーマーの下部のテキストボックスに、AWS Batch ジョブに渡す `Parameters` 構造を作成します。これらのパラメータは、ジョブの実行時にジョブコンテナのコマンドの *`Ref::S3bucket`* および *`Ref::S3key`* プレースホルダーに置き換えられます。

```
{"Parameters" : {"S3bucket": <S3BucketValue>, "S3key": <S3KeyValue>}}
```

8. ジョブに使用する既存の CloudWatch イベント の IAM ロールを選択するか、[この特定のリソースに対して新しいロールを作成する] を選択して新しいロールを作成します。詳細については、「CloudWatch イベント IAM ロール (p. 92)」を参照してください。
9. [設定の詳細] を選択し、[ルールの定義] で以下のフィールドに適切に入力してから、[ルールの作成] を選択します。
   - [名前]: ルールの名前を入力します。
   - [説明]: (オプション) ルールの説明を入力します。
   - [状態:] ルールを今すぐ有効にするか、後日まで無効にするかを選択します。

チュートリアル: AWS Batch CloudWatch イベントのリッスン

このチュートリアルでは、AWS Batch ジョブイベントをリッスンして 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、「AWS Batch の開始方法 (p. 9)」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、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. https://console.aws.amazon.com/lambda/ にある AWS Lambda コンソールを開きます。
2. [Create a Lambda function]、[Author from scratch] の順に選択します。
3. [Name] に、「batch-event-stream-handler」と入力します。
4. [Role] で、[Create a custom role]、[Allow] の順に選択します。
5. [関数の作成] を選択します。

6. [Function code] セクションで、ランタイム用に [Python 2.7] を選択し、次の例に一致するようにサンプルコードを編集します。

```
import json

def lambda_handler(event, 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 2.7 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. [Save] 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、AWS Batch リソースから送信されるジョブイベントをキャプチャする CloudWatch イベント 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内に、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) が含まれており、この情報を使用してプログラムでイベントをフィルタしてソートできます。

Note

AWS マネジメントコンソール を使用してイベントルールを作成すると、CloudWatch イベントに Lambda 関数を呼び出す権限を付与するために必要な IAM アクセス権限がコンソールによって自動的に追加されます。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 CloudWatch 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。

CloudWatch イベント ルールを作成するには

1. https://console.aws.amazon.com/cloudwatch/にある CloudWatch コンソールを開きます。
2. ナビゲーションペインで、[Events]、[Create rule] の順に選択します。
3. [Event source] で、イベントソースとして [Event Pattern] を選択し、[Build custom event pattern] を選択します。
4. 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 グループのすべての 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。

5. [Targets] で、[Add target] を選択します。[ターゲットの種類] で [Lambda 関数] を選択し、Lambda 関数を選択します。
6. [Configure details] を選択します。
7. [Rule definition] で、ルールの名前と説明を入力し、[Create rule] を選択します。

ステップ 3: 設定をテストする

最後に、ジョブキューにジョブを送信して CloudWatch イベント 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 (p. 14)」を参照してください。
3. https://console.aws.amazon.com/cloudwatch/にある CloudWatch コンソールを開きます。
4. ナビゲーションペインで、[Logs (ログ)] を選択して Lambda 関数 (/aws/lambda/ *my-function* など) 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: 失敗したジョブイベントに関する Amazon Simple Notification Service アラートを送信する

このチュートリアルでは、ジョブの状態が `FAILED` に移行したジョブイベントのみをキャプチャする CloudWatch イベント 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して Amazon SNS アラートが正しく設定されていることをテストできます。

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、「AWS Batch の開始方法 (p. 9)」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. [Topics]、[Create new topic] の順に選択します。
3. [トピック名] については、**`JobFailedAlert`** を入力し、[トピックを作成] を選択します。
4. 先ほど作成したトピックを選択します。[トピックの詳細: JobFailedAlert] 画面で、[サブスクリプションの作成] を選択します。
5. [Protocol] で [Email] を選択します。[Endpoint] に、現在利用できるメールアドレスを入力し、[Create subscription] を選択します。
6. 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] 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。

イベントルールを作成するには

1. https://console.aws.amazon.com/cloudwatch/にある CloudWatch コンソールを開きます。
2. ナビゲーションペインで、[Events]、[Create rule] の順に選択します。
3. [Show advanced options]、[edit] の順に選択します。
4. [Build a pattern that selects events for processing by your targets] で、既存のテキストを次のテキストに置き換えます。

