デベロッパーガイド

Deadline クラウド

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Deadline クラウド: デベロッパーガイド

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスはAmazon 以外の製品およびサービスに使用することはできません。また、お客様に誤解を与える可能性がある形式で、または Amazon の信用を損なう形式で使用することもできません。Amazon が所有していないその他のすべての商標は Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

Table of Contents

Deadline Cloud とは 1
オープンジョブの説明 2
概念と用語 2
Deadline Cloud ワークロードとは 6
ワークロードが本番環境からどのように発生するか 6
ワークロードの構成要素 7
ワークロードの移植性 8
入門 11
ファームを作成する 11
次のステップ 16
ワーカーエージェントを実行する 16
次のステップ 18
ジョブの送信 18
simple_job サンプルを送信する 19
パラメータを使用して送信する 22
simple_file_job ジョブを作成する 23
次のステップ 26
添付ファイルを含むジョブの送信 27
ジョブアタッチメントのキューを設定する 28
ジョブアタッチメントを使用して送信する 30
ジョブアタッチメントの保存方法 32
次のステップ 36
サービスマネージドフリートを追加する 36
次のステップ 39
ファームリソースをクリーンアップする 39
キュー環境を使用してジョブを設定する 43
ジョブ環境を制御する 44
環境変数を設定する 44
パスを設定する 49
バックグラウンドデーモンプロセスを実行する 53
ジョブにアプリケーションを提供する 59
conda チャネルからアプリケーションを取得する 60
別のパッケージマネージャーを使用する 62
S3 を使用して conda チャネルを作成する 63

パッケージ構築キューを作成する 64
パッケージ構築キューのアクセス許可を設定する 64
カスタム conda パッケージの本番キューのアクセス許可を設定する 65
conda チャネルをキュー環境に追加する 66
アプリケーションの conda パッケージを作成する 67
の conda ビルドレシピを作成する Blender 68
4.2 Blender パッケージジョブを送信する 70
4.2 Blender レンダリングジョブでパッケージをテストする 72
の conda レシピを作成する Maya 73
MtoA プラグインの conda レシピを作成する 76
Maya レンダリングジョブを使用してパッケージをテストする 77
ジョブを構築する 79
ジョブバンドル 80
ジョブテンプレートの要素 83
パラメータ値の要素 86
アセット参照要素 88
ジョブでのファイルの使用 91
サンプルプロジェクトインフラストラクチャ 92
ストレージプロファイルとパスマッピング 94
ジョブアタッチメント 102
ジョブを使用したファイルの送信 103
ジョブからの出力ファイルの取得 114
依存ステップでのファイルの使用 118
ジョブのリソース制限を作成する 120
制限の停止と削除 122
制限を作成する 123
制限とキューを関連付ける 123
制限が必要なジョブを送信する 124
ジョブを送信する 126
ターミナルから 126
スクリプトから 127
アプリケーション内から 128
スケジュールジョブ 130
フリートの互換性を確認する 130
フリートスケーリング 132
セッション 132

ステップの依存関係 ... 135
ジョブの変更 ... 136
カスタマーマネージドフリート ... 141
CMF を作成する ... 141
ワーカーホストのセットアップ ... 146
Python 環境を設定する ... 147
ワーカーエージェントをインストールする ... 148
ワーカーエージェントを設定する ... 150
ジョブのユーザーとグループを作成する ... 151
アクセスを管理する ... 154
アクセス許可の付与 ... 155
アクセスの取り消し ... 155
ジョブ用のソフトウェアをインストールする ... 156
DCC アダプターをインストールする ... 157
認証情報の設定 ... 157
ネットワークの設定 ... 160
ワーカーホストをテストする ... 161
AMI の作成 ... 164
インスタンスを準備する ... 164
の構築 AMI ... 166
フリートインフラストラクチャを作成する ... 167
フリートの自動スケーリング ... 172
フリートのヘルスチェック ... 177
サービスマネージドフリート ... 178
管理スクリプトを実行する ... 178
トラブルシューティング ... 181
ソフトウェアライセンスの使用 ... 183
SMF フリートをライセンスサーバーに接続する ... 183
ステップ 1: キュー環境を設定する ... 184
ステップ 2: (オプション) ライセンスプロキシインスタンスのセットアップ ... 190
ステップ 3: AWS CloudFormation テンプレートのセットアップ ... 192
CMF フリートをライセンスエンドポイントに接続する ... 200
ステップ 1: セキュリティグループを作成する ... 200
ステップ 2: ライセンスエンドポイントを設定する ... 201
ステップ 3: レンダリングアプリケーションをエンドポイントに接続する ... 202
ステップ 4: ライセンスエンドポイントを削除する ... 204

モニタリング 205
CloudTrail ログ 206
Deadline Cloud CloudTrail のデータイベント 208
Deadline Cloud CloudTrail での 管理イベント 210
Deadline Cloud イベントの例 213
CloudWatch によるモニタリング 214
CloudWatch メトリクス 215
を使用したイベントの管理 EventBridge 217
Deadline Cloud イベント 218
Deadline Cloud イベントの送信 219
イベントの詳細リファレンス 220
セキュリティ 235
データ保護 236
保管中の暗号化 237
転送中の暗号化 237
キー管理 237
ネットワーク間トラフィックのプライバシー 247
オプトアウト 248
Identity and Access Management 249
対象者 249
アイデンティティを使用した認証 250
ポリシーを使用したアクセスの管理 254
Deadline Cloud と IAM の連携方法 256
アイデンティティベースのポリシーの例 263
AWS マネージドポリシー 267
トラブルシューティング 271
コンプライアンス検証 273
耐障害性 275
インフラストラクチャセキュリティ 275
設定と脆弱性の分析 276
サービス間での不分別な代理処理の防止 276
AWS PrivateLink 278
考慮事項 278
Deadline Cloud エンドポイント 279
エンドポイントの作成 279
セキュリティに関するベストプラクティス 280

データ保護 281
IAM 許可 281
ユーザーおよびグループとしてジョブを実行する 282
ネットワーク 282
ジョブデータ 283
ファーム構造 283
ジョブアタッチメントキュー 284
カスタムソフトウェアバケット 286
ワーカーホスト 286
ホスト設定スクリプト 288
ワークステーション 288
ダウンロードしたソフトウェアを検証する 289
ドキュメント履歴 295
..... ccxcvii

AWS Deadline Cloud とは

AWS Deadline Cloud は、スケーラブルな処理ファームを数分で稼働させることができるフルマネージド AWS サービスです。ユーザー、ファーム、ジョブをスケジュールするためのキュー、および処理を行うワーカーのフリートを管理するための管理コンソールを提供します。

このデベロッパーガイドは、次のような幅広いユースケースのパイプライン、ツール、アプリケーションデベロッパーを対象としています。

- パイプライン開発者とテクニカルディレクターは、Deadline Cloud APIs と機能をカスタム本番パイプラインに統合できます。
- 独立系ソフトウェアベンダーは Deadline Cloud をアプリケーションに統合できるため、デジタルコンテンツ作成アーティストやユーザーは、ワークステーションから Deadline Cloud レンダージョブをシームレスに送信できます。
- ウェブおよびクラウドベースのサービスデベロッパーは、Deadline Cloud レンダリングをプラットフォームに統合できるため、お客様はアセットを提供して製品を仮想的に表示できます。

パイプラインの任意のステップを直接操作できるツールが用意されています。

- 直接またはスクリプトから使用できるコマンドラインインターフェイス。
- AWS SDK for 11 の一般的なプログラミング言語。
- アプリケーションから呼び出すことができる REST ベースのウェブインターフェイス。

AWS のサービス カスタムアプリケーションで他の を使用することもできます。例えば、次を使用できます。

- AWS CloudFormation は、ファーム、キュー、フリートの作成と削除を自動化します。
- ジョブのメトリクスを収集する Amazon CloudWatch。
- デジタルアセットとジョブ出力を保存および管理するための Amazon Simple Storage Service。
- AWS IAM Identity Center は、ファームのユーザーとグループを管理します。

オープンジョブの説明

Deadline Cloud は、Open Job Description (OpenJD) 仕様を使用してジョブの詳細を指定します。OpenJD は、ソリューション間で移植可能なジョブを定義するために開発されました。これを使用して、ワーカーホストで実行される一連のコマンドであるジョブを定義します。

Deadline Cloud が提供する送信者を使用して OpenJD ジョブテンプレートを作成することも、テンプレートを作成する任意のツールを使用することもできます。テンプレートを作成したら、Deadline Cloud に送信します。送信者を使用すると、テンプレートの送信が処理されます。別の方法でテンプレートを作成した場合は、Deadline Cloud コマンドラインアクションを呼び出すか、いずれかの AWS SDKs を使用してジョブを送信できます。いずれの場合も、Deadline Cloud は指定されたキューにジョブを追加し、作業をスケジュールします。

Deadline Cloud の概念と用語

Deadline Cloud AWS の開始に役立つように、このトピックではその主要な概念と用語の一部について説明します。

予算マネージャー

Budget Manager は Deadline Cloud モニターの一部です。予算マネージャーを使用して、予算を作成および管理します。また、これを使用してアクティビティを制限し、予算内にとどまることもできます。

Deadline クラウドクライアントライブラリ

クライアントライブラリには、Deadline Cloud を管理するためのコマンドラインインターフェイスとライブラリが含まれています。機能には、Open Job Description 仕様に基づくジョブバンドルの Deadline Cloud への送信、ジョブアタッチメント出力のダウンロード、コマンドラインインターフェイスを使用したファームのモニタリングが含まれます。

デジタルコンテンツ作成アプリケーション (DCC)

デジタルコンテンツ作成アプリケーション (DCCs) は、デジタルコンテンツを作成するサードパーティー製品です。DCCsはMaya、、Nuke、および ですHoudini。Deadline Cloud は、特定のDCCs。

ファーム

ファームは、プロジェクトリソースが配置されている です。キューとフリートで構成されます。

フリート

フリートは、レンダリングを実行するワーカーノードのグループです。ワーカーノードはジョブを処理します。フリートは複数のキューに関連付けることができ、キューは複数のフリートに関連付けることができます。

ジョブ

ジョブはレンダリングリクエストです。ユーザーはジョブを送信します。ジョブには、ステップとタスクとして概説されている特定のジョブプロパティが含まれています。

ジョブアタッチメント

ジョブアタッチメントは、ジョブの入出力を管理するために使用できる Deadline Cloud 機能です。ジョブファイルは、レンダリングプロセス中にジョブアタッチメントとしてアップロードされます。これらのファイルは、テクスチャ、3D モデル、照明リグ、その他の類似アイテムです。

ジョブの優先度

ジョブの優先度は、Deadline Cloud がキュー内のジョブを処理するおおよその順序です。ジョブの優先度は 1 ~ 100 の間で設定できます。優先度の高いジョブは通常、最初に処理されます。優先度が同じジョブは、受信した順序で処理されます。

ジョブプロパティ

ジョブプロパティは、レンダリングジョブを送信するときに定義する設定です。例としては、フレーム範囲、出力パス、ジョブアタッチメント、レンダリング可能なカメラなどがあります。プロパティは、レンダーの送信元の DCC によって異なります。

ジョブテンプレート

ジョブテンプレートは、ランタイム環境と、Deadline Cloud ジョブの一部として実行されるすべてのプロセスを定義します。

キュー

キューは、送信されたジョブが配置され、レンダリングがスケジュールされる場所です。正常なレンダリングを作成するには、キューをフリートに関連付ける必要があります。キューは複数のフリートに関連付けることができます。

キューフリートの関連付け

キューがフリートに関連付けられている場合、キューとフリートの関連付けがあります。関連付けを使用して、フリートからそのキュー内のジョブにワーカーをスケジュールします。関連付けを開始および停止して、作業のスケジュールを制御できます。

セッション

セッションは、同じジョブから一連のタスクを実行するために作成されたワーカーホスト上の一時的なランタイム環境です。セッションは、ワーカーホストがそのジョブの実行タスクを完了すると終了します。

このセッションでは、環境変数の定義やバックグラウンドプロセスやコンテナの開始など、複数のタスク実行で共有されるリソースを使用して環境を設定できます。

セッションアクション

セッションアクションは、セッション内のワーカーによって実行される個別の作業単位です。これには、タスクのコアタスク実行オペレーションを含めることも、環境設定などの準備ステップや、ティアダウンやクリーンアップなどの実行後プロセスを含めることもできます。

Step

ステップは、ジョブで実行する特定のプロセスの 1 つです。

Deadline Cloud 送信者

Deadline Cloud 送信者は、デジタルコンテンツ作成 (DCC) プラグインです。アーティストはこれを使用して、使い慣れたサードパーティーの DCC インターフェイスからジョブを送信します。

[タグ]

タグは、 AWS リソースに割り当てることができるラベルです。各タグは、お客様が定義するキーとオプション値で構成されています。

タグを使用すると、 AWS リソースをさまざまな方法で分類できます。例えば、各インスタンスの所有者とスタックレベルを追跡しやすくするため、アカウントの Amazon EC2 インスタンスに対してタグセットを定義できます。

AWS リソースを目的、所有者、または環境別に分類することもできます。このアプローチは、同じタイプのリソースが多数ある場合に便利です。割り当てたタグに基づいて、特定のリソースをすばやく識別できます。

タスク

タスクはレンダリングステップの 1 つのコンポーネントです。

使用量ベースのライセンス (UBL)

使用量ベースのライセンス (UBL) は、一部のサードパーティー製品で使用できるオンデマンドライセンスモデルです。このモデルは従量制料金で、使用した時間と分数に対して課金されます。

使用状況エクスプローラー

Usage Explorer は Deadline Cloud Monitor の機能です。コストと使用量のおおよその見積もりを提供します。

ワーカー

ワーカーはフリートに属し、Deadline Cloud に割り当てられたタスクを実行してステップとジョブを完了します。ワーカーは、タスクオペレーションのログを Amazon CloudWatch Logs に保存します。ワーカーは、ジョブアタッチメント機能を使用して、入力と出力を Amazon Simple Storage Service (Amazon S3) バケットに同期することもできます。

Deadline Cloud ワークロードとは

AWS Deadline Cloud を使用すると、ジョブを送信してクラウドでアプリケーションを実行し、データを処理して、ビジネスにとって重要なコンテンツやインサイトを生成できます。Deadline Cloud は、Open Job Description (OpenJD) をジョブテンプレートの構文として使用します。これは、ビジュアルコンピューティングパイプラインのニーズ向けに設計された仕様ですが、他の多くのユースケースに適用されます。ワークロードの例には、コンピュータグラフィックスのレンダリング、物理シミュレーション、写真測量などがあります。

ワークロードは、ユーザーが CLI または自動生成された GUI を使用してキューに送信するシンプルなジョブバンドルから、アプリケーション定義のワークロードのジョブバンドルを動的に生成する統合送信者プラグインまでスケールされます。

ワークロードが本番環境からどのように発生するか

本番稼働環境のワークロードと Deadline Cloud でワークロードをサポートする方法を理解するには、ワークロードがどのようになるかを検討してください。制作には、視覚効果、アニメーション、ゲーム、製品カタログイメージ、情報モデリング (BIM) を構築するための 3D 再構築の作成などが含まれます。このコンテンツは通常、さまざまなソフトウェアアプリケーションとカスタムスクリプトを実行する芸術的または技術的なスペシャリストのチームによって作成されます。チームのメンバーは、本番パイプラインを使用して相互にデータを渡します。パイプラインによって実行されるタスクの多くは、ユーザーのワークステーションで実行されると数日かかる大量の計算を伴います。

これらの本番パイプラインのタスクの例には、次のようなものがあります。

- 写真測量アプリケーションを使用して、テクスチャデジタルメッシュを再構築するために映画セットの撮影された写真を処理します。
- 3D シーンでパーティクルシミュレーションを実行して、テレビ番組の爆発ビジュアル効果に詳細レイヤーを追加します。
- ゲームレベルのデータを外部リリースに必要な形式にクックし、最適化と圧縮の設定を適用します。
- 色、背景、照明のバリエーションなど、製品カタログの一連のイメージをレンダリングします。
- カスタム開発スクリプトを 3D モデルで実行して、カスタムビルドされ、映画監督によって承認された外観を適用します。

これらのタスクには、アーティスティックな結果を取得したり、出力品質を微調整したりするために調整する多くのパラメータが含まれます。多くの場合、アプリケーション内でローカルでプロセス

を実行するボタンまたはメニューを使用して、これらのパラメータ値を選択する GUI があります。ユーザーがプロセスを実行すると、アプリケーションと場合によってはホストコンピュータ自体を他のオペレーションの実行に使用することはできません。これは、アプリケーションの状態をメモリで使用し、ホストコンピュータのすべての CPU とメモリリソースを消費する可能性があるためです。

多くの場合、プロセスは迅速です。本番稼働中、品質と複雑さの要件が高まると、プロセスの速度が遅くなります。開発中に 30 秒かかったキャラクターテストは、最終的な本番稼働用キャラクターに適用されると、簡単に 3 時間になります。この進行により、GUI 内で存続を開始したワークロードが大きすぎて収まらない可能性があります。Deadline Cloud に移植すると、ワークステーションを完全に制御し直し、Deadline Cloud モニターからより多くの反復を追跡できるため、これらのプロセスを実行しているユーザーの生産性を高めることができます。

Deadline Cloud でワークロードのサポートを開発する際に目指すサポートには、次の 2 つのレベルがあります。

- 並列処理や高速化なしで、ワークロードをユーザーワークステーションから Deadline Cloud ファームにオフロードします。これにより、ファームで使用可能なコンピューティングリソースが十分に活用されない場合がありますが、長時間のオペレーションをバッチ処理システムに移行できるため、ユーザーは自分のワークステーションでより多くの作業を行うことができます。
- Deadline Cloud ファームの水平スケールを利用して迅速に完了できるように、ワークロードの並列処理を最適化します。

ワークロードを並行して実行する方法が明らかになる場合があります。たとえば、コンピュータグラフィックスレンダリングの各フレームは個別に実行できます。ただし、この並列処理にとらわれないようにすることが重要です。代わりに、長時間実行されるワークロードを Deadline Cloud にオフロードすると、ワークロードを分割する明確な方法がない場合でも、大きな利点が得られることを理解してください。

ワークロードの構成要素

Deadline Cloud ワークロードを指定するには、ユーザーが Deadline Cloud CLI を使用してキューに送信するジョブバンドルを実装します。ジョブバンドルの作成作業の多くはジョブテンプレートの作成ですが、ワークロードに必要なアプリケーションを提供する方法など、さらに多くの要因があります。Deadline Cloud のワークロードを定義する際に考慮すべき重要な事項は次のとおりです。

- 実行するアプリケーション。ジョブはアプリケーションプロセスを起動できる必要があるため、利用可能なアプリケーションのインストールと、フローティングライセンスサーバーへのアクセス

など、アプリケーションが使用するライセンスが必要です。これは通常、ファーム設定の一部であり、ジョブバンドル自体に埋め込まれていません。
  - キュー環境を使用してジョブを設定する
  - カスタマーマネージドフリートをライセンスエンドポイントに接続する
- ジョブパラメータの定義。ジョブを送信するユーザーエクスペリエンスは、ジョブが提供するパラメータによって大きく影響を受けます。パラメータの例には、データファイル、ディレクトリ、アプリケーション設定などがあります。
  - ジョブバンドルのパラメータ値要素
- ファイルデータフロー。ジョブを実行すると、ユーザーから提供されたファイルから入力を読み取り、その出力を新しいファイルとして書き込みます。ジョブアタッチメントとパスマッピング機能を使用するには、ジョブでこれらの入力と出力のディレクトリまたは特定のファイルのパスを指定する必要があります。
  - ジョブでのファイルの使用
- ステップスクリプト。ステップスクリプトは、適切なコマンドラインオプションを使用してアプリケーションバイナリを実行して、指定されたジョブパラメータを適用します。また、ワークロードデータファイルに相対パス参照ではなく絶対パスが含まれている場合、パスマッピングなどの詳細も処理します。
  - ジョブバンドルのジョブテンプレート要素

ワークロードの移植性

ワークロードは、ジョブを送信するたびに変更することなく、複数の異なるシステムで実行できる場合に移植可能です。たとえば、異なる共有ファイルシステムがマウントされている異なるレンダーファーム、または Linuxや などの異なるオペレーティングシステムで実行できますWindows。ポータブルジョブバンドルを実装すると、ユーザーは特定のファームでジョブを実行したり、他のユースケースに適応したりしやすくなります。

ジョブバンドルをポータブルにする方法をいくつか紹介します。

- ジョブバンドル内のPATHジョブパラメータとアセットリファレンスを使用して、ワークロードに必要な入力データファイルを完全に指定します。これにより、ジョブは共有ファイルシステムに基づくファームや、Deadline Cloud ジョブアタッチメント機能などの入力データのコピーを作成するファームに移植できます。
- ジョブの入力ファイル用のファイルパス参照を再配置でき、さまざまなオペレーティングシステムで使用できるようにします。たとえば、ユーザーがWindowsワークステーションからジョブを送信してLinuxフリートで実行する場合などです。

  - 相対ファイルパス参照を使用するため、それらを含むディレクトリが別の場所に移動された場合でも、参照は解決されます。Blender などの一部のアプリケーションは、相対パスと絶対パスの選択をサポートしています。
  - 相対パスを使用できない場合は、OpenJD パスマッピングメタデータをサポートし、Deadline Cloud がジョブにファイルを提供する方法に従って絶対パスを変換します。
- ポータブルスクリプトを使用してジョブにコマンドを実装します。Python と bash は、この方法で使用できるスクリプト言語の 2 つの例です。フリートのすべてのワーカーホストで両方を提供することを検討する必要があります。
  - スクリプトファイル名を引数として、 `python`や などの`bash`スクリプトインタープリタバイナリを使用します。これは、 で実行ビットが設定されたスクリプトファイルを使用する場合と比較してWindows、 を含むすべてのオペレーティングシステムで機能しますLinux。
  - 以下のプラクティスを適用してポータブル bash スクリプトを作成します。
    - テンプレートパスパラメータを一重引用符で展開して、スペースとパス区切り文字を含むWindowsパスを処理します。
    - で実行する場合はWindows、MinGW 自動パス変換に関連する問題に注意してください。たとえば、 のような AWS CLI コマンド`aws logs tail /aws/deadline/...`を のようなコマンドに変換`aws logs tail "C:/Program Files/Git/aws/deadline/..."`し、ログを正しく末尾に付けません。変数を設定`MSYS_NO_PATHCONV=1`して、この動作をオフにします。
    - ほとんどの場合、同じコードはすべてのオペレーティングシステムで機能します。コードが異なる必要がある場合は、 `if/else`コンストラクトを使用してケースを処理します。

```
if [[ "$(uname)" == MINGW* || "$(uname -s)" == MSYS_NT* ]]; then
    # Code for Windows
elif [[ "$(uname)" == Darwin ]]; then
    # Code for MacOS
else
    # Code for Linux and other operating systems
fi
```

  - を使用してポータブル Python スクリプトを記述`pathlib`することで、ファイルシステムのパスの違いを処理し、運用固有の機能を回避できます。Python ドキュメントには、シグナルライブラリドキュメントなど、これに関する注釈が含まれています。 Linux固有の機能サポートは、「可用性: Linux」とマークされています。
- ジョブパラメータを使用して、アプリケーション要件を指定します。ファーム管理者がキュー環境に適用できる一貫した規則を使用します。
  - たとえば、ジョブで `CondaPackages`および/または `RezPackages`パラメータを、ジョブに必要なアプリケーションパッケージ名とバージョンを一覧表示するデフォルトのパラメータ値と

ともに使用できます。次に、サンプルの Conda または Rez キュー環境のいずれかを使用して、ジョブの仮想環境を提供できます。

Deadline Cloud リソースの開始方法

AWS Deadline Cloud のカスタムソリューションの作成を開始するには、 リソースを設定する必要があります。これには、ファーム、ファーム用の少なくとも 1 つのキュー、キューを処理するための少なくとも 1 つのワーカーフリートが含まれます。Deadline Cloud コンソールを使用してリソースを作成することも、 を使用することもできます AWS Command Line Interface。

このチュートリアルでは、 AWS CloudShell を使用してシンプルなデベロッパーファームを作成し、ワーカーエージェントを実行します。その後、パラメータとアタッチメントを使用してシンプルなジョブを送信して実行し、サービスマネージドフリートを追加し、完了したらファームリソースをクリーンアップできます。

以下のセクションでは、Deadline Cloud のさまざまな機能と、それらの機能および連携方法について説明します。これらのステップに従うことは、新しいワークロードとカスタマイズの開発とテストに役立ちます。

コンソールを使用してファームを設定する手順については、「Deadline Cloud ユーザーガイド」の「開始方法」を参照してください。

トピック

- Deadline Cloud ファームを作成する
- Deadline Cloud ワーカーエージェントを実行する
- Deadline Cloud で送信する
- Deadline Cloud でジョブアタッチメントを使用してジョブを送信する
- Deadline Cloud の開発者ファームにサービスマネージドフリートを追加する
- Deadline Cloud でファームリソースをクリーンアップする

Deadline Cloud ファームを作成する

AWS Deadline Cloud でデベロッパーファームとキューリソースを作成するには、次の手順に示すように AWS Command Line Interface （ AWS CLI) を使用します。また、 AWS Identity and Access Management (IAM) ロールとカスタマーマネージドフリート (CMF) を作成し、フリートをキューに関連付けます。次に、 を設定し AWS CLI 、ファームが指定されたとおりに設定され、動作していることを確認できます。

このファームを使用して Deadline Cloud の機能を調べ、新しいワークロード、カスタマイズ、パイプライン統合を開発してテストできます。

ファームを作成するには

1. セッションを開きます AWS CloudShell。CloudShell ウィンドウを使用して AWS Command Line Interface（AWS CLI) コマンドを入力し、このチュートリアルの例を実行します。CloudShell ウィンドウを開いたままにしておきます。
2. ファームの名前を作成し、そのファーム名を に追加します`~/.bashrc`。これにより、他のターミナルセッションで利用できるようになります。

```
echo "DEV_FARM_NAME=DeveloperFarm" >> ~/.bashrc
source ~/.bashrc
```

3. ファームリソースを作成し、そのファーム ID を に追加します`~/.bashrc`。

```
aws deadline create-farm \
    --display-name "$DEV_FARM_NAME"

echo "DEV_FARM_ID=\$(aws deadline list-farms \
        --query \"farms[?displayName=='\$DEV_FARM_NAME'].farmId \
        | [0]\" --output text)" >> ~/.bashrc
source ~/.bashrc
```

4. キューリソースを作成し、そのキュー ID を に追加します。 `~/.bashrc.`

```
aws deadline create-queue \
    --farm-id $DEV_FARM_ID \
    --display-name "$DEV_FARM_NAME Queue" \
    --job-run-as-user '{"posix": {"user": "job-user", "group": "job-group"},
 "runAs":"QUEUE_CONFIGURED_USER"}'

echo "DEV_QUEUE_ID=\$(aws deadline list-queues \
        --farm-id \$DEV_FARM_ID \
        --query \"queues[?displayName=='\$DEV_FARM_NAME Queue'].queueId \
        | [0]\" --output text)" >> ~/.bashrc
source ~/.bashrc
```

5. フリートの IAM ロールを作成します。このロールは、フリートのワーカーホストに、キューからジョブを実行するために必要なセキュリティ認証情報を提供します。

```
aws iam create-role \
    --role-name "${DEV_FARM_NAME}FleetRole" \
    --assume-role-policy-document \
        '{
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "credentials.deadline.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        }'
aws iam put-role-policy \
    --role-name "${DEV_FARM_NAME}FleetRole" \
    --policy-name WorkerPermissions \
    --policy-document \
        '{
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Action": [
                        "deadline:AssumeFleetRoleForWorker",
                        "deadline:UpdateWorker",
                        "deadline:DeleteWorker",
                        "deadline:UpdateWorkerSchedule",
                        "deadline:BatchGetJobEntity",
                        "deadline:AssumeQueueRoleForWorker"
                    ],
                    "Resource": "*",
                    "Condition": {
                        "StringEquals": {
                            "aws:PrincipalAccount": "${aws:ResourceAccount}"
                        }
                    }
                },
                {
                    "Effect": "Allow",
                    "Action": [
                        "logs:CreateLogStream"
```

```
                    ],
                    "Resource": "arn:aws:logs:*:*:*:/aws/deadline/*",
                    "Condition": {
                        "StringEquals": {
                            "aws:PrincipalAccount": "${aws:ResourceAccount}"
                        }
                    }
                },
                {
                    "Effect": "Allow",
                    "Action": [
                        "logs:PutLogEvents",
                        "logs:GetLogEvents"
                    ],
                    "Resource": "arn:aws:logs:*:*:*:/aws/deadline/*",
                    "Condition": {
                        "StringEquals": {
                            "aws:PrincipalAccount": "${aws:ResourceAccount}"
                        }
                    }
                }
            ]
        }'
```

6. カスタマーマネージドフリート (CMF) を作成し、そのフリート ID を に追加します`~/.bashrc`。

```
FLEET_ROLE_ARN="arn:aws:iam::$(aws sts get-caller-identity \
        --query "Account" --output text):role/${DEV_FARM_NAME}FleetRole"
aws deadline create-fleet \
    --farm-id $DEV_FARM_ID \
    --display-name "$DEV_FARM_NAME CMF" \
    --role-arn $FLEET_ROLE_ARN \
    --max-worker-count 5 \
    --configuration \
        '{
            "customerManaged": {
                "mode": "NO_SCALING",
                "workerCapabilities": {
                    "vCpuCount": {"min": 1},
                    "memoryMiB": {"min": 512},
                    "osFamily": "linux",
                    "cpuArchitectureType": "x86_64"
```

```
                }
            }
        }'

echo "DEV_CMF_ID=\$(aws deadline list-fleets \
        --farm-id \$DEV_FARM_ID \
        --query \"fleets[?displayName=='\$DEV_FARM_NAME CMF'].fleetId \
        | [0]\" --output text)" >> ~/.bashrc
source ~/.bashrc
```

7. CMF をキューに関連付けます。

```
aws deadline create-queue-fleet-association \
    --farm-id $DEV_FARM_ID \
    --queue-id $DEV_QUEUE_ID \
    --fleet-id $DEV_CMF_ID
```

8. Deadline Cloud コマンドラインインターフェイスをインストールします。

```
pip install deadline
```

9. デフォルトのファームをファーム ID に設定し、キューを前に作成したキュー ID に設定するには、次のコマンドを使用します。

```
deadline config set defaults.farm_id $DEV_FARM_ID
deadline config set defaults.queue_id $DEV_QUEUE_ID
```

10. (オプション) ファームが仕様に従って設定されていることを確認するには、次のコマンドを使用します。

    - すべてのファームを一覧表示する – **`deadline farm list`**
    - デフォルトファーム内のすべてのキューを一覧表示する – **`deadline queue list`**
    - デフォルトファーム内のすべてのフリートを一覧表示する – **`deadline fleet list`**
    - デフォルトのファームを取得する – **`deadline farm get`**
    - デフォルトのキューを取得する – **`deadline queue get`**
    - デフォルトキューに関連付けられているすべてのフリートを取得する – **`deadline fleet get`**

次のステップ

ファームを作成したら、フリートのホストで Deadline Cloud ワーカーエージェントを実行してジョブを処理できます。「Deadline Cloud ワーカーエージェントを実行する」を参照してください。

Deadline Cloud ワーカーエージェントを実行する

デベロッパーファームのキューに送信するジョブを実行する前に、ワーカーホストでデベロッパーモードで Deadline Cloud AWS ワーカーエージェントを実行する必要があります。

このチュートリアルの残りの部分では、2 つの AWS CloudShell タブを使用して開発者ファームで AWS CLI オペレーションを実行します。最初のタブでは、ジョブを送信できます。2 番目のタブでは、ワーカーエージェントを実行できます。

> **Note**
>
> CloudShell セッションを 20 分以上アイドル状態にすると、ワーカーエージェントはタイムアウトして停止します。ワーカーエージェントを再起動するには、以下の手順に従います。

ワーカーエージェントを起動する前に、Deadline Cloud ファーム、キュー、フリートを設定する必要があります。「Deadline Cloud ファームを作成する」を参照してください。

デベロッパーモードでワーカーエージェントを実行するには

1. ファームを最初の CloudShell タブで開いたままにして、2 番目の CloudShell タブを開き、`demoenv-logs`および `demoenv-persist` ディレクトリを作成します。

   ```
   mkdir ~/demoenv-logs
   mkdir ~/demoenv-persist
   ```

2. PyPI から Deadline Cloud ワーカーエージェントパッケージをダウンロードしてインストールします。

   > **Note**
   >
   > ではWindows、エージェントファイルが Python のグローバル site-packages ディレクトリにインストールされている必要があります。Python 仮想環境は現在サポートされていません。

```
python -m pip install deadline-cloud-worker-agent
```

3. ワーカーエージェントがジョブを実行するための一時ディレクトリを作成できるようにするには、 ディレクトリを作成します。

```
sudo mkdir /sessions
sudo chmod 750 /sessions
sudo chown cloudshell-user /sessions
```

4. Deadline Cloud ワーカーエージェントを、 に追加`DEV_CMF_ID`した変数 `DEV_FARM_ID`と を使用して開発者モードで実行します`~/.bashrc`。

```
deadline-worker-agent \
    --farm-id $DEV_FARM_ID \
    --fleet-id $DEV_CMF_ID \
    --run-jobs-as-agent-user \
    --logs-dir ~/demoenv-logs \
    --persistence-dir ~/demoenv-persist
```

ワーカーエージェントが初期化して `UpdateWorkerSchedule` API オペレーションをポーリングすると、次の出力が表示されます。

```
INFO    Worker Agent starting
[2024-03-27 15:51:01,292][INFO    ] # Worker Agent starting
[2024-03-27 15:51:01,292][INFO    ] AgentInfo
Python Interpreter: /usr/bin/python3
Python Version: 3.9.16 (main, Sep  8 2023, 00:00:00)  - [GCC 11.4.1 20230605 (Red
 Hat 11.4.1-2)]
Platform: linux
...
[2024-03-27 15:51:02,528][INFO    ] # API.Resp # [deadline:UpdateWorkerSchedule]
(200) params={'assignedSessions': {}, 'cancelSessionActions': {},
 'updateIntervalSeconds': 15} ...
[2024-03-27 15:51:17,635][INFO    ] # API.Resp # [deadline:UpdateWorkerSchedule]
(200) params=(Duplicate removed, see previous response) ...
[2024-03-27 15:51:32,756][INFO    ] # API.Resp # [deadline:UpdateWorkerSchedule]
(200) params=(Duplicate removed, see previous response) ...
...
```

5. 最初の CloudShell タブを選択し、フリート内のワーカーを一覧表示します。

```
deadline worker list --fleet-id $DEV_CMF_ID
```

次のような出力が表示されます。

```
Displaying 1 of 1 workers starting at 0

- workerId: worker-8c9af877c8734e89914047111f
  status: STARTED
  createdAt: 2023-12-13 20:43:06+00:00
```

本番稼働用設定では、Deadline Cloud ワーカーエージェントは、ホストマシンの管理ユーザーとして複数のユーザーと設定ディレクトリを設定する必要があります。アクセス可能な独自の開発ファームでジョブを実行しているため、これらの設定を上書きできます。

次のステップ

ワーカーホストでワーカーエージェントが実行されたので、ワーカーにジョブを送信できます。次のようにできます：

- Deadline Cloud で送信する シンプルな OpenJD ジョブバンドルを使用する。
- Deadline Cloud でジョブアタッチメントを使用してジョブを送信する 異なるオペレーティングシステムを使用してワークステーション間でファイルを共有する 。

Deadline Cloud で送信する

ワーカーホストで Deadline Cloud ジョブを実行するには、Open Job Description (OpenJD) ジョブバンドルを作成して使用してジョブを設定します。バンドルは、ジョブの入力ファイルやジョブの出力を書き込む場所を指定するなどして、ジョブを設定します。このトピックには、ジョブバンドルを設定する方法の例が含まれています。

このセクションの手順に従う前に、以下を完了する必要があります。

- Deadline Cloud ファームを作成する
- Deadline Cloud ワーカーエージェントを実行する

AWS Deadline Cloud を使用してジョブを実行するには、次の手順を使用します。最初の AWS CloudShell タブを使用して、デベロッパーファームにジョブを送信します。2 番目の CloudShell タブを使用して、ワーカーエージェントの出力を表示します。

トピック

- simple_job サンプルを送信する
- パラメータsimple_jobを使用して を送信する
- ファイル I/O を使用して simple_file_job ジョブバンドルを作成する
- 次のステップ

simple_job サンプルを送信する

ファームを作成してワーカーエージェントを実行したら、simple_jobサンプルを Deadline Cloud に送信できます。

Deadline Cloud にsimple_jobサンプルを送信するには

1. 最初の CloudShell タブを選択します。
2. GitHub からサンプルをダウンロードします。

```
cd ~
git clone https://github.com/aws-deadline/deadline-cloud-samples.git
```

3. ジョブバンドルサンプルディレクトリに移動します。

```
cd ~/deadline-cloud-samples/job_bundles/
```

4. simple_job サンプルを送信します。

```
deadline bundle submit simple_job
```

5. 2 番目の CloudShell タブを選択すると、 の呼び出しBatchGetJobEntities、セッションの取得、セッションアクションの実行に関するログ記録出力が表示されます。

```
...
[2024-03-27 16:00:21,846][INFO    ] # Session.Starting
 # [session-053d77cef82648fe2] Starting new Session.
 [queue-3ba4ff683ff54db09b851a2ed8327d7b/job-d34cc98a6e234b6f82577940ab4f76c6]
```

```
[2024-03-27 16:00:21,853][INFO    ] # API.Req # [deadline:BatchGetJobEntity]
 resource={'farm-id': 'farm-3e24cfc9bbcd423e9c1b6754bc1',
 'fleet-id': 'fleet-246ee60f46d44559b6cce010d05', 'worker-id':
 'worker-75e0fce9c3c344a69bff57fcd83'} params={'identifiers': [{'jobDetails':
 {'jobId': 'job-d34cc98a6e234b6f82577940ab4'}}]} request_url=https://
scheduling.deadline.us-west-2.amazonaws.com/2023-10-12/farms/
farm-3e24cfc9bbcd423e /fleets/fleet-246ee60f46d44559b1 /workers/worker-
 75e0fce9c3c344a69b /batchGetJobEntity
[2024-03-27 16:00:22,013][INFO    ] # API.Resp # [deadline:BatchGetJobEntity](200)
 params={'entities': [{'jobDetails': {'jobId': 'job-d34cc98a6e234b6f82577940ab6',
 'jobRunAsUser': {'posix': {'user': 'job-user', 'group': 'job-group'},
 'runAs': 'QUEUE_CONFIGURED_USER'}, 'logGroupName': '/aws/deadline/
farm-3e24cfc9bbcd423e9c1b6754bc1/queue-3ba4ff683ff54db09b851a2ed83', 'parameters':
 '*REDACTED*', 'schemaVersion': 'jobtemplate-2023-09'}}], 'errors': []}
 request_id=a3f55914-6470-439e-89e5-313f0c6
[2024-03-27 16:00:22,013][INFO    ] # Session.Add #
 [session-053d77cef82648fea9c69827182] Appended new SessionActions.
 (ActionIds: ['sessionaction-053d77cef82648fea9c69827182-0'])
 [queue-3ba4ff683ff54db09b851a2ed8b/job-d34cc98a6e234b6f82577940ab6]
[2024-03-27 16:00:22,014][WARNING ] # Session.User #
 [session-053d77cef82648fea9c69827182] Running as the Worker Agent's
 user. (User: cloudshell-user) [queue-3ba4ff683ff54db09b851a2ed8b/job-
d34cc98a6e234b6f82577940ac6]
[2024-03-27 16:00:22,015][WARNING ] # Session.AWSCreds #
 [session-053d77cef82648fea9c69827182] AWS Credentials are not available: Queue has
 no IAM Role. [queue-3ba4ff683ff54db09b851a2ed8b/job-d34cc98a6e234b6f82577940ab6]
[2024-03-27 16:00:22,026][INFO    ] # Session.Logs #
 [session-053d77cef82648fea9c69827182] Logs streamed to: AWS CloudWatch
 Logs. (LogDestination: /aws/deadline/farm-3e24cfc9bbcd423e9c1b6754bc1/
queue-3ba4ff683ff54db09b851a2ed83/session-053d77cef82648fea9c69827181)
 [queue-3ba4ff683ff54db09b851a2ed83/job-d34cc98a6e234b6f82577940ab4]
[2024-03-27 16:00:22,026][INFO    ] # Session.Logs #
 [session-053d77cef82648fea9c69827182] Logs streamed to: local
 file. (LogDestination: /home/cloudshell-user/demoenv-logs/
queue-3ba4ff683ff54db09b851a2ed8b/session-053d77cef82648fea9c69827182.log)
 [queue-3ba4ff683ff54db09b851a2ed83/job-d34cc98a6e234b6f82577940ab4]
...
```

> **Note**
>
> ワーカーエージェントからのログ記録出力のみが表示されます。ジョブを実行するセッションには別のログがあります。

6. 最初のタブを選択し、ワーカーエージェントが書き込むログファイルを検査します。

   a. ワーカーエージェントのログディレクトリに移動し、その内容を表示します。

```
cd ~/demoenv-logs
ls
```

   b. ワーカーエージェントが作成した最初のログファイルを出力します。

```
cat worker-agent-bootstrap.log
```

   このファイルには、Deadline Cloud API を呼び出してフリートにワーカーリソースを作成し、フリートロールを引き受けた方法に関するワーカーエージェントの出力が含まれています。

   c. ワーカーエージェントがフリートに参加するときに、ログファイルの出力を出力します。

```
cat worker-agent.log
```

   このログには、ワーカーエージェントが実行するすべてのアクションに関する出力が含まれますが、それらのリソースの IDs を除き、ジョブを実行するキューに関する出力は含まれません。

   d. キューリソース ID と同じ名前のディレクトリに、各セッションのログファイルを出力します。

```
cat $DEV_QUEUE_ID/session-*.log
```

   ジョブが成功すると、ログファイルの出力は次のようになります。

```
cat $DEV_QUEUE_ID/$(ls -t $DEV_QUEUE_ID | head -1)

2024-03-27 16:00:22,026 WARNING Session running with no AWS Credentials.
2024-03-27 16:00:22,404 INFO
2024-03-27 16:00:22,405 INFO ================================================
2024-03-27 16:00:22,405 INFO --------- Running Task
2024-03-27 16:00:22,405 INFO ================================================
2024-03-27 16:00:22,406 INFO ------------------------------------------------
2024-03-27 16:00:22,406 INFO Phase: Setup
2024-03-27 16:00:22,406 INFO ------------------------------------------------
2024-03-27 16:00:22,406 INFO Writing embedded files for Task to disk.
```

```
2024-03-27 16:00:22,406 INFO Mapping: Task.File.runScript -> /sessions/
session-053d77cef82648fea9c698271812a/embedded_fileswa_gj55_/tmp2u9yqtsz
2024-03-27 16:00:22,406 INFO Wrote: runScript -> /sessions/
session-053d77cef82648fea9c698271812a/embedded_fileswa_gj55_/tmp2u9yqtsz
2024-03-27 16:00:22,407 INFO ----------------------------------------------
2024-03-27 16:00:22,407 INFO Phase: Running action
2024-03-27 16:00:22,407 INFO ----------------------------------------------
2024-03-27 16:00:22,407 INFO Running command /sessions/
session-053d77cef82648fea9c698271812a/tmpzuzxpslm.sh
2024-03-27 16:00:22,414 INFO Command started as pid: 471
2024-03-27 16:00:22,415 INFO Output:
2024-03-27 16:00:22,420 INFO Welcome to AWS Deadline Cloud!
2024-03-27 16:00:22,571 INFO
2024-03-27 16:00:22,572 INFO ==============================================
2024-03-27 16:00:22,572 INFO --------- Session Cleanup
2024-03-27 16:00:22,572 INFO ==============================================
2024-03-27 16:00:22,572 INFO Deleting working directory: /sessions/
session-053d77cef82648fea9c698271812a
```

7. ジョブに関する情報を出力します。

```
deadline job get
```

ジョブを送信すると、システムはそれをデフォルトとして保存するため、ジョブ ID を入力する必要はありません。

パラメータsimple_jobを使用して を送信する

パラメータを使用してジョブを送信できます。次の手順では、simple_jobテンプレートを編集してカスタムメッセージを含め、 を送信しsimple_job、セッションログファイルを印刷してメッセージを表示します。

パラメータを使用してsimple_jobサンプルを送信するには

1. 最初の CloudShell タブを選択し、ジョブバンドルサンプルディレクトリに移動します。

```
cd ~/deadline-cloud-samples/job_bundles/
```

2. simple_job テンプレートの内容を出力します。

```
cat simple_job/template.yaml
```

`Message` パラメータを含む `parameterDefinitions`セクションは次のようになります。

```
parameterDefinitions:
- name: Message
  type: STRING
  default: Welcome to AWS Deadline Cloud!
```

3. パラメータ値を使用してsimple_jobサンプルを送信し、ジョブの実行が完了するまで待ちます。

```
deadline bundle submit simple_job \
    -p "Message=Greetings from the developer getting started guide."
```

4. カスタムメッセージを表示するには、最新のセッションログファイルを表示します。

```
cd ~/demoenv-logs
cat $DEV_QUEUE_ID/$(ls -t $DEV_QUEUE_ID | head -1)
```