```
{
  "detail-type": [
    "Batch Job State Change"
  ],
  "source": [
    "aws.batch"
  ],
  "detail": {
    "status": [
      "FAILED"
    ]
  }
}
```

   このコードでは、ジョブのステータスが `FAILED` であるイベントに一致する CloudWatch イベントルールを定義します。イベントパターンの詳細については、イベントとイベントパターン (Amazon CloudWatch ユーザーガイド) を参照してください。
5. [Targets] で、[Add target] を選択します。[ターゲットの種類] で、[SNS トピック]、[JobFailedAlert] の順に選択します。
6. [Configure details] を選択します。
7. [Rule definition] で、ルールの名前と説明を入力し、[Create rule] 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 (p. 14)」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。

AWS CloudTrail を使用した AWS Batch API コールのログ作成

AWS Batch は、AWS CloudTrail 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である AWS Batch と統合されています。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、リクエストの作成元の IP アドレス、リクエストの実行者、リクエストの実行日時などの詳細を調べて、AWS Batch に対してどのようなリクエストが行われたかを判断できます。

CloudTrail の詳細については、「AWS CloudTrail User Guide」を参照してください。

CloudTrail 内の AWS Batch 情報

CloudTrail は、アカウント作成時に 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)] の AWS の他のサービスのイベントとともに 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 Batchのイベントなど、AWS アカウントのイベントの継続的な記録については、証跡を作成します。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、コンソールで作成した証跡がすべての AWS リージョンに適用されます。証跡では、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がログに記録され、指定した Amazon S3 バケットにログファイルが配信されます。さらに、より詳細な分析と AWS ログで収集されたデータに基づいた行動のためにその他の CloudTrail サービスを設定できます。詳細については、以下を参照してください。

- 証跡を作成するための概要
- CloudTrail でサポートされるサービスと統合
- CloudTrail の Amazon SNS 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」と「複数のアカウントから CloudTrail ログファイルを受け取る」

AWS Batch アクションはすべて CloudTrail によって記録されます。また、これらのアクションは https://docs.aws.amazon.com/batch/latest/APIReference/で説明されています。たとえば、[`SubmitJob`]、[`ListJobs`]、[`DescribeJobs`] セクションの呼び出しは、CloudTrail ログファイルにエントリを生成します。

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。この ID 情報は以下のことを確認するのに役立ちます。

- リクエストが、ルートまたは AWS Identity and Access Management (IAM) ユーザー認証情報のどちらを使用して送信されたかどうか。
- リクエストが、ロールとフェデレーティッドユーザーのどちらの一時的なセキュリティ認証情報を使用して送信されたか.
- リクエストが、別の AWS 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

証跡は、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できる設定です。CloudTrail ログファイルには、1 つ以上のログエントリが含まれます。イベントは任意の送信元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に関する情報が含まれます。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。

以下の例は、`CreateComputeEnvironment` アクションの CloudTrail ログエントリを示しています。


```
{
  "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",
```

```
    "serviceRole": "service-role/AWSBatchServiceRole",
    "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"
}
```

チュートリアル: のパブリックサブネットとプライベートサブネットを持つ VPC を作成する

このチュートリアルでは、2 つのパブリックサブネットと 2 つのプライベートサブネットを持つ VPC の作成を案内します。この VPC では、NAT ゲートウェイを介したインターネットアクセスが可能です。

トピック

ステップ 1: NAT ゲートウェイの Elastic IP アドレスを選択する

NAT ゲートウェイでは、パブリックサブネットで Elastic IP アドレスが必要になりますが、VPC ウィザードによって自動的には作成されません。VPC ウィザードを実行する前に Elastic IP アドレスを作成します。

Elastic IP アドレスを作成するには

1. https://console.aws.amazon.com/vpc/にある Amazon VPC コンソールを開きます。
2. 左のナビゲーションペインで [Elastic IP] を選択します。
3. [新しいアドレスの割り当て]、[割り当て]、[閉じる] の順に選択します。
4. 新しく作成した Elastic IP アドレスの [割り当て ID] を書き留めます。これは後で VPC ウィザードで入力します。

ステップ 2: VPC ウィザードを実行する

VPC ウィザードにより、ほとんどの VPC リソースが自動的に作成および設定されます。

VPC ウィザードを実行するには

1. 左のナビゲーションペインの [VPC ダッシュボード] を選択します。
2. [VPC ウィザードの開始]、[パブリックとプライベートサブネットを持つ VPC]、[選択] の順に選択します。
3. [VPC 名] に、VPC の一意の名前を入力します。
4. [Elastic IP 割り当て ID] で、前に作成した Elastic IP アドレス の ID を選択します。
5. [Create VPC] を選択します。
6. ウィザードが終了したら、[OK] を選択します。VPC サブネットが作成されたアベイラビリティーゾーンを書き留めます。追加のサブネットは別のアベイラビリティーゾーンに作成します。

ステップ 3: 追加のサブネットを作成する

ウィザードでは、1 つのパブリックサブネットと 1 つのプライベートサブネットを持つ VPC が 1 つのアベイラビリティーゾーンに作成されます。可用性を高めるため、別のアベイラビリティーゾーンに各サブネットタイプを少なくとも 1 つ作成し、VPC が 2 つのアベイラビリティーゾーンにまたがって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を持つようにします。

追加のプライベートサブネットを作成するには

1. 左側のナビゲーションペインで [サブネット] を選択します。
2. [Create Subnet] を選択します。
3. [名前タグ] にサブネットの名前を入力します (Private subnet など)。
4. [VPC] で、前の手順で作成した VPC を選択します。
5. [アベイラビリティーゾーン] で、VPC の元のサブネットとは異なるアベイラビリティーゾーンを選択します。
6. [IPv4 CIDR ブロック] に、有効な CIDR ブロックを入力します。たとえば、ウィザードではデフォルトで 10.0.0.0/24 や 10.0.1.0/24 に CIDR ブロックが作成される場合、2 つ目のプライベートサブネットに 10.0.3.0/24 を使用できます。
7. [Yes, Create] を選択します。

追加のパブリックサブネットを作成するには

1. 左のナビゲーションペインで [サブネット]、[サブネットの作成] の順に選択します。
2. [名前タグ] にサブネットの名前を入力します (Public subnet など)。
3. [VPC] で、前の手順で作成した VPC を選択します。
4. [アベイラビリティーゾーン] で、前の手順で作成した同じアベイラビリティーゾーンを、追加のプライベートサブネットとして選択します。
5. [IPv4 CIDR ブロック] に、有効な CIDR ブロックを入力します。たとえば、ウィザードではデフォルトで 10.0.0.0/24 や 10.0.1.0/24 に CIDR ブロックが作成される場合、2 つ目のパブリックサブネットに 10.0.2.0/24 を使用できます。
6. [Yes, Create] を選択します。
7. 先ほど作成したパブリックサブネットを選択し、[ルートテーブル]、[編集] の順に選択します。
8. デフォルトでは、プライベートルートテーブルが選択されます。他の使用可能なルートテーブルを選択し、送信先 0.0.0.0/0 がインターネットゲートウェイ (igw-***xxxxxxxx***) にルーティングされるようにして、[保存] を選択します。
9. 2 つ目のパブリックサブネットを選択したままにし、[Subnet Actions]、[Modify auto-assign IP settings] の順に選択します。
10. [パブリック IPv4 アドレスの自動割り当てを有効にする] を選択し、[保存]、[閉じる] の順に選択します。

次のステップ

VPC を作成したら、以下に示す、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を操作する」を参照してください。

AWS Batch サービスの制限

次の表は、AWS Batch のサービス制限を示していて、これは変更できません。

リソース	制限
ジョブキューの最大数	20
コンピューティング環境の最大数	50
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

AWS Batch のトラブルシューティング

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でトラブルシューティングが必要な場合があります。この章は、AWS Batch 環境におけるトラブルシューティングと問題の解決に役立ちます。

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が不正確に設定されると、INVALID 状態となり、ジョブの配置を受け付けられません。このセクションでは、考えられる原因とその修正方法について説明します。

正しくないロール名または ARN

無効なコンピューティング環境の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正しくない名前、あるいは ARN です。これは、AWS CLI あるいは AWS SDK で作成されたコンピューティング環境に発生しやすい問題です。AWS マネジメントコンソールでコンピューティング環境を作成すると、AWS Batch が正しいサービスまたはスポットフリートロールの選択を援助するため、名前や ARN を誤って入力することがありません。

ただし、AWS CLI コマンドまたは SDK コードで IAM の名前あるいは ARN を手動で入力すると、AWS Batch は文字列を検証できずに不正な値として受取り、環境を作成しようと試みます。環境の作成に失敗したあとで、この環境は INVALID 状態に移り、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
```