ファイル I/O を使用して simple_file_job ジョブバンドルを作成する

レンダリングジョブは、シーン定義を読み取り、そこからイメージをレンダリングしてから、そのイメージを出力ファイルに保存する必要があります。このアクションをシミュレートするには、イメージをレンダリングする代わりに、ジョブに入力のハッシュを計算します。

ファイル I/O を使用して simple_file_job ジョブバンドルを作成するには

1. 最初の CloudShell タブを選択し、ジョブバンドルサンプルディレクトリに移動します。

```
cd ~/deadline-cloud-samples/job_bundles/
```

2. 新しい名前`simple_job`で のコピーを作成します`simple_file_job`。

```
cp -r simple_job simple_file_job
```

3. ジョブテンプレートを次のように編集します。

> **Note**
>
> これらのステップnanoでは を使用することをお勧めします。を使用する場合はVim、 を使用して貼り付けモードを設定する必要があります:`set paste`。

a. テキストエディタでテンプレートを開きます。

```
nano simple_file_job/template.yaml
```

b. 次の `type`、`objectType`、`dataFlow`および を追加します`parameterDefinitions`。

```
- name: InFile
  type: PATH
  objectType: FILE
  dataFlow: IN
- name: OutFile
  type: PATH
  objectType: FILE
  dataFlow: OUT
```

c. 入力ファイルから読み取り、出力ファイルに書き込むファイルの末尾に、次のbashスクリプトコマンドを追加します。

```
        # hash the input file, and write that to the output
        sha256sum "{{Param.InFile}}" > "{{Param.OutFile}}"
```

更新された `template.yaml`は、以下と完全に一致する必要があります。

```
specificationVersion: 'jobtemplate-2023-09'
name: Simple File Job Bundle Example
parameterDefinitions:
- name: Message
  type: STRING
  default: Welcome to AWS Deadline Cloud!
- name: InFile
  type: PATH
  objectType: FILE
  dataFlow: IN
- name: OutFile
```

```
    type: PATH
    objectType: FILE
    dataFlow: OUT
  steps:
  - name: WelcomeToDeadlineCloud
    script:
      actions:
        onRun:
          command: '{{Task.File.Run}}'
      embeddedFiles:
      - name: Run
        type: TEXT
        runnable: true
        data: |
          #!/usr/bin/env bash
          echo "{{Param.Message}}"

          # hash the input file, and write that to the output
          sha256sum "{{Param.InFile}}" > "{{Param.OutFile}}"
```

> **Note**
>
> の間隔を調整する場合は`template.yaml`、必ずインデントの代わりにスペースを使用してください。

d. ファイルを保存し、テキストエディタを終了します。

4. simple_file_job を送信する入出力ファイルのパラメータ値を指定します。

```
deadline bundle submit simple_file_job \
    -p "InFile=simple_job/template.yaml" \
    -p "OutFile=hash.txt"
```

5. ジョブに関する情報を出力します。

```
deadline job get
```

- 次のような出力が表示されます。

```
parameters:
  Message:
    string: Welcome to AWS Deadline Cloud!
```

```
  InFile:
    path: /local/home/cloudshell-user/BundleFiles/JobBundle-Examples/simple_job/
template.yaml
  OutFile:
    path: /local/home/cloudshell-user/BundleFiles/JobBundle-Examples/hash.txt
```

- 相対パスのみを指定しましたが、パラメータにはフルパスが設定されています。は、パスのタイプが の場合、パラメータとして指定されたパスに現在の作業ディレクトリを AWS CLI 結合しますPATH。
- 他のターミナルウィンドウで実行されているワーカーエージェントがジョブをピックアップして実行します。このアクションにより `hash.txt` ファイルが作成され、次のコマンドで表示できます。

  ```
  cat hash.txt
  ```

  このコマンドは、次のような出力を出力します。

  ```
  eaa2df5d34b54be5ac34c56a24a8c237b8487231a607eaf530a04d76b89c9cd3 /local/home/
  cloudshell-user/BundleFiles/JobBundle-Examples/simple_job/template.yaml
  ```

次のステップ

Deadline Cloud CLI を使用してシンプルなジョブを送信する方法を学習したら、以下を確認できます。

- Deadline Cloud でジョブアタッチメントを使用してジョブを送信する では、異なるオペレーティングシステムを実行しているホストでジョブを実行する方法について説明します。
- Deadline Cloud の開発者ファームにサービスマネージドフリートを追加する は、Deadline Cloud によって管理されるホストでジョブを実行します。
- Deadline Cloud でファームリソースをクリーンアップする このチュートリアルに使用したリソースをシャットダウンするには、 を使用します。

Deadline Cloud でジョブアタッチメントを使用してジョブを送信する

多くのファームは、共有ファイルシステムを使用して、ジョブを送信するホストとジョブを実行するホスト間でファイルを共有します。たとえば、前の`simple_file_job`例では、ローカルファイルシステムは、ジョブを送信するタブ 1 とワーカーエージェントを実行するタブ 2 で実行される AWS CloudShell ターミナルウィンドウ間で共有されます。

共有ファイルシステムは、送信者ワークステーションとワーカーホストが同じローカルエリアネットワーク上にある場合に便利です。それにアクセスするワークステーションの近くのオンプレミスにデータを保存する場合、クラウドベースのファームを使用すると、高レイテンシーの VPN 経由でファイルシステムを共有するか、クラウドでファイルシステムを同期する必要があります。これらのオプションはいずれも、設定や操作が容易ではありません。

AWS Deadline Cloud は、E メールの添付ファイルに似たジョブの添付ファイルを含むシンプルなソリューションを提供します。ジョブアタッチメントでは、データをジョブにアタッチします。その後、Deadline Cloud はジョブデータの転送と Amazon Simple Storage Service (Amazon S3) バケットへの保存の詳細を処理します。

コンテンツ作成ワークフローは、多くの場合反復的です。つまり、ユーザーは変更されたファイルの小さなサブセットでジョブを送信します。Amazon S3 バケットはジョブアタッチメントをコンテンツアドレス可能なストレージに保存するため、各オブジェクトの名前はオブジェクトのデータのハッシュに基づいており、ディレクトリツリーのコンテンツはジョブにアタッチされたマニフェストファイル形式で保存されます。

このセクションの手順に従う前に、以下を完了する必要があります。

- Deadline Cloud ファームを作成する
- Deadline Cloud ワーカーエージェントを実行する

ジョブアタッチメントを使用してジョブを実行するには、次の手順を実行します。

トピック

- ジョブアタッチメント設定をキューに追加する
- ジョブアタッチメント`simple_file_job`を使用して送信する
- ジョブアタッチメントを Amazon S3 に保存する方法を理解する
- 次のステップ

ジョブアタッチメント設定をキューに追加する

キューでジョブアタッチメントを有効にするには、 アカウントのキューリソースにジョブアタッチメント設定を追加します。

ジョブアタッチメント設定をキューに追加するには

1. 最初の CloudShell タブを選択し、次のいずれかのコマンドを入力して、ジョブアタッチメントに Amazon S3 バケットを使用します。
   - 既存のプライベート Amazon S3 バケットがない場合は、新しい S3 バケットを作成して使用できます。

     ```
     DEV_FARM_BUCKET=$(echo $DEV_FARM_NAME \
         | tr '[:upper:]' '[:lower:]')-$(xxd -l 16 -p /dev/urandom)
     if [ "$AWS_REGION" == "us-east-1" ]; then LOCATION_CONSTRAINT=
     else LOCATION_CONSTRAINT="--create-bucket-configuration \
         LocationConstraint=${AWS_REGION}"
     fi
     aws s3api create-bucket \
         $LOCATION_CONSTRAINT \
         --acl private \
         --bucket ${DEV_FARM_BUCKET}
     ```

   - プライベート Amazon S3 バケットがすでにある場合は、 をバケットの名前*MY_BUCKET_NAME*に置き換えて使用できます。

     ```
     DEV_FARM_BUCKET=MY_BUCKET_NAME
     ```

2. Amazon S3 バケットを作成または選択したら、バケット名を に追加`~/.bashrc`して、バケットを他のターミナルセッションで使用できるようにします。

   ```
   echo "DEV_FARM_BUCKET=$DEV_FARM_BUCKET" >> ~/.bashrc
    source ~/.bashrc
   ```

3. キューの AWS Identity and Access Management (IAM) ロールを作成します。

   ```
   aws iam create-role --role-name "${DEV_FARM_NAME}QueueRole" \
       --assume-role-policy-document \
           '{
               "Version": "2012-10-17",
               "Statement": [
   ```

```
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "credentials.deadline.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        }'
aws iam put-role-policy \
    --role-name "${DEV_FARM_NAME}QueueRole" \
    --policy-name S3BucketsAccess \
    --policy-document \
            '{
                "Version": "2012-10-17",
                "Statement": [
                {
                    "Action": [
                        "s3:GetObject*",
                        "s3:GetBucket*",
                        "s3:List*",
                        "s3:DeleteObject*",
                        "s3:PutObject",
                        "s3:PutObjectLegalHold",
                        "s3:PutObjectRetention",
                        "s3:PutObjectTagging",
                        "s3:PutObjectVersionTagging",
                        "s3:Abort*"
                    ],
                    "Resource": [
                        "arn:aws:s3:::'$DEV_FARM_BUCKET'",
                        "arn:aws:s3:::'$DEV_FARM_BUCKET'/*"
                    ],
                    "Effect": "Allow"
                }
            ]
            }'
```

4. キューを更新して、ジョブアタッチメント設定と IAM ロールを含めます。

```
QUEUE_ROLE_ARN="arn:aws:iam::$(aws sts get-caller-identity \
        --query "Account" --output text):role/${DEV_FARM_NAME}QueueRole"
aws deadline update-queue \
    --farm-id $DEV_FARM_ID \
```

```
    --queue-id $DEV_QUEUE_ID \
    --role-arn $QUEUE_ROLE_ARN \
    --job-attachment-settings \
        '{
            "s3BucketName": "'$DEV_FARM_BUCKET'",
            "rootPrefix": "JobAttachments"
        }'
```

5. キューを更新したことを確認します。

```
deadline queue get
```

次のような出力が表示されます。

```
...
jobAttachmentSettings:
  s3BucketName: DEV_FARM_BUCKET
  rootPrefix: JobAttachments
roleArn: arn:aws:iam::ACCOUNT_NUMBER:role/DeveloperFarmQueueRole
...
```

ジョブアタッチメントsimple_file_jobを使用して送信する

ジョブアタッチメントを使用する場合、ジョブバンドルは Deadline Cloud に、PATHパラメータの使用など、ジョブのデータフローを決定するのに十分な情報を提供する必要があります。の場合 simple_file_job、 `template.yaml` ファイルを編集して、データフローが入力ファイルと出力ファイルにあることを Deadline Cloud に伝えます。

ジョブアタッチメント設定をキューに追加したら、ジョブアタッチメントを含む simple_file_job サンプルを送信できます。これを行うと、ログ記録とジョブ出力を表示して、ジョブアタッチメント simple_file_jobを持つ が機能していることを確認できます。

simple_file_job ジョブバンドルをジョブアタッチメントとともに送信するには

1. 最初の CloudShell タブを選択し、 `JobBundle-Samples` ディレクトリを開きます。
2.
```
cd ~/deadline-cloud-samples/job_bundles/
```
3. simple_file_job をキューに送信します。アップロードを確認するプロンプトが表示されたら、「」と入力します**y**。

```
deadline bundle submit simple_file_job \
    -p InFile=simple_job/template.yaml \
    -p OutFile=hash-jobattachments.txt
```

4. ジョブアタッチメントのデータ転送セッションログ出力を表示するには、次のコマンドを実行します。

```
JOB_ID=$(deadline config get defaults.job_id)
SESSION_ID=$(aws deadline list-sessions \
        --farm-id $DEV_FARM_ID \
        --queue-id $DEV_QUEUE_ID \
        --job-id $JOB_ID \
        --query "sessions[0].sessionId" \
        --output text)
cat ~/demoenv-logs/$DEV_QUEUE_ID/$SESSION_ID.log
```

5. セッション内で実行されたセッションアクションを一覧表示します。

```
aws deadline list-session-actions \
    --farm-id $DEV_FARM_ID \
    --queue-id $DEV_QUEUE_ID \
    --job-id $JOB_ID \
    --session-id $SESSION_ID
```

次のような出力が表示されます。

```
{
    "sessionactions": [
        {
            "sessionActionId": "sessionaction-123-0",
            "status": "SUCCEEDED",
            "startedAt": "<timestamp>",
            "endedAt": "<timestamp>",
            "progressPercent": 100.0,
            "definition": {
                "syncInputJobAttachments": {}
            }
        },
        {
            "sessionActionId": "sessionaction-123-1",
            "status": "SUCCEEDED",
```

```
            "startedAt": "<timestamp>",
            "endedAt": "<timestamp>",
            "progressPercent": 100.0,
            "definition": {
                "taskRun": {
                    "taskId": "task-abc-0",
                    "stepId": "step-def"
                }
            }
        }
    ]
}
```

最初のセッションアクションは入力ジョブアタッチメントをダウンロードし、2 番目のアクションは前のステップのようにタスクを実行し、出力ジョブアタッチメントをアップロードします。

6. 出力ディレクトリを一覧表示します。

```
ls *.txt
```

などの出力は ディレクトリに`hash.txt`存在しますが、ジョブの出力ファイルがまだダウンロードされていないため、存在`hash-jobattachments.txt`しません。

7. 最新のジョブから出力をダウンロードします。

```
deadline job download-output
```

8. ダウンロードしたファイルの出力を表示します。

```
cat hash-jobattachments.txt
```

次のような出力が表示されます。

```
eaa2df5d34b54be5ac34c56a24a8c237b8487231a607eaf530a04d76b89c9cd3  /tmp/openjd/
session-123/assetroot-abc/simple_job/template.yaml
```

ジョブアタッチメントを Amazon S3 に保存する方法を理解する

AWS Command Line Interface（AWS CLI) を使用して、Amazon S3 バケットに保存されているジョブアタッチメントのデータをアップロードまたはダウンロードできます。Deadline Cloud が

Amazon S3 にジョブアタッチメントを保存する方法を理解すると、ワークロードとパイプラインの統合を開発するのに役立ちます。

Deadline Cloud ジョブアタッチメントが Amazon S3 にどのように保存されているかを調べるには

1. 最初の CloudShell タブを選択し、ジョブバンドルサンプルディレクトリを開きます。

```
cd ~/deadline-cloud-samples/job_bundles/
```

2. ジョブのプロパティを検査します。

```
deadline job get
```

次のような出力が表示されます。

```
parameters:
  Message:
    string: Welcome to AWS Deadline Cloud!
  InFile:
    path: /home/cloudshell-user/deadline-cloud-samples/job_bundles/simple_job/
template.yaml
  OutFile:
    path: /home/cloudshell-user/deadline-cloud-samples/job_bundles/hash-
jobattachments.txt
attachments:
  manifests:
  - rootPath: /home/cloudshell-user/deadline-cloud-samples/job_bundles/
    rootPathFormat: posix
    outputRelativeDirectories:
    - .
    inputManifestPath: farm-3040c59a5b9943d58052c29d907a645d/queue-
cde9977c9f4d4018a1d85f3e6c1a4e6e/Inputs/
f46af01ca8904cd8b514586671c79303/0d69cd94523ba617c731f29c019d16e8_input.xxh128
    inputManifestHash: f95ef91b5dab1fc1341b75637fe987ee
  fileSystem: COPIED
```

添付ファイルフィールドには、ジョブの実行時に使用する入出力データパスを記述するマニフェスト構造のリストが含まれています。ジョブを送信したマシンのローカルディレクトリパス`rootPath`を確認するには、「」を参照してください。マニフェストファイルを含む Amazon S3 オブジェクトサフィックスを表示するには、 を確認します`inputManifestFile`。マニフェ

ストファイルには、ジョブの入力データのディレクトリツリースナップショットのメタデータが含まれています。

3. Amazon S3 マニフェストオブジェクトをプリティプリントして、ジョブの入力ディレクトリ構造を確認します。

```
MANIFEST_SUFFIX=$(aws deadline get-job \
     --farm-id $DEV_FARM_ID \
     --queue-id $DEV_QUEUE_ID \
     --job-id $JOB_ID \
     --query "attachments.manifests[0].inputManifestPath" \
     --output text)
 aws s3 cp s3://$DEV_FARM_BUCKET/JobAttachments/Manifests/$MANIFEST_SUFFIX - | jq .
```

次のような出力が表示されます。

```
{
     "hashAlg": "xxh128",
     "manifestVersion": "2023-03-03",
     "paths": [
     {
         "hash": "2ec297b04c59c4741ed97ac8fb83080c",
         "mtime": 1698186190000000,
         "path": "simple_job/template.yaml",
         "size": 445
     }
     ],
     "totalSize": 445
 }
```

4. 出力ジョブアタッチメントのマニフェストを保持する Amazon S3 プレフィックスを作成し、その下にオブジェクトを一覧表示します。

```
SESSION_ACTION=$(aws deadline list-session-actions \
    --farm-id $DEV_FARM_ID \
    --queue-id $DEV_QUEUE_ID \
    --job-id $JOB_ID \
    --session-id $SESSION_ID \
    --query "sessionActions[?definition.taskRun != null] | [0]")
STEP_ID=$(echo $SESSION_ACTION | jq -r .definition.taskRun.stepId)
TASK_ID=$(echo $SESSION_ACTION | jq -r .definition.taskRun.taskId)
```

```
TASK_OUTPUT_PREFIX=JobAttachments/Manifests/$DEV_FARM_ID/$DEV_QUEUE_ID/$JOB_ID/
$STEP_ID/$TASK_ID/
aws s3api list-objects-v2 --bucket $DEV_FARM_BUCKET --prefix $TASK_OUTPUT_PREFIX
```

出力ジョブアタッチメントは、ジョブリソースから直接参照されるのではなく、ファームリソース ID に基づいて Amazon S3 バケットに配置されます。 IDs

5. 特定のセッションアクション ID の最新マニフェストオブジェクトキーを取得し、マニフェストオブジェクトをプリティープリントします。

```
SESSION_ACTION_ID=$(echo $SESSION_ACTION | jq -r .sessionActionId)
 MANIFEST_KEY=$(aws s3api list-objects-v2 \
     --bucket $DEV_FARM_BUCKET \
     --prefix $TASK_OUTPUT_PREFIX \
     --query "Contents[*].Key" --output text \
     | grep $SESSION_ACTION_ID \
     | sort | tail -1)
 MANIFEST_OBJECT=$(aws s3 cp s3://$DEV_FARM_BUCKET/$MANIFEST_KEY -)
 echo $MANIFEST_OBJECT | jq .
```

出力`hash-jobattachments.txt`には、次のような ファイルのプロパティが表示されます。

```
{
     "hashAlg": "xxh128",
     "manifestVersion": "2023-03-03",
     "paths": [
     {
         "hash": "f60b8e7d0fabf7214ba0b6822e82e08b",
         "mtime": 1698785252554950,
         "path": "hash-jobattachments.txt",
         "size": 182
     }
     ],
     "totalSize": 182
 }
```

ジョブにはタスク実行ごとに 1 つのマニフェストオブジェクトしかありませんが、一般的にタスク実行ごとにより多くのオブジェクトを持つことができます。

6. `Data` プレフィックスの下にコンテンツアドレス可能な Amazon S3 ストレージ出力を表示します。

```
FILE_HASH=$(echo $MANIFEST_OBJECT | jq -r .paths[0].hash)
FILE_PATH=$(echo $MANIFEST_OBJECT | jq -r .paths[0].path)
aws s3 cp s3://$DEV_FARM_BUCKET/JobAttachments/Data/$FILE_HASH -
```

次のような出力が表示されます。

```
eaa2df5d34b54be5ac34c56a24a8c237b8487231a607eaf530a04d76b89c9cd3  /tmp/openjd/
session-123/assetroot-abc/simple_job/template.yaml
```

次のステップ

Deadline Cloud CLI を使用して添付ファイルを含むジョブを送信する方法を学習したら、以下を確認できます。

- Deadline Cloud で送信する ワーカーホストで OpenJD バンドルを使用してジョブを実行する方法について説明します。
- Deadline Cloud の開発者ファームにサービスマネージドフリートを追加する は、Deadline Cloud によって管理されるホストでジョブを実行します。
- Deadline Cloud でファームリソースをクリーンアップする このチュートリアルで使用したリソースをシャットダウンするには、 を使用します。

Deadline Cloud の開発者ファームにサービスマネージドフリートを追加する

AWS CloudShell は、大規模なワークロードをテストするのに十分なコンピューティング容量を提供しません。また、複数のワーカーホストにタスクを分散するジョブで動作するように設定されていません。

CloudShell を使用する代わりに、Auto Scaling サービスマネージドフリート (SMF) をデベロッパーファームに追加できます。SMF は、大規模なワークロードに十分なコンピューティング容量を提供し、複数のワーカーホストにジョブタスクを分散する必要があるジョブを処理できます。

SMF を追加する前に、Deadline Cloud ファーム、キュー、フリートを設定する必要があります。「Deadline Cloud ファームを作成する」を参照してください。

サービスマネージドフリートをデベロッパーファームに追加するには

1. 最初の AWS CloudShell タブを選択し、サービスマネージドフリートを作成し、そのフリート ID を に追加します `.bashrc`。このアクションにより、他のターミナルセッションで使用できます。

```
FLEET_ROLE_ARN="arn:aws:iam::$(aws sts get-caller-identity \
         --query "Account" --output text):role/${DEV_FARM_NAME}FleetRole"
 aws deadline create-fleet \
     --farm-id $DEV_FARM_ID \
     --display-name "$DEV_FARM_NAME SMF" \
     --role-arn $FLEET_ROLE_ARN \
     --max-worker-count 5 \
     --configuration \
         '{
             "serviceManagedEc2": {
                 "instanceCapabilities": {
                     "vCpuCount": {
                         "min": 2,
                         "max": 4
                     },
                     "memoryMiB": {
                         "min": 512
                     },
                     "osFamily": "linux",
                     "cpuArchitectureType": "x86_64"
                 },
                 "instanceMarketOptions": {
                     "type": "spot"
                 }
             }
         }'

 echo "DEV_SMF_ID=$(aws deadline list-fleets \
         --farm-id $DEV_FARM_ID \
         --query "fleets[?displayName=='$DEV_FARM_NAME SMF'].fleetId \
         | [0]" --output text)" >> ~/.bashrc
 source ~/.bashrc
```

2. SMF をキューに関連付けます。

```
aws deadline create-queue-fleet-association \
     --farm-id $DEV_FARM_ID \
```

```
     --queue-id $DEV_QUEUE_ID \
     --fleet-id $DEV_SMF_ID
```

3. キューsimple_file_jobに送信します。アップロードを確認するプロンプトが表示されたら、「」と入力しますy。

```
deadline bundle submit simple_file_job \
    -p InFile=simple_job/template.yaml \
    -p OutFile=hash-jobattachments.txt
```

4. SMF が正しく動作していることを確認します。

```
deadline fleet get
```

- ワーカーの起動には数分かかる場合があります。フリートが実行されていることがわかるまで、 `deadline fleet get` コマンドを繰り返します。
- サービスマネージドフリート`queueFleetAssociationsStatus`の は になります`ACTIVE`。
- SMF `autoScalingStatus`は から `GROWING`に変更されます`STEADY`。

ステータスは次のようになります。

```
fleetId: fleet-2cc78e0dd3f04d1db427e7dc1d51ea44
farmId: farm-63ee8d77cdab4a578b685be8c5561c4a
displayName: DeveloperFarm SMF
description: ''
status: ACTIVE
autoScalingStatus: STEADY
targetWorkerCount: 0
workerCount: 0
minWorkerCount: 0
maxWorkerCount: 5
```

5. 送信したジョブのログを表示します。このログは、CloudShell ファイルシステムではなく、Amazon CloudWatch Logs のログに保存されます。

```
 JOB_ID=$(deadline config get defaults.job_id)
 SESSION_ID=$(aws deadline list-sessions \
         --farm-id $DEV_FARM_ID \
         --queue-id $DEV_QUEUE_ID \
         --job-id $JOB_ID \
```

```
        --query "sessions[0].sessionId" \
        --output text)
aws logs tail /aws/deadline/$DEV_FARM_ID/$DEV_QUEUE_ID \
    --log-stream-names $SESSION_ID
```

次のステップ

サービスマネージドフリートを作成およびテストしたら、不要な料金が発生しないように、作成したリソースを削除する必要があります。

- Deadline Cloud でファームリソースをクリーンアップする このチュートリアルで使用したリソースをシャットダウンするには、 を使用します。

Deadline Cloud でファームリソースをクリーンアップする

新しいワークロードとパイプライン統合を開発してテストするには、このチュートリアル用に作成した Deadline Cloud 開発者ファームを引き続き使用できます。開発者ファームが不要になった場合は、ファーム、フリート、キュー、 AWS Identity and Access Management (IAM) ロール、ログなどのリソースを Amazon CloudWatch Logs で削除できます。これらのリソースを削除した後、リソースを使用するにはチュートリアルを再度開始する必要があります。詳細については、「Deadline Cloud リソースの開始方法」を参照してください。

デベロッパーファームリソースをクリーンアップするには

1. 最初の CloudShell タブを選択し、キューとフリートの関連付けをすべて停止します。

```
FLEETS=$(aws deadline list-queue-fleet-associations \
        --farm-id $DEV_FARM_ID \
        --queue-id $DEV_QUEUE_ID \
        --query "queueFleetAssociations[].fleetId" \
        --output text)
for FLEET_ID in $FLEETS; do
    aws deadline update-queue-fleet-association \
        --farm-id $DEV_FARM_ID \
        --queue-id $DEV_QUEUE_ID \
        --fleet-id $FLEET_ID \
        --status STOP_SCHEDULING_AND_CANCEL_TASKS
done
```

2. キューフリートの関連付けを一覧表示します。

```
aws deadline list-queue-fleet-associations \
     --farm-id $DEV_FARM_ID \
     --queue-id $DEV_QUEUE_ID
```

出力が を報告するまでコマンドを再実行する必要がある場合があります。"status": "STOPPED"次のステップに進むことができます。このプロセスは完了までに数分かかることがあります。

```
{
    "queueFleetAssociations": [
        {
            "queueId": "queue-abcdefgh0123456789012345678901 2id",
            "fleetId": "fleet-abcdefgh0123456789012345678901 2id",
            "status": "STOPPED",
            "createdAt": "2023-11-21T20:49:19+00:00",
            "createdBy": "arn:aws:sts::123456789012:assumed-role/RoleToBeAssumed/
MySessionName",
            "updatedAt": "2023-11-21T20:49:38+00:00",
            "updatedBy": "arn:aws:sts::123456789012:assumed-role/RoleToBeAssumed/
MySessionName"
        },
        {
            "queueId": "queue-abcdefgh0123456789012345678901 2id",
            "fleetId": "fleet-abcdefgh0123456789012345678901 2id",
            "status": "STOPPED",
            "createdAt": "2023-11-21T20:32:06+00:00",
            "createdBy": "arn:aws:sts::123456789012:assumed-role/RoleToBeAssumed/
MySessionName",
            "updatedAt": "2023-11-21T20:49:39+00:00",
            "updatedBy": "arn:aws:sts::123456789012:assumed-role/RoleToBeAssumed/
MySessionName"
        }
    ]
}
```

3. キューのすべてのキューフリートの関連付けを削除します。

```
for FLEET_ID in $FLEETS; do
     aws deadline delete-queue-fleet-association \
         --farm-id $DEV_FARM_ID \
```

```
        --queue-id $DEV_QUEUE_ID \
        --fleet-id $FLEET_ID
done
```

4. キューに関連付けられているすべてのフリートを削除します。

```
for FLEET_ID in $FLEETS; do
     aws deadline delete-fleet \
         --farm-id $DEV_FARM_ID \
         --fleet-id $FLEET_ID
done
```

5. キューを削除します。

```
aws deadline delete-queue \
     --farm-id $DEV_FARM_ID \
     --queue-id $DEV_QUEUE_ID
```

6. ファームを削除します。

```
aws deadline delete-farm \
     --farm-id $DEV_FARM_ID
```

7. ファームの他の AWS リソースを削除します。

   a. フリート AWS Identity and Access Management (IAM) ロールを削除します。

   ```
   aws iam delete-role-policy \
        --role-name "${DEV_FARM_NAME}FleetRole" \
        --policy-name WorkerPermissions
   aws iam delete-role \
        --role-name "${DEV_FARM_NAME}FleetRole"
   ```

   b. キューの IAM ロールを削除します。

   ```
   aws iam delete-role-policy \
        --role-name "${DEV_FARM_NAME}QueueRole" \
        --policy-name S3BucketsAccess
   aws iam delete-role \
        --role-name "${DEV_FARM_NAME}QueueRole"
   ```

   c. Amazon CloudWatch Logs ロググループを削除します。各キューとフリートには独自のロググループがあります。

```
aws logs delete-log-group \
     --log-group-name "/aws/deadline/$DEV_FARM_ID/$DEV_QUEUE_ID"
aws logs delete-log-group \
     --log-group-name "/aws/deadline/$DEV_FARM_ID/$DEV_CMF_ID"
aws logs delete-log-group \
     --log-group-name "/aws/deadline/$DEV_FARM_ID/$DEV_SMF_ID"
```

キュー環境を使用してジョブを設定する

AWS Deadline Cloud はキュー環境を使用して、ワーカーにソフトウェアを設定します。環境では、セッション内のすべてのタスクに対して、セットアップやティアダウンなどの時間のかかるタスクを 1 回実行できます。セッションを開始または停止するときにワーカーに対して実行するアクションを定義します。キューの環境、キューで実行されるジョブ、ジョブの個々のステップを設定できます。

環境は、キュー環境またはジョブ環境として定義します。Deadline Cloud コンソールまたは deadline:CreateQueueEnvironment オペレーションを使用してキュー環境を作成し、送信するジョブのジョブテンプレートでジョブ環境を定義します。環境の Open Job Description (OpenJD) 仕様に従います。詳細については、GitHub の OpenJD 仕様の「<Environment>」を参照してください。

name および に加えてdescription、各環境にはホスト上の環境を定義する 2 つのフィールドが含まれています。具体的には次の 2 つです。

- script – この環境がワーカーで実行されたときに実行されるアクション。
- variables – 環境に入るときに設定される環境変数の名前と値のペアのセット。

または の少なくとも 1 script つを設定する必要がありますvariables。

ジョブテンプレートで複数の環境を定義できます。各環境は、テンプレートにリストされている順序で適用されます。これを使用して、環境の複雑さを管理できます。

Deadline Cloud のデフォルトのキュー環境では、conda パッケージマネージャーを使用して環境にソフトウェアをロードしますが、他のパッケージマネージャーを使用できます。デフォルトの環境では、ロードするソフトウェアを指定する 2 つのパラメータを定義します。これらの変数は Deadline Cloud が提供する送信者によって設定されますが、デフォルトの環境を使用する独自のスクリプトやアプリケーションで設定できます。具体的には次の 2 つです。

- CondaPackages – conda パッケージのスペース区切りリストは、ジョブにインストールする仕様と一致します。たとえば、Blender 送信者は blender=3.6を Blender 3.6 のレンダーフレームに追加します。
- CondaChannels – パッケージをインストールする conda チャネルのスペース区切りリスト。サービスマネージドフリートの場合、パッケージはdeadline-cloudチャネルからインストールされます。他のチャネルを追加できます。

トピック

- OpenJD キュー環境でジョブ環境を制御する
- ジョブにアプリケーションを提供する

OpenJD キュー環境でジョブ環境を制御する

キュー環境を使用して、レンダリングジョブ用にカスタマイズされた環境を定義できます。キュー環境は、特定のキューで実行されているジョブの環境変数、ファイルマッピング、およびその他の設定を制御するテンプレートです。これにより、キューに送信されたジョブの実行環境をワークロードの要件に合わせて調整できます。 AWS Deadline Cloud には、オープンジョブ記述 (OpenJD) 環境を適用できる 3 つのネストされたレベルがあります。キュー、ジョブ、ステップです。キュー環境を定義することで、さまざまなタイプのジョブに対して一貫した最適なパフォーマンスを確保し、リソース割り当てを合理化し、キュー管理を簡素化できます。

キュー環境は、 AWS 管理コンソールまたは を使用して AWS アカウントのキューにアタッチするテンプレートです AWS CLI。キューに 1 つの環境を作成することも、実行環境を作成するために が適用した複数のキュー環境を作成することもできます。これにより、 環境をステップで作成してテストし、ジョブで正しく動作することを確認することができます。

ジョブ環境とステップ環境は、キューにジョブを作成するために使用するジョブテンプレートで定義されます。OpenJD 構文は、これらの異なる形式の環境で同じです。このセクションでは、ジョブテンプレート内にそれらを表示します。

トピック

- キュー環境で環境変数を設定する
- キュー環境でパスを設定する
- キュー環境からバックグラウンドデーモンプロセスを実行する

キュー環境で環境変数を設定する

Open Job Description (OpenJD) 環境では、スコープ内のすべてのタスクコマンドが使用する環境変数を設定できます。多くのアプリケーションとフレームワークは、機能設定、ログ記録レベルなどを制御するために環境変数をチェックします。

例えば、Qt Framework は多くのデスクトップアプリケーションに GUI 機能を提供します。インタラクティブディスプレイなしでワーカーホストでこれらのアプリケーションを実行する場合、ワーカーがディスプレイを探さないoffscreenように環境変数QT_QPA_PLATFORMを に設定する必要がある場合があります。

この例では、Deadline Cloud サンプルディレクトリのサンプルジョブバンドルを使用して、ジョブの環境変数を設定および表示します。

前提条件

Deadline Cloud サンプル github リポジトリの環境変数を使用してサンプルジョブバンドルを実行するには、次の手順を実行します。

1. キューと関連する Linux フリートを持つ Deadline Cloud ファームがない場合は、Deadline Cloud コンソールのガイド付きオンボーディングエクスペリエンスに従って、デフォルト設定で作成します。
2. ワークステーションに Deadline Cloud CLI と Deadline Cloud モニターがない場合は、ユーザーガイドの「Deadline Cloud 送信者の設定」の手順に従ってください。
3. `git` を使用して Deadline Cloud サンプル GitHub リポジトリのクローンを作成します。

```
git clone https://github.com/aws-deadline/deadline-cloud-samples.git
 Cloning into 'deadline-cloud-samples'...
 ...
cd deadline-cloud-samples/job_bundles
```

環境変数サンプルを実行する

1. Deadline Cloud CLI を使用して`job_env_vars`サンプルを送信します。

```
deadline bundle submit job_env_vars
 Submitting to Queue: MySampleQueue
 ...
```

2. Deadline Cloud モニターでは、新しいジョブを表示し、進行状況をモニタリングできます。キューに関連付けられたLinuxフリートがジョブのタスクを実行できるワーカーを持つと、ジョブは数秒で完了します。タスクを選択し、タスクパネルの右上メニューでログの表示オプションを選択します。

   右側には、Launch JobEnv、Launch StepEnv、Task run の 3 つのセッションアクションがあります。ウィンドウの中央のログビューは、右側の選択したセッションアクションに対応します。

セッションアクションとその定義を比較する

このセクションでは、Deadline Cloud モニターを使用して、セッションアクションをジョブテンプレートで定義されている場所と比較します。これは前のセクションから続きます。

テキストエディタで job_env_vars/template.yaml ファイルを開きます。これは、セッションアクションを定義するジョブテンプレートです。

1. Deadline Cloud Monitor で Launch JobEnv セッションアクションを選択します。次のログ出力が表示されます。

```
 024/07/16 16:18:27-07:00
 2024/07/16 16:18:27-07:00 ================================================
 2024/07/16 16:18:27-07:00 --------- Entering Environment: JobEnv
 2024/07/16 16:18:27-07:00 ================================================
 2024/07/16 16:18:27-07:00 Setting: JOB_VERBOSITY=MEDIUM
 2024/07/16 16:18:27-07:00 Setting: JOB_EXAMPLE_PARAM=An example parameter value
 2024/07/16 16:18:27-07:00 Setting: JOB_PROJECT_ID=project-12
 2024/07/16 16:18:27-07:00 Setting: JOB_ENDPOINT_URL=https://internal-host-name/some/
path
 2024/07/16 16:18:27-07:00 Setting: QT_QPA_PLATFORM=offscreen
```

   ジョブテンプレートの次の行で、このアクションが指定されました。

```
jobEnvironments:
 - name: JobEnv
   description: Job environments apply to everything in the job.
   variables:
     # When applications have options as environment variables, you can set them
 here.
     JOB_VERBOSITY: MEDIUM
     # You can use the value of job parameters when setting environment variables.
     JOB_EXAMPLE_PARAM: "{{Param.ExampleParam}}"
     # Some more ideas.
     JOB_PROJECT_ID: project-12
     JOB_ENDPOINT_URL: https://internal-host-name/some/path
     # This variable lets applications using the Qt Framework run without a display
     QT_QPA_PLATFORM: offscreen
```

2. Deadline Cloud Monitor で StepEnv セッション起動アクションを選択します。次のログ出力が表示されます。

```
2024/07/16 16:18:27-07:00
2024/07/16 16:18:27-07:00 ================================================
2024/07/16 16:18:27-07:00 --------- Entering Environment: StepEnv
2024/07/16 16:18:27-07:00 ================================================
2024/07/16 16:18:27-07:00 Setting: STEP_VERBOSITY=HIGH
2024/07/16 16:18:27-07:00 Setting: JOB_PROJECT_ID=step-project-12
```

ジョブテンプレートの次の行で、このアクションが指定されました。

```
  stepEnvironments:
  - name: StepEnv
    description: Step environments apply to all the tasks in the step.
    variables:
      # These environment variables are only set within this step, not other steps.
      STEP_VERBOSITY: HIGH
      # Replace a variable value defined at the job level.
      JOB_PROJECT_ID: step-project-12
```

3. Deadline Cloud Monitor でタスク実行セッションアクションを選択します。次の出力が表示されます。

```
2024/07/16 16:18:27-07:00
2024/07/16 16:18:27-07:00 ================================================
2024/07/16 16:18:27-07:00 --------- Running Task
2024/07/16 16:18:27-07:00 ================================================
2024/07/16 16:18:27-07:00 ------------------------------------------------
2024/07/16 16:18:27-07:00 Phase: Setup
2024/07/16 16:18:27-07:00 ------------------------------------------------
2024/07/16 16:18:27-07:00 Writing embedded files for Task to disk.
2024/07/16 16:18:27-07:00 Mapping: Task.File.Run -> /sessions/session-
b4bd451784674c0987be82c5f7d5642deupf6tk9/embedded_files08cdnuyt/tmpmdiajwvh
2024/07/16 16:18:27-07:00 Wrote: Run -> /sessions/session-
b4bd451784674c0987be82c5f7d5642deupf6tk9/embedded_files08cdnuyt/tmpmdiajwvh
2024/07/16 16:18:27-07:00 ------------------------------------------------
2024/07/16 16:18:27-07:00 Phase: Running action
2024/07/16 16:18:27-07:00 ------------------------------------------------
2024/07/16 16:18:27-07:00 Running command sudo -u job-user -i setsid -w /sessions/
session-b4bd451784674c0987be82c5f7d5642deupf6tk9/tmpiqbrsby4.sh
2024/07/16 16:18:27-07:00 Command started as pid: 2176
2024/07/16 16:18:27-07:00 Output:
2024/07/16 16:18:28-07:00 Running the task
```

```
2024/07/16 16:18:28-07:00
2024/07/16 16:18:28-07:00 Environment variables starting with JOB_*:
2024/07/16 16:18:28-07:00 JOB_ENDPOINT_URL=https://internal-host-name/some/path
2024/07/16 16:18:28-07:00 JOB_EXAMPLE_PARAM='An example parameter value'
2024/07/16 16:18:28-07:00 JOB_PROJECT_ID=step-project-12
2024/07/16 16:18:28-07:00 JOB_VERBOSITY=MEDIUM
2024/07/16 16:18:28-07:00
2024/07/16 16:18:28-07:00 Environment variables starting with STEP_*:
2024/07/16 16:18:28-07:00 STEP_VERBOSITY=HIGH
2024/07/16 16:18:28-07:00
2024/07/16 16:18:28-07:00 Done running the task
2024/07/16 16:18:28-07:00 ----------------------------------------------
2024/07/16 16:18:28-07:00 Uploading output files to Job Attachments
2024/07/16 16:18:28-07:00 ----------------------------------------------
```

ジョブテンプレートの次の行で、このアクションが指定されました。

```
  script:
    actions:
      onRun:
        command: bash
        args:
        - '{{Task.File.Run}}'
    embeddedFiles:
    - name: Run
      type: TEXT
      data: |
        echo Running the task
        echo ""

        echo Environment variables starting with JOB_*:
        set | grep ^JOB_
        echo ""

        echo Environment variables starting with STEP_*:
        set | grep ^STEP_
        echo ""

        echo Done running the task
```

キュー環境でパスを設定する

OpenJD 環境を使用して、 環境で新しいコマンドを提供します。まず、スクリプトファイルを含むディレクトリを作成し、そのディレクトリをPATH環境変数に追加します。これにより、スクリプト内の実行可能ファイルは、毎回ディレクトリパスを指定することなく実行できます。環境定義の変数のリストは変数を変更する方法を提供しないため、代わりにスクリプトを実行して変更します。スクリプトが をセットアップして変更するとPATH、 コマンド を使用して変数を OpenJD ランタイムにエクスポートしますecho "openjd_env: PATH=$PATH"。

前提条件

Deadline Cloud サンプル github リポジトリの環境変数を使用してサンプルジョブバンドルを実行するには、次の手順を実行します。

1. キューと関連する Linux フリートを持つ Deadline Cloud ファームがない場合は、Deadline Cloud コンソールのガイド付きオンボーディングエクスペリエンスに従って、デフォルト設定で作成します。
2. ワークステーションに Deadline Cloud CLI と Deadline Cloud モニターがない場合は、ユーザーガイドの「Deadline Cloud 送信者の設定」の手順に従ってください。
3. `git` を使用して Deadline Cloud サンプル GitHub リポジトリのクローンを作成します。

```
git clone https://github.com/aws-deadline/deadline-cloud-samples.git
 Cloning into 'deadline-cloud-samples'...
 ...
cd deadline-cloud-samples/job_bundles
```

パスサンプルを実行する

1. Deadline Cloud CLI を使用して`job_env_with_new_command`サンプルを送信します。

```
 $ deadline bundle submit job_env_with_new_command
 Submitting to Queue: MySampleQueue
 ...
```

2. Deadline Cloud モニターに新しいジョブが表示され、進行状況をモニタリングできます。キューに関連付けられたLinuxフリートがジョブのタスクを実行できるワーカーを持つと、ジョブは数秒で完了します。タスクを選択し、タスクパネルの右上メニューでログの表示オプションを選択します。

右側には、Launch RandomSleepCommand と Task run の 2 つのセッションアクションがあります。 ウィンドウの中央にあるログビューワーは、右側の選択したセッションアクションに対応します。

セッションアクションとその定義を比較する

このセクションでは、Deadline Cloud モニターを使用して、セッションアクションをジョブテンプレートで定義されている場所と比較します。これは前のセクションから続きます。

テキストエディタで job_env_with_new_command/template.yaml ファイルを開きます。セッションアクションをジョブテンプレートで定義されている場所と比較します。

1. Deadline Cloud モニターで Launch RandomSleepCommand セッションアクションを選択します。ログ出力は次のように表示されます。

```
 2024/07/16 17:25:32-07:00
 2024/07/16 17:25:32-07:00 ================================================
 2024/07/16 17:25:32-07:00 --------- Entering Environment: RandomSleepCommand
 2024/07/16 17:25:32-07:00 ================================================
 2024/07/16 17:25:32-07:00 ------------------------------------------------
 2024/07/16 17:25:32-07:00 Phase: Setup
 2024/07/16 17:25:32-07:00 ------------------------------------------------
 2024/07/16 17:25:32-07:00 Writing embedded files for Environment to disk.
 2024/07/16 17:25:32-07:00 Mapping: Env.File.Enter -> /sessions/session-
ab132a51b9b54d5da22cbe839dd946baaw1c8hk5/embedded_filesf3tq_1os/tmpbt8j_c3f
 2024/07/16 17:25:32-07:00 Mapping: Env.File.SleepScript -> /sessions/session-
ab132a51b9b54d5da22cbe839dd946baaw1c8hk5/embedded_filesf3tq_1os/tmperastlp4
 2024/07/16 17:25:32-07:00 Wrote: Enter -> /sessions/session-
ab132a51b9b54d5da22cbe839dd946baaw1c8hk5/embedded_filesf3tq_1os/tmpbt8j_c3f
 2024/07/16 17:25:32-07:00 Wrote: SleepScript -> /sessions/session-
ab132a51b9b54d5da22cbe839dd946baaw1c8hk5/embedded_filesf3tq_1os/tmperastlp4
 2024/07/16 17:25:32-07:00 ------------------------------------------------
 2024/07/16 17:25:32-07:00 Phase: Running action
 2024/07/16 17:25:32-07:00 ------------------------------------------------
 2024/07/16 17:25:32-07:00 Running command sudo -u job-user -i setsid -w /sessions/
session-ab132a51b9b54d5da22cbe839dd946baaw1c8hk5/tmpbwrquq5u.sh
 2024/07/16 17:25:32-07:00 Command started as pid: 2205
 2024/07/16 17:25:32-07:00 Output:
 2024/07/16 17:25:33-07:00 openjd_env: PATH=/sessions/session-
ab132a51b9b54d5da22cbe839dd946baaw1c8hk5/bin:/opt/conda/condabin:/home/job-
```

```
user/.local/bin:/home/job-user/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/
bin:/sbin:/bin:/var/lib/snapd/snap/bin
 No newer logs at this moment.
```

ジョブテンプレートの次の行で、このアクションが指定されました。

```
 jobEnvironments:
 - name: RandomSleepCommand
   description: Adds a command 'random-sleep' to the environment.
   script:
     actions:
       onEnter:
         command: bash
         args:
         - "{{Env.File.Enter}}"
     embeddedFiles:
     - name: Enter
       type: TEXT
       data: |
         #!/bin/env bash
         set -euo pipefail

         # Make a bin directory inside the session's working directory for providing
 new commands
         mkdir -p '{{Session.WorkingDirectory}}/bin'

         # If this bin directory is not already in the PATH, then add it
         if ! [[ ":$PATH:" == *':{{Session.WorkingDirectory}}/bin:'* ]]; then
           export "PATH={{Session.WorkingDirectory}}/bin:$PATH"

           # This message to Open Job Description exports the new PATH value to the
 environment
           echo "openjd_env: PATH=$PATH"
         fi

         # Copy the SleepScript embedded file into the bin directory
         cp '{{Env.File.SleepScript}}' '{{Session.WorkingDirectory}}/bin/random-
sleep'
         chmod u+x '{{Session.WorkingDirectory}}/bin/random-sleep'
     - name: SleepScript
       type: TEXT
       runnable: true
       data: |
```

```
        ...
```

2. Deadline Cloud モニターで StepEnv セッション起動アクションを選択します。ログ出力は次のように表示されます。

```
2024/07/16 17:25:33-07:00
2024/07/16 17:25:33-07:00 ==============================================
2024/07/16 17:25:33-07:00 --------- Running Task
2024/07/16 17:25:33-07:00 ==============================================
2024/07/16 17:25:33-07:00 ----------------------------------------------
2024/07/16 17:25:33-07:00 Phase: Setup
2024/07/16 17:25:33-07:00 ----------------------------------------------
2024/07/16 17:25:33-07:00 Writing embedded files for Task to disk.
2024/07/16 17:25:33-07:00 Mapping: Task.File.Run -> /sessions/session-
ab132a51b9b54d5da22cbe839dd946baaw1c8hk5/embedded_filesf3tq_1os/tmpdrwuehjf
2024/07/16 17:25:33-07:00 Wrote: Run -> /sessions/session-
ab132a51b9b54d5da22cbe839dd946baaw1c8hk5/embedded_filesf3tq_1os/tmpdrwuehjf
2024/07/16 17:25:33-07:00 ----------------------------------------------
2024/07/16 17:25:33-07:00 Phase: Running action
2024/07/16 17:25:33-07:00 ----------------------------------------------
2024/07/16 17:25:33-07:00 Running command sudo -u job-user -i setsid -w /sessions/
session-ab132a51b9b54d5da22cbe839dd946baaw1c8hk5/tmpz81iaqfw.sh
2024/07/16 17:25:33-07:00 Command started as pid: 2256
2024/07/16 17:25:33-07:00 Output:
2024/07/16 17:25:34-07:00 + random-sleep 12.5 27.5
2024/07/16 17:26:00-07:00 Sleeping for duration 26.90
2024/07/16 17:26:00-07:00 ----------------------------------------------
2024/07/16 17:26:00-07:00 Uploading output files to Job Attachments
2024/07/16 17:26:00-07:00 ----------------------------------------------
```

3. ジョブテンプレートの次の行で、このアクションが指定されました。

```
steps:
- name: EnvWithCommand
  script:
    actions:
      onRun:
        command: bash
        args:
        - '{{Task.File.Run}}'
    embeddedFiles:
    - name: Run
      type: TEXT
```

```
      data: |
        set -xeuo pipefail

        # Run the script installed into PATH by the job environment
        random-sleep 12.5 27.5
  hostRequirements:
    attributes:
    - name: attr.worker.os.family
      anyOf:
      - linux
```

キュー環境からバックグラウンドデーモンプロセスを実行する

レンダリングの多くのユースケースでは、アプリケーションとシーンデータのロードにかなりの時間がかかる場合があります。ジョブがフレームごとに再ロードすると、ほとんどの時間がオーバーヘッドに費やされます。多くの場合、アプリケーションをバックグラウンドデーモンプロセスとして 1 回ロードし、シーンデータをロードしてから、プロセス間通信 (IPC) を介してコマンドを送信してレンダリングを実行することができます。

オープンソースの Deadline Cloud 統合の多くは、このパターンを使用します。Open Job Description プロジェクトは、サポートされているすべてのオペレーティングシステムで堅牢な IPC パターンを持つアダプターランタイムライブラリを提供します。

このパターンを示すために、Python と bash コードを使用してバックグラウンドデーモンと IPC を実装する自己完結型のサンプルジョブバンドルがあります。デーモンは Python に実装され、タスクを処理するタイミングについて POSIX SIGUSR1 シグナルをリッスンします。タスクの詳細は特定の JSON ファイルのデーモンに渡され、タスクの実行結果は別の JSON ファイルとして返されます。

前提条件

Deadline Cloud サンプル github リポジトリからデーモンプロセスを使用してサンプルジョブバンドルを実行するには、次の手順を実行します。

1. キューと関連する Linux フリートを持つ Deadline Cloud ファームがない場合は、Deadline Cloud コンソールのガイド付きオンボーディングエクスペリエンスに従って、デフォルト設定で作成します。
2. ワークステーションに Deadline Cloud CLI と Deadline Cloud モニターがない場合は、ユーザーガイドの「Deadline Cloud 送信者の設定」の手順に従ってください。
3. `git` を使用して Deadline Cloud サンプル GitHub リポジトリのクローンを作成します。

```
git clone https://github.com/aws-deadline/deadline-cloud-samples.git
 Cloning into 'deadline-cloud-samples'...
 ...
cd deadline-cloud-samples/job_bundles
```

デーモンサンプルを実行する

1. Deadline Cloud CLI を使用して`job_env_daemon_process`サンプルを送信します。

```
 git clone https://github.com/aws-deadline/deadline-cloud-samples.git
Cloning into 'deadline-cloud-samples'...
 ...
cd deadline-cloud-samples/job_bundles
```

2. Deadline Cloud Monitor アプリケーションには、新しいジョブが表示され、進行状況をモニタリングできます。キューに関連付けられたLinuxフリートがジョブのタスクを実行できるワーカーを持つと、約 1 分で完了します。いずれかのタスクを選択し、タスクパネルの右上メニューでログの表示オプションを選択します。

   右側には、Launch DaemonProcess と Task run の 2 つのセッションアクションがあります。ウィンドウの中央にあるログビューワーは、右側の選択したセッションアクションに対応します。

   オプション すべてのタスクのログを表示する を選択します。タイムラインには、セッションの一部として実行された残りのタスクと、環境を終了した`Shut down DaemonProcess`アクションが表示されます。

デーモンログを表示する

1. このセクションでは、Deadline Cloud モニターを使用して、セッションアクションをジョブテンプレートで定義されている場所と比較します。これは前のセクションから続きます。

   テキストエディタで job_env_daemon_process/template.yaml ファイルを開きます。セッションアクションをジョブテンプレートで定義されている場所と比較します。

2. Deadline Cloud Monitor で`Launch DaemonProcess`セッションアクションを選択します。ログ出力は次のように表示されます。

```
 2024/07/17 16:27:20-07:00
```

```
 2024/07/17 16:27:20-07:00 ================================================
 2024/07/17 16:27:20-07:00 --------- Entering Environment: DaemonProcess
 2024/07/17 16:27:20-07:00 ================================================
 2024/07/17 16:27:20-07:00 ------------------------------------------------
 2024/07/17 16:27:20-07:00 Phase: Setup
 2024/07/17 16:27:20-07:00 ------------------------------------------------
 2024/07/17 16:27:20-07:00 Writing embedded files for Environment to disk.
 2024/07/17 16:27:20-07:00 Mapping: Env.File.Enter -> /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/enter-daemon-
process-env.sh
 2024/07/17 16:27:20-07:00 Mapping: Env.File.Exit -> /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/exit-daemon-
process-env.sh
 2024/07/17 16:27:20-07:00 Mapping: Env.File.DaemonScript -> /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/daemon-
script.py
 2024/07/17 16:27:20-07:00 Mapping: Env.File.DaemonHelperFunctions -> /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/daemon-
helper-functions.sh
 2024/07/17 16:27:20-07:00 Wrote: Enter -> /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/enter-daemon-
process-env.sh
 2024/07/17 16:27:20-07:00 Wrote: Exit -> /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/exit-daemon-
process-env.sh
 2024/07/17 16:27:20-07:00 Wrote: DaemonScript -> /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/daemon-
script.py
 2024/07/17 16:27:20-07:00 Wrote: DaemonHelperFunctions -> /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/daemon-
helper-functions.sh
 2024/07/17 16:27:20-07:00 ------------------------------------------------
 2024/07/17 16:27:20-07:00 Phase: Running action
 2024/07/17 16:27:20-07:00 ------------------------------------------------
 2024/07/17 16:27:20-07:00 Running command sudo -u job-user -i setsid -w /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/tmp_u8slys3.sh
 2024/07/17 16:27:20-07:00 Command started as pid: 2187
 2024/07/17 16:27:20-07:00 Output:
 2024/07/17 16:27:21-07:00 openjd_env: DAEMON_LOG=/sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/daemon.log
 2024/07/17 16:27:21-07:00 openjd_env: DAEMON_PID=2223
```

```
 2024/07/17 16:27:21-07:00 openjd_env: DAEMON_BASH_HELPER_SCRIPT=/sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/daemon-
helper-functions.sh
```

ジョブテンプレートの次の行で、このアクションが指定されました。

```
  stepEnvironments:
  - name: DaemonProcess
    description: Runs a daemon process for the step's tasks to share.
    script:
      actions:
        onEnter:
          command: bash
          args:
          - "{{Env.File.Enter}}"
        onExit:
          command: bash
          args:
          - "{{Env.File.Exit}}"
      embeddedFiles:
      - name: Enter
        filename: enter-daemon-process-env.sh
        type: TEXT
        data: |
          #!/bin/env bash
          set -euo pipefail

          DAEMON_LOG='{{Session.WorkingDirectory}}/daemon.log'
          echo "openjd_env: DAEMON_LOG=$DAEMON_LOG"
          nohup python {{Env.File.DaemonScript}} > $DAEMON_LOG 2>&1 &
          echo "openjd_env: DAEMON_PID=$!"
          echo "openjd_env:
 DAEMON_BASH_HELPER_SCRIPT={{Env.File.DaemonHelperFunctions}}"

          echo 0 > 'daemon_log_cursor.txt'
    ...
```

3. Deadline Cloud Monitor でタスク実行: N セッションアクションのいずれかを選択します。ログ出力は次のように表示されます。

```
2024/07/17 16:27:22-07:00
 2024/07/17 16:27:22-07:00 ================================================
 2024/07/17 16:27:22-07:00 --------- Running Task
```

```
 2024/07/17 16:27:22-07:00 ==============================================
 2024/07/17 16:27:22-07:00 Parameter values:
 2024/07/17 16:27:22-07:00 Frame(INT) = 2
 2024/07/17 16:27:22-07:00 ----------------------------------------------
 2024/07/17 16:27:22-07:00 Phase: Setup
 2024/07/17 16:27:22-07:00 ----------------------------------------------
 2024/07/17 16:27:22-07:00 Writing embedded files for Task to disk.
 2024/07/17 16:27:22-07:00 Mapping: Task.File.Run -> /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/run-task.sh
 2024/07/17 16:27:22-07:00 Wrote: Run -> /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/embedded_fileswy00x5ra/run-task.sh
 2024/07/17 16:27:22-07:00 ----------------------------------------------
 2024/07/17 16:27:22-07:00 Phase: Running action
 2024/07/17 16:27:22-07:00 ----------------------------------------------
 2024/07/17 16:27:22-07:00 Running command sudo -u job-user -i setsid -w /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/tmpv4obfkhn.sh
 2024/07/17 16:27:22-07:00 Command started as pid: 2301
 2024/07/17 16:27:22-07:00 Output:
 2024/07/17 16:27:23-07:00 Daemon PID is 2223
 2024/07/17 16:27:23-07:00 Daemon log file is /sessions/
session-972e21d98dde45e59c7153bd9258a64dohwg4yg1/daemon.log
 2024/07/17 16:27:23-07:00
 2024/07/17 16:27:23-07:00 === Previous output from daemon
 2024/07/17 16:27:23-07:00 ===
 2024/07/17 16:27:23-07:00
 2024/07/17 16:27:23-07:00 Sending command to daemon
 2024/07/17 16:27:23-07:00 Received task result:
 2024/07/17 16:27:23-07:00 {
 2024/07/17 16:27:23-07:00   "result": "SUCCESS",
 2024/07/17 16:27:23-07:00   "processedTaskCount": 1,
 2024/07/17 16:27:23-07:00   "randomValue": 0.2578537967668988,
 2024/07/17 16:27:23-07:00   "failureRate": 0.1
 2024/07/17 16:27:23-07:00 }
 2024/07/17 16:27:23-07:00
 2024/07/17 16:27:23-07:00 === Daemon log from running the task
 2024/07/17 16:27:23-07:00 Loading the task details file
 2024/07/17 16:27:23-07:00 Received task details:
 2024/07/17 16:27:23-07:00 {
 2024/07/17 16:27:23-07:00  "pid": 2329,
 2024/07/17 16:27:23-07:00  "frame": 2
 2024/07/17 16:27:23-07:00 }
 2024/07/17 16:27:23-07:00 Processing frame number 2
 2024/07/17 16:27:23-07:00 Writing result
 2024/07/17 16:27:23-07:00 Waiting until a USR1 signal is sent...
```

```
2024/07/17 16:27:23-07:00 ===
2024/07/17 16:27:23-07:00
2024/07/17 16:27:23-07:00 ----------------------------------------------
2024/07/17 16:27:23-07:00 Uploading output files to Job Attachments
2024/07/17 16:27:23-07:00 ----------------------------------------------
```

ジョブテンプレートの次の行は、このアクションを指定したものです。「」ステップ：

```
steps:
- name: EnvWithDaemonProcess
  parameterSpace:
    taskParameterDefinitions:
    - name: Frame
      type: INT
      range: "{{Param.Frames}}"

  stepEnvironments:
    ...

  script:
    actions:
      onRun:
        timeout: 60
        command: bash
        args:
        - '{{Task.File.Run}}'
    embeddedFiles:
    - name: Run
      filename: run-task.sh
      type: TEXT
      data: |
        # This bash script sends a task to the background daemon process,
        # then waits for it to respond with the output result.

        set -euo pipefail

        source "$DAEMON_BASH_HELPER_SCRIPT"

        echo "Daemon PID is $DAEMON_PID"
        echo "Daemon log file is $DAEMON_LOG"

        print_daemon_log "Previous output from daemon"
```

```
        send_task_to_daemon "{\"pid\": $$, \"frame\": {{Task.Param.Frame}} }"
        wait_for_daemon_task_result

        echo Received task result:
        echo "$TASK_RESULT" | jq .

        print_daemon_log "Daemon log from running the task"

    hostRequirements:
      attributes:
      - name: attr.worker.os.family
        anyOf:
        - linux
```

ジョブにアプリケーションを提供する

キュー環境を使用して、ジョブを処理するアプリケーションをロードできます。Deadline Cloud コンソールを使用してサービスマネージドフリートを作成する場合、conda パッケージマネージャーを使用してアプリケーションをロードするキュー環境を作成するオプションがあります。

別のパッケージマネージャーを使用する場合は、そのマネージャーのキュー環境を作成できます。Rez の使用例については、「」を参照してください別のパッケージマネージャーを使用する。

Deadline Cloud は、選択したレンダリングアプリケーションを環境にロードするための conda チャネルを提供します。Deadline Cloud がデジタルコンテンツ作成アプリケーションに提供する送信者をサポートします。

ジョブで使用する conda-forge 用のソフトウェアをロードすることもできます。次の例は、Deadline Cloud が提供するキュー環境を使用してジョブを実行する前にアプリケーションをロードするジョブテンプレートを示しています。

トピック

- conda チャネルからアプリケーションを取得する
- 別のパッケージマネージャーを使用する

conda チャネルからアプリケーションを取得する

選択したソフトウェアをインストールする Deadline Cloud ワーカーのカスタムキュー環境を作成できます。このサンプルキュー環境の動作は、サービスマネージドフリートのコンソールで使用される環境と同じです。conda を直接実行して環境を作成します。

環境は、ワーカーで実行される Deadline Cloud セッションごとに新しい conda 仮想環境を作成し、完了すると環境を削除します。

Conda はダウンロードしたパッケージをキャッシュするため、再度ダウンロードする必要はありませんが、各セッションではすべてのパッケージを環境にリンクする必要があります。

環境は、Deadline Cloud がワーカーでセッションを開始するときに実行される 3 つのスクリプトを定義します。最初のスクリプトは、 `onEnter`アクションが呼び出されたときに実行されます。他の 2 つを呼び出して環境変数を設定します。スクリプトの実行が完了すると、conda 環境は指定されたすべての環境変数セットで使用できます。

この例の最新バージョンについては、GitHub の deadline-cloud-samples リポジトリの conda_queue_env_console_equivalent.yaml を参照してください。

conda チャネルで利用できないアプリケーションを使用する場合は、Amazon S3 で conda チャネルを作成し、そのアプリケーション用に独自のパッケージを構築できます。詳細については、「S3 を使用して conda チャネルを作成する」を参照してください。

conda-forge からオープンソースライブラリを取得する

このセクションでは、`conda-forge`チャネルからオープンソースライブラリを使用する方法について説明します。次の例は、Python `polars` パッケージを使用するジョブテンプレートです。

ジョブは、パッケージを取得する場所を Deadline Cloud に指示するキュー環境で定義された `CondaPackages`および `CondaChannels`パラメータを設定します。

パラメータを設定するジョブテンプレートの セクションは次のとおりです。

```
- name: CondaPackages
  description: A list of conda packages to install. The job expects a Queue Environment
 to handle this.
  type: STRING
  default: polars
- name: CondaChannels
```

```
  description: A list of conda channels to get packages from. The job expects a Queue
 Environment to handle this.
  type: STRING
  default: conda-forge
```

完全なサンプルジョブテンプレートの最新バージョンについては、「stage_1_self_contained_template/template.yaml」を参照してください。conda パッケージをロードするキュー環境の最新バージョンについては、GitHub の deadline-cloud-samples リポジトリの conda_queue_env_console_equivalent.yaml を参照してください。

deadline-cloud チャネルBlenderから取得する

次の例は、`deadline-cloud`conda チャネルBlenderから取得するジョブテンプレートを示しています。このチャネルは、Deadline Cloud がデジタルコンテンツ作成ソフトウェアに提供する送信者をサポートしますが、同じチャネルを使用して独自の使用のためにソフトウェアをロードできます。

`deadline-cloud` チャネルが提供するソフトウェアのリストについては、AWS 「Deadline Cloud ユーザーガイド」の「デフォルトキュー環境」を参照してください。

このジョブは、キュー環境で定義された `CondaPackages` パラメータを設定して、環境にロードするように Deadline Cloud Blenderに指示します。

パラメータを設定するジョブテンプレートの セクションは次のとおりです。

```
- name: CondaPackages
  type: STRING
  userInterface:
    control: LINE_EDIT
    label: Conda Packages
    groupLabel: Software Environment
  default: blender
  description: >
    Tells the queue environment to install Blender from the deadline-cloud conda
 channel.
```

ジョブテンプレートの完全な例の最新バージョンについては、「Blender_render/template.yaml」を参照してください。conda パッケージをロードするキュー環境の最新バージョンについては、 の deadline-cloud-samples リポジトリの conda_queue_env_console_equivalent.yaml を参照してくださいGitHub。

別のパッケージマネージャーを使用する

Deadline Cloud のデフォルトのパッケージマネージャーは conda です。などの別のパッケージマネージャーを使用する必要がある場合はRez、代わりにパッケージマネージャーを使用するスクリプトを含むカスタムキュー環境を作成できます。

このサンプルキュー環境は、 コンソールがサービスマネージドフリートに使用する環境と同じ動作を提供します。conda パッケージマネージャーを に置き換えますRez。

環境は、Deadline Cloud がワーカーでセッションを開始するときに実行される 3 つのスクリプトを定義します。最初のスクリプトは、 `onEnter`アクションが呼び出されたときに実行されます。他の 2 つを呼び出して環境変数を設定します。スクリプトの実行が完了すると、Rez環境は指定されたすべての環境変数セットで使用できます。

この例では、Rez パッケージに共有ファイルシステムを使用するカスタマーマネージドフリートがあることを前提としています。

この例の最新バージョンについては、 の deadline-cloud-samples リポジトリの rez_queue_env.yaml を参照してくださいGitHub。

S3 を使用して conda チャネルを作成する

`deadline-cloud` または `conda-forge`チャネルで使用できないアプリケーションのカスタムパッケージがある場合は、環境が使用するパッケージを含む conda チャネルを作成できます。パッケージを Amazon S3 バケットに保存し、アクセス AWS Identity and Access Management 許可を使用してチャネルへのアクセスを制御できます。

Deadline Cloud キューを使用して conda チャネルのパッケージを構築し、アプリケーションパッケージの更新と保守を容易にすることができます。

このアプローチの主な利点は、CUDA サポートの有無にかかわらず、パッケージ構築キューが複数の異なるオペレーティングシステム用のパッケージを作成できることです。これに対して、ワークステーションでパッケージを構築する場合は、これらのケースに対して異なるワークステーションを作成および管理する必要があります。

次の例は、 環境の および アプリケーションを提供する conda チャネルを作成する方法を示しています。例のアプリケーションは 4Blender.2 ですが、どの Deadline Cloud 統合アプリケーションも使用できます。

AWS CloudFormation テンプレートを使用して、パッケージ構築キューを含む Deadline Cloud ファームを作成するか、以下の手順に従ってサンプルファームを自分で作成できます。 AWS CloudFormation テンプレートについては、GitHub の AWS Deadline Cloud サンプルリポジトリの「スターター Deadline Cloud ファーム」を参照してください。

トピック

- パッケージ構築キューを作成する
- カスタム conda パッケージの本番キューのアクセス許可を設定する
- conda チャネルをキュー環境に追加する
- アプリケーションの conda パッケージを作成する
- の conda ビルドレシピを作成する Blender
- の conda ビルドレシピを作成する Autodesk Maya
- Autodesk Maya to Arnold (MtoA) プラグインの conda ビルドレシピを作成する

パッケージ構築キューを作成する

この例では、Deadline Cloud キューを作成して 4.2 Blender アプリケーションを構築します。これにより、conda チャネルとして使用される Amazon S3 バケットへの完成したパッケージの配信が簡単になり、既存のフリートを使用してパッケージを構築できます。これにより、管理するインフラストラクチャコンポーネントの数が減ります。

Deadline Cloud ユーザーガイドの「キューの作成」の手順に従います。以下の変更を加えます。

- ステップ 5 で、既存の S3 バケットを選択します。ビルドアーティファクトが通常の Deadline Cloud アタッチメントとは別のまま**`DeadlineCloudPackageBuild`**になるように、 などのルートフォルダ名を指定します。
- ステップ 6 では、パッケージ構築キューを既存のフリートに関連付けることも、現在のフリートが適切でない場合はまったく新しいフリートを作成することもできます。
- ステップ 9 で、パッケージ構築キューの新しいサービスロールを作成します。アクセス許可を変更して、パッケージのアップロードと conda チャネルのインデックス再作成に必要なアクセス許可をキューに付与します。

パッケージ構築キューのアクセス許可を設定する

パッケージビルドキューがキューの S3 バケット内の`/Conda`プレフィックスにアクセスできるようにするには、キューのロールを変更して読み取り/書き込みアクセスを許可する必要があります。ロールには、パッケージビルドジョブが新しいパッケージをアップロードし、チャネルのインデックスを再作成できるように、次のアクセス許可が必要です。

- `s3:GetObject`
- `s3:PutObject`
- `s3:ListBucket`
- `s3:GetBucketLocation`
- `s3:DeleteObject`

1. Deadline Cloud コンソールを開き、パッケージビルドキューのキューの詳細ページに移動します。
2. キューサービスロールを選択し、キューの編集を選択します。

3. キューサービスロールセクションまでスクロールし、IAM コンソールでこのロールを表示するを選択します。
4. アクセス許可ポリシーのリストから、キューの AmazonDeadlineCloudQueuePolicy を選択します。
5. アクセス許可タブから、編集 を選択します。
6. キューサービスロールを次のように更新します。*amzn-s3-demo-bucket* と *111122223333* を独自のバケットとアカウントに置き換えます。

```
{
   "Effect": "Allow",
   "Sid": "CustomCondaChannelReadWrite",
   "Action": [
    "s3:GetObject",
    "s3:PutObject",
    "s3:DeleteObject",
    "s3:ListBucket",
    "s3:GetBucketLocation"
   ],
   "Resource": [
    "arn:aws:s3:::amzn-s3-demo-bucket",
    "arn:aws:s3:::amzn-s3-demo-bucket/Conda/*"   ],
   "Condition": {
    "StringEquals": {
     "aws:ResourceAccount": "111122223333"
    }
   }
  },
```

カスタム conda パッケージの本番キューのアクセス許可を設定する

本番キューには、キューの S3 バケットの /Conda プレフィックスへの読み取り専用アクセス許可が必要です。本番キューに関連付けられたロールの AWS Identity and Access Management (IAM) ページを開き、以下を使用してポリシーを変更します。

1. Deadline Cloud コンソールを開き、パッケージビルドキューのキューの詳細ページに移動します。
2. キューサービスロールを選択し、キューの編集を選択します。

3. キューサービスロールセクションまでスクロールし、IAM コンソールでこのロールを表示するを選択します。
4. アクセス許可ポリシーのリストから、キューの AmazonDeadlineCloudQueuePolicy を選択します。
5. アクセス許可タブから、編集を選択します。
6. 次のような新しいセクションをキューサービスロールに追加します。*amzn-s3-demo-bucket* と *111122223333* を独自のバケットとアカウントに置き換えます。

```
{
   "Effect": "Allow",
   "Sid": "CustomCondaChannelReadOnly",
   "Action": [
    "s3:GetObject",
    "s3:ListBucket"
   ],
   "Resource": [
    "arn:aws:s3:::amzn-s3-demo-bucket",
    "arn:aws:s3:::amzn-s3-demo-bucket/Conda/*"
   ],
   "Condition": {
    "StringEquals": {
     "aws:ResourceAccount": "111122223333"
    }
   }
  },
```

conda チャネルをキュー環境に追加する

S3 conda チャネルを使用するには、Deadline Cloud に送信するジョブの `CondaChannels`パラメータに`s3://amzn-s3-demo-bucket/Conda/Default`チャネルの場所を追加する必要があります。Deadline Cloud で提供される送信者は、カスタム conda チャネルとパッケージを指定するためのフィールドを提供します。

本番稼働用キューの conda キュー環境を編集することで、すべてのジョブの変更を回避できます。サービスマネージドキューの場合は、次の手順を使用します。

1. Deadline Cloud コンソールを開き、本番キューのキューの詳細ページに移動します。
2. 環境タブを選択します。

3. Conda キュー環境を選択し、編集を選択します。
4. JSON エディタを選択し、スクリプトで のパラメータ定義を見つけます`CondaChannels`。
5. 新しく作成された S3 conda チャネルで始まる`default: "deadline-cloud"`ように行を編集します。

```
default: "s3://amzn-s3-demo-bucket/Conda/Default deadline-cloud"
```

サービスマネージドフリートは、デフォルトで conda の厳密なチャネル優先度を有効にします。新しい S3 チャネルを使用すると、conda は`deadline-cloud`チャネルの使用を停止します。`deadline-cloud` チャネル`blender=3.6`から を使用して正常に完了したジョブは、4.2 Blender を使用しているため失敗します。

カスタマーマネージドフリートの場合、Deadline Cloud サンプルGitHubリポジトリの Conda キュー環境サンプルのいずれかを使用して、conda パッケージの使用を有効にできます。

アプリケーションの conda パッケージを作成する

依存関係を含むアプリケーション全体を conda パッケージに結合できます。Deadline Cloud がサービスマネージドフリートの期限クラウドチャネルで提供するパッケージは、このバイナリ再パッケージ化アプローチを使用します。これにより、conda 仮想環境に合わせてインストールと同じファイルが整理されます。

conda のアプリケーションを再パッケージ化する場合、次の 2 つの目標があります。

- アプリケーションのほとんどのファイルは、プライマリ conda 仮想環境構造とは別にする必要があります。その後、環境はアプリケーションを conda-forge などの他のソースのパッケージと混在させることができます。
- conda 仮想環境がアクティブ化されると、アプリケーションは PATH 環境変数から使用可能になります。

conda のアプリケーションを再パッケージ化するには

1. conda のアプリケーションを再パッケージ化するには、アプリケーションをインストールする conda ビルドレシピを などのサブディレクトリに書き込みます`$CONDA_PREFIX/opt/<application-name>`。これにより、 `bin`や などの標準プレフィックスディレクトリから分離されます`lib`。

2. 次に、シンボリックリンクまたは起動スクリプトを に追加`$CONDA_PREFIX/bin`して、アプリケーションバイナリを実行します。

   または、`conda activate`コマンドが実行する activate.d スクリプトを作成して、アプリケーションバイナリディレクトリを PATH に追加します。シンボリックリンクがサポートされていない ではWindows、環境を作成できるすべての場所で、代わりにアプリケーションの起動または activate.d スクリプトを使用します。
3. 一部のアプリケーションは、Deadline Cloud のサービスマネージドフリートにデフォルトでインストールされていないライブラリに依存します。例えば、通常、X11 ウィンドウシステムは非インタラクティブジョブには必要ありませんが、一部のアプリケーションではグラフィカルインターフェイスなしで実行する必要があります。これらの依存関係は、作成するパッケージ内で指定する必要があります。
4. パッケージ化するアプリケーションの著作権およびライセンス契約に従ってください。conda チャネルにはプライベート Amazon S3 バケットを使用して、ディストリビューションを制御し、ファームへのパッケージアクセスを制限することをお勧めします。

の conda ビルドレシピを作成する Blender

さまざまなアプリケーションを使用して conda ビルドレシピを作成できます。 Blenderは無料で使用でき、conda で簡単にパッケージ化できます。Blender Foundation は、複数のオペレーティングシステム用のアプリケーションアーカイブを提供します。Windows .zip ファイルと Linux .tar.xz ファイルを使用するサンプル conda ビルドレシピを作成しました。このセクションでは、Blender4.2 conda ビルドレシピを使用する方法について説明します。

deadline-cloud.yaml ファイルは、パッケージジョブを Deadline Cloud に送信するための conda プラットフォームおよびその他のメタデータを指定します。このレシピには、その仕組みを示すローカルソースアーカイブ情報が含まれています。linux-64 conda プラットフォームは、最も一般的な設定に合わせてデフォルトのジョブ送信でビルドするように設定されています。deadline-cloud.yaml は次のようになります。

```
condaPlatforms:
  - platform: linux-64
    defaultSubmit: true
    sourceArchiveFilename: blender-4.2.1-linux-x64.tar.xz
    sourceDownloadInstructions: 'Run "curl -LO https://download.blender.org/release/
Blender4.2/blender-4.2.1-linux-x64.tar.xz"'
  - platform: win-64
    defaultSubmit: false
```

```
    sourceArchiveFilename: blender-4.2.1-windows-x64.zip
    sourceDownloadInstructions: 'Run "curl -LO https://download.blender.org/release/
Blender4.2/blender-4.2.1-windows-x64.zip"'
```

recipe ディレクトリ内のファイルを確認します。レシピのメタデータは recipe/meta.yaml にあります。また、conda build meta.yaml ドキュメントを参照して、ファイルが YAML を生成するテンプレートである方法などの詳細を確認することもできます。テンプレートは、バージョン番号を 1 回だけ指定し、オペレーティングシステムに基づいて異なる値を提供するために使用されます。

で選択したビルドオプションを確認してmeta.yaml、さまざまなバイナリ再配置と動的共有オブジェクト (DSO) リンクチェックをオフにできます。これらのオプションは、任意のディレクトリプレフィックスの conda 仮想環境にパッケージがインストールされている場合のパッケージの動作を制御します。デフォルト値は、すべての依存関係ライブラリを別のパッケージにパッケージ化することを簡素化しますが、アプリケーションをバイナリで再パッケージ化する場合は、変更する必要があります。

パッケージ化するアプリケーションに追加の依存関係ライブラリが必要な場合、またはアプリケーションのプラグインを個別にパッケージ化している場合、DSO エラーが発生する可能性があります。これらのエラーは、依存関係が必要な実行可能ファイルまたはライブラリのライブラリ検索パスにない場合に発生します。アプリケーションは、システムにインストール/usr/libされたときに、/libや などのグローバルに定義されたパスにあるライブラリに依存します。ただし、conda 仮想環境はどこにでも配置できるため、絶対的に使用するパスはありません。Conda は、 Linuxと の両方macOSがサポートする相対的な RPATH 機能を使用してこれを処理します。詳細については、パッケージの再配置に関する conda ビルドドキュメントを参照してください。

Blender アプリケーションアーカイブはこれを念頭に置いて構築されているため、 は RPATH の調整を必要としません。これを必要とするアプリケーションの場合、conda ビルドと同じツールを使用できます。Linux patchelfでは 、 install_name_tool では ですmacOS。

パッケージのビルド中、build.sh または build_win.sh (で呼び出されますbld.bat) スクリプトが実行され、パッケージの依存関係で準備された環境にファイルがインストールされます。これらのスクリプトは、インストールファイルをコピーし、 からシンボリックリンクを作成し$PREFIX/bin、アクティベーションスクリプトを設定します。ではWindows、シンボリックリンクは作成されず、代わりにアクティベーションスクリプトの PATH に Blender ディレクトリを追加します。

conda ビルドレシピWindowsの一部には cmd.exe .bat ファイルbashの代わりに を使用します。これにより、ビルドスクリプト間の一貫性が向上します。bash での の使用に関するヒントについては、Deadline Cloud 開発者ガイドのワークロードの移植性に関する推奨事項を参照してください

Windows。git for Windows をインストールして deadline-cloud-samples git リポジトリのクローンを作成している場合は、既に にアクセスできますbash。

conda ビルド環境変数のドキュメントには、ビルドスクリプトで使用できる値が一覧表示されています。これらの値には、ソースアーカイブデータ$SRC_DIR、$PREFIXインストールディレクトリ、レシピからの他のファイル$RECIPE_DIRへのアクセス、$PKG_NAME$PKG_VERSIONパッケージ名とバージョン、ターゲット conda プラットフォーム$target_platformが含まれます。

4.2 Blender パッケージジョブを送信する

Blender アーカイブをダウンロードし、パッケージ構築キューにジョブを送信することで、独自の Blender 4.2 conda パッケージを構築してジョブをレンダリングできます。キューは、関連付けられたフリートにジョブを送信してパッケージを構築し、conda チャネルのインデックスを再作成します。

これらの手順では、bash 互換シェルから git を使用して、Deadline Cloud サンプルGitHubリポジトリから OpenJD パッケージビルドジョブといくつかの conda レシピを取得します。また、以下も必要になります:

- を使用している場合はWindows、git をインストールするときに bash のバージョン git BASH がインストールされます。
- Deadline Cloud CLI がインストールされている必要があります。
- Deadline Cloud モニターにログインする必要があります。

1. 次のコマンドを使用して Deadline Cloud 設定 GUI を開き、デフォルトのファームとキューをパッケージ構築キューに設定します。

```
deadline config gui
```

2. 次のコマンドを使用して、Deadline Cloud サンプルGitHUbリポジトリのクローンを作成します。

```
git clone https://github.com/aws-deadline/deadline-cloud-samples.git
```

3. ディレクトリの deadline-cloud-samples ディレクトリに変更conda_recipesします。

```
cd deadline-cloud-samples/conda_recipes
```

4. というスクリプトを実行しますsubmit-package-job。スクリプトは、スクリプトBlenderを初めて実行するときにダウンロードする手順を提供します。

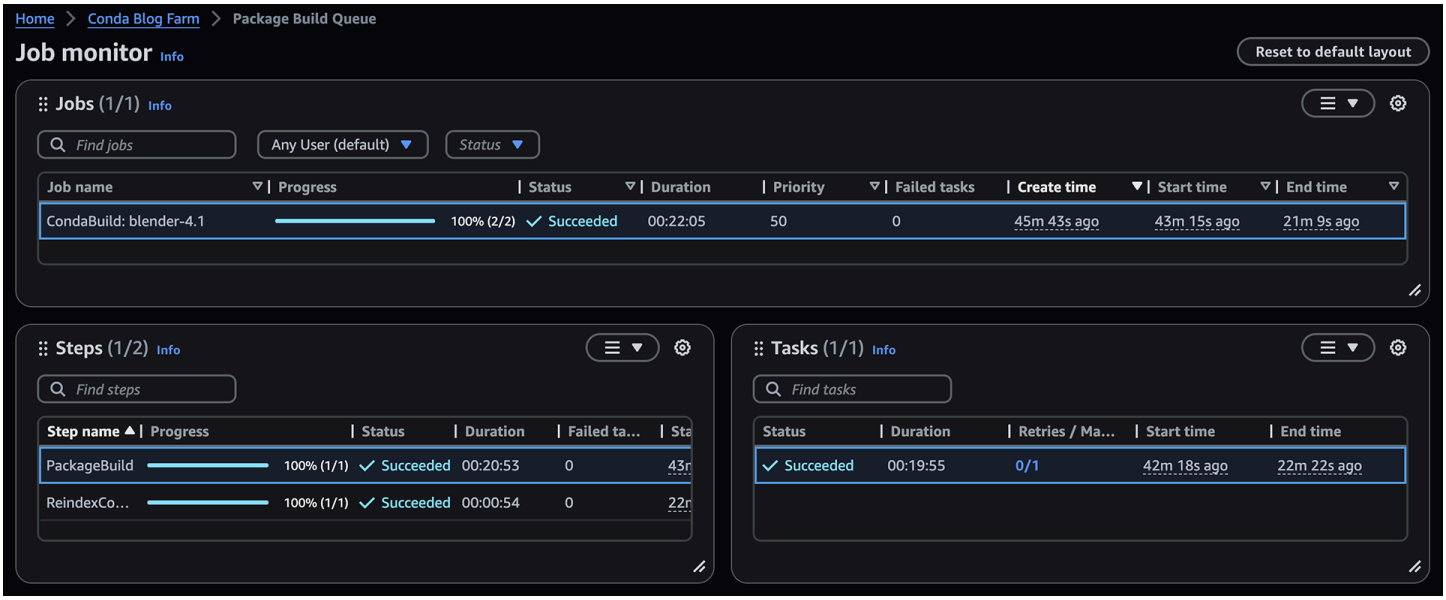
```
./submit-package-job blender-4.2/
```

5. をダウンロードする手順に従いますBlender。アーカイブを取得したら、submit-package-jobスクリプトを再度実行します。

```
./submit-package-job blender-4.2/
```

ジョブを送信したら、Deadline Cloud モニターを使用して、実行中のジョブの進行状況とステータスを表示します。

モニタの左下には、ジョブの 2 つのステップ、パッケージの構築、インデックスの再作成が表示されます。右下には、各タスクの個々のステップが表示されます。この例では、タスクごとに 1 つのステップがあります。



モニタの左下には、 ジョブの 2 つのステップがあります。パッケージを構築し、conda チャネルのインデックスを再作成します。右下には、各ステップの個々のタスクが表示されます。この例では、ステップごとに 1 つのタスクしかありません。

パッケージ構築ステップのタスクを右クリックし、ログの表示を選択すると、モニターに、タスクがワーカーでどのようにスケジュールされているかを示すセッションアクションのリストが表示されます。アクションは次のとおりです。

- アタッチメントの同期 – このアクションは、ジョブアタッチメントファイルシステムに使用される設定に応じて、入力ジョブアタッチメントをコピーするか、仮想ファイルシステムをマウントします。
- Launch Conda – このアクションは、キューの作成時にデフォルトで追加されたキュー環境からのものです。ジョブは conda パッケージを指定しないため、すぐに終了し、conda 仮想環境は作成されません。
- CondaBuild Env を起動する – このアクションは、conda パッケージを構築し、チャネルのインデックスを再作成するために必要なソフトウェアを含むカスタム conda 仮想環境を作成します。conda-forge チャネルからインストールされます。
- タスク実行 – このアクションはBlenderパッケージをビルドし、結果を Amazon S3 にアップロードします。

アクションを実行すると、構造化された形式のログが Amazon CloudWatch に送信されます。ジョブが完了したら、すべてのタスクのログを表示を選択して、ジョブが実行される環境の設定と削除に関する追加のログを表示します。

4.2 Blender レンダリングジョブでパッケージをテストする

Blender 4.2 パッケージを構築し、S3 conda チャネルを使用するように本番キューを設定したら、ジョブを送信してパッケージでレンダリングできます。Blender シーンがない場合は、Blenderデモファイルページから Blender 3.5 - コージーキッチンシーンをダウンロードします。

前にダウンロードした Deadline Cloud サンプルGitHubリポジトリには、次のコマンドを使用してBlenderシーンをレンダリングするサンプルジョブが含まれています。

```
deadline bundle submit blender_render \
     -p CondaPackages=blender=4.2 \
     -p BlenderSceneFile=/path/to/downloaded/blender-3.5-splash.blend \
     -p Frames=1
```

Deadline Cloud モニターを使用して、ジョブの進行状況を追跡できます。

1. モニターで、送信したジョブのタスクを選択し、ログを表示するオプションを選択します。
2. ログビューの右側で、Launch Conda セッションアクションを選択します。

アクションがキュー環境に設定された 2 つの conda チャネルで Blender 4.2 を検索し、S3 チャネルでパッケージが見つかったことがわかります。

の conda ビルドレシピを作成する Autodesk Maya

商用アプリケーションを conda パッケージとしてパッケージ化できます。 の conda ビルドレシピを作成するBlenderでは、シンプルな再配置可能なアーカイブファイルとして、オープンソースのライセンス条項の下で利用可能なアプリケーションをパッケージ化する方法を学習しました。商用アプリケーションは多くの場合、インストーラを介して配布され、使用するライセンス管理システムがある場合があります。

次のリストは、商用アプリケーションのパッケージングに一般的に関連する要件を持つアプリケーションの conda パッケージを作成するで説明されている基本に基づいています。サブ箇条書きの詳細は、ガイドラインを に適用する方法を示していますMaya。

- アプリケーションのライセンス権限と制限を理解します。ライセンス管理システムの設定が必要になる場合があります。アプリケーションに強制が含まれていない場合は、権限に従ってファームを設定する必要があります。
  - Autodesk Cloud Rights に関するサブスクリプション特典に関するよくある質問をお読みになり、 に適用されるMaya可能性のある のクラウド権限を理解してください。必要に応じて Deadline Cloud ファームを設定します。
  - Autodesk 製品は、 という名前のファイルに依存しています`ProductInformation.pit`。このファイルのほとんどの設定には、 システムへの管理者アクセスが必要です。これは、サービスマネージドフリートでは利用できません。シンクライアントの製品機能は、これを処理する再配置可能な方法を提供します。詳細については、「Maya および MotionBuilder のシンクライアントライセンス」を参照してください。
- 一部のアプリケーションは、サービスマネージドフリートワーカーホストにインストールされていないライブラリに依存するため、パッケージで提供する必要があります。これは、アプリケーションパッケージ内に直接配置することも、別の依存関係パッケージに配置することもできます。
  - Maya は、Freetype や fontconfig など、このようなライブラリの数に依存します。これらのライブラリが AL2023 `dnf`の など、システムパッケージマネージャーで使用できる場合は、アプリケーションのソースとして使用できます。これらの RPM パッケージは再配置できるように構築されていないため、 などのツールを使用して`patchelf`、Mayaインストールプレフィックス内で依存関係が解決されるようにする必要があります。
- インストールには管理者アクセスが必要になる場合があります。サービスマネージドフリートは管理者アクセスを提供しないため、このアクセスを持つシステムでインストールを実行する必要があります。次に、パッケージビルドジョブが使用するのに必要なファイルのアーカイブを作成します。

- のWindowsインストーラには管理者アクセスMayaが必要なため、conda パッケージを構築するには、まずこのようなアーカイブを作成する手動プロセスが必要です。
- プラグインの登録方法を含むアプリケーション設定は、オペレーティングシステムまたはユーザーレベルで定義できます。conda 仮想環境に配置する場合、プラグインには、含まれている方法でアプリケーションと統合する方法が必要であり、仮想環境プレフィックスの外部にファイルやその他のデータを書き込むことはありません。これをアプリケーションの conda パッケージから設定することをお勧めします。
  - サンプルMayaパッケージは、環境変数を定義`MAYA_NO_HOME=1`してユーザーレベル設定から分離し、モジュール検索パスを に追加`MAYA_MODULE_PATH`して、個別にパッケージ化されたプラグインを仮想環境内から統合できるようにします。サンプルMtoAパッケージは、Maya起動時にロードするこれらのディレクトリの 1 つに .mod ファイルを配置します。

レシピメタデータを記述する

1. ブラウザまたはリポジトリのローカルクローンのテキストエディタで GitHub deadline-cloud-samples/conda_recipes/maya-2025 ディレクトリを開きます。

   ファイルは、 のパッケージを構築する conda ビルドプラットフォームと、アプリケーションを取得する場所を`deadline-cloud.yaml`記述します。レシピサンプルはLinux、 と の両方のWindowsビルドを指定し、デフォルトでLinuxのみ送信されます。