この問題によくある原因の 1 つとしては、AWS CLI または AWS SDK に、完全な ARN ではなく、IAM ロールにのみ名前を指定することが挙げられます。これは、ロールを作成した方法によって、ARN に service-role パスプレフィックスが含まれる場合があるからです。たとえば、「AWS Batch サービス IAM ロール (p. 88)」の手順を使用して AWS Batch サービスロールを手動で作成すると、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/service-role/AWSBatchServiceRole
```

AWS CLI または AWS SDK を使用する場合に IAM 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判断します。このため、コンピューティング環境を作成するときには、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の修復 (p. 109)」を参照してください。

INVALID コンピューティング環境の修復

コンピューティング環境が INVALID 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 (p. 108)」に対応する場合、正しいサービスロールで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. AWS Batch コンソール (https://console.aws.amazon.com/batch/) 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、[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] 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、[編集] を選択します。
5. [コンピューティング環境の更新] ページの [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. [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 状態よりも先に進まない場合、実際に配置されるコンピューティングリソースから何かがジョブを妨げています。この問題の一般的な原因を次に示します。

awslogs ログドライバが、コンピューティングリソースで設定されていない

AWS Batch ジョブは、ログ情報を CloudWatch Logs に送信します。この機能を有効にするには、awslogs 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。Amazon ECS に最適化された AMI (または Amazon Linux) で基本コンピューティングリソース AMI をオフにすると、このドライバはデフォルトで ecs-init パッケージに登録されます。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 ログドライバが ECS_AVAILABLE_LOGGING_DRIVERS 環境変数で使用可能なログドライバとして指定されていることを確認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 (p. 58)」および「コンピューティングリソース AMI の作成 (p. 58)」を参照してください。

リソースが不十分である

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 またはメモリーリソースがジョブ定義に指定されていると、ジョブは配置されません。たとえば、ジョブが 4 GiB のメモリーを指定し、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、このジョブをこのコンピューティングリソースに配置することはできません。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、「 メモリ管理 (p. 75)」を参照してください。

コンピューティングリソースのインターネットアクセスがありません

は、Amazon ECS サービスエンドポイントと通信するためのアクセス権限を必要とします。この操作は、インターフェイス VPC エンドポイントを通じて、またはパブリック IP アドレスを持つを通じて実行することができます。

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 Developer Guideの「」を参照してください。

インターフェイス VPC エンドポイントを設定しておらず、にパブリック IP アドレスがない場合、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、の「NAT ゲートウェイ」およびこのガイドの「Amazon VPC ユーザーガイド詳細については、「チュートリアル: のパブリックサブネットとプライベートサブネットを持つ VPC を作成する (p. 105)」を参照してください。

Amazon EC2 インスタンス制限に到達

アカウントで AWS リージョンで起動できる Amazon EC2 インスタンスの数は、EC2イ ンスタンスの制限によって決定されます。特定のインスタンスタイプには、インスタンスタイプごとの制限もあります。アカウントの Amazon EC2 インスタンス制限の詳細 (制限の引き上げをリクエストする方法を含む) については、Linux インスタンス用 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

`RUNNABLE` ステータスで止まっているジョブの診断の詳細については、AWS ナリッジセンターで「AWS Batch ジョブが RUNNABLE ステータスで止まっているのはなぜですか?」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 日にサポートが開始されました。このサポート以前に推奨されていた Amazon EC2 スポットフリート用の 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。

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。その後、そのロールで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. https://console.aws.amazon.com/iam/ にある IAM コンソールを開きます。
2. [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. [ポリシーのアタッチ] を選択します。
4. [AmazonEC2SpotFleetTaggingRole] を選択し、[Attach policy] 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、[Detach] を選択します。

ドキュメント履歴

次の表に、AWS Batch の最初のリリース以後に行われた、ドキュメントの重要な変更を示します。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。

update-history-change	update-history-description	update-history-date
EFA のサポート	AWS Batch によって Elastic Fabric Adapter (EFA) デバイスのサポートが追加されます。	August 2, 2019
GPU スケジューリング	GPU ジョブを使用して各ジョブに必要な GPU の数を指定できます。AWS Batch は、ジョブに応じてインスタンスをスケールアップします。	April 4, 2019
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。	November 19, 2018
リソースレベルのアクセス許可	AWS Batch で複数の 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がサポートされるようになりました。	November 12, 2018
Amazon EC2 起動テンプレートのサポート	AWS Batch に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。	November 12, 2018
AWS Batch 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。	April 5, 2018
CloudWatch イベント ターゲットとしての AWS Batch ジョブ	AWS Batch ジョブは CloudWatch イベント 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、ルールに一致したイベントに応じて AWS Batch ジョブを送信できます。	March 1, 2018
AWS Batch の CloudTrail 監査	CloudTrail では、AWS Batch API アクションに対する呼び出しを監査できます。	January 10, 2018
配列ジョブ	AWS Batch は配列ジョブをサポートします。これはパラメータスイープおよび Monte Carlo ワークロードに役立ちます。	November 28, 2017

拡張された AWS Batch のタグ付け	AWS Batch を使用すると、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。	October 26, 2017
CloudWatch イベント の AWS Batch イベントストリーム	CloudWatch イベント の 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現在の状態に関する通知を、ほぼリアルタイムで受け取ります。	October 24, 2017
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、ジョブが失敗した場合に自動的に再試行できます。	March 28, 2017
AWS Batch の一般提供 (p. 111)	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。	January 5, 2017

AWS の用語集

最新の AWS の用語については、『AWS General Reference』の「AWS の用語集」を参照してください。