2. Autodesk ログインから完全なMayaインストーラをダウンロードします。の場合Linux、パッケージビルドはアーカイブを直接使用できるため、`conda_recipes/archive_files`ディレクトリに直接配置します。の場合Windows、インストーラを実行するには管理者アクセスが必要です。インストーラを実行し、使用するパッケージレシピのアーカイブに必要なファイルを収集する必要があります。レシピの README.md ファイルは、このアーティファクトを作成するための繰り返し可能な手順を文書化します。この手順では、新しく起動された Amazon EC2 インスタンスを使用して、インストール用のクリーンな環境を提供し、その結果を保存した後に終了できます。管理者アクセスを必要とする他のアプリケーションをパッケージ化するには、アプリケーションに必要なファイルのセットを決定したら、同様の手順に従います。

3. recipe/recipe.yaml ファイルと recipe/meta.yaml ファイルを開き、rattler-build と conda-build の設定を確認または編集します。パッケージ化するアプリケーションのパッケージ名とバージョンを設定できます。

   ソースセクションには、ファイルの sha256 ハッシュを含むアーカイブへの参照が含まれています。これらのファイルを新しいバージョンなどに変更するたびに、これらの値を計算して更新する必要があります。

ビルドセクションには、パッケージが使用する特定のライブラリおよびバイナリディレクトリの自動メカニズムが正しく機能しないため、デフォルトのバイナリ再配置オプションをオフにするオプションが主に含まれています。

最後に、 about セクションでは、conda チャネルのコンテンツを参照または処理するときに使用できるアプリケーションに関するメタデータを入力できます。

パッケージビルドスクリプトを記述する

1. Maya サンプル conda ビルドレシピのパッケージビルドスクリプトには、スクリプトが実行するステップを説明するコメントが含まれています。コメントとコマンドを読んで、以下を確認してください。

   - レシピが から RPM ファイルを処理する方法 Autodesk
   - レシピがインストール先である conda 仮想環境にインストールを再配置できるようにするために適用される変更
   - レシピが `MAYA_LOCATION`や などのユーティリティ変数を設定する方法。ソフトウェア`MAYA_VERSION`はこれを使用して実行中の Maya を理解できます。

2. で Linuxrecipe/build.sh ファイルを開き、パッケージビルドスクリプトを確認または編集します。

   ではWindows、recipe/build_win.sh ファイルを開いて、パッケージビルドスクリプトを確認または編集します。

Maya パッケージを構築するジョブを送信する

1. GitHub deadline-cloud-samples リポジトリのクローンに `conda_recipes` ディレクトリを入力します。
2. Deadline Cloud ファームが Deadline Cloud CLI 用に設定されていることを確認します。Amazon S3 を使用して conda チャネルを作成するステップに従った場合は、CLI 用にファームを設定する必要があります。
3. 次のコマンドを実行して、 Linux と Windowsパッケージの両方を構築するジョブを送信します。

   ```
   ./submit-package-job maya-2025 --all-platforms
   ```

Autodesk Maya to Arnold (MtoA) プラグインの conda ビルドレシピを作成する

商用アプリケーション用のプラグインを conda パッケージとしてパッケージ化できます。プラグインは、アプリケーションが提供するアプリケーションバイナリインターフェイス (ABI) を使用してそのアプリケーションの機能を拡張する動的にロードされたライブラリです。Maya to Arnold (MtoA) プラグインは、Arnoldレンダラーを 内のオプションとして追加しますMaya。

プラグインのパッケージの作成はアプリケーションのパッケージ化に似ていますが、パッケージは別のパッケージに含まれるホストアプリケーションと統合されます。次のリストでは、この作業を行うための要件について説明します。

- ビルドレシピと に、ビルド`meta.yaml`と実行の両方の依存関係としてホストアプリケーションパッケージを含めます`recipe.yaml`。バージョン制約を使用して、ビルドレシピが互換性のあるパッケージでのみインストールされるようにします。
  - MtoA サンプルビルドレシピはMayaパッケージに依存し、 バージョンの==制約を使用します。
- プラグインを登録するには、ホストアプリケーションパッケージの規則に従います。
  - Maya パッケージは、プラグインが `.mod`ファイルを配置するためのMayaモジュールパス`$PREFIX/usr/autodesk/maya$MAYA_VERSION/modules`を仮想環境 に設定します。MtoA サンプルビルドレシピは、このディレクトリ`mtoa.mod`に ファイルを作成します。

レシピメタデータを記述する

1. ブラウザまたはリポジトリのローカルクローンのテキストエディタで GitHub deadline-cloud-samples/conda_recipes/maya-mtoa-2025 ディレクトリを開きます。

   レシピは Maya conda ビルドレシピと同じパターンに従い、同じソースアーカイブを使用してプラグインをインストールします。

2. recipe/recipe.yaml ファイルと recipe/meta.yaml ファイルを開き、rattler-build と conda-build の設定を確認または編集します。これらのファイルは、パッケージのビルド`maya`時およびプラグインを実行する仮想環境の作成時に への依存関係を指定します。

パッケージビルドスクリプトを記述する

- MtoA サンプル conda ビルドレシピのパッケージビルドスクリプトには、スクリプトが実行するステップを説明するコメントが含まれています。コメントとコマンドを読み、レシピがMaya

パッケージで指定されたディレクトリ`mtoa.mod`にファイルをインストールMtoAして作成する方法について説明します。

Arnold と は同じライセンス技術Mayaを使用するため、Mayaconda ビルドレシピにはすでに が必要とする情報が含まれていますArnold。

Linux とWindowsビルドスクリプトの違いは、Mayaconda ビルドレシピの違いと似ています。

Maya MtoA プラグインパッケージを構築するジョブを送信する

1. GitHub deadline-cloud-samples リポジトリのクローンに `conda_recipes` ディレクトリを入力します。
2. 前のセクションでMayaホストアプリケーションのパッケージを構築したことを確認します。
3. Deadline Cloud ファームが Deadline Cloud CLI 用に設定されていることを確認します。Amazon S3 を使用して conda チャネルを作成するステップに従った場合は、CLI 用にファームを設定する必要があります。
4. 次のコマンドを実行して、 Linux と Windowsパッケージの両方を構築するジョブを送信します。

```
./submit-package-job maya-mtoa-2025 --all-platforms
```

Maya レンダリングジョブを使用してパッケージをテストする

2025 パッケージと Maya MtoAパッケージを構築したら、 パッケージでレンダリングするジョブを送信できます。ジョブバンドルのサンプルを含むターンテーブルMaya/Arnoldは、 Mayaと を含むアニメーションをレンダリングしますArnold。このサンプルでは、FFmpeg を使用して動画をエンコードします。conda キュー環境のデフォルトリストに `CondaChannels` conda-forge チャネルを追加して、`ffmpeg`パッケージのソースを提供できます。

deadlinedeadline-cloud-samples の git クローンの `job_bundles` ディレクトリから、次のコマンドを実行します。

```
deadline bundle submit turntable_with_maya_arnold
```

Deadline Cloud モニターを使用して、ジョブの進行状況を追跡できます。

1. モニターで、送信したジョブのタスクを選択し、ログを表示するオプションを選択します。

2. ログビューの右側で、Launch Conda セッションアクションを選択します。

アクションがキュー環境用に設定された conda チャネルmaya-mtoaで mayaと を検索し、S3 チャネルでパッケージを見つけたことを確認できます。

Deadline Cloud に送信するジョブを構築する

ジョブバンドルを使用して Deadline Cloud にジョブを送信します。ジョブバンドルは、Open Job Description (OpenJD) ジョブテンプレートや、ジョブのレンダリングに必要なアセットファイルを含むファイルのコレクションです。

ジョブテンプレートは、ワーカーがアセットを処理してアクセスする方法を説明し、ワーカーが実行するスクリプトを提供します。ジョブバンドルを使用すると、アーティスト、テクニカルディレクター、パイプライン開発者は、ローカルワークステーションまたはオンプレミスレンダーファームから Deadline Cloud に複雑なジョブを簡単に送信できます。これは、スケーラブルなオンデマンドコンピューティングリソースを必要とする大規模な視覚効果、アニメーション、またはその他のメディアレンダリングプロジェクトに取り組むチームにとって特に便利です。

ローカルファイルシステムを使用してジョブバンドルを作成し、ファイルを保存し、テキストエディタを使用してジョブテンプレートを作成できます。バンドルを作成したら、Deadline Cloud CLI または Deadline Cloud 送信者などのツールを使用して Deadline Cloud にジョブを送信します。

ワーカー間で共有されているファイルシステムにアセットを保存することも、Deadline Cloud ジョブアタッチメントを使用して、ワーカーがアクセスできる S3 バケットへのアセットの移動を自動化することもできます。ジョブアタッチメントは、出力をジョブからワークステーションに戻すのにも役立ちます。

以下のセクションでは、ジョブバンドルを作成して Deadline Cloud に送信する詳細な手順について説明します。

トピック

- Deadline Cloud の Open Job Description (OpenJD) テンプレート
- ジョブでのファイルの使用
- ジョブアタッチメントを使用してファイルを共有する
- ジョブのリソース制限を作成する
- Deadline Cloud にジョブを送信する方法
- Deadline Cloud でジョブをスケジュールする
- Deadline Cloud でジョブを変更する

Deadline Cloud の Open Job Description (OpenJD) テンプレート

ジョブバンドルは、Deadline Cloud のジョブを定義するために使用するツールの 1 AWS つです。Open Job Description (OpenJD) テンプレートを、ジョブアタッチメントで使用するファイルやディレクトリなどの追加情報でグループ化します。Deadline Cloud コマンドラインインターフェイス (CLI) を使用して、ジョブバンドルを使用してキューを実行するジョブを送信します。

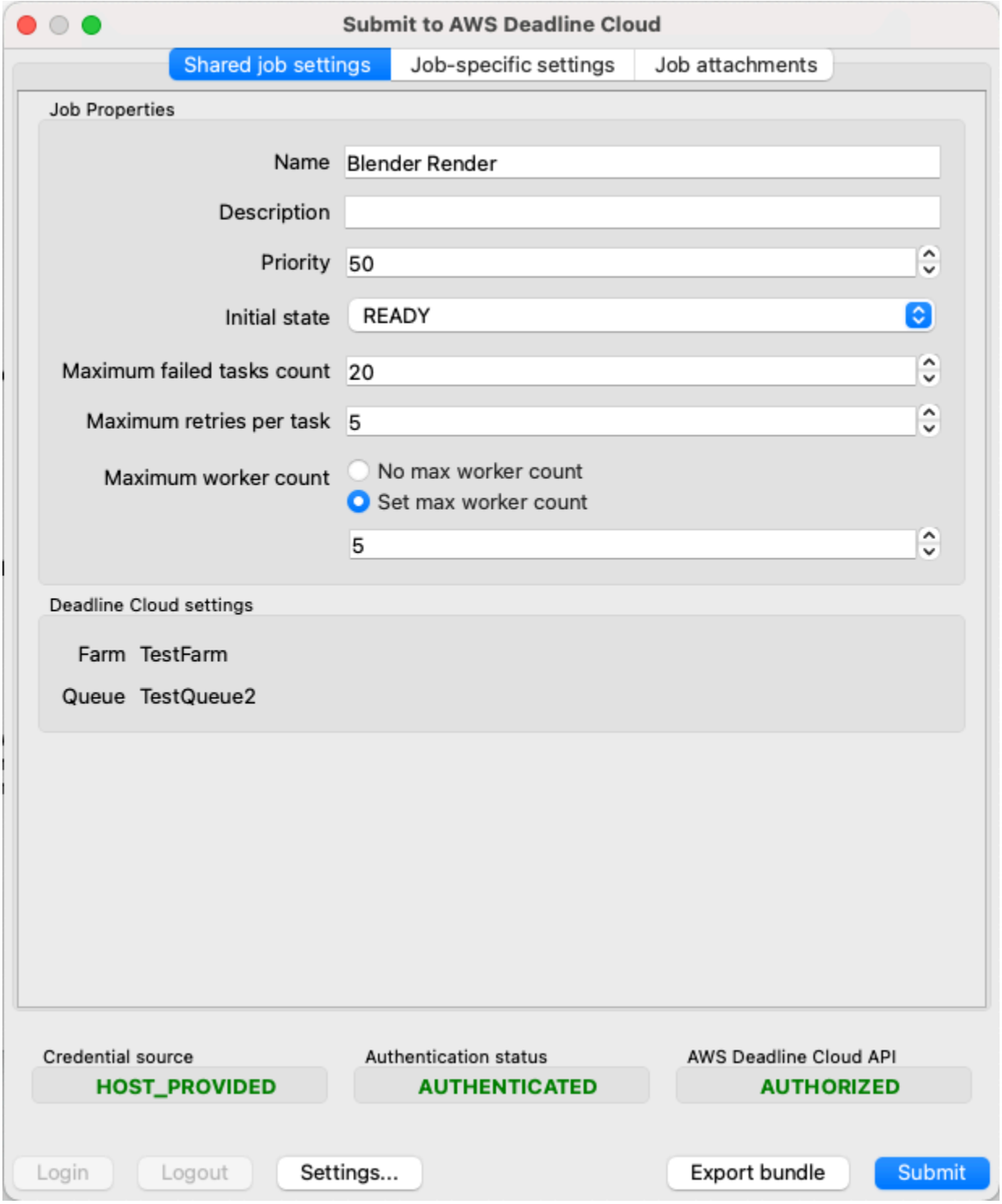
ジョブバンドルは、OpenJD ジョブテンプレート、ジョブを定義する他のファイル、ジョブの入力に必要なジョブ固有のファイルを含むディレクトリ構造です。ジョブを YAML ファイルまたは JSON ファイルとして定義するファイルを指定できます。

必要なファイルは `template.yaml`または のみです`template.json`。次のファイルを含めることもできます。

```
/template.yaml (or template.json)
/asset_references.yaml (or asset_references.json)
/parameter_values.yaml (or parameter_values.json)
/other job-specific files and directories
```

Deadline Cloud CLI とジョブアタッチメントでカスタムジョブの送信にジョブバンドルを使用するか、グラフィカルな送信インターフェイスを使用できます。例えば、GitHub の Blender サンプルは次のとおりです。Blender サンプルディレクトリの folloowing コマンドを使用してサンプルを実行するには：

```
deadline bundle gui-submit blender_render
```



ジョブ固有の設定パネルは、ジョブテンプレートで定義されたジョブパラメータのuserInterfaceプロパティから生成されます。

コマンドラインを使用してジョブを送信するには、次のようなコマンドを使用できます。

```
deadline bundle submit \
    --yes \
    --name Demo \
     -p BlenderSceneFile=location of scene file \
     -p OutputDir=file pathe for job output \
      blender_render/
```

または、Python `deadline` パッケージで `deadline.client.api.create_job_from_job_bundle`関数を使用できます。

Autodesk Maya プラグインなど、Deadline Cloud に付属しているすべてのジョブ送信者プラグインは、送信用のジョブバンドルを生成し、Deadline Cloud Python パッケージを使用してジョブを Deadline Cloud に送信します。送信されたジョブバンドルは、ワークステーションのジョブ履歴ディレクトリまたは送信者を使用して確認できます。ジョブ履歴ディレクトリは、次のコマンドで確認できます。

```
deadline config get settings.job_history_dir
```

ジョブが Deadline Cloud ワーカーで実行されている場合、ジョブに関する情報を提供する環境変数にアクセスできます。環境変数は次のとおりです。

変数名	利用可能
DEADLINE_FARM_ID	すべてのアクション
DEADLINE_FLEET_ID	すべてのアクション
DEADLINE_WORKER_ID	すべてのアクション
DEADLINE_QUEUE_ID	すべてのアクション
DEADLINE_JOB_ID	すべてのアクション
DEADLINE_SESSION_ID	すべてのアクション
DEADLINE_SESSIONACTION_ID	すべてのアクション
DEADLINE_TASK_ID	タスクアクション

トピック

- ジョブバンドルのジョブテンプレート要素
- ジョブバンドルのパラメータ値要素
- ジョブバンドルのアセット参照要素

ジョブバンドルのジョブテンプレート要素

ジョブテンプレートは、ランタイム環境と、Deadline Cloud ジョブの一部として実行されるプロセスを定義します。テンプレートにパラメータを作成して、プログラミング言語の関数と同様に、入力値のみが異なるジョブを作成することができます。

Deadline Cloud にジョブを送信すると、キューに適用されているキュー環境で実行されます。キュー環境は、Open Job Description (OpenJD) 外部環境仕様を使用して構築されます。詳細については、OpenJD GitHub リポジトリの「環境テンプレート」を参照してください。

OpenJD ジョブテンプレートを使用したジョブの作成の概要については、OpenJD GitHub リポジトリでのジョブの作成の概要」を参照してください。詳細については、「ジョブの実行方法」を参照してください。OpenJD GitHub リポジトリの `samples` ディレクトリの にジョブテンプレートのサンプルがあります。

ジョブテンプレートは、YAML 形式 (`template.yaml`) または JSON 形式 () で定義できます`template.json`。このセクションの例は YAML 形式で示されています。

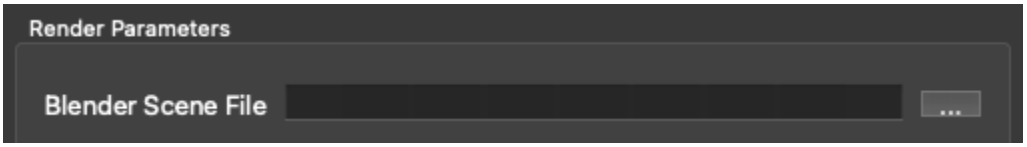
たとえば、`blender_render`サンプルのジョブテンプレートでは、入力パラメータをファイルパス`BlenderSceneFile`として定義します。

```
- name: BlenderSceneFile
  type: PATH
  objectType: FILE
  dataFlow: IN
  userInterface:
    control: CHOOSE_INPUT_FILE
    label: Blender Scene File
    groupLabel: Render Parameters
    fileFilters:
    - label: Blender Scene Files
      patterns: ["*.blend"]
    - label: All Files
      patterns: ["*"]
  description: >
```

```
    Choose the Blender scene file to render. Use the 'Job Attachments' tab
    to add textures and other files that the job needs.
```

userInterface プロパティは、 コマンドを使用し、Autodesk Maya などのアプリケーションのジョブ送信プラグイン内でdeadline bundle gui-submit、コマンドラインの両方で自動的に生成されるユーザーインターフェイスの動作を定義します。

この例では、 BlenderSceneFileパラメータの値を入力するための UI ウィジェットは、ファイルのみを表示する.blendファイル選択ダイアログです。



userInteface 要素を使用するその他の例については、GitHub の deadline-cloud-samples リポジトリにある gui_control_showcase サンプルを参照してください。

プロパティobjectTypeと dataFlowプロパティは、ジョブバンドルからジョブを送信するときのジョブアタッチメントの動作を制御します。この場合、 objectType: FILEとBlenderSceneFileは、 の値がジョブアタッチメントの入力ファイルであるdataFlow:INことを意味します。

対照的に、 OutputDirパラメータの定義には objectType: DIRECTORYと がありますdataFlow: OUT。

```
- name: OutputDir
  type: PATH
  objectType: DIRECTORY
  dataFlow: OUT
  userInterface:
    control: CHOOSE_DIRECTORY
    label: Output Directory
    groupLabel: Render Parameters
  default: "./output"
  description: Choose the render output directory.
```

OutputDir パラメータの値は、ジョブが出力ファイルを書き込むディレクトリとしてジョブアタッチメントによって使用されます。

objectType および dataFlowプロパティの詳細については、Open Job Description 仕様のJobPathParameterDefinition」を参照してください。

残りのblender_renderジョブテンプレートサンプルでは、ジョブのワークフローを単一のステップとして定義し、アニメーションの各フレームを個別のタスクとしてレンダリングします。

```
steps:
- name: RenderBlender
  parameterSpace:
    taskParameterDefinitions:
    - name: Frame
      type: INT
      range: "{{Param.Frames}}"
  script:
    actions:
      onRun:
        command: bash
        # Note: {{Task.File.Run}} is a variable that expands to the filename on the
 worker host's
        # disk where the contents of the 'Run' embedded file, below, is written.
        args: ['{{Task.File.Run}}']
    embeddedFiles:
      - name: Run
        type: TEXT
        data: |
          # Configure the task to fail if any individual command fails.
          set -xeuo pipefail

          mkdir -p '{{Param.OutputDir}}'

          blender --background '{{Param.BlenderSceneFile}}' \
                  --render-output '{{Param.OutputDir}}/{{Param.OutputPattern}}' \
                  --render-format {{Param.Format}} \
                  --use-extension 1 \
                  --render-frame {{Task.Param.Frame}}
```

たとえば、 Framesパラメータの値が の場合1-10、10 個のタスクを定義します。タスクごとにFrameパラメータの値が異なります。タスクを実行するには：

1. 埋め込みファイルの dataプロパティのすべての変数参照が展開されます。たとえば、 です--render-frame 1。
2. data プロパティの内容は、ディスク上のセッション作業ディレクトリのファイルに書き込まれます。

3. タスクの onRun コマンドは に解決されbash *location of embedded file*、実行されます。

埋め込みファイル、セッション、パスマップされた場所の詳細については、「Open Job Description」仕様の「How jobs are run」を参照してください。

deadlinedeadline-cloud-samples/job_bundles リポジトリには、ジョブテンプレートの例と、Open Job Descriptions 仕様で提供されているテンプレートサンプルがあります。

ジョブバンドルのパラメータ値要素

パラメータファイルを使用して、ジョブテンプレートまたはジョブバンドルの CreateJob オペレーションリクエスト引数の一部のジョブパラメータの値を設定できるため、ジョブの送信時に値を設定する必要はありません。ジョブ送信用の UI を使用すると、これらの値を変更できます。

ジョブテンプレートは、YAML 形式 (`parameter_values.yaml`) または JSON 形式 () で定義できますparameter_values.json。このセクションの例は YAML 形式で示されています。

YAML では、ファイルの形式は次のとおりです。

```
parameterValues:
- name: <string>
  value: <integer>, <float>, or <string>
- name: <string>
  value: <integer>, <float>, or <string>ab
... repeating as necessary
```

`parameterValues` リストの各要素は、次のいずれかである必要があります。

- ジョブテンプレートで定義されたジョブパラメータ。
- ジョブを送信するキューのキュー環境で定義されたジョブパラメータ。
- ジョブの作成時に `CreateJob`オペレーションに渡される特別なパラメータ。
  - `deadline:priority` – 値は整数である必要があります。優先度パラメータとして `CreateJob`オペレーションに渡されます。
  - `deadline:targetTaskRunStatus` – 値は文字列である必要があります。これは targetTaskRunStatus パラメータとして `CreateJob`オペレーションに渡されます。
  - `deadline:maxFailedTasksCount` – 値は整数である必要があります。これは、maxFailedTasksCount パラメータとして `CreateJob`オペレーションに渡されます。

- `deadline:maxRetriesPerTask` – 値は整数である必要があります。maxRetriesPerTask パラメータとして `CreateJob`オペレーションに渡されます。
- `deadline:maxWorkercount` – 値は整数である必要があります。mazWorkerCount パラメータとして `CreateJob`オペレーションに渡されます。

ジョブテンプレートは常に、実行する特定のジョブではなくテンプレートです。パラメータ値ファイルを使用すると、一部のパラメータにこのファイルで定義された値がない場合はテンプレートとして、すべてのパラメータに値がある場合は特定のジョブ送信として、ジョブバンドルが機能します。

たとえば、Blender_render サンプルにはパラメータファイルがなく、そのジョブテンプレートはデフォルト値なしでパラメータを定義します。このテンプレートは、ジョブを作成するためのテンプレートとして使用する必要があります。このジョブバンドルを使用してジョブを作成すると、Deadline Cloud は新しいジョブバンドルをジョブ履歴ディレクトリに書き込みます。

たとえば、次のコマンドを使用してジョブを送信する場合です。

```
deadline bundle gui-submit blender_render/
```

新しいジョブバンドルには、指定されたパラメータを含む `parameter_values.yaml` ファイルが含まれています。

```
% cat ~/.deadline/job_history/\(default\)/2024-06/2024-06-20-01-JobBundle-Demo/
parameter_values.yaml
parameterValues:
- name: deadline:targetTaskRunStatus
  value: READY
- name: deadline:maxFailedTasksCount
  value: 10
- name: deadline:maxRetriesPerTask
  value: 5
- name: deadline:priority
  value: 75
- name: BlenderSceneFile
  value: /private/tmp/bundle_demo/bmw27_cpu.blend
- name: Frames
  value: 1-10
- name: OutputDir
  value: /private/tmp/bundle_demo/output
- name: OutputPattern
  value: output_####
```

```
- name: Format
  value: PNG
- name: CondaPackages
  value: blender
- name: RezPackages
  value: blender
```

次のコマンドを使用して、同じジョブを作成できます。

```
deadline bundle submit ~/.deadline/job_history/\(default\)/2024-06/2024-06-20-01-
JobBundle-Demo/
```

> **Note**
>
> 送信したジョブバンドルは、ジョブ履歴ディレクトリに保存されます。次のコマンドを使用して、そのディレクトリの場所を見つけることができます。
>
> ```
> deadline config get settings.job_history_dir
> ```

ジョブバンドルのアセット参照要素

Deadline Cloud ジョブアタッチメントを使用して、ワークステーションと Deadline Cloud 間でファイルを送受信できます。アセットリファレンスファイルには、添付ファイルの入力ファイルとディレクトリ、および出力ディレクトリが一覧表示されます。このファイル内のすべてのファイルとディレクトリを一覧表示しない場合は、 `deadline bundle gui-submit` コマンドを使用してジョブを送信するときにそれらを選択できます。

ジョブアタッチメントを使用していない場合、このファイルは効果がありません。

ジョブテンプレートは、YAML 形式 (`asset_references.yaml`) または JSON 形式 () で定義できます`asset_references.json`。このセクションの例は YAML 形式で示されています。

YAML では、ファイルの形式は次のとおりです。

```
assetReferences:
    inputs:
        # Filenames on the submitting workstation whose file contents are needed as
        # inputs to run the job.
```

```
        filenames:
        - list of file paths
        # Directories on the submitting workstation whose contents are needed as inputs
        # to run the job.
        directories:
        - list of directory paths

    outputs:
        # Directories on the submitting workstation where the job writes output files
        # if running locally.
        directories:
        - list of directory paths

    # Paths referenced by the job, but not necessarily input or output.
    # Use this if your job uses the name of a path in some way, but does not explicitly
 need
    # the contents of that path.
    referencedPaths:
    - list of directory paths
```

Amazon S3 にアップロードする入力ファイルまたは出力ファイルを選択すると、Deadline Cloud はファイルパスをストレージプロファイルにリストされているパスと比較します。ストレージプロファイルの各 SHAREDタイプのファイルシステムの場所は、ワークステーションとワーカーホストにマウントされたネットワークファイル共有を抽象化します。Deadline Cloud は、これらのファイル共有の 1 つにないファイルのみをアップロードします。

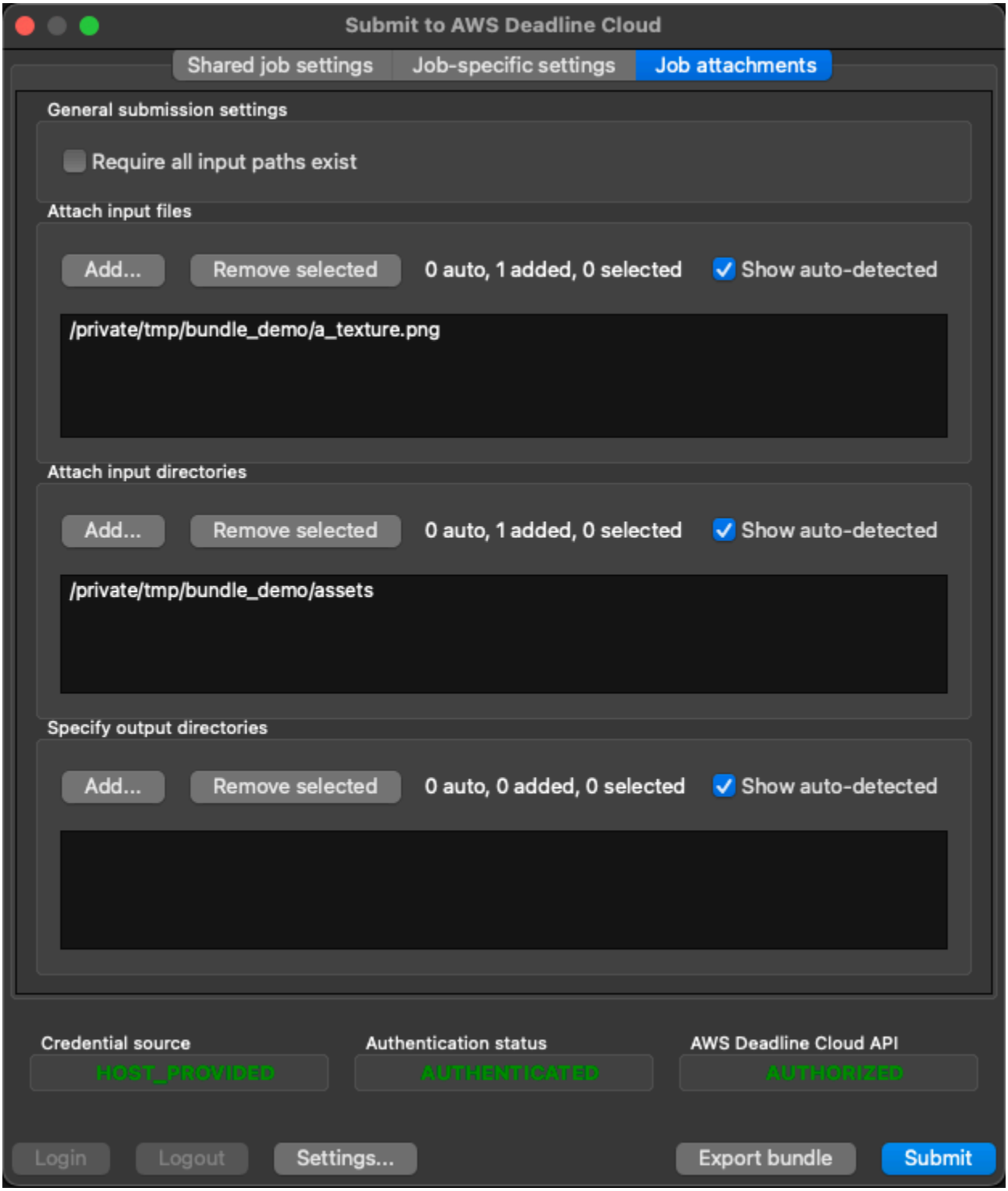
ストレージプロファイルの作成と使用の詳細については、「Deadline Cloud ユーザーガイド」の「Deadline Cloud の共有ストレージ」を参照してください。 AWS

Example - Deadline Cloud GUI によって作成されたアセットリファレンスファイル

次のコマンドを使用して、Blender_render サンプルを使用してジョブを送信します。

```
deadline bundle gui-submit blender_render/
```

ジョブアタッチメントタブのジョブにいくつかのファイルを追加します。



ジョブを送信すると、ジョブ履歴ディレクトリのジョブバンドル内の `asset_references.yaml` ファイルを参照して、YAML ファイル内のアセットを確認できます。

```
% cat ~/.deadline/job_history/\(default\)/2024-06/2024-06-20-01-JobBundle-Demo/
asset_references.yaml
```

```
assetReferences:
  inputs:
    filenames:
    - /private/tmp/bundle_demo/a_texture.png
    directories:
    - /private/tmp/bundle_demo/assets
  outputs:
    directories: []
  referencedPaths: []
```

ジョブでのファイルの使用

AWS Deadline Cloud に送信するジョブの多くは、入力ファイルと出力ファイルがあります。入力ファイルと出力ディレクトリは、共有ファイルシステムとローカルドライブの組み合わせにある場合があります。ジョブは、それらの場所にあるコンテンツを見つける必要があります。Deadline Cloud には、ジョブのアタッチメントとストレージプロファイルの 2 つの機能があり、ジョブが連携して必要なファイルを見つけるのに役立ちます。

ジョブアタッチメントにはいくつかの利点があります

- Amazon S3 を使用してホスト間でファイルを移動する
- ワークステーションからワーカーホストにファイルを転送する、またはその逆
- この機能を有効にするキュー内のジョブで使用可能
- 主にサービスマネージドフリートで使用されますが、カスタマーマネージドフリートとも互換性があります。

ストレージプロファイルを使用して、ワークステーションとワーカーホスト上の共有ファイルシステムの場所のレイアウトをマッピングします。これにより、 ベースのワークステーションと Windows ベースのワーカーホストを使用したクロスプラットフォーム設定など、ワークステーションとワーカーホスト間で場所が異なる場合に、ジョブが共有ファイルやディレクトリを見つけることができますLinux。ファイルシステム設定のストレージプロファイルのマップは、Amazon S3 を介してホスト間で転送する必要があるファイルを識別するために、ジョブアタッチメントでも使用されます。

ジョブアタッチメントを使用しておらず、ワークステーションとワーカーホスト間でファイルとディレクトリの場所を再マッピングする必要がない場合は、ストレージプロファイルを使用してファイル共有をモデル化する必要はありません。

トピック

- サンプルプロジェクトインフラストラクチャ
- ストレージプロファイルとパスマッピング

サンプルプロジェクトインフラストラクチャ

ジョブアタッチメントとストレージプロファイルの使用をデモンストレーションするには、2 つの異なるプロジェクトでテスト環境を設定します。Deadline Cloud コンソールを使用して、テストリソースを作成できます。

1. まだ作成していない場合は、テストファームを作成します。ファームを作成するには、「ファームの作成」の手順に従います。
2. 2 つのプロジェクトのそれぞれで、ジョブ用に 2 つのキューを作成します。キューを作成するには、「キューの作成」の手順に従います。

   a. という名前の最初のキューを作成します**Q1**。次の設定を使用して、他のすべての項目にデフォルトを使用します。

      - ジョブアタッチメントで、新しい Amazon S3 バケットを作成するを選択します。
      - カスタマーマネージドフリートとの関連付けを有効にするを選択します。
      - ユーザーとして実行する場合は、POSIX ユーザーとグループ**jobuser**の両方に を入力します。
      - キューサービスロールに、 という名前の新しいロールを作成します。 **AssetDemoFarm-Q1-Role**
      - デフォルトの Conda キュー環境チェックボックスをオフにします。

   b. という名前の 2 番目のキューを作成します**Q2**。次の設定を使用して、他のすべての項目にデフォルトを使用します。

      - ジョブアタッチメントで、新しい Amazon S3 バケットを作成するを選択します。
      - カスタマーマネージドフリートとの関連付けを有効にするを選択します。
      - ユーザーとして実行する場合は、POSIX ユーザーとグループ**jobuser**の両方に を入力します。
      - キューサービスロールには、 という名前の新しいロールを作成します。 **AssetDemoFarm-Q2-Role**
      - デフォルトの Conda キュー環境チェックボックスをオフにします。

3. 両方のキューからジョブを実行する単一のカスタマーマネージドフリートを作成します。フリートを作成するには、「カスタマーマネージドフリートを作成する」の手順に従います。次の設定を使用します。
    - 名前には、 を使用します**DemoFleet**。
    - フリートタイプ カスタマーマネージド を選択する
    - フリートサービスロールには、AssetDemoFarm-Fleet-Role という名前の新しいロールを作成します。
    - フリートをキューに関連付けないでください。

テスト環境では、ネットワークファイル共有を使用してホスト間で共有されているファイルシステムが 3 つあることを前提としています。この例では、ロケーションの名前は次のとおりです。

- FSCommon - 両方のプロジェクトに共通の入力ジョブアセットが含まれます。
- FS1 - プロジェクト 1 の入力ジョブアセットと出力ジョブアセットが含まれます。
- FS2 - プロジェクト 2 の入力ジョブアセットと出力ジョブアセットが含まれます。

テスト環境では、次のように 3 つのワークステーションがあることも前提としています。

- WSAll - 開発者がすべてのプロジェクトで使用する Linuxベースのワークステーション。共有ファイルシステムの場所は次のとおりです。
    - FSCommon: /shared/common
    - FS1: /shared/projects/project1
    - FS2: /shared/projects/project2
- WS1 - プロジェクト 1 に使用される Windowsベースのワークステーション。共有ファイルシステムの場所は次のとおりです。
    - FSCommon: S:\
    - FS1: Z:\
    - FS2: 利用できません
- WS1 - プロジェクト 2 に使用される macOSベースのワークステーション。共有ファイルシステムの場所は次のとおりです。
    - FSCommon: /Volumes/common
    - FS1: 利用できません

- FS2: /Volumes/projects/project2

最後に、フリート内のワーカーの共有ファイルシステムの場所を定義します。以下の例では、この設定を と呼んでいますWorkerConfig。共有場所は次のとおりです。

- FSCommon: /mnt/common
- FS1: /mnt/projects/project1
- FS2: /mnt/projects/project2

この設定に一致する共有ファイルシステム、ワークステーション、またはワーカーをセットアップする必要はありません。デモンストレーションのために共有場所が存在する必要はありません。

ストレージプロファイルとパスマッピング

ストレージプロファイルを使用して、ワークステーションとワーカーホストのファイルシステムをモデル化します。各ストレージプロファイルは、いずれかのシステム設定のオペレーティングシステムとファイルシステムレイアウトを記述します。このトピックでは、Deadline Cloud がジョブのパスマッピングルールを生成できるように、ストレージプロファイルを使用してホストのファイルシステム設定をモデル化する方法と、それらのパスマッピングルールをストレージプロファイルから生成する方法について説明します。

Deadline Cloud にジョブを送信するときに、ジョブのオプションのストレージプロファイル ID を指定できます。このストレージプロファイルは、送信するワークステーションのファイルシステムを記述します。ジョブテンプレートのファイルパスが使用する元のファイルシステム設定について説明します。

ストレージプロファイルをフリートに関連付けることもできます。ストレージプロファイルは、フリート内のすべてのワーカーホストのファイルシステム設定を記述します。ファイルシステム設定が異なるワーカーがある場合は、それらのワーカーをファーム内の別のフリートに割り当てる必要があります。

パスマッピングルールは、ジョブで指定されたパスからワーカーホスト上のパスの実際の場所にパスを再マッピングする方法を説明します。Deadline Cloud は、ジョブのストレージプロファイルで説明されているファイルシステム設定と、ジョブを実行しているフリートのストレージプロファイルを比較して、これらのパスマッピングルールを取得します。

トピック

- ストレージプロファイルを使用して共有ファイルシステムの場所をモデル化する
- フリートのストレージプロファイルを設定する
- キューのストレージプロファイルを設定する
- ストレージプロファイルからパスマッピングルールを取得する

ストレージプロファイルを使用して共有ファイルシステムの場所をモデル化する

ストレージプロファイルは、ホスト設定のいずれかのファイルシステム設定をモデル化します。サンプルプロジェクトインフラストラクチャには 4 つの異なるホスト設定があります。この例では、それぞれに個別のストレージプロファイルを作成します。ストレージプロファイルは、次のいずれかを使用して作成できます。

- CreateStorageProfile API
- AWS::Deadline::StorageProfile AWS CloudFormation リソース
- AWS コンソール

ストレージプロファイルは、ファイルシステムの場所のリストで構成され、それぞれがホストから送信または実行されるジョブに関連するファイルシステムの場所とタイプを Deadline Cloud に伝えます。ストレージプロファイルは、ジョブに関連する場所のみをモデル化する必要があります。たとえば、共有FSCommon場所は WS1 のワークステーションにあるためS:\、対応するファイルシステムの場所は次のとおりです。

```
{
    "name": "FSCommon",
    "path": "S:\\",
    "type": "SHARED"
}
```

次のコマンドを使用して、ワークステーション設定 WS1、WS2、WS3および AWS CLIの WorkerConfigを使用してワーカー設定のストレージプロファイルを作成しますAWS CloudShell。

```
# Change the value of FARM_ID to your farm's identifier
FARM_ID=farm-00112233445566778899aabbccddeeff

aws deadline create-storage-profile --farm-id $FARM_ID \
  --display-name WSAll \
  --os-family LINUX \
```

```
  --file-system-locations \
  '[
      {"name": "FSCommon", "type":"SHARED", "path":"/shared/common"},
      {"name": "FS1", "type":"SHARED", "path":"/shared/projects/project1"},
      {"name": "FS2", "type":"SHARED", "path":"/shared/projects/project2"}
  ]'

aws deadline create-storage-profile --farm-id $FARM_ID \
  --display-name WS1 \
  --os-family WINDOWS \
  --file-system-locations \
  '[
      {"name": "FSCommon", "type":"SHARED", "path":"S:\\"},
      {"name": "FS1", "type":"SHARED", "path":"Z:\\"}
   ]'

aws deadline create-storage-profile --farm-id $FARM_ID \
  --display-name WS2 \
  --os-family MACOS \
  --file-system-locations \
  '[
      {"name": "FSCommon", "type":"SHARED", "path":"/Volumes/common"},
      {"name": "FS2", "type":"SHARED", "path":"/Volumes/projects/project2"}
  ]'

aws deadline create-storage-profile --farm-id $FARM_ID \
  --display-name WorkerCfg \
  --os-family LINUX \
  --file-system-locations \
  '[
      {"name": "FSCommon", "type":"SHARED", "path":"/mnt/common"},
      {"name": "FS1", "type":"SHARED", "path":"/mnt/projects/project1"},
      {"name": "FS2", "type":"SHARED", "path":"/mnt/projects/project2"}
  ]'
```

> **Note**
>
> ファーム内のすべてのストレージプロファイルで nameプロパティに同じ値を使用して、ストレージプロファイル内のファイルシステムの場所を参照する必要があります。Deadline Cloud は名前を比較して、パスマッピングルールを生成するときに、異なるストレージプロファイルのファイルシステムの場所が同じ場所を参照しているかどうかを確認します。

フリートのストレージプロファイルを設定する

フリート内のすべてのワーカーのファイルシステムの場所をモデル化するストレージプロファイルを含めるようにフリートを設定できます。フリート内のすべてのワーカーのホストファイルシステム設定は、フリートのストレージプロファイルと一致する必要があります。ファイルシステム設定が異なるワーカーは、別々のフリートに存在する必要があります。

WorkerConfig ストレージプロファイルを使用するようにフリートの設定を設定するには、 AWS CLI で を使用しますAWS CloudShell。

```
# Change the value of FARM_ID to your farm's identifier
FARM_ID=farm-00112233445566778899aabbccddeeff
# Change the value of FLEET_ID to your fleet's identifier
FLEET_ID=fleet-00112233445566778899aabbccddeeff
# Change the value of WORKER_CFG_ID to your storage profile named WorkerConfig
WORKER_CFG_ID=sp-00112233445566778899aabbccddeeff

FLEET_WORKER_MODE=$( \
  aws deadline get-fleet --farm-id $FARM_ID --fleet-id $FLEET_ID \
   --query '.configuration.customerManaged.mode' \
)
FLEET_WORKER_CAPABILITIES=$( \
  aws deadline get-fleet --farm-id $FARM_ID --fleet-id $FLEET_ID \
   --query '.configuration.customerManaged.workerCapabilities' \
)

aws deadline update-fleet --farm-id $FARM_ID --fleet-id $FLEET_ID \
  --configuration \
  "{
    \"customerManaged\": {
      \"storageProfileId\": \"$WORKER_CFG_ID\",
      \"mode\": $FLEET_WORKER_MODE,
      \"workerCapabilities\": $FLEET_WORKER_CAPABILITIES
    }
  }"
```

キューのストレージプロファイルを設定する

キューの設定には、キューに送信されたジョブがアクセスする必要がある共有ファイルシステムの場所の大文字と小文字を区別する名前のリストが含まれます。たとえば、キューに送信されたジョブにはファイルシステムの場所FSCommonと Q1が必要ですFS1。キューに送信されるジョブには、ファイルシステムの場所 FSCommon と Q2が必要ですFS2。

これらのファイルシステムの場所を要求するようにキューの設定を設定するには、次のスクリプトを使用します。

```
# Change the value of FARM_ID to your farm's identifier
FARM_ID=farm-00112233445566778899aabbccddeeff
# Change the value of QUEUE1_ID to queue Q1's identifier
QUEUE1_ID=queue-00112233445566778899aabbccddeeff
# Change the value of QUEUE2_ID to queue Q2's identifier
QUEUE2_ID=queue-00112233445566778899aabbccddeeff

aws deadline update-queue --farm-id $FARM_ID --queue-id $QUEUE1_ID \
  --required-file-system-location-names-to-add FSComm FS1

aws deadline update-queue --farm-id $FARM_ID --queue-id $QUEUE2_ID \
  --required-file-system-location-names-to-add FSComm FS2
```

キューの設定には、そのキューに送信されたジョブと、そのキューに関連付けられたフリートに適用される許可されたストレージプロファイルのリストも含まれます。キューに必要なすべてのファイルシステムの場所のファイルシステムの場所を定義するストレージプロファイルのみが、キューの許可されたストレージプロファイルのリストで許可されます。

キューの許可されたストレージプロファイルのリストにないストレージプロファイルでジョブを送信すると、ジョブは失敗します。ストレージプロファイルのないジョブは、いつでもキューに送信できます。とラベル付けされたワークステーション設定にはWSAll、キュー に必要なファイルシステムの場所 (FSCommon と FS1) WS1がありますQ1。キューへのジョブの送信を許可する必要があります。同様に、ワークステーション設定 WSAllと WS2はキュー の要件を満たしていますQ2。ジョブをそのキューに送信することを許可する必要があります。両方のキュー設定を更新して、次のスクリプトを使用して、これらのストレージプロファイルでジョブを送信できるようにします。

```
# Change the value of WSALL_ID to the identifier of the WSAll storage profile
WSALL_ID=sp-00112233445566778899aabbccddeeff
# Change the value of WS1 to the identifier of the WS1 storage profile
WS1_ID=sp-00112233445566778899aabbccddeeff
# Change the value of WS2 to the identifier of the WS2 storage profile
WS2_ID=sp-00112233445566778899aabbccddeeff

aws deadline update-queue --farm-id $FARM_ID --queue-id $QUEUE1_ID \
  --allowed-storage-profile-ids-to-add $WSALL_ID $WS1_ID

aws deadline update-queue --farm-id $FARM_ID --queue-id $QUEUE2_ID \
```

```
  --allowed-storage-profile-ids-to-add $WSALL_ID $WS2_ID
```

WS2 ストレージプロファイルをキューの許可されたストレージプロファイルのリストに追加するQ1と、失敗します。

```
$ aws deadline update-queue --farm-id $FARM_ID --queue-id $QUEUE1_ID \
  --allowed-storage-profile-ids-to-add $WS2_ID

An error occurred (ValidationException) when calling the UpdateQueue operation: Storage
 profile id: sp-00112233445566778899aabbccddeeff does not have required file system
 location: FS1
```

これは、WS2ストレージプロファイルに、キューFS1がQ1必要とする という名前のファイルシステムの場所の定義が含まれていないためです。

キューの許可されたストレージプロファイルのリストに含まれていないストレージプロファイルに設定されたフリートの関連付けも失敗します。例:

```
$ aws deadline create-queue-fleet-association --farm-id $FARM_ID \
   --fleet-id $FLEET_ID \
   --queue-id $QUEUE1_ID

An error occurred (ValidationException) when calling the CreateQueueFleetAssociation
 operation: Mismatch between storage profile ids.
```

エラーを修正するには、 という名前のストレージプロファイルWorkerConfigをキューQ1とキューの両方の許可されたストレージプロファイルのリストに追加しますQ2。次に、フリートのワーカーが両方のキューからジョブを実行できるように、フリートをこれらのキューに関連付けます。

```
# Change the value of FLEET_ID to your fleet's identifier
FLEET_ID=fleet-00112233445566778899aabbccddeeff
# Change the value of WORKER_CFG_ID to your storage profile named WorkerCfg
WORKER_CFG_ID=sp-00112233445566778899aabbccddeeff

aws deadline update-queue --farm-id $FARM_ID --queue-id $QUEUE1_ID \
  --allowed-storage-profile-ids-to-add $WORKER_CFG_ID

aws deadline update-queue --farm-id $FARM_ID --queue-id $QUEUE2_ID \
  --allowed-storage-profile-ids-to-add $WORKER_CFG_ID

aws deadline create-queue-fleet-association --farm-id $FARM_ID \
```

```
  --fleet-id $FLEET_ID \
  --queue-id $QUEUE1_ID

aws deadline create-queue-fleet-association --farm-id $FARM_ID \
  --fleet-id $FLEET_ID \
  --queue-id $QUEUE2_ID
```

ストレージプロファイルからパスマッピングルールを取得する

パスマッピングルールは、ジョブからワーカーホスト上のパスの実際の場所にパスを再マッピングする方法を説明します。タスクがワーカーで実行されている場合、ジョブのストレージプロファイルがワーカーのフリートのストレージプロファイルと比較され、タスクのパスマッピングルールが取得されます。

Deadline Cloud は、キューの設定に必要なファイルシステムの場所ごとにマッピングルールを作成します。たとえば、`WSAll`ストレージプロファイルを使用してキューに送信されたジョブ`Q1`には、パスマッピングルールがあります。

- `FSComm`: `/shared/common -> /mnt/common`
- `FS1`: `/shared/projects/project1 -> /mnt/projects/project1`

Deadline Cloud は、 `FSComm`と `FS1` ファイルシステムの場所のルールを作成しますが、 `WSAll`と の両方の`WorkerConfig`ストレージプロファイルで が定義されている場合でも、`FS2`ファイルシステムの場所は作成しません`FS2`。これは、キュー`Q1`の必要なファイルシステムの場所のリストが であるためです`["FSComm", "FS1"]`。

オープンジョブの説明のパスマッピングルールファイルを出力するジョブを送信し、ジョブの完了後にセッションログを読み取ることで、特定のストレージプロファイルで送信されたジョブで使用できるパスマッピングルールを確認できます。

```
# Change the value of FARM_ID to your farm's identifier
FARM_ID=farm-00112233445566778899aabbccddeeff
# Change the value of QUEUE1_ID to queue Q1's identifier
QUEUE1_ID=queue-00112233445566778899aabbccddeeff
# Change the value of WSALL_ID to the identifier of the WSALL storage profile
WSALL_ID=sp-00112233445566778899aabbccddeeff

aws deadline create-job --farm-id $FARM_ID --queue-id $QUEUE1_ID \
  --priority 50 \\
  --storage-profile-id $WSALL_ID \
```

```
  --template-type JSON --template \
  '{
    "specificationVersion": "jobtemplate-2023-09",
    "name": "DemoPathMapping",
    "steps": [
      {
        "name": "ShowPathMappingRules",
        "script": {
          "actions": {
            "onRun": {
              "command": "/bin/cat",
              "args": [ "{{Session.PathMappingRulesFile}}" ]
            }
          }
        }
      }
    ]
  }'
```

Deadline Cloud CLI を使用してジョブを送信する場合、そのsettings.storage_profile_id設定により、CLI で送信されたジョブが持つストレージプロファイルが設定されます。WSAll ストレージプロファイルを使用してジョブを送信するには、以下を設定します。

```
deadline config set settings.storage_profile_id $WSALL_ID
```

サンプルインフラストラクチャで実行されているかのようにカスタマーマネージドワーカーを実行するには、Deadline Cloud ユーザーガイドの「ワーカーエージェントの実行」の手順に従ってワーカーを実行します AWS CloudShell。前述の手順に従っている場合は、まず ~/demoenv-logsおよび ~/demoenv-persist ディレクトリを削除します。また、指示が参照する DEV_FARM_IDおよび DEV_CMF_ID 環境変数の値を次のように設定してから、設定します。

```
DEV_FARM_ID=$FARM_ID
DEV_CMF_ID=$FLEET_ID
```

ジョブの実行後、ジョブのログファイルにパスマッピングルールが表示されます。

```
cat demoenv-logs/${QUEUE1_ID}/*.log
...
JJSON log results (see below)
...
```

ログには、 FS1と FSComm ファイルシステムの両方のマッピングが含まれています。読みやすくするために再フォーマットされたログエントリは次のようになります。

```
{
    "version": "pathmapping-1.0",
    "path_mapping_rules": [
        {
            "source_path_format": "POSIX",
            "source_path": "/shared/projects/project1",
            "destination_path": "/mnt/projects/project1"
        },
        {
            "source_path_format": "POSIX",
            "source_path": "/shared/common",
            "destination_path": "/mnt/common"
        }
    ]
```

異なるストレージプロファイルを持つジョブを送信して、パスマッピングルールがどのように変化するかを確認できます。

ジョブアタッチメントを使用してファイルを共有する

ジョブアタッチメントを使用して、共有ディレクトリにないファイルをジョブで使用できるようにし、共有ディレクトリに書き込まれない場合は出力ファイルをキャプチャします。ジョブアタッチメントはAmazon S3 を使用してホスト間でファイルをバッファリングします。ファイルは S3 バケットに保存されるため、コンテンツが変更されていない場合はファイルをアップロードする必要はありません。

ホストはファイルシステムの場所を共有しないため、サービスマネージドフリートでジョブを実行するときは、ジョブアタッチメントを使用する必要があります。ジョブアタッチメントは、ジョブバンドルにシェルや Python スクリプトが含まれている場合など、共有ネットワークファイルシステムに保存されているジョブの入力ファイルまたは出力ファイルがある場合にも、カスタマーマネージドフリートで役立ちます。

Deadline Cloud CLI または Deadline Cloud 送信者のいずれかを使用してジョブバンドルを送信すると、ジョブアタッチメントはジョブのストレージプロファイルとキューの必要なファイルシステムの場所を使用して、ワーカーホストになく、ジョブ送信の一部として Amazon S3 にアップロードする必要がある入力ファイルを識別します。これらのストレージプロファイルは、Deadline Cloud が

ワークステーションで使用できるように Amazon S3 にアップロードする必要があるワーカーホストロケーション内の出力ファイルを識別するのにも役立ちます。

ジョブアタッチメントの例では、 および のファーム、フリート、キュー、ストレージプロファイル設定を使用しますサンプルプロジェクトインフラストラクチャストレージプロファイルとパスマッピング。この前に、これらのセクションを確認してください。

次の例では、サンプルジョブバンドルを開始点として使用し、ジョブアタッチメントの機能を調べるために変更します。ジョブバンドルは、ジョブがジョブアタッチメントを使用する最適な方法です。ディレクトリ内の Open Job Description ジョブテンプレートと、ジョブバンドルを使用するジョブに必要なファイルとディレクトリを一覧表示する追加ファイルを組み合わせて使用します。ジョブバンドルの詳細については、「」を参照してくださいDeadline Cloud の Open Job Description (OpenJD) テンプレート。

ジョブを使用したファイルの送信

Deadline Cloud を使用すると、ジョブワークフローを有効にして、ワーカーホストの共有ファイルシステムロケーションで使用できない入力ファイルにアクセスできます。ジョブアタッチメントを使用すると、レンダリングジョブはローカルワークステーションドライブまたはサービスマネージドフリート環境のみにあるファイルにアクセスできます。ジョブバンドルを送信するときに、ジョブに必要な入力ファイルとディレクトリのリストを含めることができます。Deadline Cloud は、これらの共有されていないファイルを識別し、ローカルマシンから Amazon S3 にアップロードして、ワーカーホストにダウンロードします。これにより、入力アセットをレンダリングノードに転送するプロセスが合理化され、分散ジョブの実行に必要なすべてのファイルにアクセスできるようになります。

ジョブバンドルでジョブのファイルを直接指定し、環境変数またはスクリプトを使用して指定したジョブテンプレートでパラメータを使用し、ジョブの `assets_references` ファイルを使用できます。これらのメソッドのいずれか、または 3 つすべての組み合わせを使用できます。ジョブのバンドルのストレージプロファイルを指定して、ローカルワークステーションで変更されたファイルのみをアップロードできます。

このセクションでは、GitHub のジョブバンドルの例を使用して、Deadline Cloud がアップロードするジョブ内のファイルを識別する方法、それらのファイルを Amazon S3 で整理する方法、およびジョブを処理するワーカーホストがそれらをどのように利用できるかを示します。

トピック

- Deadline Cloud が Amazon S3 にファイルをアップロードする方法
- Deadline Cloud がアップロードするファイルを選択する方法
- ジョブがジョブアタッチメント入力ファイルを検索する方法

Deadline Cloud が Amazon S3 にファイルをアップロードする方法

この例では、Deadline Cloud がワークステーションまたはワーカーホストから Amazon S3 にファイルをアップロードして共有できるようにする方法を示します。GitHub と Deadline Cloud CLI のサンプルジョブバンドルを使用してジョブを送信します。

まず、Deadline Cloud サンプル GitHub リポジトリをAWS CloudShell環境にクローンし、`job_attachments_devguide`ジョブバンドルをホームディレクトリにコピーします。

```
git clone https://github.com/aws-deadline/deadline-cloud-samples.git
cp -r deadline-cloud-samples/job_bundles/job_attachments_devguide ~/
```

Deadline Cloud CLI をインストールしてジョブバンドルを送信します。

```
pip install deadline --upgrade
```

`job_attachments_devguide` ジョブバンドルには、ファイルシステムの場所がジョブパラメータとして渡される bash シェルスクリプトを実行するタスクを含む 1 つのステップがあります。ジョブパラメータの定義は次のとおりです。

```
...
- name: ScriptFile
  type: PATH
  default: script.sh
  dataFlow: IN
  objectType: FILE
...
```

`dataFlow` プロパティの`IN`値は、 `ScriptFile`パラメータの値がジョブへの入力であることをジョブアタッチメントに伝えます。`default` プロパティの値は、ジョブバンドルの ディレクトリに対する相対的な場所ですが、絶対パスにすることもできます。このパラメータ定義は、ジョブバンドルの ディレクトリにある `script.sh` ファイルを、ジョブの実行に必要な入力ファイルとして宣言します。

次に、Deadline Cloud CLI にストレージプロファイルが設定されていないことを確認し、ジョブをキュー に送信します`Q1`。

```
# Change the value of FARM_ID to your farm's identifier
FARM_ID=farm-00112233445566778899aabbccddeeff
# Change the value of QUEUE1_ID to queue Q1's identifier
```

```
QUEUE1_ID=queue-00112233445566778899aabbccddeeff

deadline config set settings.storage_profile_id ''

deadline bundle submit --farm-id $FARM_ID --queue-id $QUEUE1_ID
 job_attachments_devguide/
```

このコマンドの実行後の Deadline Cloud CLI からの出力は次のようになります。

```
Submitting to Queue: Q1
...
Hashing Attachments  [####################################]  100%
Hashing Summary:
    Processed 1 file totaling 39.0 B.
    Skipped re-processing 0 files totaling 0.0 B.
    Total processing time of 0.0327 seconds at 1.19 KB/s.

Uploading Attachments  [####################################]  100%
Upload Summary:
    Processed 1 file totaling 39.0 B.
    Skipped re-processing 0 files totaling 0.0 B.
    Total processing time of 0.25639 seconds at 152.0 B/s.

Waiting for Job to be created...
Submitted job bundle:
   job_attachments_devguide/
Job creation completed successfully
job-74148c13342e4514b63c7a7518657005
```

ジョブを送信すると、Deadline Cloud はまず`script.sh`ファイルをハッシュし、Amazon S3 にアップロードします。

Deadline Cloud は、S3 バケットをコンテンツアドレス可能なストレージとして扱います。ファイルは S3 オブジェクトにアップロードされます。オブジェクト名は、ファイルの内容のハッシュから派生します。2 つのファイルに同じコンテンツがある場合、ファイルの場所や名前に関係なく、同じハッシュ値を持ちます。これにより、Deadline Cloud は、ファイルがすでに利用可能な場合、ファイルのアップロードを回避できます。

AWS CLI を使用して、Amazon S3 にアップロードされたオブジェクトを表示できます。

```
# The name of queue `Q1`'s job attachments S3 bucket
Q1_S3_BUCKET=$(
```

```
  aws deadline get-queue --farm-id $FARM_ID --queue-id $QUEUE1_ID \
    --query 'jobAttachmentSettings.s3BucketName' | tr -d '"'
)

aws s3 ls s3://$Q1_S3_BUCKET --recursive
```

2 つのオブジェクトが S3 にアップロードされました。

- `DeadlineCloud/Data/87cb19095dd5d78fcaf56384ef0e6241.xxh128` – の内容`script.sh`。オブジェクトキー`87cb19095dd5d78fcaf56384ef0e6241`の値はファイルの内容のハッシュであり、拡張子はハッシュ値が 128 ビット xx ハッシュとして計算された`xxh128`ことを示します。
- `DeadlineCloud/Manifests/<farm-id>/<queue-id>/Inputs/<guid>/a1d221c7fd97b08175b3872a37428e8c_input` – ジョブ送信のマニフェストオブジェクト。値`<farm-id>`、`<queue-id>`、 `<guid>`は、ファーム識別子、キュー識別子、およびランダムな 16 進値です。この例`a1d221c7fd97b08175b3872a37428e8c`の値は、 が配置されているディレクトリである文字列 から計算`/home/cloudshell-user/job_attachments_devguide`されたハッシュ値`script.sh`です。

マニフェストオブジェクトには、ジョブの送信の一部として S3 にアップロードされた特定のルートパスの入力ファイルに関する情報が含まれています。このマニフェストファイル () をダウンロードします`aws s3 cp s3://$Q1_S3_BUCKET/<objectname>`。その内容は次のようになります。

```
{
    "hashAlg": "xxh128",
    "manifestVersion": "2023-03-03",
    "paths": [
        {
            "hash": "87cb19095dd5d78fcaf56384ef0e6241",
            "mtime": 1721147454416085,
            "path": "script.sh",
            "size": 39
        }
    ],
    "totalSize": 39
}
```

これは、ファイルがアップロード`script.sh`され、そのファイルのコンテンツのハッシュが であることを示します`87cb19095dd5d78fcaf56384ef0e6241`。

このハッシュ値は、オブジェクト名 の値と一致しますDeadlineCloud/Data/87cb19095dd5d78fcaf56384ef0e6241.xxh128。Deadline Cloud は、このファイルの内容に対してどのオブジェクトをダウンロードするかを知るために使用されます。

このファイルの完全なスキーマは GitHub で入手できます。

CreateJob オペレーションを使用すると、マニフェストオブジェクトの場所を設定できます。GetJob オペレーションを使用して、場所を確認できます。

```
{
    "attachments": {
        "file system": "COPIED",
        "manifests": [
            {
                "inputManifestHash": "5b0db3d311805ea8de7787b64cbbe8b3",
                "inputManifestPath": "<farm-id>/<queue-id>/Inputs/<guid>/
a1d221c7fd97b08175b3872a37428e8c_input",
                "rootPath": "/home/cloudshell-user/job_attachments_devguide",
                "rootPathFormat": "posix"
            }
        ]
    },
    ...
}
```

Deadline Cloud がアップロードするファイルを選択する方法

ジョブアタッチメントが Amazon S3 へのアップロードをジョブへの入力と見なすファイルとディレクトリは次のとおりです。

- IN または の値を持つジョブバンドルのジョブテンプレートで定義されたすべての PATHタイプのジョブパラメータdataFlowの値INOUT。
- ジョブバンドルのアセットリファレンスファイルの入力としてリストされているファイルとディレクトリ。

ストレージプロファイルのないジョブを送信すると、アップロードの対象となるすべてのファイルがアップロードされます。ストレージプロファイルを使用してジョブを送信する場合、ファイルは、キューに必要なファイルシステムの場所であるストレージプロファイルの SHAREDタイプのファイルシステムの場所にある場合、Amazon S3 にアップロードされません。これらの場所は、ジョブを実

行するワーカーホストで使用できることが予想されるため、S3 にアップロードする必要はありません。

この例では、AWS CloudShell 環境で WSAllにSHAREDファイルシステムの場所を作成し、それらのファイルシステムの場所にファイルを追加します。以下のコマンドを使用します。

```
# Change the value of WSALL_ID to the identifier of the WSAll storage profile
WSALL_ID=sp-00112233445566778899aabbccddeeff

sudo mkdir -p /shared/common /shared/projects/project1 /shared/projects/project2
sudo chown -R cloudshell-user:cloudshell-user /shared

for d in /shared/common /shared/projects/project1 /shared/projects/project2; do
  echo "File contents for $d" > ${d}/file.txt
done
```

次に、ジョブの入力として作成したすべてのファイルを含むアセット参照ファイルをジョブバンドルに追加します。以下のコマンドを使用します。

```
cat > ${HOME}/job_attachments_devguide/asset_references.yaml << EOF
assetReferences:
  inputs:
    filenames:
    - /shared/common/file.txt
    directories:
    - /shared/projects/project1
    - /shared/projects/project2
EOF
```

次に、WSAllストレージプロファイルを使用してジョブを送信するように Deadline Cloud CLI を設定し、ジョブバンドルを送信します。

```
# Change the value of FARM_ID to your farm's identifier
FARM_ID=farm-00112233445566778899aabbccddeeff
# Change the value of QUEUE1_ID to queue Q1's identifier
QUEUE1_ID=queue-00112233445566778899aabbccddeeff
# Change the value of WSALL_ID to the identifier of the WSAll storage profile
WSALL_ID=sp-00112233445566778899aabbccddeeff

deadline config set settings.storage_profile_id $WSALL_ID
```

```
deadline bundle submit --farm-id $FARM_ID --queue-id $QUEUE1_ID
 job_attachments_devguide/
```

Deadline Cloud は、ジョブの送信時に 2 つのファイルを Amazon S3 にアップロードします。ジョブのマニフェストオブジェクトを S3 からダウンロードして、アップロードされたファイルを表示できます。

```
for manifest in $( \
  aws deadline get-job --farm-id $FARM_ID --queue-id $QUEUE1_ID --job-id $JOB_ID \
    --query 'attachments.manifests[].inputManifestPath' \
    | jq -r '.[]'
); do
  echo "Manifest object: $manifest"
  aws s3 cp --quiet s3://$Q1_S3_BUCKET/DeadlineCloud/Manifests/$manifest /dev/stdout |
 jq .
done
```

この例では、次の内容のマニフェストファイルが 1 つあります。

```
{
    "hashAlg": "xxh128",
    "manifestVersion": "2023-03-03",
    "paths": [
        {
            "hash": "87cb19095dd5d78fcaf56384ef0e6241",
            "mtime": 1721147454416085,
            "path": "home/cloudshell-user/job_attachments_devguide/script.sh",
            "size": 39
        },
        {
            "hash": "af5a605a3a4e86ce7be7ac5237b51b79",
            "mtime": 1721163773582362,
            "path": "shared/projects/project2/file.txt",
            "size": 44
        }
    ],
    "totalSize": 83
}
```

マニフェストの GetJob オペレーションを使用して、 `rootPath`が「/」であることを確認します。

```
aws deadline get-job --farm-id $FARM_ID --queue-id $QUEUE1_ID --job-id $JOB_ID --query
 'attachments.manifests[*]'
```

入力ファイルのセットのルートパスは、常にそれらのファイルの最長の共通サブパスです。Windows ジョブが から送信され、異なるドライブにあったために共通のサブパスを持たない入力ファイルがある場合、各ドライブに個別のルートパスが表示されます。マニフェスト内のパスは常にマニフェストのルートパスを基準にしているため、アップロードされた入力ファイルは次のとおりです。

- /home/cloudshell-user/job_attachments_devguide/script.sh – ジョブバンドル内のスクリプトファイル。
- /shared/projects/project2/file.txt – キュー に必要なSHAREDファイルシステムの場所のリストに含まれていないWSAllストレージプロファイル内のファイルシステムの場所にあるファイルQ1。

ファイルシステムの場所 FSCommon (/shared/common/file.txt) と FS1 (/shared/projects/project1/file.txt) のファイルはリストにありません。これは、これらのファイルシステムの場所がWSAllストレージプロファイルSHAREDにあり、どちらもキュー の必要なファイルシステムの場所のリストにあるためですQ1。

GetStorageProfileForQueue オペレーションを使用して、特定のストレージプロファイルで送信されたジョブSHAREDで考慮されるファイルシステムの場所を確認できます。キューWSAllのストレージプロファイルをクエリするには、次のコマンドQ1を使用します。

```
aws deadline get-storage-profile --farm-id $FARM_ID --storage-profile-id $WSALL_ID

aws deadline get-storage-profile-for-queue --farm-id $FARM_ID --queue-id $QUEUE1_ID --
storage-profile-id $WSALL_ID
```

ジョブがジョブアタッチメント入力ファイルを検索する方法

ジョブの添付ファイルを使用して Deadline Cloud が Amazon S3 にアップロードするファイルを使用するには、ジョブでワーカーホストのファイルシステムを介して利用可能なファイルが必要です。ジョブのセッションがワーカーホストで実行されると、Deadline Cloud はジョブの入力ファイルをワーカーホストのローカルドライブの一時ディレクトリにダウンロードし、ジョブの各ルートパスのパスマッピングルールをローカルドライブのファイルシステムの場所に追加します。

この例では、AWS CloudShell タブで Deadline Cloud ワーカーエージェントを起動します。以前に送信したジョブの実行を終了し、ログディレクトリからジョブログを削除します。

```
rm -rf ~/devdemo-logs/queue-*
```

次のスクリプトは、セッションの一時作業ディレクトリ内のすべてのファイルとパスマッピングルールファイルの内容を表示するようにジョブバンドルを変更し、変更されたバンドルでジョブを送信します。

```
# Change the value of FARM_ID to your farm's identifier
FARM_ID=farm-00112233445566778899aabbccddeeff
# Change the value of QUEUE1_ID to queue Q1's identifier
QUEUE1_ID=queue-00112233445566778899aabbccddeeff
# Change the value of WSALL_ID to the identifier of the WSAll storage profile
WSALL_ID=sp-00112233445566778899aabbccddeeff

deadline config set settings.storage_profile_id $WSALL_ID

cat > ~/job_attachments_devguide/script.sh << EOF
#!/bin/bash

echo "Session working directory is: \$(pwd)"
echo
echo "Contents:"
find . -type f
echo
echo "Path mapping rules file: \$1"
jq . \$1
EOF

cat > ~/job_attachments_devguide/template.yaml << EOF
specificationVersion: jobtemplate-2023-09
name: "Job Attachments Explorer"
parameterDefinitions:
- name: ScriptFile
  type: PATH
  default: script.sh
  dataFlow: IN
  objectType: FILE
steps:
- name: Step
  script:
    actions:
```

```
      onRun:
        command: /bin/bash
        args:
        - "{{Param.ScriptFile}}"
        - "{{Session.PathMappingRulesFile}}"
EOF

deadline bundle submit --farm-id $FARM_ID --queue-id $QUEUE1_ID
 job_attachments_devguide/
```

AWS CloudShell 環境内のワーカーによって実行されたジョブの実行ログを確認できます。

```
cat demoenv-logs/queue-*/session*.log
```

ログは、セッションで最初に発生するのは、ジョブの 2 つの入力ファイルがワーカーにダウンロードされることを示しています。

```
2024-07-17 01:26:37,824 INFO ================================================
2024-07-17 01:26:37,825 INFO --------- Job Attachments Download for Job
2024-07-17 01:26:37,825 INFO ================================================
2024-07-17 01:26:37,825 INFO Syncing inputs using Job Attachments
2024-07-17 01:26:38,116 INFO Downloaded 142.0 B / 186.0 B of 2 files (Transfer rate:
 0.0 B/s)
2024-07-17 01:26:38,174 INFO Downloaded 186.0 B / 186.0 B of 2 files (Transfer rate:
 733.0 B/s)
2024-07-17 01:26:38,176 INFO Summary Statistics for file downloads:
Processed 2 files totaling 186.0 B.
Skipped re-processing 0 files totaling 0.0 B.
Total processing time of 0.09752 seconds at 1.91 KB/s.
```

次に、ジョブによって`script.sh`実行された からの出力を示します。

- ジョブの送信時にアップロードされた入力ファイルは、セッションの一時ディレクトリの「アサルート」で始まる名前のディレクトリにあります。
- 入力ファイルのパスは、ジョブの入力マニフェスト () のルートパスではなく、「アセルート」ディレクトリに対して再配置されました"/"。
- パスマッピングルールファイルには、「assetroot」ディレクトリの絶対パス"/"に再マッピングする追加のルールが含まれています。

例：

```
2024-07-17 01:26:38,264 INFO Output:
2024-07-17 01:26:38,267 INFO Session working directory is: /sessions/session-5b33f
2024-07-17 01:26:38,267 INFO
2024-07-17 01:26:38,267 INFO Contents:
2024-07-17 01:26:38,269 INFO ./tmp_xdhbsdo.sh
2024-07-17 01:26:38,269 INFO ./tmpdi00052b.json
2024-07-17 01:26:38,269 INFO ./assetroot-assetroot-3751a/shared/projects/project2/
file.txt
2024-07-17 01:26:38,269 INFO ./assetroot-assetroot-3751a/home/cloudshell-user/
job_attachments_devguide/script.sh
2024-07-17 01:26:38,269 INFO
2024-07-17 01:26:38,270 INFO Path mapping rules file: /sessions/session-5b33f/
tmpdi00052b.json
2024-07-17 01:26:38,282 INFO {
2024-07-17 01:26:38,282 INFO   "version": "pathmapping-1.0",
2024-07-17 01:26:38,282 INFO   "path_mapping_rules": [
2024-07-17 01:26:38,282 INFO     {
2024-07-17 01:26:38,282 INFO       "source_path_format": "POSIX",
2024-07-17 01:26:38,282 INFO       "source_path": "/shared/projects/project1",
2024-07-17 01:26:38,283 INFO       "destination_path": "/mnt/projects/project1"
2024-07-17 01:26:38,283 INFO     },
2024-07-17 01:26:38,283 INFO     {
2024-07-17 01:26:38,283 INFO       "source_path_format": "POSIX",
2024-07-17 01:26:38,283 INFO       "source_path": "/shared/common",
2024-07-17 01:26:38,283 INFO       "destination_path": "/mnt/common"
2024-07-17 01:26:38,283 INFO     },
2024-07-17 01:26:38,283 INFO     {
2024-07-17 01:26:38,283 INFO       "source_path_format": "POSIX",
2024-07-17 01:26:38,283 INFO       "source_path": "/",
2024-07-17 01:26:38,283 INFO       "destination_path": "/sessions/session-5b33f/
assetroot-assetroot-3751a"
2024-07-17 01:26:38,283 INFO     }
2024-07-17 01:26:38,283 INFO   ]
2024-07-17 01:26:38,283 INFO }
```

> **Note**
>
> 送信するジョブに異なるルートパスを持つ複数のマニフェストがある場合、ルートパスごとに異なる「assetroot」という名前のディレクトリがあります。

入力ファイル、ディレクトリ、またはファイルシステムの場所のいずれかの再配置されたファイルシステムの場所を参照する必要がある場合は、ジョブのパスマッピングルールファイルを処理して再マッピングを自分で実行するか、ジョブバンドルのジョブテンプレートにPATHタイプジョブパラメータを追加して、そのパラメータの値として再マッピングする必要がある値を渡すことができます。たとえば、次の例では、これらのジョブパラメータのいずれかを持つようにジョブバンドルを変更し、ファイルシステムの場所を値/shared/projects/project2としてジョブを送信します。

```
cat > ~/job_attachments_devguide/template.yaml << EOF
specificationVersion: jobtemplate-2023-09
name: "Job Attachments Explorer"
parameterDefinitions:
- name: LocationToRemap
  type: PATH
steps:
- name: Step
  script:
    actions:
      onRun:
        command: /bin/echo
        args:
        - "The location of {{RawParam.LocationToRemap}} in the session is
 {{Param.LocationToRemap}}"
EOF

deadline bundle submit --farm-id $FARM_ID --queue-id $QUEUE1_ID
 job_attachments_devguide/ \
  -p LocationToRemap=/shared/projects/project2
```

このジョブの実行のログファイルには、出力が含まれています。

```
2024-07-17 01:40:35,283 INFO Output:
2024-07-17 01:40:35,284 INFO The location of /shared/projects/project2 in the session
 is /sessions/session-5b33f/assetroot-assetroot-3751a
```

ジョブからの出力ファイルの取得

この例では、Deadline Cloud がジョブが生成する出力ファイルを識別する方法、それらのファイルを Amazon S3 にアップロードするかどうかを決定する方法、およびワークステーションでそれらの出力ファイルを取得する方法を示します。

この例では、job_attachments_devguide_outputジョブバンドルの代わりにjob_attachments_devguideジョブバンドルを使用します。まず、Deadline Cloud サンプル GitHub リポジトリのクローンから AWS CloudShell 環境内のバンドルのコピーを作成します。

```
cp -r deadline-cloud-samples/job_bundles/job_attachments_devguide_output ~/
```

このジョブバンドルとjob_attachments_devguideジョブバンドルの重要な違いは、ジョブテンプレートに新しいジョブパラメータを追加することです。

```
...
parameterDefinitions:
...
- name: OutputDir
  type: PATH
  objectType: DIRECTORY
  dataFlow: OUT
  default: ./output_dir
  description: This directory contains the output for all steps.
...
```

パラメータの dataFlowプロパティには値 がありますOUT。Deadline Cloud は、dataFlowジョブの出力INOUTとして OUTまたは の値を持つジョブパラメータの値を使用します。これらの種類のジョブパラメータに値として渡されたファイルシステムの場所が、ジョブを実行するワーカーのローカルファイルシステムの場所に再マッピングされた場合、Deadline Cloud は、その場所で新しいファイルを検索し、ジョブ出力として Amazon S3 にアップロードします。

これがどのように機能するかを確認するには、まず AWS CloudShell タブで Deadline Cloud ワーカーエージェントを起動します。以前に送信したジョブの実行を終了します。次に、ログディレクトリからジョブログを削除します。

```
rm -rf ~/devdemo-logs/queue-*
```

次に、このジョブバンドルを使用してジョブを送信します。CloudShell で実行されているワーカーが実行されたら、ログを確認します。

```
# Change the value of FARM_ID to your farm's identifier
FARM_ID=farm-00112233445566778899aabbccddeeff
# Change the value of QUEUE1_ID to queue Q1's identifier
QUEUE1_ID=queue-00112233445566778899aabbccddeeff
```

```
# Change the value of WSALL_ID to the identifier of the WSAll storage profile
WSALL_ID=sp-00112233445566778899aabbccddeeff

deadline config set settings.storage_profile_id $WSALL_ID

deadline bundle submit --farm-id $FARM_ID --queue-id $QUEUE1_ID ./
job_attachments_devguide_output
```

ログは、ファイルが出力として検出され、Amazon S3 にアップロードされたことを示しています。

```
2024-07-17 02:13:10,873 INFO ----------------------------------------------
2024-07-17 02:13:10,873 INFO Uploading output files to Job Attachments
2024-07-17 02:13:10,873 INFO ----------------------------------------------
2024-07-17 02:13:10,873 INFO Started syncing outputs using Job Attachments
2024-07-17 02:13:10,955 INFO Found 1 file totaling 117.0 B in output directory: /
sessions/session-7efa/assetroot-assetroot-3751a/output_dir
2024-07-17 02:13:10,956 INFO Uploading output manifest to
 DeadlineCloud/Manifests/farm-0011/queue-2233/job-4455/step-6677/
task-6677-0/2024-07-17T02:13:10.835545Z_sessionaction-8899-1/
c6808439dfc59f86763aff5b07b9a76c_output
2024-07-17 02:13:10,988 INFO Uploading 1 output file to S3: s3BucketName/DeadlineCloud/
Data
2024-07-17 02:13:11,011 INFO Uploaded 117.0 B / 117.0 B of 1 file (Transfer rate: 0.0
 B/s)
2024-07-17 02:13:11,011 INFO Summary Statistics for file uploads:
Processed 1 file totaling 117.0 B.
Skipped re-processing 0 files totaling 0.0 B.
Total processing time of 0.02281 seconds at 5.13 KB/s.
```

ログには、Deadline Cloud がキュー のジョブアタッチメントで使用するように設定された Amazon S3 バケットに新しいマニフェストオブジェクトを作成したことも示されていますQ1。マニフェストオブジェクトの名前は、出力を生成したタスクのファーム、キュー、ジョブ、ステップ、タスク、タイムスタンプ、`sessionaction`識別子から派生します。このマニフェストファイルをダウンロードして、Deadline Cloud がこのタスクの出力ファイルをどこに配置したかを確認します。

```
# The name of queue `Q1`'s job attachments S3 bucket
Q1_S3_BUCKET=$(
  aws deadline get-queue --farm-id $FARM_ID --queue-id $QUEUE1_ID \
    --query 'jobAttachmentSettings.s3BucketName' | tr -d '"'
)

# Fill this in with the object name from your log
```

```
OBJECT_KEY="DeadlineCloud/Manifests/..."

aws s3 cp --quiet s3://$Q1_S3_BUCKET/$OBJECT_KEY /dev/stdout | jq .
```

マニフェストは次のようになります。

```
{
  "hashAlg": "xxh128",
  "manifestVersion": "2023-03-03",
  "paths": [
    {
      "hash": "34178940e1ef9956db8ea7f7c97ed842",
      "mtime": 1721182390859777,
      "path": "output_dir/output.txt",
      "size": 117
    }
  ],
  "totalSize": 117
}
```

これは、出力ファイルのコンテンツが、ジョブ入力ファイルを保存するのと同じ方法で Amazon S3 に保存されることを示しています。入力ファイルと同様に、出力ファイルは ファイルのハッシュと プレフィックス を含むオブジェクト名で S3 に保存されますDeadlineCloud/Data。

```
$ aws s3 ls --recursive s3://$Q1_S3_BUCKET | grep 34178940e1ef9956db8ea7f7c97ed842
2024-07-17 02:13:11        117 DeadlineCloud/
Data/34178940e1ef9956db8ea7f7c97ed842.xxh128
```

Deadline Cloud モニターまたは Deadline Cloud CLI を使用して、ジョブの出力をワークステーションにダウンロードできます。

```
deadline job download-output --farm-id $FARM_ID --queue-id $QUEUE1_ID --job-id $JOB_ID
```

送信されたOutputDirジョブのジョブパラメータの値は であるため./output_dir、出力はジョブバンドルディレクトリoutput_dir内の というディレクトリにダウンロードされます。の値として絶対パスまたは異なる相対位置を指定した場合OutputDir、代わりに出力ファイルがその場所にダウンロードされます。

```
$ deadline job download-output --farm-id $FARM_ID --queue-id $QUEUE1_ID --job-id
 $JOB_ID
Downloading output from Job 'Job Attachments Explorer: Output'

Summary of files to download:
    /home/cloudshell-user/job_attachments_devguide_output/output_dir/output.txt (1
 file)

You are about to download files which may come from multiple root directories. Here are
 a list of the current root directories:
[0] /home/cloudshell-user/job_attachments_devguide_output
> Please enter the index of root directory to edit, y to proceed without changes, or n
 to cancel the download (0, y, n) [y]:

Downloading Outputs  [####################################]  100%
Download Summary:
    Downloaded 1 files totaling 117.0 B.
    Total download time of 0.14189 seconds at 824.0 B/s.
    Download locations (total file counts):
        /home/cloudshell-user/job_attachments_devguide_output (1 file)
```

依存ステップのステップからのファイルの使用

この例では、ジョブの 1 つのステップが同じジョブで依存するステップからの出力にアクセスする方法を示します。

あるステップの出力を別のステップで使用できるようにするために、Deadline Cloud はセッションにアクションを追加して、セッションでタスクを実行する前にそれらの出力をダウンロードします。出力を使用する必要があるステップの依存関係としてこれらのステップを宣言することで、出力をからダウンロードするステップを指示します。

この例では、 `job_attachments_devguide_output` ジョブバンドルを使用します。まず、Deadline Cloud サンプル GitHub リポジトリのクローンから AWS CloudShell 環境にコピーを作成します。既存のステップの後にのみ実行され、そのステップの出力を使用する依存ステップを追加するように変更します。

```
cp -r deadline-cloud-samples/job_bundles/job_attachments_devguide_output ~/

cat >> job_attachments_devguide_output/template.yaml << EOF
- name: DependentStep
  dependencies:
  - dependsOn: Step
```

```
  script:
    actions:
      onRun:
        command: /bin/cat
        args:
        - "{{Param.OutputDir}}/output.txt"
EOF
```

この変更されたジョブバンドルで作成されたジョブは、2 つの別々のセッションとして実行されます。1 つはステップ「Step」のタスク用で、もう 1 つはステップDependentStep」のタスク用です。

まず、CloudShell タブで Deadline Cloud ワーカーエージェントを起動します。以前に送信したジョブの実行を終了してから、ログディレクトリからジョブログを削除します。

```
rm -rf ~/devdemo-logs/queue-*
```

次に、変更されたジョブバンドルを使用してjob_attachments_devguide_outputジョブを送信します。CloudShell 環境のワーカーでの実行が完了するまで待ちます。2 つのセッションのログを確認します。

```
# Change the value of FARM_ID to your farm's identifier
FARM_ID=farm-00112233445566778899aabbccddeeff
# Change the value of QUEUE1_ID to queue Q1's identifier
QUEUE1_ID=queue-00112233445566778899aabbccddeeff
# Change the value of WSALL_ID to the identifier of the WSAll storage profile
WSALL_ID=sp-00112233445566778899aabbccddeeff

deadline config set settings.storage_profile_id $WSALL_ID

deadline bundle submit --farm-id $FARM_ID --queue-id $QUEUE1_ID ./
job_attachments_devguide_output

# Wait for the job to finish running, and then:

cat demoenv-logs/queue-*/session-*
```

というステップのタスクのセッションログにはDependentStep、2 つの異なるダウンロードアクションが実行されます。

```
2024-07-17 02:52:05,666 INFO ================================================
```

```
2024-07-17 02:52:05,666 INFO --------- Job Attachments Download for Job
2024-07-17 02:52:05,667 INFO ================================================
2024-07-17 02:52:05,667 INFO Syncing inputs using Job Attachments
2024-07-17 02:52:05,928 INFO Downloaded 207.0 B / 207.0 B of 1 file (Transfer rate: 0.0
 B/s)
2024-07-17 02:52:05,929 INFO Summary Statistics for file downloads:
Processed 1 file totaling 207.0 B.
Skipped re-processing 0 files totaling 0.0 B.
Total processing time of 0.03954 seconds at 5.23 KB/s.

2024-07-17 02:52:05,979 INFO
2024-07-17 02:52:05,979 INFO ================================================
2024-07-17 02:52:05,979 INFO --------- Job Attachments Download for Step
2024-07-17 02:52:05,979 INFO ================================================
2024-07-17 02:52:05,980 INFO Syncing inputs using Job Attachments
2024-07-17 02:52:06,133 INFO Downloaded 117.0 B / 117.0 B of 1 file (Transfer rate: 0.0
 B/s)
2024-07-17 02:52:06,134 INFO Summary Statistics for file downloads:
Processed 1 file totaling 117.0 B.
Skipped re-processing 0 files totaling 0.0 B.
Total processing time of 0.03227 seconds at 3.62 KB/s.
```

最初のアクションは、「Step」という名前のステップで使用される`script.sh`ファイルをダウンロードします。2 番目のアクションは、そのステップからの出力をダウンロードします。Deadline Cloud は、そのステップで生成された出力マニフェストを入力マニフェストとして使用して、ダウンロードするファイルを決定します。

同じログに遅れると、「DependentStep」という名前のステップからの出力が表示されます。

```
2024-07-17 02:52:06,213 INFO Output:
2024-07-17 02:52:06,216 INFO Script location: /sessions/session-5b33f/
assetroot-assetroot-3751a/script.sh
```

ジョブのリソース制限を作成する

Deadline Cloud に送信されるジョブは、複数のジョブ間で共有されるリソースに依存する場合があります。たとえば、ファームには、特定のリソースのフローティングライセンスよりも多くのワーカーが存在する場合があります。または、共有ファイルサーバーは、限られた数のワーカーにのみ同時にデータを供給できます。場合によっては、1 つ以上のジョブがこれらのリソースをすべて要求し、新しいワーカーの起動時にリソースが使用できないために エラーが発生することがあります。

これを解決するために、これらの制約されたリソースの制限を使用できます。Deadline Cloud は、制約されたリソースの可用性を考慮し、その情報を使用して、新しいワーカーの起動時にリソースが利用可能になるようにします。これにより、リソースが利用できないためにジョブが失敗する可能性が低くなります。

制限はファーム全体に作成されます。キューに送信されたジョブは、キューに関連付けられた制限のみを取得できます。キューに関連付けられていないジョブの制限を指定すると、ジョブは互換性がなく、実行されません。

制限を使用するには、

- 制限を作成する
- 制限とキューを関連付ける
- 制限が必要なジョブを送信する

> **Note**
>
> 制限に関連付けられていないキューに制約されたリソースがあるジョブを実行すると、そのジョブはすべてのリソースを消費できます。制約されたリソースがある場合は、リソースを使用するキュー内のジョブのすべてのステップが制限に関連付けられていることを確認してください。

ファームで定義され、キューに関連付けられ、ジョブで指定されている制限の場合、次の 4 つのことのいずれかが発生する可能性があります。

- 制限を作成してキューに関連付け、ジョブのテンプレートで制限を指定すると、ジョブが実行され、制限で定義されたリソースのみが使用されます。
- 制限を作成してジョブテンプレートで指定し、制限をキューに関連付けない場合、ジョブは互換性がないとマークされ、実行されません。
- 制限を作成し、キューに関連付けず、ジョブのテンプレートで制限を指定しない場合、ジョブは実行されますが、制限は使用されません。
- 制限をまったく使用しない場合、ジョブが実行されます。

制限を複数のキューに関連付けると、キューは制限によって制約されるリソースを共有します。たとえば、100 の制限を作成し、1 つのキューが 60 個のリソースを使用している場合、他のキューは 40

個のリソースしか使用できません。リソースが解放されると、任意のキューからタスクがリソースを取得できます。

Deadline Cloud には、制限によって提供されるリソースのモニタリングに役立つ 2 つの AWS CloudFormation メトリクスが用意されています。使用中のリソースの現在の数と、制限内で使用可能なリソースの最大数をモニタリングできます。詳細については、「Deadline Cloud Developer Guide」の「Resource limit metrics」を参照してください。

ジョブテンプレートのジョブステップに制限を適用します。ステップの `amounts`セクションで制限の量要件名を指定し、同じ を持つ制限`amountRequirementName`がジョブ`hostRequirements`のキューに関連付けられている場合、このステップでスケジュールされるタスクはリソースの制限によって制約されます。

ステップで制限に達したリソースが必要な場合、そのステップのタスクは追加のワーカーによって取得されません。

ジョブステップには複数の制限を適用できます。たとえば、ステップで 2 つの異なるソフトウェアライセンスを使用している場合、ライセンスごとに個別の制限を適用できます。ステップに 2 つの制限が必要で、いずれかのリソースの制限に達した場合、そのステップのタスクは、リソースが利用可能になるまで追加のワーカーによって取得されません。

制限の停止と削除

キューと制限間の関連付けを停止または削除すると、制限を使用するジョブは、この制限を必要とするステップのタスクのスケジュールを停止し、ステップの新しいセッションの作成をブロックします。

READY 状態のタスクは準備完了のままで、キューと制限との関連付けによってタスクが自動的に再開され、再びアクティブになります。ジョブを再キューに入れる必要はありません。

キューと制限間の関連付けを停止または削除する場合、実行中のタスクを停止する方法には 2 つの選択肢があります。

- タスクの停止とキャンセル – 制限を取得したセッションを持つワーカーは、すべてのタスクをキャンセルします。
- 実行中のタスクを停止して終了する – 制限を取得したセッションを持つワーカーはタスクを完了します。

コンソールを使用して制限を削除すると、ワーカーはまずタスクの実行をすぐに停止するか、タスクの完了時に最終的に停止します。関連付けが削除されると、次のようになります。

- 制限が必要なステップは互換性がないとマークされます。
- 制限を必要としないステップを含め、これらのステップを含むジョブ全体がキャンセルされます。
- ジョブは互換性がないとマークされています。

制限に関連付けられたキューに、制限の量要件名と一致するフリート機能を持つフリートが関連付けられている場合、そのフリートは指定された制限でジョブを処理し続けます。

制限を作成する

Deadline Cloud コンソールまたは Deadline Cloud API の CreateLimit オペレーションを使用して制限を作成します。制限はファームに対して定義されますが、キューに関連付けられます。制限を作成したら、1 つ以上のキューに関連付けることができます。

制限を作成するには

1. Deadline Cloud コンソール (https://console.aws.amazon.com/deadlinecloud/home) ダッシュボードから、キューを作成するファームを選択します。
2. 制限を追加するファームを選択し、制限タブを選択し、制限の作成を選択します。
3. 制限の詳細を入力します。金額要件名は、制限を識別するためにジョブテンプレートで使用される名前です。プレフィックスで始まり、その後に金額名が**amount.**続く必要があります。金額要件名は、制限に関連付けられたキュー内で一意である必要があります。
4. 最大量を設定する を選択した場合、これはこの制限で許可されるリソースの合計数です。最大量なしを選択した場合、リソース使用量は制限されません。リソースの使用が制限されていない場合でも、`CurrentCount`Amazon CloudWatch メトリクスが出力されるため、使用状況を追跡できます。詳細については、「Deadline CloudWatch metrics」を参照してください。
5. 制限を使用するキューが既にわかっている場合は、今すぐ選択できます。制限を作成するためにキューを関連付ける必要はありません。
6. 制限の作成 を選択します。

制限とキューを関連付ける

制限を作成したら、1 つ以上のキューを制限に関連付けることができます。制限に関連付けられているキューのみが、制限で指定された値を使用します。

Deadline Cloud コンソールまたは Deadline Cloud API の CreateQueueLimitAssociation オペレーションを使用して、キューとの関連付けを作成します。

キューを制限に関連付けるには

1. Deadline Cloud コンソール (https://console.aws.amazon.com/deadlinecloud/home) ダッシュボードから、制限をキューに関連付けるファームを選択します。
2. 制限タブを選択し、キューを関連付ける制限を選択し、制限の編集を選択します。
3. キューの関連付けセクションで、制限に関連付けるキューを選択します。
4. [Save changes] (変更の保存) をクリックします。

制限が必要なジョブを送信する

制限を適用するには、ジョブまたはジョブステップのホスト要件として指定します。ステップで制限を指定せず、そのステップが関連付けられたリソースを使用する場合、ジョブがスケジュールされたとき、ステップの使用量は制限に対してカウントされません。

一部の Deadline Cloud 送信者では、ホスト要件を設定できます。送信者で制限金額要件名を指定して、制限を適用できます。

送信者がホスト要件の追加をサポートしていない場合は、ジョブのジョブテンプレートを編集して制限を適用することもできます。

ジョブバンドルのジョブステップに制限を適用するには

1. テキストエディタを使用して、ジョブのジョブテンプレートを開きます。ジョブテンプレートは、ジョブのジョブバンドルディレクトリにあります。詳細については、Deadline Cloud デベロッパーガイドの「ジョブバンドル」を参照してください。
2. 制限を適用するステップのステップ定義を見つけます。
3. ステップ定義に以下を追加します。*amount.name* を、制限の金額要件名に置き換えます。一般的な使用では、min値を 1 に設定する必要があります。

   YAML

   ```
   hostRequirements:
     amounts:
     - name: amount.name
       min: 1
   ```

JSON

```
"hostRequirements": {
    "amounts": [
        {
            "name": "amount.name",
            "min": "1"
        }
    }
}
```

次のように、ジョブステップに複数の制限を追加できます。*amount.name_1* と *amount.name_2* を、制限の金額要件名に置き換えます。

YAML

```
  hostRequirements:
    amounts:
    - name: amount.name_1
      min: 1
    - name: amount.name_2
      min: 1
```

JSON

```
"hostRequirements": {
    "amounts": [
        {
            "name": "amount.name_1",
            "min": "1"
        },
        {
            "name": "amount.name_2",
            "min": "1"
        }
    }
}
```

4. 変更をジョブテンプレートに保存します。

Deadline Cloud にジョブを送信する方法

AWS Deadline Cloud にジョブを送信するには、さまざまな方法があります。このセクションでは、Deadline Cloud が提供するツールを使用するか、ワークロード用に独自のカスタムツールを作成してジョブを送信する方法をいくつか説明します。

- ターミナルから – ジョブバンドルを初めて開発する場合、またはジョブを送信するユーザーがコマンドラインの使用に慣れている場合
- スクリプトから – ワークロードのカスタマイズと自動化
- アプリケーションから – ユーザーの作業がアプリケーションにある場合、またはアプリケーションのコンテキストが重要な場合。

次の例では、Python `deadline` ライブラリと`deadline`コマンドラインツールを使用します。どちらも PyPi から利用でき、GitHub でホストされています。

トピック

- ターミナルから Deadline Cloud にジョブを送信する
- スクリプトを使用して Deadline Cloud にジョブを送信する
- アプリケーション内でジョブを送信する

ターミナルから Deadline Cloud にジョブを送信する

ジョブバンドルと Deadline Cloud CLI のみを使用すると、ユーザーまたはより技術的なユーザーは、ジョブバンドルの書き込みをすばやく繰り返してジョブの送信をテストできます。ジョブバンドルを送信するには、次のコマンドを使用します。

```
deadline bundle submit <path-to-job-bundle>
```

バンドルにデフォルトがないパラメータを含むジョブバンドルを送信する場合は、/ `-p --parameter`オプションで指定できます。

```
deadline bundle submit <path-to-job-bundle> -p <parameter-name>=<parameter-value> -
p ...
```

使用可能なオプションの完全なリストについては、ヘルプコマンドを実行します。

```
deadline bundle submit --help
```

GUI を使用して Deadline Cloud にジョブを送信する

Deadline Cloud CLI には、ジョブを送信する前にユーザーが指定する必要があるパラメータを表示できるようにするグラフィカルユーザーインターフェイスも付属しています。ユーザーがコマンドラインを操作したくない場合は、特定のジョブバンドルを送信するダイアログを開くデスクトップショートカットを記述できます。

```
deadline bundle gui-submit <path-to-job-bundle>
```

`--browse` オプションを使用すると、ユーザーはジョブバンドルを選択できます。

```
deadline bundle gui-submit --browse
```

使用可能なオプションの完全なリストについては、ヘルプコマンドを実行します。

```
deadline bundle gui-submit --help
```

スクリプトを使用して Deadline Cloud にジョブを送信する

Deadline Cloud へのジョブの送信を自動化するには、bash、Powershell、バッチファイルなどのツールを使用してジョブをスクリプト化できます。

環境変数や他のアプリケーションからジョブパラメータを入力するなどの機能を追加できます。複数のジョブを連続して送信したり、送信するジョブバンドルの作成をスクリプト化したりすることもできます。

Python を使用してジョブを送信する

Deadline Cloud には、 サービスとやり取りするためのオープンソースの Python ライブラリもあります。ソースコードは GitHub で入手できます。

ライブラリは pip () 経由で pypi で使用できます`pip install deadline`。Deadline Cloud CLI ツールで使用されるのと同じライブラリです。

```
from deadline.client import api

job_bundle_path = "/path/to/job/bundle"
job_parameters = [
```

```
    {
        "name": "parameter_name",
        "value": "parameter_value"
    },
]

job_id = api.create_job_from_job_bundle(
    job_bundle_path,
    job_parameters
)
print(job_id)
```

deadline bundle gui-submit コマンドのようなダイアログを作成するには、 から show_job_bundle_submitter関数を使用できますdeadline.client.ui.job_bundle_submitter。

次の例では、Qt アプリケーションを起動し、ジョブバンドル送信者を示します。

```
# The GUI components must be installed with pip install "deadline[gui]"
import sys
from qtpy.QtWidgets import QApplication
from deadline.client.ui.job_bundle_submitter import show_job_bundle_submitter

app = QApplication(sys.argv)
submitter = show_job_bundle_submitter(browse=True)
submitter.show()
app.exec()
print(submitter.create_job_response)
```

独自のダイアログを作成するには、 で SubmitJobToDeadlineDialog クラスを使用できますdeadline.client.ui.dialogs.submit_job_to_deadline_dialog。値を渡す、独自のジョブ固有のタブを埋め込む、ジョブバンドルの作成 (または渡される) 方法を決定できます。

アプリケーション内でジョブを送信する

ユーザーがジョブを簡単に送信できるように、アプリケーションが提供するスクリプトランタイムまたはプラグインシステムを使用できます。ユーザーには使い慣れたインターフェイスがあり、ワークロードの送信時にユーザーを支援する強力なツールを作成できます。

アプリケーションにジョブバンドルを埋め込む

この例では、アプリケーションで利用できるジョブバンドルを送信する方法を示します。

これらのジョブバンドルへのアクセス権をユーザーに付与するには、Deadline Cloud CLI を起動するメニュー項目に埋め込まれたスクリプトを作成します。

次のスクリプトを使用すると、ユーザーはジョブバンドルを選択できます。

```
deadline bundle gui-submit --install-gui
```

代わりにメニュー項目で特定のジョブバンドルを使用するには、以下を使用します。

```
deadline bundle gui-submit </path/to/job/bundle> --install-gui
```

これにより、ユーザーがジョブパラメータ、入力、出力を変更し、ジョブを送信できるダイアログが開きます。ユーザーがアプリケーションで送信するジョブバンドルごとに異なるメニュー項目を設定できます。

ジョブバンドルで送信するジョブに、送信間で同様のパラメータとアセット参照が含まれている場合は、基盤となるジョブバンドルのデフォルト値を入力できます。

アプリケーションから情報を取得する

ユーザーが送信に手動で追加する必要がないようにアプリケーションから情報を取得するには、Deadline Cloud をアプリケーションと統合して、ユーザーがアプリケーションを終了したりコマンドラインツールを使用したりすることなく、使い慣れたインターフェイスを使用してジョブを送信できるようにします。

アプリケーションに Python と pyside/pyqt をサポートするスクリプトランタイムがある場合は、Deadline Cloud クライアントライブラリの GUI コンポーネントを使用して UI を作成できます。例については、GitHub の「Deadline Cloud for Maya integration」を参照してください。

Deadline Cloud クライアントライブラリには、強力な統合ユーザーエクスペリエンスを提供するために以下を実行するオペレーションが用意されています。

- プルキュー環境パラメータ、ジョブパラメータ、アセットリファレンスは、アプリケーション SDK を呼び出して環境変数を形成します。
- ジョブバンドルのパラメータを設定します。元のバンドルを変更しないようにするには、バンドルのコピーを作成し、コピーを送信する必要があります。

`deadline bundle gui-submit` コマンドを使用してジョブバンドルを送信する場合は、`parameter_values.yaml`および `asset_references.yaml` ファイルをプログラムで使用して、

アプリケーションから情報を渡す必要があります。これらのファイルの詳細については、「」を参照してください[Deadline Cloud の Open Job Description (OpenJD) テンプレート](#)。

OpenJD が提供するコントロールよりも複雑なコントロールが必要な場合、ユーザーからジョブを抽象化する必要がある場合、または統合をアプリケーションのビジュアルスタイルと一致させる場合は、Deadline Cloud クライアントライブラリを呼び出してジョブを送信する独自のダイアログを作成できます。

Deadline Cloud でジョブをスケジュールする

ジョブが作成されると、 AWS Deadline Cloud はキューに関連付けられた 1 つ以上のフリートで処理されるようにスケジュールします。特定のタスクを処理するフリートは、フリート用に設定された機能と特定のステップのホスト要件に基づいて選択されます。

キュー内のジョブは、ベストエフォートの優先順位で、最も高い順にスケジュールされます。2 つのジョブの優先度が同じ場合、最も古いジョブが最初にスケジュールされます。

以下のセクションでは、ジョブをスケジュールするプロセスの詳細について説明します。

フリートの互換性を確認する

ジョブが作成されると、Deadline Cloud はジョブの各ステップのホスト要件を、ジョブが送信されたキューに関連付けられたフリートの機能と照合します。フリートがホスト要件を満たしている場合、ジョブは `READY`状態になります。

ジョブ内のいずれかのステップに、キューに関連付けられたフリートが満たすことができない要件がある場合、ステップのステータスは に設定されます`NOT_COMPATIBLE`。さらに、ジョブの残りのステップはキャンセルされます。

フリートの機能はフリートレベルで設定されます。フリートのワーカーがジョブの要件を満たしていても、フリートがジョブの要件を満たしていない場合、ジョブからタスクは割り当てられません。

次のジョブテンプレートには、ステップのホスト要件を指定するステップがあります。

```
name: Sample Job With Host Requirements
specificationVersion: jobtemplate-2023-09
steps:
- name: Step 1
  script:
    actions:
```

```
      onRun:
        args:
        - '1'
        command: /usr/bin/sleep
  hostRequirements:
    amounts:
    # Capabilities starting with "amount." are amount capabilities. If they start with
 "amount.worker.",
    # they are defined by the OpenJD specification. Other names are free for custom
 usage.
    - name: amount.worker.vcpu
      min: 4
      max: 8
    attributes:
    - name: attr.worker.os.family
      anyOf:
      - linux
```

このジョブは、次の機能を備えたフリートにスケジュールできます。

```
{
    "vCpuCount": {"min": 4, "max": 8},
    "memoryMiB": {"min": 1024},
    "osFamily": "linux",
    "cpuArchitectureType": "x86_64"
}
```

このジョブは、次のいずれかの機能を持つフリートにスケジュールすることはできません。

```
{
    "vCpuCount": {"min": 4},
    "memoryMiB": {"min": 1024},
    "osFamily": "linux",
    "cpuArchitectureType": "x86_64"
}
    The vCpuCount has no maximum, so it exceeds the maximum vCPU host requirement.

{
    "vCpuCount": {"max": 8},
    "memoryMiB": {"min": 1024},
    "osFamily": "linux",
    "cpuArchitectureType": "x86_64"
}
```

```
    The vCpuCount has no minimum, so it doesn't satisfy the minimum vCPU host
 requirement.

{
    "vCpuCount": {"min": 4, "max": 8},
    "memoryMiB": {"min": 1024},
    "osFamily": "windows",
    "cpuArchitectureType": "x86_64"
}
    The osFamily doesn't match.
```

フリートスケーリング

ジョブが互換性のあるサービスマネージドフリートに割り当てられると、フリートは自動スケーリングされます。フリート内のワーカーの数は、フリートが実行できるタスクの数に基づいて変わります。

ジョブがカスタマーマネージドフリートに割り当てられると、ワーカーがすでに存在しているか、イベントベースの自動スケーリングを使用して作成できます。詳細については、Amazon EC2 Auto Scaling ユーザーガイド」のEventBridge を使用して自動スケーリングイベントを処理する」を参照してください。

セッション

ジョブのタスクは 1 つ以上のセッションに分割されます。ワーカーはセッションを実行して環境をセットアップし、タスクを実行し、環境を破棄します。各セッションは、ワーカーが実行する必要がある 1 つ以上のアクションで構成されます。

ワーカーがセクションアクションを完了すると、追加のセッションアクションをワーカーに送信できます。ワーカーは、セッション内の既存の環境とジョブアタッチメントを再利用して、タスクをより効率的に完了します。

ジョブアタッチメントは、Deadline Cloud CLI ジョブバンドルの一部として使用する送信者によって作成されます。`create-job` AWS CLI コマンドの `--attachments`オプションを使用してジョブアタッチメントを作成することもできます。環境は、特定のキューにアタッチされたキュー環境と、ジョブテンプレートで定義されたジョブ環境とステップ環境の 2 つの場所で定義されます。

セッションアクションには 4 つのタイプがあります。

- `syncInputJobAttachments` – 入力ジョブの添付ファイルをワーカーにダウンロードします。
- `envEnter` – 環境の`onEnter`アクションを実行します。

- taskRun – タスクのonRunアクションを実行します。
- envExit – 環境のonExitアクションを実行します。

次のジョブテンプレートにはステップ環境があります。ステップ環境を設定するonEnter定義、実行するタスクを定義するonRun定義、ステップ環境を破棄するonExit定義があります。このジョブ用に作成されたセッションには、 envEnterアクション、1 つ以上の taskRun アクション、envExitアクションが含まれます。

```
name: Sample Job with Maya Environment
specificationVersion: jobtemplate-2023-09
steps:
- name: Maya Step
  stepEnvironments:
  - name: Maya
    description: Runs Maya in the background.
    script:
      embeddedFiles:
      - name: initData
        filename: init-data.yaml
        type: TEXT
        data: |
          scene_file: MyAwesomeSceneFile
          renderer: arnold
          camera: persp
      actions:
        onEnter:
          command: MayaAdaptor
          args:
          - daemon
          - start
          - --init-data
          - file://{{Env.File.initData}}
        onExit:
          command: MayaAdaptor
          args:
          - daemon
          - stop
  parameterSpace:
    taskParameterDefinitions:
    - name: Frame
      range: 1-5
      type: INT
```

```
  script:
    embeddedFiles:
    - name: runData
      filename: run-data.yaml
      type: TEXT
      data: |
        frame: {{Task.Param.Frame}}
    actions:
      onRun:
        command: MayaAdaptor
        args:
        - daemon
        - run
        - --run-data
        - file://{{ Task.File.runData }}
```

セッションアクションのパイプライン化

セッションアクションのパイプラインにより、スケジューラは複数のセッションアクションをワーカーに事前割り当てできます。その後、ワーカーはこれらのアクションを順番に実行し、タスク間のアイドル時間を短縮または排除できます。

初期割り当てを作成するには、スケジューラが 1 つのタスクでセッションを作成し、ワーカーがタスクを完了してから、スケジューラがタスク期間を分析して今後の割り当てを決定します。

スケジューラを有効にするには、タスク期間ルールがあります。1 分未満のタスクの場合、スケジューラは Power-of-2 成長パターンを使用します。たとえば、1 秒のタスクの場合、スケジューラは 2 つの新しいタスクを割り当て、次に 4、次に 8 を割り当てます。1 分間のタスクの場合、スケジューラは新しいタスクを 1 つだけ割り当て、パイプライン作成は無効のままになります。

パイプラインサイズを計算するために、スケジューラは以下を実行します。

- 完了したタスクの平均タスク期間を使用します
- ワーカーを 1 分間ビジー状態に保つことを目指します。
- 同じセッション内のタスクのみを考慮します
- ワーカー間で期間データを共有しない

セッションアクションが枯渇すると、ワーカーは新しいタスクをすぐに開始し、スケジューラリクエスト間の待機時間はありません。また、長時間実行されるプロセスのワーカー効率が向上し、タスク分散が向上します。

さらに、新しい優先度の高いジョブが利用可能である場合、ワーカーは現在のセッションが終了し、優先度の高いジョブからの新しいセッションが割り当てられる前に、以前に割り当てられたすべての作業を完了します。

ステップの依存関係

Deadline Cloud はステップ間の依存関係の定義をサポートしているため、あるステップは別のステップが完了するまで待機してから開始します。ステップには複数の依存関係を定義できます。依存関係を持つステップは、すべての依存関係が完了するまでスケジュールされません。

ジョブテンプレートが循環依存関係を定義している場合、ジョブは拒否され、ジョブのステータスはに設定されますCREATE_FAILED。

次のジョブテンプレートは、2 つのステップでジョブを作成します。 StepBは に依存しStepAます。 が正常にStepA完了した後にStepBのみ実行されます。

ジョブが作成されると、 StepAは READY状態になり、 StepBは PENDING状態になります。StepAが終了すると、 は READY状態StepBに移行します。がStepA失敗した場合、または StepAがキャンセルされた場合、 は CANCELED状態StepBに移行します。

複数のステップに依存関係を設定できます。たとえば、 StepCが StepAと の両方に依存する場合StepB、 StepCは他の 2 つのステップが完了するまで開始しません。

```
name: Step-Step Dependency Test
specificationVersion: 'jobtemplate-2023-09'
steps:
- name: A
  script:
    actions:
      onRun:
        command: bash
        args: ['{{ Task.File.run }}']
    embeddedFiles:
      - name: run
        type: TEXT
        data: |
          #!/bin/env bash

          set -euo pipefail

          sleep 1
          echo Task A Done!
```

```
- name: B
  dependencies:
  - dependsOn: A # This means Step B depends on Step A
  script:
    actions:
      onRun:
        command: bash
        args: ['{{ Task.File.run }}']
    embeddedFiles:
      - name: run
        type: TEXT
        data: |
          #!/bin/env bash

          set -euo pipefail

          sleep 1
          echo Task B Done!
```

Deadline Cloud でジョブを変更する

次の AWS Command Line Interface （AWS CLI) update コマンドを使用して、ジョブの設定を変更するか、ジョブ、ステップ、またはタスクのターゲットステータスを設定できます。

- `aws deadline update-job`
- `aws deadline update-step`
- `aws deadline update-task`

次のupdateコマンドの例では、それぞれを独自の情報に置き換え*user input placeholder*ます。

Example – ジョブを再キューに入れる

ステップの依存関係がない限り、ジョブ内のすべてのタスクは READYステータスに切り替わります。依存関係を持つステップは、復元PENDING時に READYまたは のいずれかに切り替わります。

```
aws deadline update-job \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
```

```
--target-task-run-status PENDING
```

Example – ジョブをキャンセルする

ステータスがない、SUCCEEDEDまたは とマークFAILEDされているジョブ内のすべてのタスクCANCELED。

```
aws deadline update-job \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--target-task-run-status CANCELED
```

Example – ジョブが失敗したことを示す

ステータスが のジョブ内のすべてのタスクSUCCEEDEDは変更されません。他のすべてのタスクは とマークされますFAILED。

```
aws deadline update-job \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--target-task-run-status FAILED
```

Example – ジョブを成功としてマークする

ジョブ内のすべてのタスクが SUCCEEDED状態に移行します。

```
aws deadline update-job \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--target-task-run-status SUCCEEDED
```

Example – ジョブを停止する

、SUCCEEDED、CANCELEDまたは FAILED状態のジョブのタスクは変更されません。他のすべてのタスクは とマークされますSUSPENDED。

```
aws deadline update-job \
--farm-id farmID \
```

```
--queue-id queueID \
--job-id jobID \
--target-task-run-status SUSPENDED
```

Example – ジョブの優先度を変更する

キュー内のジョブの優先度を更新して、スケジュールされている順序を変更します。優先度の高いジョブは通常、最初にスケジュールされます。

```
aws deadline update-job \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--priority 100
```

Example – 失敗したタスクの許容数を変更する

残りのタスクがキャンセルされるまでにジョブが保持できる失敗したタスクの最大数を更新します。

```
aws deadline update-job \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--max-failed-tasks-count 200
```

Example – 許可されるタスク再試行の数を変更する

タスクが失敗する前に、タスクの最大再試行回数を更新します。再試行の最大数に達したタスクは、この値が増加するまで再キューに入れることはできません。

```
aws deadline update-job \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--max-retries-per-task 10
```

Example – ジョブをアーカイブする

ジョブのライフサイクルステータスを に更新しますARCHIVED。アーカイブされたジョブはスケジュールまたは変更できません。アーカイブできるのは、FAILED、、CANCELED、SUCCEEDEDまたは SUSPENDED状態のジョブのみです。

```
aws deadline update-job \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--lifecycle-status ARCHIVED
```

Example – ステップを再キューに入れる

ステップの依存関係がない限り、ステップ内のすべてのタスクは READY状態に切り替わります。依存関係を持つステップのタスクは READYまたは のいずれかに切り替わりPENDING、タスクは復元されます。

```
aws deadline update-step \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--step-id stepID \
--target-task-run-status PENDING
```

Example – ステップをキャンセルする

ステータスがない、SUCCEEDEDまたは FAILEDとマークされているステップ内のすべてのタスクCANCELED。

```
aws deadline update-step \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--step-id stepID \
--target-task-run-status CANCELED
```

Example – ステップのマークに失敗しました

ステータスが のステップのすべてのタスクSUCCEEDEDは変更されません。他のすべてのタスクは とマークされますFAILED。

```
aws deadline update-step \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--step-id stepID \
```

```
--target-task-run-status FAILED
```

Example – ステップを成功としてマークする

ステップのすべてのタスクは とマークされますSUCCEEDED。

```
aws deadline update-step \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--step-id stepID \
--target-task-run-status SUCCEEDED
```

Example – ステップを停止する

、SUCCEEDED、CANCELEDまたは FAILED状態の ステップのタスクは変更されません。他のすべてのタスクは とマークされますSUSPENDED。

```
aws deadline update-step \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--step-id stepID \
--target-task-run-status SUSPENDED
```

Example – タスクのステータスを変更する

update-task Deadline Cloud CLI コマンドを使用すると、タスクは指定されたステータスに切り替わります。

```
aws deadline update-task \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--step-id stepID \
--task-id taskID \
--target-task-run-status SUCCEEDED | SUSPENDED | CANCELED | FAILED | PENDING
```

Deadline Cloud カスタマーマネージドフリートを作成して使用する

カスタマーマネージドフリート (CMF) を作成すると、処理パイプラインを完全に制御できます。各ワーカーのネットワーク環境とソフトウェア環境を定義します。Deadline Cloud は、ジョブのリポジトリとスケジューラとして機能します。

ワーカーは、Amazon Elastic Compute Cloud (Amazon EC2) インスタンス、コロケーション施設のワーカー、またはオンプレミスワーカーです。各ワーカーは Deadline Cloud ワーカーエージェントを実行する必要があります。すべてのワーカーは、Deadline Cloud サービスエンドポイントにアクセスできる必要があります。

以下のトピックでは、Amazon EC2 インスタンスを使用して基本的な CMF を作成する方法について説明します。

トピック

- カスタマーマネージドフリートを作成する
- ワーカーホストのセットアップと設定
- Windows ジョブユーザーシークレットへのアクセスを管理する
- ジョブに必要なソフトウェアをインストールして設定する
- AWS 認証情報の設定
- AWS エンドポイント接続を許可するようにネットワークを設定する
- ワーカーホストの設定をテストする
- Amazon Machine Image の作成
- Amazon EC2 Auto Scaling グループを使用してフリートインフラストラクチャを作成する

カスタマーマネージドフリートを作成する

カスタマーマネージドフリート (CMF) を作成するには、次の手順を実行します。

Deadline Cloud console

Deadline Cloud コンソールを使用してカスタマーマネージドフリートを作成するには

1. Deadline Cloud コンソールを開きます。

2. Farms を選択します。使用可能なファームのリストが表示されます。
3. 作業するファームの名前を選択します。
4. フリートタブを選択し、フリートの作成を選択します。
5. フリートの名前を入力します。
6. (オプション) フリートの説明を入力します。
7. フリートタイプにカスタマーマネージドを選択します。
8. フリートのサービスアクセスを選択します。
   a. アクセス許可をより詳細に制御するには、フリートごとに新しいサービスロールの作成と使用オプションを使用することをお勧めします。このオプションはデフォルト選択。
   b. サービスロールの選択を選択して、既存のサービスロールを使用することもできます。
9. 選択内容を確認し、次へを選択します。
10. フリートのオペレーティングシステムを選択します。フリートのすべてのワーカーには、共通のオペレーティングシステムが必要です。
11. ホスト CPU アーキテクチャを選択します。
12. フリートのワークロードの需要を満たすために、vCPU とメモリのハードウェア機能の最小値と最大値を選択します。
13. Auto Scaling タイプを選択します。詳細については、EventBridge を使用して Auto Scaling イベントを処理する」を参照してください。
    - スケーリングなし: オンプレミスフリートを作成し、Deadline Cloud Auto Scaling をオプトアウトしたいと考えています。
    - スケーリングの推奨事項: Amazon Elastic Compute Cloud (Amazon EC2) フリートを作成しています。
14. (オプション) 矢印を選択して、機能の追加セクションを展開します。
15. (オプション) GPU 機能の追加 - オプション のチェックボックスを選択し、GPUs とメモリの最小値と最大値を入力します。
16. 選択内容を確認し、次へを選択します。
17. (オプション) カスタムワーカー機能を定義し、次へを選択します。
18. ドロップダウンを使用して、フリートに関連付ける 1 つ以上のキューを選択します。

> **Note**
>
> フリートは、すべて同じ信頼境界にあるキューにのみ関連付けることをお勧めします。これにより、同じワーカーで実行中のジョブ間の強力なセキュリティ境界が確保されます。

19. キューの関連付けを確認し、次へを選択します。
20. (オプション) デフォルトの Conda キュー環境では、ジョブによってリクエストされた Conda パッケージをインストールするキューの環境を作成します。

> **Note**
>
> Conda キュー環境は、ジョブによってリクエストされた Conda パッケージをインストールするために使用されます。CMFs にはデフォルトで必要な Conda コマンドがインストールされないため、通常、CMFs に関連付けられたキューで Conda キュー環境のチェックを解除する必要があります。

21. (オプション) CMF にタグを追加します。詳細については、「AWS リソースのタグ付け」を参照してください。
22. フリート設定を確認して変更を加え、フリートの作成を選択します。
23. フリートタブを選択し、フリート ID を書き留めます。

AWS CLI

を使用してカスタマーマネージドフリート AWS CLI を作成するには

1. ターミナルを開きます。
2. 新しいエディタ`fleet-trust-policy.json`で を作成します。

   a. 次の IAM ポリシーを追加し、*`ITALICIZED`* テキストをアカウント ID と Deadline Cloud ファーム ID に置き換えます AWS 。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Principal": {
                "Service": "credentials.deadline.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "ACCOUNT_ID"
                },
                "ArnEquals": {
                    "aws:SourceArn":
 "arn:aws:deadline:*:ACCOUNT_ID:farm/FARM_ID"
                }
            }
        }
    ]
}
```

b. 変更内容を保存します。

3. `fleet-policy.json` を作成します。

a. 次の IAM ポリシーを追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "deadline:AssumeFleetRoleForWorker",
                "deadline:UpdateWorker",
                "deadline:DeleteWorker",
                "deadline:UpdateWorkerSchedule",
                "deadline:BatchGetJobEntity",
                "deadline:AssumeQueueRoleForWorker"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalAccount": "${aws:ResourceAccount}"
                }
            }
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream"
            ],
            "Resource": "arn:aws:logs:*:*:*:/aws/deadline/*",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalAccount": "${aws:ResourceAccount}"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:GetLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:*:/aws/deadline/*",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalAccount": "${aws:ResourceAccount}"
                }
            }
        }
    ]
}
```

b. 変更内容を保存します。

4. フリート内のワーカーが使用する IAM ロールを追加します。

```
aws iam create-role --role-name FleetWorkerRoleName --assume-role-policy-
document file://fleet-trust-policy.json
aws iam put-role-policy --role-name FleetWorkerRoleName --policy-name
 FleetWorkerPolicy --policy-document file://fleet-policy.json
```

5. `create-fleet-request.json` を作成します。

a. 次の IAM ポリシーを追加し、ITALICIZED テキストを CMF の値に置き換えます。

> **Note**
>
> *ROLE_ARN* は にありますcreate-cmf-fleet.json。

OS_FAMILY の場合は、linux、、macosまたは のいずれかを選択する必要があります windows。

```
{
    "farmId": "FARM_ID",
    "displayName": "FLEET_NAME",
    "description": "FLEET_DESCRIPTION",
    "roleArn": "ROLE_ARN",
    "minWorkerCount": 0,
    "maxWorkerCount": 10,
    "configuration": {
        "customerManaged": {
            "mode": "NO_SCALING",
            "workerCapabilities": {
                "vCpuCount": {
                    "min": 1,
                    "max": 4
                },
                "memoryMiB": {
                    "min": 1024,
                    "max": 4096
                },
                "osFamily": "OS_FAMILY",
                "cpuArchitectureType": "x86_64",
            },
        },
    }
}
```

b. 変更内容を保存します。

6. フリートを作成します。

```
aws deadline create-fleet --cli-input-json file://create-fleet-request.json
```

ワーカーホストのセットアップと設定

ワーカーホストとは、Deadline Cloud ワーカーを実行するホストマシンを指します。このセクションでは、ワーカーホストをセットアップし、特定のニーズに合わせて設定する方法について説明しま

す。各ワーカーホストは、ワーカーエージェントと呼ばれるプログラムを実行します。ワーカーエージェントは以下の責任を負います。

- ワーカーのライフサイクルの管理。
- 割り当てられた作業、その進行状況、結果を同期します。
- 実行中の作業のモニタリング。
- 設定された送信先にログを転送する。

提供された Deadline Cloud ワーカーエージェントを使用することをお勧めします。ワーカーエージェントはオープンソースであり、機能リクエストをお勧めしますが、ニーズに合わせて開発およびカスタマイズすることもできます。

以下のセクションのタスクを完了するには、以下が必要です。

Linux

- Linuxベースの Amazon Elastic Compute Cloud (Amazon EC2) インスタンス。Amazon Linux 2023 をお勧めします。
- sudo 権限
- Python 3.9 以降

Windows

- Windowsベースの Amazon Elastic Compute Cloud (Amazon EC2) インスタンス。をお勧めしますWindows Server 2022。
- ワーカーホストへの管理者アクセス
- すべてのユーザーにインストールされた Python 3.9 以降

Python 仮想環境を作成して設定する

Python 3.9 以降をインストールして に配置しLinuxている場合は、 で Python 仮想環境を作成できますPATH。

> **Note**
>
> ではWindows、エージェントファイルは Python のグローバル site-packages ディレクトリにインストールする必要があります。Python 仮想環境は現在サポートされていません。

Python 仮想環境を作成してアクティブ化するには

1. `root` ユーザーとしてターミナルを開きます (または `sudo` / を使用しますsu) 。
2. Python 仮想環境を作成してアクティブ化します。

```
python3 -m venv /opt/deadline/worker
source /opt/deadline/worker/bin/activate
pip install --upgrade pip
```

Deadline Cloud ワーカーエージェントをインストールする

Python をセットアップし、 に仮想環境を作成したらLinux、Deadline Cloud ワーカーエージェント Python パッケージをインストールします。

ワーカーエージェントの Python パッケージをインストールするには

リナックス

1. `root` ユーザーとしてターミナルを開きます (または `sudo` / を使用しますsu) 。
2. PyPI から Deadline Cloud ワーカーエージェントパッケージをダウンロードしてインストールします。

```
/opt/deadline/worker/bin/python -m pip install deadline-cloud-worker-agent
```

Windows

1. 管理者コマンドプロンプトまたは PowerShell ターミナルを開きます。
2. PyPI から Deadline Cloud ワーカーエージェントパッケージをダウンロードしてインストールします。

```
python -m pip install deadline-cloud-worker-agent
```

Windows ワーカーホストに長いパス名 (250 文字以上) が必要な場合は、次のように長いパス名を有効にする必要があります。

Windows ワーカーホストのロングパスを有効にするには

1. ロングパスレジストリキーが有効になっていることを確認します。詳細については、Microsoft ウェブサイトの「ログパスを有効にするレジストリ設定」を参照してください。
2. Windows SDK for Desktop C++ x86 アプリをインストールします。詳細については、Windows 開発センターのWindows「 SDK」を参照してください。
3. ワーカーエージェントがインストールされている環境で Python のインストール場所を開きます。デフォルトは `C:\Program Files\Python311` です。という名前の実行可能ファイルがありますpythonservice.exe。
4. `pythonservice.exe.manifest` 同じ場所に という名前の新しいファイルを作成します。以下を追加します。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
  <assemblyIdentity type="win32" name="pythonservice" processorArchitecture="x86"
 version="1.0.0.0"/>
  <application xmlns="urn:schemas-microsoft-com:asm.v3">
    <windowsSettings>
      <longPathAware xmlns="http://schemas.microsoft.com/SMI/2016/
WindowsSettings">true</longPathAware>
    </windowsSettings>
  </application>
</assembly>
```

5. コマンドプロンプトを開き、作成したマニフェストファイルの場所で次のコマンドを実行します。

```
"C:\Program Files (x86)\Windows Kits\10\bin\10.0.26100.0\x86\mt.exe" -manifest
 pythonservice.exe.manifest -outputresource:pythonservice.exe;#1
```

次のような出力が表示されます:

```
Microsoft (R) Manifest Tool
Copyright (c) Microsoft Corporation.
All rights reserved.
```

ワーカーは長いパスにアクセスできるようになりました。クリーンアップするには、`pythonservice.exe.manifest` ファイルを削除して SDK をアンインストールします。

Deadline Cloud ワーカーエージェントを設定する

Deadline Cloud ワーカーエージェントの設定は、3 つの方法で設定できます。`install-deadline-worker` ツールを実行して、オペレーティングシステムのセットアップを使用することをお勧めします。

ワーカーエージェントは、Windows でのドメインユーザーとしての実行をサポートしていません。ドメインユーザーとしてジョブを実行するには、ジョブを実行するキューユーザーを設定するときにドメインユーザーアカウントを指定できます。詳細については、「Deadline Cloud ユーザーガイド」の「Deadline Cloud キュー」のステップ 7 を参照してください。 AWS

コマンドライン引数 — コマンドラインから Deadline Cloud ワーカーエージェントを実行するときに引数を指定できます。一部の設定は、コマンドライン引数では利用できません。使用可能なすべてのコマンドライン引数を表示するには、「」と入力します`deadline-worker-agent --help`。

環境変数 — で始まる環境変数を設定することで、Deadline Cloud ワーカーエージェントを設定できます`DEADLINE_WORKER_`。たとえば、使用可能なすべてのコマンドライン引数を表示するには、ワーカーエージェントの出力を詳細に設定`export DEADLINE_WORKER_VERBOSE=true`するために使用できます。その他の例と情報については、`/etc/amazon/deadline/worker.toml.example`Linux「」または`C:\ProgramData\Amazon\Deadline\Config\worker.toml.example`「」を参照してくださいWindows。

設定ファイル — ワーカーエージェントをインストールすると、 `/etc/amazon/deadline/worker.toml` の Linuxまたは `C:\ProgramData\Amazon\Deadline\Config\worker.toml`にある設定ファイルが作成されますWindows。ワーカーエージェントは、起動時にこの設定ファイルをロードします。サンプル設定ファイル (`/etc/amazon/deadline/worker.toml.example` Linuxまたは `C:\ProgramData\Amazon\Deadline\Config\worker.toml.example` Windows) を使用して、特定のニーズに合わせてデフォルトのワーカーエージェント設定ファイルをカスタマイズできます。

最後に、ソフトウェアがデプロイされ、期待どおりに動作したら、ワーカーエージェントの自動シャットダウンを有効にすることをお勧めします。これにより、ワーカーフリートは必要に応じてスケールアップし、ジョブが終了したらシャットダウンできます。自動スケーリングは、必要なリソースのみを使用するのに役立ちます。自動スケーリンググループによって開始されたインスタンスをシャットダウンできるようにするには、 `worker.toml` を設定ファイル`shutdown_on_stop=true`に追加する必要があります。

自動シャットダウンを有効にするには

root ユーザーとして：

- パラメータ を使用してワーカーエージェントをインストールします**--allow-shutdown**。

  リナックス

  次のとおりに入力します。

  ```
  /opt/deadline/worker/bin/install-deadline-worker \
    --farm-id FARM_ID \
    --fleet-id FLEET_ID \
    --region REGION \
    --allow-shutdown
  ```

  Windows

  次のとおりに入力します。

  ```
  install-deadline-worker ^
    --farm-id FARM_ID ^
    --fleet-id FLEET_ID ^
    --region REGION ^
    --allow-shutdown
  ```

ジョブのユーザーとグループを作成する

このセクションでは、エージェントユーザーとキューで`jobRunAsUser`定義されている との間の必要なユーザーとグループの関係について説明します。

Deadline Cloud ワーカーエージェントは、ホスト上のエージェント固有の専用ユーザーとして実行する必要があります。Deadline Cloud キューの `jobRunAsUser`プロパティを設定して、ワーカーがキュージョブを特定のオペレーティングシステムのユーザーおよびグループとして実行できるようにする必要があります。つまり、ジョブが持つ共有ファイルシステムのアクセス許可を制御できます。また、ジョブとワーカーエージェントユーザー間の重要なセキュリティ境界として も提供します。

Linux ジョブのユーザーとグループ

ローカルワーカーエージェントユーザーと を設定するにはjobRunAsUser、次の要件を満たしていることを確認してください。Active Directory や LDAP などの Linux Pluggable Authentication Module (PAM) を使用している場合、手順は異なる場合があります。

ワーカーエージェントをインストールすると、ワーカーエージェントユーザーと共有jobRunAsUserグループが設定されます。デフォルトは deadline-worker-agentと ですがdeadline-job-users、ワーカーエージェントをインストールするときに変更できます。

```
install-deadline-worker \
    --user AGENT_USER_NAME \
    --group JOB_USERS_GROUP
```

コマンドはルートユーザーとして実行する必要があります。

- 各 には、一致するプライマリグループjobRunAsUserが必要です。adduser コマンドを使用してユーザーを作成すると、通常、一致するプライマリグループが作成されます。

  ```
  adduser -r -m jobRunAsUser
  ```

- のプライマリグループは、ワーカーエージェントユーザーのセカンダリグループjobRunAsUserです。共有グループを使用すると、ワーカーエージェントは実行中にジョブでファイルを使用できるようになります。

  ```
  usermod -a -G jobRunAsUser deadline-worker-agent
  ```

- は、共有ジョブグループのメンバーjobRunAsUserである必要があります。

  ```
  usermod -a -G deadline-job-users jobRunAsUser
  ```

- jobRunAsUser は、ワーカーエージェントユーザーのプライマリグループに属してはいけません。ワーカーエージェントによって書き込まれた機密ファイルは、エージェントのプライマリグループによって所有されます。jobRunAsUser がこのグループの一部である場合、ワーカーエージェントファイルには、ワーカーで実行されているジョブがアクセスできる可能性があります。
- デフォルトは、ワーカーが属するファームのリージョンと一致する AWS リージョン 必要があります。これは、ワーカーのすべてのjobRunAsUserアカウントに適用する必要があります。

  ```
  sudo -u jobRunAsUser aws configure set default.region aws-region
  ```

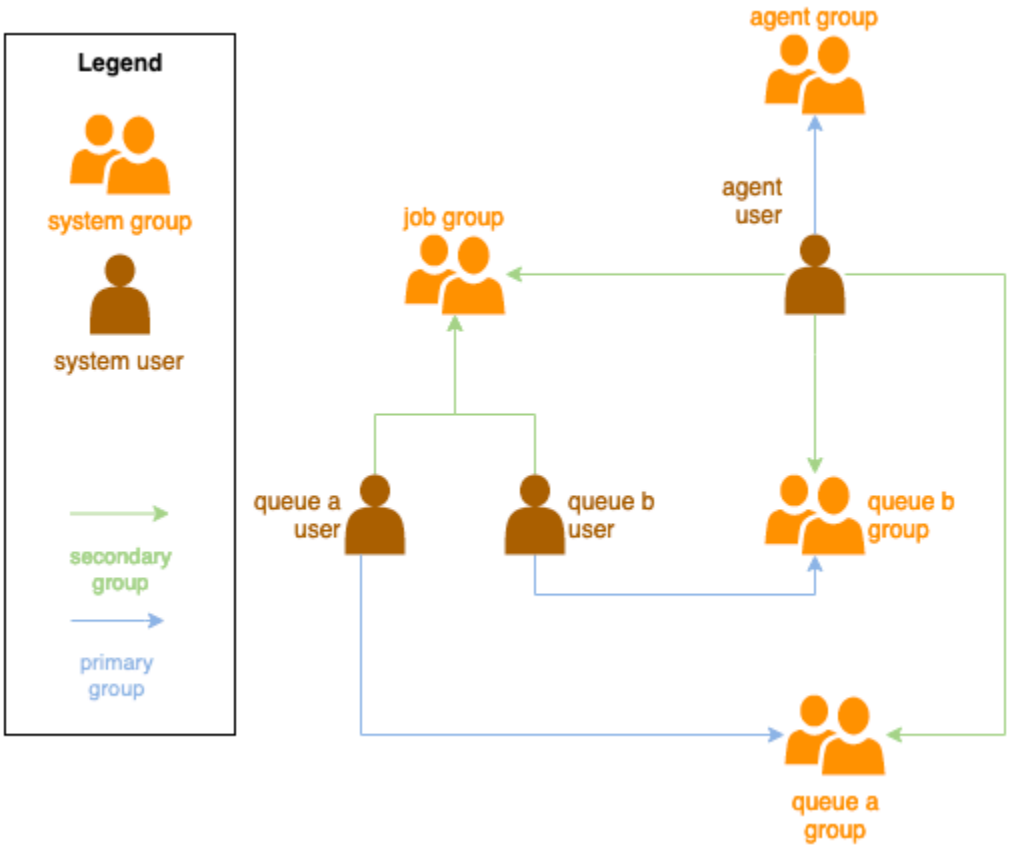
- ワーカーエージェントユーザーは、 としてsudoコマンドを実行できる必要がありますjobRunAsUser。次のコマンドを実行してエディタを開き、新しい sudoers ルールを作成します。

```
visudo -f /etc/sudoers.d/deadline-worker-job-user
```

ファイルに以下を追加します。

```
# Allows the Deadline Cloud worker agent OS user to run commands
 # as the queue OS user without requiring a password.
deadline-worker-agent ALL=(jobRunAsUser) NOPASSWD:ALL
```

次の図は、フリートに関連付けられたキューのエージェントユーザーとjobRunAsUserユーザーおよびグループの関係を示しています。



Windows ユーザー

Windows ユーザーを として使用するにはjobRunAsUser、次の要件を満たしている必要があります。

- すべてのキューjobRunAsUserユーザーが存在している必要があります。
- パスワードは、キューの JobRunAsUserフィールドで指定されたシークレットの値と一致する必要があります。手順については、「Deadline Cloud ユーザーガイド」の「Deadline Cloud キュー」のステップ 7 を参照してください。 AWS
- エージェントユーザーは、それらのユーザーとしてログオンできる必要があります。

Windows ジョブユーザーシークレットへのアクセスを管理する

を使用してキューを設定するときはjobRunAsUser、 AWS Secrets Manager Windows シークレットを指定する必要があります。このシークレットの値は、 形式の JSON エンコードされたオブジェクトであることが期待されます。

```
{
    "password": "JOB_USER_PASSWORD"
}
```

ワーカーがキューに設定された としてジョブを実行するにはjobRunAsUser、フリートの IAM ロールにシークレットの値を取得するためのアクセス許可が必要です。シークレットがカスタマー管理の KMS キーを使用して暗号化されている場合、フリートの IAM ロールには KMS キーを使用して復号するアクセス許可も必要です。

これらのシークレットの最小特権の原則に従うことを強くお勧めします。つまり、キューの jobRunAsUser → windows → のシークレット値を取得するためのアクセスpasswordArnは次のようになります。

- フリートとキューの間にキューフリートの関連付けが作成されたときにフリートロールに付与される
- フリートとキュー間のキューフリートの関連付けが削除されたときにフリートロールから取り消された

さらに、jobRunAsUserパスワードを含む AWS Secrets Manager シークレットは、使用されなくなったときに削除する必要があります。

パスワードシークレットへのアクセスを許可する

Deadline Cloud フリートは、キューとフリートが関連付けられているときに、キューのjobRunAsUserパスワードシークレットに保存されているパスワードにアクセスする必要があります。Secrets Manager リソースポリシーを使用して AWS 、フリートロールへのアクセスを許可することをお勧めします。このガイドラインに厳密に従うと、シークレットにアクセスできるフリートロールを簡単に判断できます。

シークレットへのアクセスを許可するには

1. AWS Secret Manager コンソールを開いてシークレットを開きます。
2. 「リソースのアクセス許可」セクションに、 形式のポリシーステートメントを追加します。

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    //...
    {
      "Effect" : "Allow",
      "Principal" : {
        "AWS" : "FLEET_ROLE_ARN"
      },
      "Action" : "secretsmanager:GetSecretValue",
      "Resource" : "*"
    }
    //...
  ]
}
```

パスワードシークレットへのアクセスを取り消す

フリートがキューへのアクセスが不要になった場合は、キューのパスワードシークレット へのアクセスを削除しますjobRunAsUser。 AWS Secrets Manager リソースポリシーを使用して、フリートロールへのアクセスを許可することをお勧めします。このガイドラインに厳密に従うと、シークレットにアクセスできるフリートロールを簡単に判断できます。

シークレットへのアクセスを取り消すには

1. AWS Secret Manager コンソールを開いてシークレットを開きます。

2. リソースのアクセス許可セクションで、 フォームのポリシーステートメントを削除します。

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    //...
    {
      "Effect" : "Allow",
      "Principal" : {
        "AWS" : "FLEET_ROLE_ARN"
      },
      "Action" : "secretsmanager:GetSecretValue",
      "Resource" : "*"
    }
    //...
  ]
}
```

ジョブに必要なソフトウェアをインストールして設定する

Deadline Cloud ワーカーエージェントを設定したら、ジョブの実行に必要なソフトウェアを使用してワーカーホストを準備できます。

関連付けられた を持つキューにジョブを送信するとjobRunAsUser、ジョブはそのユーザーとして実行されます。絶対パスではないコマンドでジョブを送信する場合、そのコマンドはそのユーザーの PATH にある必要があります。

Linux では、次のいずれかでPATHユーザーの を指定できます。

- ~/.bashrc または ~/.bash_profile
- /etc/profile.d/* や などのシステム設定ファイル /etc/profile
- シェル起動スクリプト: /etc/bashrc。

Windows では、次のいずれかでPATHユーザーの を指定できます。

- ユーザー固有の環境変数
- システム全体の環境変数

デジタルコンテンツ作成ツールアダプターをインストールする

Deadline Cloud は、一般的なデジタルコンテンツ作成 (DCC) アプリケーションを使用するための OpenJobDescription アダプターを提供します。これらのアダプターをカスタマーマネージドフリートで使用するには、DCC ソフトウェアとアプリケーションアダプターをインストールする必要があります。次に、ソフトウェアの実行可能プログラムがシステム検索パス (PATH環境変数など) で使用可能であることを確認します。

カスタマーマネージドフリートに DCC アダプターをインストールするには

1. ターミナルを開きます。
   a. Linux では、`root`ユーザーとしてターミナルを開きます (または `sudo` / を使用しますsu)
   b. Windows で、管理者コマンドプロンプトまたは PowerShell ターミナルを開きます。
2. Deadline Cloud アダプターパッケージをインストールします。

```
pip install deadline deadline-cloud-for-maya deadline-cloud-for-nuke deadline-
cloud-for-blender
```

AWS 認証情報の設定

ワーカーライフサイクルの初期フェーズはブートストラップです。このフェーズでは、ワーカーエージェントソフトウェアがフリートにワーカーを作成し、フリートのロールから AWS 認証情報を取得して、さらなるオペレーションを行います。

AWS credentials for Amazon EC2

Deadline Cloud ワーカーホストのアクセス許可を持つ Amazon EC2 の IAM ロールを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで、ナビゲーションペインでロールを選択し、ロールの作成を選択します。
3. AWS サービスを選択します。
4. サービスまたはユースケースとして EC2 を選択し、次へを選択します。
5. 必要なアクセス許可を付与するには、 `AWSDeadlineCloud-WorkerHost` AWS 管理ポリシーをアタッチします。

On-premises AWS credentials

オンプレミスワーカーは認証情報を使用して Deadline Cloud にアクセスします。最も安全なアクセスのために、IAM Roles Anywhere を使用してワーカーを認証することをお勧めします。詳細については、「IAM Roles Anywhere」を参照してください。

テストには、 AWS 認証情報に IAM ユーザーアクセスキーを使用できます。制限付きインラインポリシーを含めることで、IAM ユーザーの有効期限を設定することをお勧めします。

> **Important**
>
> 次の警告に注意してください。
>
> - アカウントのルート認証情報を使用して AWS リソースにアクセスしないでください。これらの認証情報は無制限のアカウントアクセスを提供し、取り消すのが困難です。
> - アプリケーションファイルにリテラルアクセスキーや認証情報を配置しないでください。これを行うと、パブリックリポジトリにプロジェクトをアップロードするなど、誤って認証情報が公開されるリスクが発生します。
> - プロジェクト領域に認証情報を含むファイルを含めないでください。
> - アクセスキーを保護します。アカウント識別子を確認するためであっても、アクセスキーを認可されていない当事者に提供しないでください。提供すると、第三者がアカウントへの永続的なアクセスを取得する場合があります。
> - 共有 AWS 認証情報ファイルに保存されている認証情報はすべてプレーンテキストで保存されることに注意してください。

詳細については、 AWS 全般のリファレンスの AWS 「アクセスキーを管理するためのベストプラクティス」を参照してください。

IAM ユーザーの作成

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで、ユーザーを選択し、ユーザーの作成を選択します。
3. ユーザーの名前。へのユーザーアクセスを許可する AWS Management Consoleのチェックボックスをオフにし、次へを選択します。
4. ポリシーを直接アタッチするを選択します。

5. アクセス許可ポリシーのリストから、AWSDeadlineCloud-WorkerHost ポリシーを選択し、次へを選択します。
6. ユーザーの詳細を確認し、ユーザーの作成を選択します。

制限された時間枠にユーザーアクセスを制限する

作成する IAM ユーザーアクセスキーは長期的な認証情報です。これらの認証情報が誤って処理された場合に有効期限が切れるようにするには、キーの有効性が切れる日付を指定したインラインポリシーを作成して、これらの認証情報に期限を設定することができます。

1. 先ほど作成した IAM ユーザーを開きます。アクセス許可タブで、アクセス許可の追加を選択し、インラインポリシーの作成を選択します。
2. JSON エディタで、次のアクセス許可を指定します。このポリシーを使用するには、ポリシー例の`aws:CurrentTime`タイムスタンプ値を独自の日時に置き換えます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "DateGreaterThan": {
          "aws:CurrentTime": "2024-01-01T00:00:00Z"
        }
      }
    }
  ]
}
```

アクセスキーの作成

1. ユーザーの詳細ページで、セキュリティ認証情報タブを選択します。[Access keys (アクセスキー)] セクションで、[Create access key (アクセスキーを作成)] を選択します。
2. Other に キーを使用するように指定し、Next を選択し、Create access key を選択します。
3. アクセスキーの取得ページで、表示を選択してユーザーのシークレットアクセスキーの値を表示します。認証情報をコピーするか、csv ファイルをダウンロードできます。

ユーザーアクセスキーを保存する

- ユーザーアクセスキーをワーカーホストシステムのエージェントユーザーの AWS 認証情報ファイルに保存します。
  - ではLinux、ファイルは にあります。 `~/.aws/credentials`
  - ではWindows、ファイルは にあります。 `%USERPROVILE\.aws\credentials`

  次のキーを置き換えます。

```
[default]
aws_access_key_id=ACCESS_KEY_ID
aws_access_key_id=SECRET_ACCESS_KEY
```

> **Important**
>
> この IAM ユーザーが不要になった場合は、AWS セキュリティのベストプラクティスに合わせて削除することをお勧めします。にアクセスするAWS IAM Identity Centerときは、を通じて一時的な認証情報を使用することを人間のユーザーに要求することをお勧めします AWS。

AWS エンドポイント接続を許可するようにネットワークを設定する

Deadline Cloud では、適切なオペレーションを行うために、さまざまな AWS サービスエンドポイントへの安全な接続が必要です。Deadline Cloud を使用するには、ネットワーク環境で Deadline Cloud ワーカーがこれらのエンドポイントに接続できることを確認する必要があります。

アウトバウンド接続をブロックするネットワークファイアウォールの設定がある場合は、特定のエンドポイントにファイアウォール例外を追加する必要がある場合があります。Deadline Cloud では、次のサービスの例外を追加する必要があります。

- Deadline Cloud エンドポイント
- Amazon CloudWatch Logs エンドポイント
- Amazon Simple Storage Service エンドポイント

ジョブで他のサービスを使用している場合は AWS 、それらのサービスにも例外を追加する必要があります。これらのエンドポイントは、AWS 全般のリファレンスガイドの「サービスエンドポイントとクォータ」の章で確認できます。必要なエンドポイントを特定したら、ファイアウォールにアウトバウンドルールを作成して、これらの特定のエンドポイントへのトラフィックを許可します。

適切なオペレーションには、これらのエンドポイントにアクセスできることを確認する必要があります。さらに、必要な Deadline Cloud トラフィックを許可しながら安全な環境を維持するために、仮想プライベートクラウド (VPCs)、セキュリティグループ、ネットワークアクセスコントロールリスト (ACLs) を使用するなど、適切なセキュリティ対策を実装することを検討してください。

ワーカーホストの設定をテストする

ワーカーエージェントをインストールし、ジョブの処理に必要なソフトウェアをインストールし、ワーカーエージェントの AWS 認証情報を設定したら、フリートAMIの を作成する前に、インストールがジョブを処理できることをテストする必要があります。以下をテストする必要があります。

- Deadline Cloud ワーカーエージェントは、システムサービスとして実行されるように適切に設定されています。
- ワーカーが関連するキューの作業をポーリングすること。
- ワーカーがフリートに関連付けられたキューに送信されたジョブを正常に処理すること。

設定をテストし、代表的なジョブを正常に処理できたら、設定されたワーカーを使用して Amazon EC2 ワーカーAMI用の を作成するか、オンプレミスワーカーのモデルとして を作成できます。

> **Note**
>
> Auto Scaling フリートのワーカーホスト設定をテストする場合、次の状況ではワーカーのテストが困難な場合があります。
>
> - キューに作業がない場合、Deadline Cloud はワーカーの起動直後にワーカーエージェントを停止します。
> - ワーカーエージェントが停止時にホストをシャットダウンするように設定されている場合、キューに作業がない場合、エージェントはマシンをシャットダウンします。

これらの問題を回避するには、自動スケーリングを行わないステージングフリートを使用してワーカーを設定およびテストします。ワーカーホストをテストしたら、 をベーキングする前に、必ず正しいフリート ID を設定してくださいAMI。

ワーカーホスト設定をテストするには

1. オペレーティングシステムサービスを起動してワーカーエージェントを実行します。

   Linux

   ルートシェルから次のコマンドを実行します。

   ```
   systemctl start deadline-worker
   ```

   Windows

   管理者コマンドプロンプトまたはPowerShellターミナルから、次のコマンドを入力します。

   ```
   sc.exe start DeadlineWorker
   ```

2. ワーカーをモニタリングして、ワーカーが起動し、作業をポーリングすることを確認します。

   Linux

   ルートシェルから次のコマンドを実行します。

   ```
   systemctl status deadline-worker
   ```

   コマンドは次のようなレスポンスを返します。

   ```
   Active: active (running) since Wed 2023-06-14 14:44:27 UTC; 7min ago
   ```

   レスポンスがそうでない場合は、次のコマンドを使用してログファイルを検査します。

   ```
   tail -n 25 /var/log/amazon/deadline/worker-agent.log
   ```

Windows

管理者コマンドプロンプトまたはPowerShellターミナルから、次のコマンドを入力します。

```
sc.exe query DeadlineWorker
```

コマンドは次のようなレスポンスを返します。

```
STATE   : 4 RUNNING
```

レスポンスに が含まれていない場合はRUNNING、ワーカーログファイルを検査します。管理者PowerShellプロンプトを開き、次のコマンドを実行します。

```
Get-Content -Tail 25 -Path $env:PROGRAMDATA\Amazon\Deadline\Logs\worker-
agent.log
```

3. フリートに関連付けられたキューにジョブを送信します。ジョブは、フリートが処理するジョブを表す必要があります。
4. Deadline Cloud モニターまたは CLI を使用してジョブの進行状況をモニタリングします。ジョブが失敗した場合、セッションログとワーカーログを確認します。
5. ジョブが正常に完了するまで、必要に応じてワーカーホストの設定を更新します。
6. テストジョブが成功すると、ワーカーを停止できます。

Linux

ルートシェルから次のコマンドを実行します。

```
systemctl stop deadline-worker
```

Windows

管理者コマンドプロンプトまたはPowerShellターミナルから、次のコマンドを入力します。

```
sc.exe stop DeadlineWorker
```

Amazon Machine Image の作成

Amazon Elastic Compute Cloud (Amazon EC2Amazon Machine ImageAMI) カスタマーマネージドフリート (CMF) で使用する () を作成するには、このセクションのタスクを完了します。先に進む前に、Amazon EC2 インスタンス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「インスタンスの起動Amazon EC2」を参照してください。

> **Important**
>
> を作成すると、Amazon EC2 インスタンスのアタッチされたボリュームのスナップショットAMIが作成されます。インスタンスにインストールされたソフトウェアは保持されるため、インスタンスは からインスタンスを起動するときに再利用されますAMI。フリートに適用する前に、パッチ適用戦略を採用し、更新されたソフトウェアAMIで新しい を定期的に更新することをお勧めします。

Amazon EC2 インスタンスを準備する

を構築する前にAMI、ワーカーの状態を削除する必要があります。ワーカーの状態は、ワーカーエージェントの起動間で維持されます。この状態が に保持される場合AMI、そこから起動されたすべてのインスタンスが同じ状態を共有します。

既存のログファイルも削除することをお勧めします。AMI を準備するときに、ログファイルを Amazon EC2 インスタンスに残すことができます。これらのファイルを削除すると、AMI を使用するワーカーフリートで発生する可能性のある問題を診断する際の混乱を最小限に抑えることができます。

また、Amazon EC2 の起動時に Deadline Cloud ワーカーエージェントが起動するように、ワーカーエージェントシステムサービスを有効にする必要があります。

最後に、ワーカーエージェントの自動シャットダウンを有効にすることをお勧めします。これにより、ワーカーフリートは必要に応じてスケールアップし、レンダリングジョブが終了したらシャットダウンできます。この自動スケーリングは、必要に応じてリソースのみを使用するようにするのに役立ちます。

Amazon EC2 インスタンスを準備するには

1. Amazon EC2 コンソールを開きます。

2. Amazon EC2 インスタンスの起動 詳細については、「インスタンスを起動する」を参照してください。
3. ID プロバイダー (IdP) に接続するようにホストを設定し、必要な共有ファイルシステムをマウントします。
4. Deadline Cloud ワーカーエージェントをインストールする「」、ワーカーエージェントを設定する「」、「」のチュートリアルに従いますジョブのユーザーとグループを作成する。
5. VFX リファレンスプラットフォームと互換性のあるソフトウェアを実行するために Amazon Linux 2023 AMIに基づく を準備する場合は、いくつかの要件を更新する必要があります。詳細については、AWS 「Deadline Cloud ユーザーガイド」の「VFX リファレンスプラットフォームの互換性」を参照してください。
6. ターミナルを開きます。
   a. Linux では、`root`ユーザーとしてターミナルを開きます (または `sudo` / を使用しますsu) 。
   b. でWindows、管理者コマンドプロンプトまたは PowerShell ターミナルを開きます。
7. ワーカーサービスが実行されておらず、起動時に起動するように設定されていることを確認します。
   a. Linux では、 を実行します。

```
systemctl stop deadline-worker
systemctl enable deadline-worker
```

   b. でWindows、 を実行します。

```
sc.exe stop DeadlineWorker
sc.exe config DeadlineWorker start= auto
```

8. ワーカーの状態を削除します。
   a. Linux では、 を実行します。

```
rm -rf /var/lib/deadline/*
```

   b. でWindows、 を実行します。

```
del /Q /S %PROGRAMDATA%\Amazon\Deadline\Cache\*
```

9. ログファイルを削除します。

a. Linux では、 を実行します。

```
rm -rf /var/log/amazon/deadline/*
```

b. でWindows、 を実行します。

```
del /Q /S %PROGRAMDATA%\Amazon\Deadline\Logs\*
```

10. ではWindows、スタートメニューにある Amazon EC2Launch Settings アプリケーションを実行して、インスタンスの最終ホスト準備とシャットダウンを完了することをお勧めします。

> **Note**
>
> Sysprep なしでシャットダウンを選択し、Sysprep でシャットダウンを選択しないでください。Sysprep でシャットダウンすると、すべてのローカルユーザーが使用できなくなります。詳細については、Windows インスタンス用ユーザーガイドの「カスタム AMI の作成」トピックの「開始する前に」セクションを参照してください。

の構築 AMI

を構築するには AMI

1. Amazon EC2 コンソールを開きます。
2. ナビゲーションペインでインスタンスを選択し、インスタンスを選択します。
3. 「インスタンスの状態」を選択し、「インスタンスを停止」を選択します。
4. インスタンスが停止したら、アクションを選択します。
5. 画像とテンプレートを選択し、画像を作成します。
6. イメージ名を入力します。
7. (オプション) イメージの説明を入力します。
8. [イメージを作成] を選択してください。

Amazon EC2 Auto Scaling グループを使用してフリートインフラストラクチャを作成する

このセクションでは、Amazon EC2 Auto Scaling フリートを作成する方法について説明します。

以下の AWS CloudFormation YAML テンプレートを使用して、Amazon EC2 Auto Scaling (Auto Scaling) グループ、2 つのサブネット、インスタンスプロファイル、およびインスタンスアクセスロールを持つ Amazon Virtual Private Cloud (Amazon VPC) を作成します。これらは、サブネットで Auto Scaling を使用してインスタンスを起動するために必要です。

レンダリングニーズに合わせて、インスタンスタイプのリストを確認して更新する必要があります。

CloudFormation YAML テンプレートで使用されるリソースとパラメータの詳細については、「 AWS CloudFormation ユーザーガイド」の「Deadline Cloud resource type reference」を参照してください。

Amazon EC2 Auto Scaling フリートを作成するには

1. 次の例を使用して、`FarmID`、、`FleetID`および `AMIId`パラメータを定義する CloudFormation テンプレートを作成します。テンプレートをローカルコンピュータの`.YAML`ファイルに保存します。

```
AWSTemplateFormatVersion: 2010-09-09
Description: Amazon Deadline Cloud customer-managed fleet
Parameters:
  FarmId:
    Type: String
    Description: Farm ID
  FleetId:
    Type: String
    Description: Fleet ID
  AMIId:
    Type: String
    Description: AMI ID for launching workers
Resources:
  deadlineVPC:
    Type: 'AWS::EC2::VPC'
    Properties:
      CidrBlock: 100.100.0.0/16
  deadlineWorkerSecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
```

```
    Properties:
      GroupDescription: !Join
        - ' '
        - - Security group created for Deadline Cloud workers in the fleet
          - !Ref FleetId
      GroupName: !Join
        - ''
        - - deadlineWorkerSecurityGroup-
          - !Ref FleetId
      SecurityGroupEgress:
        - CidrIp: 0.0.0.0/0
          IpProtocol: '-1'
      SecurityGroupIngress: []
      VpcId: !Ref deadlineVPC
  deadlineIGW:
    Type: 'AWS::EC2::InternetGateway'
    Properties: {}
  deadlineVPCGatewayAttachment:
    Type: 'AWS::EC2::VPCGatewayAttachment'
    Properties:
      VpcId: !Ref deadlineVPC
      InternetGatewayId: !Ref deadlineIGW
  deadlinePublicRouteTable:
    Type: 'AWS::EC2::RouteTable'
    Properties:
      VpcId: !Ref deadlineVPC
  deadlinePublicRoute:
    Type: 'AWS::EC2::Route'
    Properties:
      RouteTableId: !Ref deadlinePublicRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref deadlineIGW
    DependsOn:
      - deadlineIGW
      - deadlineVPCGatewayAttachment
  deadlinePublicSubnet0:
    Type: 'AWS::EC2::Subnet'
    Properties:
      VpcId: !Ref deadlineVPC
      CidrBlock: 100.100.16.0/22
      AvailabilityZone: !Join
        - ''
        - - !Ref 'AWS::Region'
          - a
```

```
  deadlineSubnetRouteTableAssociation0:
    Type: 'AWS::EC2::SubnetRouteTableAssociation'
    Properties:
      RouteTableId: !Ref deadlinePublicRouteTable
      SubnetId: !Ref deadlinePublicSubnet0
  deadlinePublicSubnet1:
    Type: 'AWS::EC2::Subnet'
    Properties:
      VpcId: !Ref deadlineVPC
      CidrBlock: 100.100.20.0/22
      AvailabilityZone: !Join
        - ''
        - - !Ref 'AWS::Region'
          - c
  deadlineSubnetRouteTableAssociation1:
    Type: 'AWS::EC2::SubnetRouteTableAssociation'
    Properties:
      RouteTableId: !Ref deadlinePublicRouteTable
      SubnetId: !Ref deadlinePublicSubnet1
  deadlineInstanceAccessAccessRole:
    Type: 'AWS::IAM::Role'
    Properties:
      RoleName: !Join
        - '-'
        - - deadline
          - InstanceAccess
          - !Ref FleetId
      AssumeRolePolicyDocument:
        Statement:
          - Effect: Allow
            Principal:
              Service: ec2.amazonaws.com
            Action:
              - 'sts:AssumeRole'
      Path: /
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy'
        - 'arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore'
        - 'arn:aws:iam::aws:policy/AWSDeadlineCloud-WorkerHost'
  deadlineInstanceProfile:
    Type: 'AWS::IAM::InstanceProfile'
    Properties:
      Path: /
      Roles:
```

```
        - !Ref deadlineInstanceAccessAccessRole
  deadlineLaunchTemplate:
    Type: 'AWS::EC2::LaunchTemplate'
    Properties:
      LaunchTemplateName: !Join
        - ''
        - - deadline-LT-
          - !Ref FleetId
      LaunchTemplateData:
        NetworkInterfaces:
          - DeviceIndex: 0
            AssociatePublicIpAddress: true
            Groups:
              - !Ref deadlineWorkerSecurityGroup
            DeleteOnTermination: true
        ImageId: !Ref AMIId
        InstanceInitiatedShutdownBehavior: terminate
        IamInstanceProfile:
          Arn: !GetAtt
            - deadlineInstanceProfile
            - Arn
        MetadataOptions:
          HttpTokens: required
          HttpEndpoint: enabled

  deadlineAutoScalingGroup:
    Type: 'AWS::AutoScaling::AutoScalingGroup'
    Properties:
      AutoScalingGroupName: !Join
        - ''
        - - deadline-ASG-autoscalable-
          - !Ref FleetId
      MinSize: 0
      MaxSize: 10
      VPCZoneIdentifier:
        - !Ref deadlinePublicSubnet0
        - !Ref deadlinePublicSubnet1
      NewInstancesProtectedFromScaleIn: true
      MixedInstancesPolicy:
        InstancesDistribution:
          OnDemandBaseCapacity: 0
          OnDemandPercentageAboveBaseCapacity: 0
          SpotAllocationStrategy: capacity-optimized
          OnDemandAllocationStrategy: lowest-price
```

```
        LaunchTemplate:
          LaunchTemplateSpecification:
            LaunchTemplateId: !Ref deadlineLaunchTemplate
            Version: !GetAtt
              - deadlineLaunchTemplate
              - LatestVersionNumber
          Overrides:
            - InstanceType: m5.large
            - InstanceType: m5d.large
            - InstanceType: m5a.large
            - InstanceType: m5ad.large
            - InstanceType: m5n.large
            - InstanceType: m5dn.large
            - InstanceType: m4.large
            - InstanceType: m3.large
            - InstanceType: r5.large
            - InstanceType: r5d.large
            - InstanceType: r5a.large
            - InstanceType: r5ad.large
            - InstanceType: r5n.large
            - InstanceType: r5dn.large
            - InstanceType: r4.large
      MetricsCollection:
        - Granularity: 1Minute
          Metrics:
            - GroupMinSize
            - GroupMaxSize
            - GroupDesiredCapacity
            - GroupInServiceInstances
            - GroupTotalInstances
            - GroupInServiceCapacity
            - GroupTotalCapacity
```

2. https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソールを開きます。

   AWS CloudFormation コンソールを使用して、作成したテンプレートファイルをアップロードする手順を使用してスタックを作成します。詳細については、AWS CloudFormation 「 ユーザーガイド」の「 AWS CloudFormation コンソールでのスタックの作成」を参照してください。

> **Note**
>
> - ワーカーの Amazon EC2 インスタンスにアタッチされている IAM ロールからの認証情報は、ジョブを含む、そのワーカーで実行されているすべてのプロセスで使用できます。ワーカーには、操作する最小限の権限が必要です。 `deadline:CreateWorker` `deadline:AssumeFleetRoleForWorker`.
> - ワーカーエージェントはキューロールの認証情報を取得し、ジョブを実行して使用するために設定します。Amazon EC2 インスタンスプロファイルロールには、ジョブに必要なアクセス許可を含めないでください。

Deadline Cloud スケールレコメンデーション機能を使用して Amazon EC2 フリートを自動スケーリングする

Deadline Cloud は、Amazon EC2 Auto Scaling (Auto Scaling) グループを活用して、Amazon EC2 カスタマーマネージドフリート (CMF) を自動的にスケーリングします。フリートモードを設定し、必要なインフラストラクチャをアカウントにデプロイして、フリートを自動スケーリングする必要があります。デプロイしたインフラストラクチャはすべてのフリートで機能するため、設定する必要があるのは 1 回だけです。

基本的なワークフローは、フリートモードを自動スケーリングするように設定すると、Deadline Cloud は推奨フリートサイズが変更されるたびにそのフリートの EventBridge イベントを送信します (1 つのイベントにはフリート ID、推奨フリートサイズ、およびその他のメタデータが含まれます) 。関連するイベントをフィルタリングする EventBridge ルールと、それらを使用する Lambda があります。Lambda は Amazon EC2 Auto Scaling と統合`AutoScalingGroup`され、Amazon EC2 フリートを自動的にスケーリングします。

フリートモードを に設定する `EVENT_BASED_AUTO_SCALING`

フリートモードを に設定します`EVENT_BASED_AUTO_SCALING`。コンソールを使用してこれを行うか、 AWS CLI を使用して `CreateFleet`または `UpdateFleet` API を直接呼び出すことができます。モードが設定されると、Deadline Cloud は推奨フリートサイズが変更されるたびに EventBridge イベントの送信を開始します。

- `UpdateFleet` コマンドの例：

```
aws deadline update-fleet \
```

```
  --farm-id FARM_ID \
  --fleet-id FLEET_ID \
  --configuration file://configuration.json
```

- `CreateFleet` コマンドの例：

```
aws deadline create-fleet \
  --farm-id FARM_ID \
  --display-name "Fleet name" \
  --max-worker-count 10 \
  --configuration file://configuration.json
```

以下は、上記の CLI コマンド () `configuration.json`で使用される の例です`--configuration file://configuration.json`。

- フリートで Auto Scaling を有効にするには、 モードを に設定する必要がありますEVENT_BASED_AUTO_SCALING。
- `workerCapabilities` は、作成時に CMF に割り当てられたデフォルト値です。CMF で使用できるリソースを増やす必要がある場合は、これらの値を変更できます。

フリートモードを設定すると、Deadline Cloud はそのフリートのフリートサイズのレコメンデーションイベントの出力を開始します。

```
{
    "customerManaged": {
        "mode": "EVENT_BASED_AUTO_SCALING",
        "workerCapabilities": {
            "vCpuCount": {
                "min": 1,
                "max": 4
            },
            "memoryMiB": {
                "min": 1024,
                "max": 4096
            },
            "osFamily": "linux",
            "cpuArchitectureType": "x86_64"
        }
    }
}
```

AWS CloudFormation テンプレートを使用して Auto Scaling スタックをデプロイする

イベントをフィルタリングする EventBridge ルール、イベントを消費して Auto Scaling を制御する Lambda、未処理のイベントを保存する SQS キューを設定できます。次の AWS CloudFormation テンプレートを使用して、すべてをスタックにデプロイします。リソースを正常にデプロイすると、ジョブを送信でき、フリートは自動的にスケールアップします。

```
Resources:
  AutoScalingLambda:
    Type: 'AWS::Lambda::Function'
    Properties:
      Code:
        ZipFile: |-
          """
          This lambda is configured to handle "Fleet Size Recommendation Change"
          messages. It will handle all such events, and requires
          that the ASG is named based on the fleet id. It will scale up/down the fleet
          based on the recommended fleet size in the message.

          Example EventBridge message:
          {
              "version": "0",
              "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
              "detail-type": "Fleet Size Recommendation Change",
              "source": "aws.deadline",
              "account": "111122223333",
              "time": "2017-12-22T18:43:48Z",
              "region": "us-west-1",
              "resources": [],
              "detail": {
                  "farmId": "farm-12345678900000000000000000000000",
                  "fleetId": "fleet-12345678900000000000000000000000",
                  "oldFleetSize": 1,
                  "newFleetSize": 5,
              }
          }
          """

          import json
          import boto3
          import logging

          logger = logging.getLogger()
```

```
          logger.setLevel(logging.INFO)

          auto_scaling_client = boto3.client("autoscaling")

          def lambda_handler(event, context):
              logger.info(event)
              event_detail = event["detail"]
              fleet_id = event_detail["fleetId"]
              desired_capacity = event_detail["newFleetSize"]

              asg_name = f"deadline-ASG-autoscalable-{fleet_id}"
              auto_scaling_client.set_desired_capacity(
                  AutoScalingGroupName=asg_name,
                  DesiredCapacity=desired_capacity,
                  HonorCooldown=False,
              )

              return {
                  'statusCode': 200,
                  'body': json.dumps(f'Successfully set desired_capacity for {asg_name}
 to {desired_capacity}')
              }
     Handler: index.lambda_handler
     Role: !GetAtt
       - AutoScalingLambdaServiceRole
       - Arn
     Runtime: python3.11
   DependsOn:
     - AutoScalingLambdaServiceRoleDefaultPolicy
     - AutoScalingLambdaServiceRole
 AutoScalingEventRule:
   Type: 'AWS::Events::Rule'
   Properties:
     EventPattern:
       source:
         - aws.deadline
       detail-type:
         - Fleet Size Recommendation Change
     State: ENABLED
     Targets:
       - Arn: !GetAtt
           - AutoScalingLambda
           - Arn
         DeadLetterConfig:
```

```
            Arn: !GetAtt
              - UnprocessedAutoScalingEventQueue
              - Arn
          Id: Target0
          RetryPolicy:
            MaximumRetryAttempts: 15
  AutoScalingEventRuleTargetPermission:
    Type: 'AWS::Lambda::Permission'
    Properties:
      Action: 'lambda:InvokeFunction'
      FunctionName: !GetAtt
        - AutoScalingLambda
        - Arn
      Principal: events.amazonaws.com
      SourceArn: !GetAtt
        - AutoScalingEventRule
        - Arn
  AutoScalingLambdaServiceRole:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: 'sts:AssumeRole'
            Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
        Version: 2012-10-17
      ManagedPolicyArns:
        - !Join
          - ''
          - - 'arn:'
            - !Ref 'AWS::Partition'
            - ':iam::aws:policy/service-role/AWSLambdaBasicExecutionRole'
  AutoScalingLambdaServiceRoleDefaultPolicy:
    Type: 'AWS::IAM::Policy'
    Properties:
      PolicyDocument:
        Statement:
          - Action: 'autoscaling:SetDesiredCapacity'
            Effect: Allow
            Resource: '*'
        Version: 2012-10-17
      PolicyName: AutoScalingLambdaServiceRoleDefaultPolicy
      Roles:
```

```
        - !Ref AutoScalingLambdaServiceRole
  UnprocessedAutoScalingEventQueue:
    Type: 'AWS::SQS::Queue'
    Properties:
      QueueName: deadline-unprocessed-autoscaling-events
    UpdateReplacePolicy: Delete
    DeletionPolicy: Delete
  UnprocessedAutoScalingEventQueuePolicy:
    Type: 'AWS::SQS::QueuePolicy'
    Properties:
      PolicyDocument:
        Statement:
          - Action: 'sqs:SendMessage'
            Condition:
              ArnEquals:
                'aws:SourceArn': !GetAtt
                  - AutoScalingEventRule
                  - Arn
            Effect: Allow
            Principal:
              Service: events.amazonaws.com
            Resource: !GetAtt
              - UnprocessedAutoScalingEventQueue
              - Arn
        Version: 2012-10-17
      Queues:
        - !Ref UnprocessedAutoScalingEventQueue
```

フリートのヘルスチェックを実行する

フリートを作成したら、カスタムヘルスチェックを構築して、フリートが正常で停止したインスタンスがないことを確認し、不要なコストを防ぐ必要があります。GitHub の「Deadline Cloud フリートヘルスチェックのデプロイ」を参照してください。これにより、、起動テンプレートAmazon Machine Image、またはネットワーク設定が誤って変更され、検出されないリスクが軽減されます。

Deadline Cloud のサービスマネージドフリートを設定して使用する

サービスマネージドフリート (SMF) は、Deadline Cloud によって管理されるワーカーのコレクションです。SMF を使用すると、処理の需要に応じてフリートスケーリングを管理したり、タスク完了後にフリートサイズを削減したりする必要がなくなります。

SMF がデフォルトの Conda キュー環境を使用してキューに関連付けられている場合、Deadline Cloud はフリート内のワーカーを適切なソフトウェアパッケージで設定します。サポートされているパートナーアプリケーションについては、AWS Deadline Cloud ユーザーガイドの「デフォルトの Conda キュー環境」を参照してください。

ほとんどの場合、ワークロードを処理するために SMF を変更する必要はありません。ただし、状況によってはフリートを変更する必要がある場合があります。具体的には次のとおりです。

- ソフトウェアまたはDockerコンテナをインストールするための昇格されたアクセス許可を必要とするスクリプトの実行。

トピック

- 管理者としてスクリプトを実行してワーカーを設定する

管理者としてスクリプトを実行してワーカーを設定する

カスタムフリートホスト設定スクリプトを使用すると、サービスマネージドフリートワーカーでソフトウェアのインストールなどの管理タスクを実行できます。これらのスクリプトは昇格された権限で実行されるため、システムのワーカーを柔軟に設定できます。

Deadline Cloud は、ワーカーが `STARTING`状態になった後、タスクを実行する前にスクリプトを実行します。

> **Important**
>
> スクリプトは、Linuxシステムsudo上では昇格されたアクセス許可で実行され、Windowsシステム上では「管理者」で実行されます。スクリプトでセキュリティ上の問題が発生しないようにするのはユーザーの責任です。
> 管理スクリプトを使用する場合、フリートの状態をモニタリングする責任があります。

スクリプトの一般的な用途は次のとおりです。

- 管理者アクセスを必要とするソフトウェアのインストール
- Docker コンテナのインストール

ホスト設定スクリプトは、 コンソールまたは を使用して作成および更新できます AWS CLI。

Console

1. フリートの詳細ページで、設定タブを選択します。
2. スクリプト フィールドに、昇格されたアクセス許可で実行するスクリプトを入力します。インポートを選択して、ワークステーションからスクリプトをロードできます。
3. スクリプトを実行するタイムアウト期間を秒単位で設定します。デフォルトは 300 秒 (5 分) です。
4. 変更を保存を選択してスクリプトを保存します。

Create with CLI

ホスト設定スクリプトを使用してフリートを作成するには、次の AWS CLI コマンドを使用します。*#######*テキストを自分の情報に置き換えます。

```
aws deadline-internal create-fleet \
--farm-id farm-12345 \
--display-name "fleet-name" \
--max-worker-count 1 \
--configuration '{
  "serviceManagedEc2": {
    "instanceCapabilities": {
      "vCpuCount": {"min": 2},
      "memoryMiB": {"min": 4096},
      "osFamily": "linux",
      "cpuArchitectureType": "x86_64"
    },
    "instanceMarketOptions": {"type":"spot"}
  }
}' \
--role-arn arn:aws:iam::111122223333:role/role-name \
--host-configuration '{ "scriptBody": "script body", "scriptTimeoutSeconds": timeout
 value}'
```

Update with CLI

フリートのホスト設定スクリプトを更新するには、次の AWS CLI コマンドを使用します。*#### ####*テキストを自分の情報に置き換えます。

```
aws deadline update-fleet \
--farm-id farm-12345 \
--fleet-id fleet-455678 \
--host-configuration '{ "scriptBody": "script body", "scriptTimeoutSeconds": timeout
 value}'
```

次のスクリプトは、以下を示しています。

- スクリプトで使用できる環境変数
- その AWS 認証情報がシェルで機能している
- スクリプトが昇格されたシェルで実行されていること

リナックス

次のスクリプトを使用して、スクリプトが `root`権限で実行されていることを示します。

```
# Print environment variables
set
# Check AWS Credentials
aws sts get-caller-identity
```

Windows

次のPowerShellスクリプトを使用して、スクリプトが管理者権限で実行されていることを示します。

```
Get-ChildItem env: | ForEach-Object { "$($_.Name)=$($_.Value)" }
aws sts get-caller-identity
function Test-AdminPrivileges {
    $currentUser = New-Object
 Security.Principal.WindowsPrincipal([Security.Principal.WindowsIdentity]::GetCurrent())
    $isAdmin =
 $currentUser.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator)

    return $isAdmin
```

```
}

if (Test-AdminPrivileges) {
    Write-Host "The current PowerShell session is elevated (running as
 Administrator)."
} else {
    Write-Host "The current PowerShell session is not elevated (not running as
 Administrator)."
}
exit 0
```

ホスト設定スクリプトのトラブルシューティング

ホスト設定スクリプトを実行する場合：

- 成功時: ワーカーがジョブを実行する
- 障害発生時 (ゼロ以外の終了コードまたはクラッシュ):
  - ワーカーがシャットダウンする

  フリートは、最新のホスト設定スクリプトを使用して新しいワーカーを自動的に起動します。

スクリプトをモニタリングするには：

1. Deadline Cloud コンソールでフリートページを開く
2. ワーカーの表示を選択して Deadline Cloud モニターを開きます
3. モニターページでワーカーのステータスを表示する

重要な注意事項：

- エラーによりシャットダウンしたワーカーは、モニターのワーカーのリストでは使用できません。CloudWatch Logs を使用して、次のロググループのワーカーログを表示します。

  ```
  /aws/deadline/farm-XXXXX/fleet-YYYYY
  ```

  そのロググループ内には、

  ```
  worker-ZZZZZ
  ```

- CloudWatch Logs は、設定した保持期間に従ってワーカーログを保持します。

Deadline Cloud でのソフトウェアライセンスの使用

Deadline Cloud には、ジョブにソフトウェアライセンスを提供する 2 つの方法があります。

- 使用状況ベースのライセンス (UBL) – は、フリートがジョブの処理に使用した時間数に基づいて追跡および請求を行います。ライセンスの数は設定されていないため、フリートは必要に応じてスケールできます。UBL は、サービスマネージドフリートの標準です。カスタマーマネージドフリートの場合、UBL の Deadline Cloud ライセンスエンドポイントを接続できます。UBL は Deadline Cloud ワーカーがレンダリングするライセンスを提供し、DCC アプリケーションのライセンスは提供しません。
- Bring your own license (BYOL) – サービスまたはカスタマーマネージドフリートで既存のソフトウェアライセンスを使用できます。BYOL を使用して、Deadline Cloud 使用状況ベースのライセンスでサポートされていないソフトウェアのライセンスサーバーに接続できます。カスタムライセンスサーバーに接続することで、サービスマネージドフリートで BYOL を使用できます。

トピック

- サービスマネージドフリートをカスタムライセンスサーバーに接続する
- カスタマーマネージドフリートをライセンスエンドポイントに接続する

サービスマネージドフリートをカスタムライセンスサーバーに接続する

Deadline Cloud のサービスマネージドフリートで使用する独自のライセンスサーバーを持ち込むことができます。独自のライセンスを取得するには、ファームのキュー環境を使用してライセンスサーバーを設定できます。ライセンスサーバーを設定するには、ファームとキューが既にセットアップされている必要があります。

ソフトウェアライセンスサーバーへの接続方法は、フリートの設定とソフトウェアベンダーの要件によって異なります。通常、次の 2 つの方法のいずれかでサーバーにアクセスします。

- ライセンスサーバーに直接送信します。ワーカーは、インターネットを使用してソフトウェアベンダーのライセンスサーバーからライセンスを取得します。すべてのワーカーがサーバーに接続できる必要があります。
- ライセンスプロキシ経由。ワーカーはローカルネットワークのプロキシサーバーに接続します。インターネット経由でベンダーのライセンスサーバーに接続できるのはプロキシサーバーのみです。

以下の手順では、Amazon EC2 Systems Manager (SSM) を使用して、ワーカーインスタンスからライセンスサーバーまたはプロキシインスタンスにポートを転送します。

トピック

- ステップ 1: キュー環境を設定する
- ステップ 2: (オプション) ライセンスプロキシインスタンスのセットアップ
- ステップ 3: AWS CloudFormation テンプレートのセットアップ

ステップ 1: キュー環境を設定する

ライセンスサーバーにアクセスするようにキュー内のキュー環境を設定できます。まず、次のいずれかの方法を使用して、ライセンスサーバーアクセスが設定されている AWS インスタンスがあることを確認します。

- ライセンスサーバー – インスタンスはライセンスサーバーを直接ホストします。
- ライセンスプロキシ – インスタンスはライセンスサーバーへのネットワークアクセスを持ち、ライセンスサーバーポートをライセンスサーバーに転送します。ライセンスプロキシインスタンスの設定方法の詳細については、「」を参照してくださいステップ 2: (オプション) ライセンスプロキシインスタンスのセットアップ。

必要なアクセス許可をキューロールに追加するには

1. Deadline Cloud コンソールから、ダッシュボードに移動を選択します。
2. ダッシュボードから、ファームを選択し、設定するキューを選択します。
3. キューの詳細 > サービスロールから、ロールを選択します。
4. アクセス許可を追加 を選択し、インラインポリシーの作成 を選択します。
5. JSON ポリシーエディタを選択し、次のテキストをコピーしてエディタに貼り付けます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Action": [
                "ssm:StartSession"
```

```
            ],
            "Resource": [
                "arn:aws:ssm:region::document/AWS-StartPortForwardingSession",
                "arn:aws:ec2:region:account_id:instance/instance_id"
            ]
        }
    ]
}
```

6. 新しいポリシーを保存する前に、ポリシーテキストで次の値を置き換えます。
   - をファームがある AWS リージョンregionに置き換えます。
   - を、使用しているライセンスサーバーまたはプロキシインスタンスのインスタンス ID instance_idに置き換えます。
   - をファームを含む AWS アカウント番号account_idに置き換えます。
7. [次へ] を選択します。
8. ポリシー名には、 と入力します**LicenseForwarding**。
9. ポリシーの作成を選択して変更を保存し、必要なアクセス許可を持つポリシーを作成します。

新しいキュー環境をキューに追加するには

1. Deadline Cloud コンソールから、まだダッシュボードに移動していない場合は、「ダッシュボードに移動」を選択します。
2. ダッシュボードから、ファームを選択し、設定するキューを選択します。
3. キュー環境 > アクション > YAML で新しい を作成する を選択します。
4. 次のテキストをコピーして YAML スクリプトエディタに貼り付けます。

   Windows

```
specificationVersion: "environment-2023-09"
parameterDefinitions:
  - name: LicenseInstanceId
    type: STRING
    description: >
      The Instance ID of the license server/proxy instance
    default: ""
```

```
  - name: LicenseInstanceRegion
    type: STRING
    description: >
      The region containing this farm
    default: ""
  - name: LicensePorts
    type: STRING
    description: >
      Comma-separated list of ports to be forwarded to the license server/proxy
 instance.
      Example: "2700,2701,2702"
    default: ""
environment:
  name: BYOL License Forwarding
  variables:
    example_LICENSE: 2700@localhost
  script:
    actions:
      onEnter:
        command: powershell
        args: [ "{{Env.File.Enter}}"]
      onExit:
        command: powershell
        args: [ "{{Env.File.Exit}}" ]
    embeddedFiles:
      - name: Enter
        filename: enter.ps1
        type: TEXT
        runnable: True
        data: |
          $ZIP_NAME="SessionManagerPlugin.zip"
          Invoke-WebRequest -Uri "https://s3.amazonaws.com/session-manager-
downloads/plugin/latest/windows/$ZIP_NAME" -OutFile $ZIP_NAME
          Expand-Archive -Path $ZIP_NAME
          Expand-Archive -Path .\SessionManagerPlugin\package.zip
          conda activate
          python {{Env.File.StartSession}} {{Session.WorkingDirectory}}\package
\bin\session-manager-plugin.exe
      - name: Exit
        filename: exit.ps1
        type: TEXT
        runnable: True
        data: |
          Write-Output "Killing SSM Manager Plugin PIDs: $env:BYOL_SSM_PIDS"
```

```
          "$env:BYOL_SSM_PIDS".Split(",") | ForEach {
            Write-Output "Killing $_"
            Stop-Process -Id $_ -Force
          }
      - name: StartSession
        type: TEXT
        data: |
          import boto3
          import json
          import subprocess
          import sys

          instance_id = "{{Param.LicenseInstanceId}}"
          region = "{{Param.LicenseInstanceRegion}}"
          license_ports_list = "{{Param.LicensePorts}}".split(",")

          ssm_client = boto3.client("ssm", region_name=region)
          pids = []

          for port in license_ports_list:
            session_response = ssm_client.start_session(
              Target=instance_id,
              DocumentName="AWS-StartPortForwardingSession",
              Parameters={"portNumber": [port], "localPortNumber": [port]}
            )

            cmd = [
              sys.argv[1],
              json.dumps(session_response),
              region,
              "StartSession",
              "",
              json.dumps({"Target": instance_id}),
              f"https://ssm.{region}.amazonaws.com"
            ]

            process = subprocess.Popen(cmd, stdout=subprocess.DEVNULL,
stderr=subprocess.DEVNULL)
            pids.append(process.pid)
            print(f"SSM Port Forwarding Session started for port {port}")

          print(f"openjd_env: BYOL_SSM_PIDS={','.join(str(pid) for pid in
pids)}")
```

Linux

```
specificationVersion: "environment-2023-09"
parameterDefinitions:
  - name: LicenseInstanceId
    type: STRING
    description: >
      The Instance ID of the license server/proxy instance
    default: ""
  - name: LicenseInstanceRegion
    type: STRING
    description: >
      The region containing this farm
    default: ""
  - name: LicensePorts
    type: STRING
    description: >
      Comma-separated list of ports to be forwarded to the license server/proxy
 instance.
      Example: "2700,2701,2702"
    default: ""
environment:
  name: BYOL License Forwarding
  variables:
    example_LICENSE: 2700@localhost
  script:
    actions:
      onEnter:
        command: bash
        args: [ "{{Env.File.Enter}}"]
      onExit:
        command: bash
        args: [ "{{Env.File.Exit}}" ]
    embeddedFiles:
      - name: Enter
        type: TEXT
        runnable: True
        data: |
          curl https://s3.amazonaws.com/session-manager-downloads/plugin/
latest/linux_64bit/session-manager-plugin.rpm -Ls | rpm2cpio - | cpio -iv
```

```
 --to-stdout ./usr/local/sessionmanagerplugin/bin/session-manager-plugin >
 {{Session.WorkingDirectory}}/session-manager-plugin
          chmod +x {{Session.WorkingDirectory}}/session-manager-plugin
          conda activate
          python {{Env.File.StartSession}} {{Session.WorkingDirectory}}/session-
manager-plugin
      - name: Exit
        type: TEXT
        runnable: True
        data: |
          echo Killing SSM Manager Plugin PIDs: $BYOL_SSM_PIDS
          for pid in ${BYOL_SSM_PIDS//,/ }; do kill $pid; done
      - name: StartSession
        type: TEXT
        data: |
          import boto3
          import json
          import subprocess
          import sys

          instance_id = "{{Param.LicenseInstanceId}}"
          region = "{{Param.LicenseInstanceRegion}}"
          license_ports_list = "{{Param.LicensePorts}}".split(",")

          ssm_client = boto3.client("ssm", region_name=region)
          pids = []

          for port in license_ports_list:
            session_response = ssm_client.start_session(
              Target=instance_id,
              DocumentName="AWS-StartPortForwardingSession",
              Parameters={"portNumber": [port], "localPortNumber": [port]}
            )

            cmd = [
              sys.argv[1],
              json.dumps(session_response),
              region,
              "StartSession",
              "",
              json.dumps({"Target": instance_id}),
              f"https://ssm.{region}.amazonaws.com"
            ]
```

```
            process = subprocess.Popen(cmd, stdout=subprocess.DEVNULL,
stderr=subprocess.DEVNULL)
            pids.append(process.pid)
            print(f"SSM Port Forwarding Session started for port {port}")

         print(f"openjd_env: BYOL_SSM_PIDS={',.join(str(pid) for pid in
pids)}")
```

5. キュー環境を保存する前に、必要に応じて環境テキストに次の変更を加えます。
   - 以下のパラメータのデフォルト値を更新して、環境を反映します。
     - LicenseInstanceID – ライセンスサーバーまたはプロキシインスタンスの Amazon EC2 インスタンス ID
     - LicenseInstanceRegion – ファームを含む AWS リージョン
     - LicensePorts – ライセンスサーバーまたはプロキシインスタンスに転送されるポートのカンマ区切りリスト (2700,2701 など)
   - 必要なライセンス環境変数を変数セクションに追加します。これらの変数は、ライセンスサーバーポートの localhost に DCCs を向ける必要があります。たとえば、Foundry ライセンスサーバーがポート 6101 でリッスンしている場合、変数を として追加します**foundry_LICENSE: 6101@localhost**。
6. (オプション) Priority を 0 のままにすることも、複数のキュー環境間で優先度の順序を異なるように変更することもできます。
7. キュー環境の作成を選択して、新しい環境を保存します。

   キュー環境が設定されている場合、このキューに送信されたジョブは、設定されたライセンスサーバーからライセンスを取得します。

ステップ 2: (オプション) ライセンスプロキシインスタンスのセットアップ

ライセンスサーバーを使用する代わりに、ライセンスプロキシを使用できます。ライセンスプロキシを作成するには、ライセンスサーバーへのネットワークアクセス権を持つ新しい Amazon Linux 2023 インスタンスを作成します。必要に応じて、VPN 接続を使用してこのアクセスを設定できます。詳細については、「Amazon VPC ユーザーガイド」の「VPN 接続」を参照してください。

Deadline Cloud のライセンスプロキシインスタンスを設定するには、この手順のステップに従います。この新しいインスタンスで次の設定ステップを実行して、ライセンスサーバーへのライセンストラフィックの転送を有効にします。

1. HAProxy パッケージをインストールするには、「」と入力します。

```
sudo yum install haproxy
```

2. /etc/haproxy/haproxy.cfg 設定ファイルの listen license-server セクションを次のように更新します。

   a. LicensePort1 と LicensePort2 をライセンスサーバーに転送するポート番号に置き換えます。必要な数のポートに対応するために、カンマ区切りの値を追加または削除します。

   b. LicenseServerHost をライセンスサーバーのホスト名または IP アドレスに置き換えます。

```
lobal
    log         127.0.0.1 local2
    chroot      /var/lib/haproxy
    user        haproxy
    group       haproxy
    daemon

defaults
    timeout queue           1m
    timeout connect         10s
    timeout client          1m
    timeout server          1m
    timeout http-keep-alive 10s
    timeout check           10s

listen license-server
     bind *:LicensePort1,*:LicensePort2
     server license-server LicenseServerHost
```

3. HAProxy サービスを有効にして起動するには、次のコマンドを実行します。

```
sudo systemctl enable haproxy
sudo service haproxy start
```

ステップを完了したら、転送キュー環境から localhost に送信されたライセンスリクエストを、指定されたライセンスサーバーに転送する必要があります。

ステップ 3: AWS CloudFormation テンプレートのセットアップ

AWS CloudFormation テンプレートを使用して、独自のライセンスを使用するようにファーム全体を設定できます。

1. 次のステップで提供されたテンプレートを変更して、BYOLQueueEnvironment の変数セクションに必要なライセンス環境変数を追加します。
2. 次の AWS CloudFormation テンプレートを使用します。

```
AWSTemplateFormatVersion: 2010-09-09
Description: "Create Deadline Cloud resources for BYOL"

Parameters:
  LicenseInstanceId:
    Type: AWS::EC2::Instance::Id
    Description: Instance ID for the license server/proxy instance
  LicensePorts:
    Type: String
    Description: Comma-separated list of ports to forward to the license instance

Resources:
  JobAttachmentBucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: !Sub byol-example-ja-bucket-${AWS::AccountId}-${AWS::Region}
      BucketEncryption:
        ServerSideEncryptionConfiguration:
          - ServerSideEncryptionByDefault:
              SSEAlgorithm: AES256

  Farm:
    Type: AWS::Deadline::Farm
    Properties:
      DisplayName: BYOLFarm

  QueuePolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      ManagedPolicyName: BYOLQueuePolicy
      PolicyDocument:
        Version: 2012-10-17
```

```
        Statement:
          - Effect: Allow
            Action:
              - s3:GetObject
              - s3:PutObject
              - s3:ListBucket
              - s3:GetBucketLocation
            Resource:
              - !Sub ${JobAttachmentBucket.Arn}
              - !Sub ${JobAttachmentBucket.Arn}/job-attachments/*
            Condition:
              StringEquals:
                aws:ResourceAccount: !Sub ${AWS::AccountId}
          - Effect: Allow
            Action: logs:GetLogEvents
            Resource: !Sub arn:aws:logs:${AWS::Region}:${AWS::AccountId}:log-
group:/aws/deadline/${Farm.FarmId}/*
          - Effect: Allow
            Action:
              - s3:ListBucket
              - s3:GetObject
            Resource:
              - "*"
            Condition:
              ArnLike:
                s3:DataAccessPointArn:
                  - arn:aws:s3:*:*:accesspoint/deadline-software-*
              StringEquals:
                s3:AccessPointNetworkOrigin: VPC

  BYOLSSMPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      ManagedPolicyName: BYOLSSMPolicy
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Action:
              - ssm:StartSession
            Resource:
              - !Sub arn:aws:ssm:${AWS::Region}::document/AWS-
StartPortForwardingSession
```

```
              - !Sub arn:aws:ec2:${AWS::Region}:${AWS::AccountId}:instance/
${LicenseInstanceId}


  WorkerPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      ManagedPolicyName: BYOLWorkerPolicy
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Action:
              - logs:CreateLogStream
            Resource: !Sub arn:aws:logs:${AWS::Region}:${AWS::AccountId}:log-
group:/aws/deadline/${Farm.FarmId}/*
            Condition:
              ForAnyValue:StringEquals:
                aws:CalledVia:
                  - deadline.amazonaws.com
          - Effect: Allow
            Action:
              - logs:PutLogEvents
              - logs:GetLogEvents
            Resource: !Sub arn:aws:logs:${AWS::Region}:${AWS::AccountId}:log-
group:/aws/deadline/${Farm.FarmId}/*


  QueueRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: BYOLQueueRole
      ManagedPolicyArns:
        - !Ref QueuePolicy
        - !Ref BYOLSSMPolicy
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Action:
              - sts:AssumeRole
            Principal:
              Service:
                - credentials.deadline.amazonaws.com
```

```
                - deadline.amazonaws.com
            Condition:
              StringEquals:
                aws:SourceAccount: !Sub ${AWS::AccountId}
              ArnEquals:
                aws:SourceArn: !Ref Farm

  WorkerRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: BYOLWorkerRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AWSDeadlineCloud-FleetWorker
        - !Ref WorkerPolicy
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Action:
              - sts:AssumeRole
            Principal:
              Service: credentials.deadline.amazonaws.com


  Queue:
    Type: AWS::Deadline::Queue
    Properties:
      DisplayName: BYOLQueue
      FarmId: !GetAtt Farm.FarmId
      RoleArn: !GetAtt QueueRole.Arn
      JobRunAsUser:
        Posix:
          Group: ""
          User: ""
        RunAs: WORKER_AGENT_USER
      JobAttachmentSettings:
        RootPrefix: job-attachments
        S3BucketName: !Ref JobAttachmentBucket

  Fleet:
    Type: AWS::Deadline::Fleet
    Properties:
      DisplayName: BYOLFleet
      FarmId: !GetAtt Farm.FarmId
```

```
      MinWorkerCount: 1
      MaxWorkerCount: 2
      Configuration:
        ServiceManagedEc2:
          InstanceCapabilities:
            VCpuCount:
              Min: 4
              Max: 16
            MemoryMiB:
              Min: 4096
              Max: 16384
            OsFamily: LINUX
            CpuArchitectureType: x86_64
          InstanceMarketOptions:
            Type: on-demand
      RoleArn: !GetAtt WorkerRole.Arn

  QFA:
    Type: AWS::Deadline::QueueFleetAssociation
    Properties:
      FarmId: !GetAtt Farm.FarmId
      FleetId: !GetAtt Fleet.FleetId
      QueueId: !GetAtt Queue.QueueId

  CondaQueueEnvironment:
    Type: AWS::Deadline::QueueEnvironment
    Properties:
      FarmId: !GetAtt Farm.FarmId
      Priority: 5
      QueueId: !GetAtt Queue.QueueId
      TemplateType: YAML
      Template: |
        specificationVersion: 'environment-2023-09'
        parameterDefinitions:
        - name: CondaPackages
          type: STRING
          description: >
            This is a space-separated list of Conda package match specifications to
 install for the job.
            E.g. "blender=3.6" for a job that renders frames in Blender 3.6.

            See https://docs.conda.io/projects/conda/en/latest/user-guide/concepts/
pkg-specs.html#package-match-specifications
          default: ""
```

```
          userInterface:
            control: LINE_EDIT
            label: Conda Packages
        - name: CondaChannels
          type: STRING
          description: >
            This is a space-separated list of Conda channels from which to install
 packages. Deadline Cloud SMF packages are
            installed from the "deadline-cloud" channel that is configured by
 Deadline Cloud.

            Add "conda-forge" to get packages from the https://conda-forge.org/
 community, and "defaults" to get packages
            from Anaconda Inc (make sure your usage complies with https://
www.anaconda.com/terms-of-use).
          default: "deadline-cloud"
          userInterface:
            control: LINE_EDIT
            label: Conda Channels
        environment:
          name: Conda
          script:
            actions:
              onEnter:
                command: "conda-queue-env-enter"
                args: ["{{Session.WorkingDirectory}}/.env", "--packages",
 "{{Param.CondaPackages}}", "--channels", "{{Param.CondaChannels}}"]
              onExit:
                command: "conda-queue-env-exit"

  BYOLQueueEnvironment:
    Type: AWS::Deadline::QueueEnvironment
    Properties:
      FarmId: !GetAtt Farm.FarmId
      Priority: 10
      QueueId: !GetAtt Queue.QueueId
      TemplateType: YAML
      Template: !Sub |
        specificationVersion: "environment-2023-09"
        parameterDefinitions:
          - name: LicenseInstanceId
            type: STRING
            description: >
              The Instance ID of the license server/proxy instance
```

```
            default: "${LicenseInstanceId}"
          - name: LicenseInstanceRegion
            type: STRING
            description: >
              The region containing this farm
            default: "${AWS::Region}"
          - name: LicensePorts
            type: STRING
            description: >
              Comma-separated list of ports to be forwarded to the license server/
proxy instance.
              Example: "2700,2701,2702"
            default: "${LicensePorts}"
        environment:
          name: BYOL License Forwarding
          variables:
            example_LICENSE: 2700@localhost
          script:
            actions:
              onEnter:
                command: bash
                args: [ "{{Env.File.Enter}}"]
              onExit:
                command: bash
                args: [ "{{Env.File.Exit}}" ]
            embeddedFiles:
              - name: Enter
                type: TEXT
                runnable: True
                data: |
                  curl https://s3.amazonaws.com/session-manager-downloads/
plugin/latest/linux_64bit/session-manager-plugin.rpm -Ls | rpm2cpio - | cpio
 -iv --to-stdout ./usr/local/sessionmanagerplugin/bin/session-manager-plugin >
 {{Session.WorkingDirectory}}/session-manager-plugin
                  chmod +x {{Session.WorkingDirectory}}/session-manager-plugin
                  conda activate
                  python {{Env.File.StartSession}} {{Session.WorkingDirectory}}/
session-manager-plugin
              - name: Exit
                type: TEXT
                runnable: True
                data: |
                  echo Killing SSM Manager Plugin PIDs: $BYOL_SSM_PIDS
                  for pid in ${!BYOL_SSM_PIDS//,/ }; do kill $pid; done
```

```
              - name: StartSession
                type: TEXT
                data: |
                  import boto3
                  import json
                  import subprocess
                  import sys

                  instance_id = "{{Param.LicenseInstanceId}}"
                  region = "{{Param.LicenseInstanceRegion}}"
                  license_ports_list = "{{Param.LicensePorts}}".split(",")

                  ssm_client = boto3.client("ssm", region_name=region)
                  pids = []

                  for port in license_ports_list:
                    session_response = ssm_client.start_session(
                      Target=instance_id,
                      DocumentName="AWS-StartPortForwardingSession",
                      Parameters={"portNumber": [port], "localPortNumber": [port]}
                    )

                    cmd = [
                      sys.argv[1],
                      json.dumps(session_response),
                      region,
                      "StartSession",
                      "",
                      json.dumps({"Target": instance_id}),
                      f"https://ssm.{region}.amazonaws.com"
                    ]

                    process = subprocess.Popen(cmd, stdout=subprocess.DEVNULL,
stderr=subprocess.DEVNULL)
                    pids.append(process.pid)
                    print(f"SSM Port Forwarding Session started for port {port}")

                  print(f"openjd_env: BYOL_SSM_PIDS={','.join(str(pid) for pid in
pids)}")
```

3. AWS CloudFormation テンプレートをデプロイするときは、次のパラメータを指定します。

- LicenseInstanceID をライセンスサーバーまたはプロキシインスタンスの Amazon EC2 インスタンス ID で更新する
- LicensePorts をライセンスサーバーまたはプロキシインスタンスに転送するポートのカンマ区切りリストで更新する (2700,2701 など)

4. テンプレートをデプロイして、独自のライセンス機能を使用してファームをセットアップします。

カスタマーマネージドフリートをライセンスエンドポイントに接続する

AWS Deadline Cloud 使用状況ベースのライセンスサーバーは、一部のサードパーティー製品のオンデマンドライセンスを提供します。使用量ベースのライセンスでは、いつでも料金を支払うことができます。使用した時間に対してのみ課金されます。使用状況ベースのライセンスは、Deadline Cloud ワーカーがレンダリングするライセンスを提供しますが、DCC アプリケーションのライセンスは提供しません。

Deadline Cloud の使用状況ベースのライセンスサーバーは、Deadline Cloud ワーカーがライセンスサーバーと通信できる限り、任意のフリートタイプで使用できます。これは、サービスマネージドフリートで自動的に設定されます。この設定は、カスタマーマネージドフリートでのみ必要です。

ライセンスサーバーを作成するには、サードパーティーライセンスのトラフィックを許可するファームの VPC のセキュリティグループが必要です。

トピック

- ステップ 1: セキュリティグループを作成する
- ステップ 2: ライセンスエンドポイントを設定する
- ステップ 3: レンダリングアプリケーションをエンドポイントに接続する
- ステップ 4: ライセンスエンドポイントを削除する

ステップ 1: セキュリティグループを作成する

Amazon VPC コンソールを使用して、ファームの VPC のセキュリティグループを作成します。次のインバウンドルールを許可するようにセキュリティグループを設定します。

- Autodesk Maya と Arnold – 2701 - 2702、TCP、IPv4, IPv6

- Autodesk 3ds Max – 2704、TCP、IPv4, IPv6
- Cinema 4D – 7057、TCP、IPv4, IPv6
- KeyShot – 2703、TCP、IPv4, IPv6
- Foundry Nuke – 6101、TCP、IPv4, IPv6
- Redshift – 7054、TCP、IPv4, IPv6
- SideFX Houdini、Mantra、および Karma – 1715 - 1717、TCP、IPv4, IPv6

各インバウンドルールのソースは、フリートのワーカーセキュリティグループです。

セキュリティグループの作成の詳細については、Amazon Virtual Private Cloud ユーザーガイド」の「セキュリティグループの作成」を参照してください。

ステップ 2: ライセンスエンドポイントを設定する

ライセンスエンドポイントは、サードパーティー製品のライセンスサーバーへのアクセスを提供します。ライセンスリクエストはライセンスエンドポイントに送信されます。エンドポイントは、それらを適切なライセンスサーバーにルーティングします。ライセンスサーバーは、使用制限と使用権限を追跡します。Deadline Cloud でライセンスエンドポイントを作成すると、VPC に AWS PrivateLink インターフェイスエンドポイントがプロビジョニングされます。これらのエンドポイントは、標準 AWS PrivateLink 料金に従って請求されます。詳細については、AWS PrivateLink 料金表を参照してください。

適切なアクセス許可があれば、ライセンスエンドポイントを作成できます。ライセンスエンドポイントの作成に必要なポリシーについては、「ライセンスエンドポイントの作成を許可するポリシー」を参照してください。

Deadline Cloud コンソールのダッシュボードからライセンスエンドポイントを作成できます。

1. 左側のナビゲーションペインで、ライセンスエンドポイントを選択し、ライセンスエンドポイントの作成を選択します。
2. ライセンスエンドポイントの作成ページから、次の手順を実行します。
   - [VPC] を選択します。
   - Deadline Cloud ワーカーを含むサブネットを選択します。最大 10 個のサブネットを選択できます。
   - ステップ 1 で作成したセキュリティグループを選択します。より複雑なシナリオでは、最大 10 個のセキュリティグループを選択できます。

- (オプション) 新しいタグを追加を選択し、1 つ以上のタグを追加します。最大 50 個のタグを追加できます。

3. ライセンスエンドポイントの作成 を選択します。ライセンスエンドポイントが作成されると、ライセンスエンドポイントページに表示されます。
4. 計測済み製品セクションで、製品の追加を選択し、ライセンスエンドポイントに追加する製品を選択します。[Add] (追加) を選択します。

ライセンスエンドポイントから製品を削除するには、計測済み製品セクションで製品を選択し、削除を選択します。確認で、もう一度削除を選択します。

ステップ 3: レンダリングアプリケーションをエンドポイントに接続する

ライセンスエンドポイントを設定すると、アプリケーションはサードパーティーのライセンスサーバーを使用するのと同じ方法でそれを使用します。通常、アプリケーションのライセンスサーバーを設定するには、環境変数または Microsoft Windows レジストリキーなどの他のシステム設定をライセンスサーバーのポートとアドレスに設定します。

ライセンスエンドポイントの DNS 名を取得するには、コンソールでライセンスエンドポイントを選択し、DNS 名セクションでコピーアイコンを選択します。

設定例

Example – Autodesk Maya と Arnold

環境変数`ADSKFLEX_LICENSE_FILE`を次のように設定します。

```
2702@VPC_Endpoint_DNS_Name:2701@VPC_Endpoint_DNS_Name
```

> **Note**
> Windows ワーカーの場合は、コロン (:) の代わりにセミコロン (;) を使用してエンドポイントを区切ります。

Example – Autodesk 3ds Max

環境変数`ADSKFLEX_LICENSE_FILE`を次のように設定します。

```
2704@VPC_Endpoint_DNS_Name
```

Example – シネマ 4D

環境変数`g_licenseServerRLM`を次のように設定します。

```
VPC_Endpoint_DNS_Name:7057
```

環境変数を作成したら、次のようなコマンドラインを使用してイメージをレンダリングできます。

```
"C:\Program Files\Maxon Cinema 4D 2025\Commandline.exe" -render ^
    "C:\Users\User\MyC4DFileWithRedshift.c4d" -frame 0 ^
    -oimage "C:\Users\Administrator\User\MyOutputImage.png
```

Example – KeyShot

環境変数`LUXION_LICENSE_FILE`を次のように設定します。

```
2703@VPC_Endpoint_DNS_Name
```

インストールKeyShotして実行したら、次のコマンドを使用してライセンスが動作していることをテスト`pip install deadline-cloud-for-keyshot`できます。スクリプトは設定を検証しますが、何もレンダリングしません。

```
"C:\Program Files\KeyShot12\bin\keyshot_headless.exe" ^
    -floating_feature keyshot2 ^
    -floating_license_server 2703@VPC_Endpoint_DNS_Name ^
    -script "C:\Program Files\Python311\Lib\site-packages\deadline\keyshot_adaptor
\KeyShotClient\keyshot_handler.py"
```

レスポンスには、エラーメッセージなしで以下を含める必要があります。

```
Connecting to floating license server
```

Example – Foundry Nuke

環境変数`foundry_LICENSE`を次のように設定します。

```
6101@VPC_Endpoint_DNS_Name
```

ライセンスが正常に動作していることをテストするには、ターミナルで Nuke を実行します。

```
~/nuke/Nuke14.0v5/Nuke14.0 -x
```

Example – Redshift

環境変数`redshift_LICENSE`を次のように設定します。

```
7054@VPC_Endpoint_DNS_Name
```

環境変数を作成したら、次のようなコマンドラインを使用してイメージをレンダリングできます。

```
C:\ProgramData\redshift\bin\redshiftCmdLine.exe ^
    C:\demo\proxy\RS_Proxy_Demo.rs ^
    -oip C:\demo\proxy\images
```

Example – SideFX Houdini、Mantra、および Karma

次のコマンドを実行してください。

```
/opt/hfs19.5.640/bin/hserver -S
 "http://VPC_Endpoint_DNS_Name:1715;http://VPC_Endpoint_DNS_Name:1716;http://
VPC_Endpoint_DNS_Name:1717;"
```

ライセンスが正常に機能していることをテストするには、次のコマンドを使用して Houdini シーンをレンダリングします。

```
/opt/hfs19.5.640/bin/hython ~/forpentest.hip -c "hou.node('/out/mantra1').render()"
```

ステップ 4: ライセンスエンドポイントを削除する

カスタマーマネージドフリートを削除するときは、ライセンスエンドポイントを削除してください。ライセンスエンドポイントを削除しない場合、固定コストが AWS PrivateLink 引き続き課金されます。

Deadline Cloud コンソールのダッシュボードからライセンスエンドポイントを削除できます。

1. 左側のナビゲーションペインから、ライセンスエンドポイントを選択します。
2. 削除するエンドポイントを選択し、削除を選択し、もう一度削除を選択して確認します。

AWS Deadline Cloud のモニタリング

モニタリングは、 AWS Deadline Cloud (Deadline Cloud) と AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。マルチポイント障害が発生した場合は、その障害をより簡単にデバッグできるように、 AWS ソリューションのすべての部分からモニタリングデータを収集します。Deadline Cloud のモニタリングを開始する前に、以下の質問に対する回答を含むモニタリング計画を作成する必要があります。

- どのような目的でモニタリングしますか?
- どのリソースをモニタリングしますか?
- どのくらいの頻度でこれらのリソースをモニタリングしますか?
- どのモニタリングツールを使用しますか?
- 誰がモニタリングタスクを実行しますか?
- 問題が発生したときに誰が通知を受け取りますか?

AWS および Deadline Cloud には、リソースをモニタリングし、潜在的なインシデントに対応するために使用できるツールが用意されています。これらのツールの中には、モニタリングを行うものもあれば、手動による介入を必要とするものもあります。モニタリングタスクはできるだけ自動化する必要があります。

- Amazon CloudWatch は、 AWS リソースと で実行されるアプリケーションを AWS リアルタイムでモニタリングします。メトリクスの収集と追跡、カスタマイズしたダッシュボードの作成、および指定したメトリクスが指定したしきい値に達したときに通知またはアクションを実行するアラームの設定を行うことができます。例えば、CloudWatch で Amazon EC2 インスタンスの CPU 使用率などのメトリクスを追跡し、必要に応じて新しいインスタンスを自動的に起動できます。詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。

  Deadline Cloud には 3 つの CloudWatch メトリクスがあります。
- Amazon CloudWatch Logs では、Amazon EC2 インスタンス、CloudTrail、およびその他のソースからのログファイルをモニタリング、保存、およびアクセスできます。CloudWatch Logs は、ログファイル内の情報をモニタリングし、特定のしきい値が満たされたときに通知します。高い耐久性を備えたストレージにログデータをアーカイブすることも可能です。詳細については、「Amazon CloudWatch Logs ユーザーガイド」を参照してください。
- Amazon EventBridge を使用して AWS サービスを自動化し、アプリケーションの可用性の問題やリソースの変更などのシステムイベントに自動的に対応できます。 AWS サービスからのイベント

は、ほぼリアルタイムで EventBridge に配信されます。簡単なルールを記述して、注目するイベントと、イベントがルールに一致した場合に自動的に実行するアクションを指定できます。詳細については、「Amazon EventBridge ユーザーガイド」を参照してください。

- AWS CloudTrail は、 AWS アカウントによって、またはアカウントに代わって行われた API コールおよび関連イベントをキャプチャし、指定した Amazon S3 バケットにログファイルを配信します。が呼び出したユーザーとアカウント AWS、呼び出し元のソース IP アドレス、呼び出しの発生日時を特定できます。詳細については、「AWS CloudTrail ユーザーガイド」を参照してください。

トピック

- を使用した Deadline Cloud API コールのログ記録 AWS CloudTrail
- CloudWatch によるモニタリング
- を使用した Deadline Cloud イベントの管理 Amazon EventBridge

を使用した Deadline Cloud API コールのログ記録 AWS CloudTrail

Deadline Cloud は、ユーザーAWS CloudTrail、ロール、または 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 と統合されています AWS のサービス。CloudTrail は、 のすべての API コールをイベント Deadline Cloud 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、Deadline Cloud コンソールからの呼び出しと Deadline Cloud API オペレーションへのコード呼び出しが含まれます。CloudTrail によって収集された情報を使用して、リクエストの実行元の IP アドレス Deadline Cloud、リクエストの実行日時などの詳細を確認できます。

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。アイデンティティ情報は、以下を判別するのに役立ちます。

- ルートユーザーまたはユーザー認証情報のどちらを使用してリクエストが送信されたか。
- リクエストが IAM Identity Center ユーザーに代わって行われたかどうか。
- リクエストがロールまたはフェデレーションユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。
- リクエストが、別の AWS のサービスによって送信されたかどうか。

アカウントを作成するとCloudTrail AWS アカウント は でアクティブになり、CloudTrail イベント履歴に自動的にアクセスできます。CloudTrail の [イベント履歴] では、 AWS リージョンで過去 90 日間に記録された 管理イベントの表示、検索、およびダウンロードが可能で、変更不可能な記録を確

認できます。詳細については、「AWS CloudTrail ユーザーガイド」の「CloudTrail イベント履歴の使用」を参照してください。[イベント履歴] の閲覧には CloudTrail の料金はかかりません。

AWS アカウント 過去 90 日間のイベントの継続的な記録については、証跡または CloudTrail Lake イベントデータストアを作成します。

CloudTrail 証跡

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。を使用して作成されたすべての証跡 AWS Management Console はマルチリージョンです。 AWS CLIを使用する際は、単一リージョンまたは複数リージョンの証跡を作成できます。アカウント AWS リージョン 内のすべての でアクティビティをキャプチャするため、マルチリージョン証跡を作成することをお勧めします。単一リージョンの証跡を作成する場合、証跡の AWS リージョンに記録されたイベントのみを表示できます。証跡の詳細については、「AWS CloudTrail ユーザーガイド」の「AWS アカウントの証跡の作成」および「組織の証跡の作成」を参照してください。

証跡を作成すると、進行中の管理イベントのコピーを 1 つ無料で CloudTrail から Amazon S3 バケットに配信できますが、Amazon S3 ストレージには料金がかかります。CloudTrail の料金の詳細については、「AWS CloudTrail の料金」を参照してください。Amazon S3 の料金に関する詳細については、「Amazon S3 の料金」を参照してください。

CloudTrail Lake イベントデータストア

[CloudTrail Lake] を使用すると、イベントに対して SQL ベースのクエリを実行できます。CloudTrail Lake は、行ベースの JSON 形式の既存のイベントを Apache ORC 形式に変換します。ORC は、データを高速に取得するために最適化された単票ストレージ形式です。イベントは、イベントデータストアに集約されます。イベントデータストアは、高度なイベントセレクタを適用することによって選択する条件に基づいた、イベントのイミュータブルなコレクションです。どのイベントが存続し、クエリに使用できるかは、イベントデータストアに適用するセレクタが制御します。CloudTrail Lake の詳細については、 AWS CloudTrail ユーザーガイドのAWS CloudTrail 「Lake の使用」を参照してください。

CloudTrail Lake のイベントデータストアとクエリにはコストがかかります。イベントデータストアを作成する際に、イベントデータストアに使用する料金オプションを選択します。料金オプションによって、イベントの取り込みと保存にかかる料金、および、そのイベントデータストアのデフォルトと最長の保持期間が決まります。CloudTrail の料金の詳細については、「AWS CloudTrail の料金」を参照してください。

Deadline Cloud CloudTrail のデータイベント

データイベントでは、リソース上またはリソース内で実行されるリソースオペレーション (Amazon S3 オブジェクトの読み取りまたは書き込みなど) についての情報が得られます。これらのイベントは、データプレーンオペレーションとも呼ばれます。データイベントは、多くの場合、高ボリュームのアクティビティです。デフォルトでは、CloudTrail はデータイベントをログ記録しません。CloudTrail [イベント履歴] にはデータイベントは記録されません。

追加の変更がイベントデータに適用されます。CloudTrail の料金の詳細については、「AWS CloudTrail の料金」を参照してください。

CloudTrail コンソール、または CloudTrail API オペレーションを使用して AWS CLI、 Deadline Cloud リソースタイプのデータイベントを記録できます。データイベントをログに記録する方法の詳細については、「AWS CloudTrail ユーザーガイド」の「AWS Management Consoleを使用したデータイベントのログ記録」および「AWS Command Line Interfaceを使用したデータイベントのログ記録」を参照してください。

次の表に、データイベントを記録できる Deadline Cloud リソースタイプを示します。データイベントタイプ (コンソール) 列には、CloudTrail コンソールの[データイベントタイプ]リストから選択する値が表示されます。resources.type 値列には、 AWS CLI または CloudTrail APIs を使用して高度なイベントセレクタを設定するときに指定する `resources.type`値が表示されます。CloudTrail に記録されたデータ API 列には、リソース タイプの CloudTrail にログ記録された API コールが表示されます。

データイベントタイプ (コンソール)	resources.type 値	CloudTrail にログ記録されたデータ API
期限フリート	`AWS::Deadline::Fleet`	• SearchWorkers
期限キュー	`AWS::Deadline::Fleet`	• SearchJobs
期限ワーカー	`AWS::Deadline::Worker`	• GetWorker • ListSessionsForWorker • UpdateWorkerSchedule • BatchGetJobEntity • ListWorkers
期限ジョブ	`AWS::Deadline::Job`	• ListStepConsumers

データイベントタイプ (コンソール)	resources.type 値	CloudTrail にログ記録されたデータ API
		• UpdateTask • ListJobs • GetStep • ListSteps • GetJob • GetTask • GetSession • ListSessions • CreateJob • ListSessionActions • ListTasks • CopyJobTemplate • UpdateSession • UpdateStep • UpdateJob • ListJobParameterDefinitions • GetSessionAction • ListStepDependencies • SearchTasks • SearchSteps

`eventName`、`readOnly`、および `resources.ARN` フィールドでフィルタリングして、自分にとって重要なイベントのみをログに記録するように高度なイベントセレクタを設定できます。オブジェクトの詳細については、「AWS CloudTrail API リファレンス」の「AdvancedFieldSelector」を参照してください。

Deadline Cloud CloudTrail での 管理イベント

管理イベントは、 のリソースで実行される管理オペレーションに関する情報を提供します AWS アカウント。これらのイベントは、コントロールプレーンオペレーションとも呼ばれます。CloudTrail は、デフォルトで管理イベントをログ記録します。

AWS Deadline Cloud は、次の Deadline Cloud コントロールプレーンオペレーションを管理イベントとして CloudTrail に記録します。

- associate-member-to-farm
- associate-member-to-fleet
- associate-member-to-job
- associate-member-to-queue
- assume-fleet-role-for-read
- assume-fleet-role-for-worker
- assume-queue-role-for-read
- assume-queue-role-for-user
- assume-queue-role-for-worker
- 予算の作成
- ファームの作成
- create-fleet
- create-license-endpoint
- 作成制限
- モニターの作成
- キューの作成
- create-queue-environment
- create-queue-fleet-association
- create-queue-limit-association
- create-storage-profile
- ワーカーの作成
- 予算の削除

- delete-farm
- delete-fleet
- delete-license-endpoint
- 削除制限
- delete-metered-product
- delete-monitor
- delete-queue
- delete-queue-environment
- delete-queue-fleet-association
- delete-queue-limit-association
- delete-storage-profile
- delete-worker
- disassociate-member-from-farm
- disassociate-member-from-fleet
- disassociate-member-from-job
- disassociate-member-from-queue
- get-application-version
- 予算の取得
- get-farm
- get-feature-map
- フリートの取得
- get-license-endpoint
- get-limit
- get-monitor
- get-queue
- get-queue-environment
- get-queue-fleet-association
- get-queue-limit-association
- get-sessions-statistics-aggregation

- get-storage-profile
- get-storage-profile-for-queue
- list-available-metered-products
- リスト予算
- list-farm-members
- list-farms
- list-fleet-members
- フリートのリスト
- list-job-members
- list-license-endpoints
- リスト制限
- list-metered-products
- モニターのリスト
- list-queue-environments
- list-queue-fleet-associations
- list-queue-limit-associations
- list-queue-members
- list-queues
- list-storage-profiles
- list-storage-profiles-for-queue
- list-tags-for-resource
- put-metered-product
- start-sessions-statistics-aggregation
- タグリソース
- タグなしリソース
- アップデート予算
- update-farm
- 更新フリート
- 更新制限

- モニターの更新
- 更新キュー
- update-queue-environment
- update-queue-fleet-association
- update-queue-limit-association
- update-storage-profile
- ワーカーの更新

Deadline Cloud イベントの例

各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 API オペレーション、オペレーションの日時、リクエストパラメータなどに関する情報を含みます。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、イベントは特定の順序で表示されません。

次の例は、`CreateFarm` オペレーションを示す CloudTrail イベントを示しています。

```
{
    "eventVersion": "0",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE-PrincipalID:EXAMPLE-Session",
        "arn": "arn:aws:sts::111122223333:assumed-role/EXAMPLE-UserName/EXAMPLE-
Session",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE-accessKeyId",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE-PrincipalID",
                "arn": "arn:aws:iam::111122223333:role/EXAMPLE-UserName",
                "accountId": "111122223333",
                "userName": "EXAMPLE-UserName"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2021-03-08T23:25:49Z"
            }
```

```
        }
    },
    "eventTime": "2021-03-08T23:25:49Z",
    "eventSource": "deadline.amazonaws.com",
    "eventName": "CreateFarm",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "EXAMPLE-userAgent",
    "requestParameters": {
        "displayName": "example-farm",
        "kmsKeyArn": "arn:aws:kms:us-west-2:111122223333:key/111122223333",
        "X-Amz-Client-Token": "12abc12a-1234-1abc-123a-1a11bc1111a",
        "description": "example-description",
        "tags": {
            "purpose_1": "e2e"
            "purpose_2": "tag_test"
        }
    },
    "responseElements": {
        "farmId": "EXAMPLE-farmID"
    },
    "requestID": "EXAMPLE-requestID",
    "eventID": "EXAMPLE-eventID",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333"
    "eventCategory": "Management",
}
```

JSON の例は AWS リージョン、、IP アドレス、およびイベントを識別するのに役立つ`requestParameters`「」や`displayName`「」などの他の`kmsKeyArn`「」を示しています。

CloudTrail レコードの内容については、「AWS CloudTrail ユーザーガイド」の「CloudTrail record contents」を参照してください。

CloudWatch によるモニタリング

Amazon CloudWatch (CloudWatch) は raw データを収集し、ほぼリアルタイムの読み取り可能なメトリクスに加工します。https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開き、Deadline Cloud メトリクスを表示およびフィルタリングできます。

これらの統計は 15 か月間保持されるため、履歴情報にアクセスして、ウェブアプリケーションまたはサービスのパフォーマンスをより的確に把握できます。また、特定のしきい値を監視するアラームを設定し、これらのしきい値に達したときに通知を送信したりアクションを実行したりできます。詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。

Deadline Cloud には、タスクログとワーカーログの 2 種類のログがあります。タスクログは、スクリプトまたは DCC の実行時に実行ログを実行する場合です。タスクログには、アセットのロード、タイルのレンダリング、テクスチャが見つからないなどのイベントが表示される場合があります。

ワーカーログには、ワーカーエージェントのプロセスが表示されます。これには、ワーカーエージェントの起動、自己登録、進行状況の報告、設定のロード、タスクの完了などが含まれます。

これらのログの名前空間は です`/aws/deadline/*`。

Deadline Cloud の場合、ワーカーはこれらのログを CloudWatch Logs にアップロードします。デフォルトでは、ログは期限切れになりません。ジョブが大量のデータを出力する場合、追加のコストが発生する可能性があります。詳細については、「Amazon CloudWatch の料金」を参照してください。

各ロググループの保持ポリシーを調整できます。保持期間を短くすると、古いログが削除され、ストレージコストを削減できます。ログを保持するには、ログを削除する前に Amazon Simple Storage Service にアーカイブします。詳細については、「Amazon CloudWatch ユーザーガイド」の「コンソールを使用してログデータを Amazon S3 にエクスポートする」を参照してください。 Amazon CloudWatch

> **Note**
>
> CloudWatch ログの読み取りは によって制限されます AWS。多くのアーティストをオンボーディングする場合は、 AWS カスタマーサポートに連絡し、CloudWatch `GetLogEvents` でクォータの引き上げをリクエストすることをお勧めします。さらに、デバッグしていない場合は、ログテーリングポータルを閉じることをお勧めします。

詳細については、「Amazon CloudWatch ユーザーガイド」の「CloudWatch Logs クォータ」を参照してください。 Amazon CloudWatch

CloudWatch メトリクス

Deadline Cloud は Amazon CloudWatch にメトリクスを送信します。Deadline Cloud が CloudWatch に送信するメトリクスを一覧表示するには AWS Management Console、 AWS CLI、、または API

を使用できます。デフォルトでは、各データポイントはアクティビティの開始時刻に続く 1 分をカバーします。 AWS Management Console または を使用して使用可能なメトリクスを表示する方法については AWS CLI、Amazon CloudWatch ユーザーガイド」の「使用可能なメトリクスを表示する」を参照してください。

カスタマーマネージドフリートメトリクス

`AWS/DeadlineCloud` 名前空間には、カスタマーマネージドフリートの次のメトリクスが含まれます。

メトリクス	説明	[単位]
`RecommendedFleetSize`	Deadline Cloud がジョブの処理に使用することを推奨するワーカーの数。このメトリクスを使用して、フリートのワーカー数を拡張または縮小できます。	カウント
`UnhealthyWorkerCount`	正常でないジョブの処理に割り当てられたワーカーの数。	カウント

次のディメンションを使用して、カスタマーマネージドフリートメトリクスを絞り込むことができます。

ディメンション	説明
FarmId	このディメンションは、指定したファームにリクエストしたデータをフィルタリングします。
FleetId	このディメンションは、指定したワーカーフリートにリクエストしたデータをフィルタリングします。

リソース制限メトリクス

`AWS/DeadlineCloud` 名前空間には、リソース制限に関する以下のメトリクスが含まれています。

メトリクス	説明	[単位]
`CurrentCount`	使用中のこの制限によってモデル化されたリソースの数。	カウント
`MaxCount`	この制限によってモデル化されたリソースの最大数。API を使用して`maxCount`値を -1 に設定すると、Deadline Cloud は`MaxCount`メトリクスを出力しません。	カウント

次のディメンションを使用して、同時制限メトリクスを絞り込むことができます。

ディメンション	説明
FarmId	このディメンションは、指定したファームにリクエストしたデータをフィルタリングします。
LimitId	このディメンションは、指定した制限までリクエストしたデータをフィルタリングします。

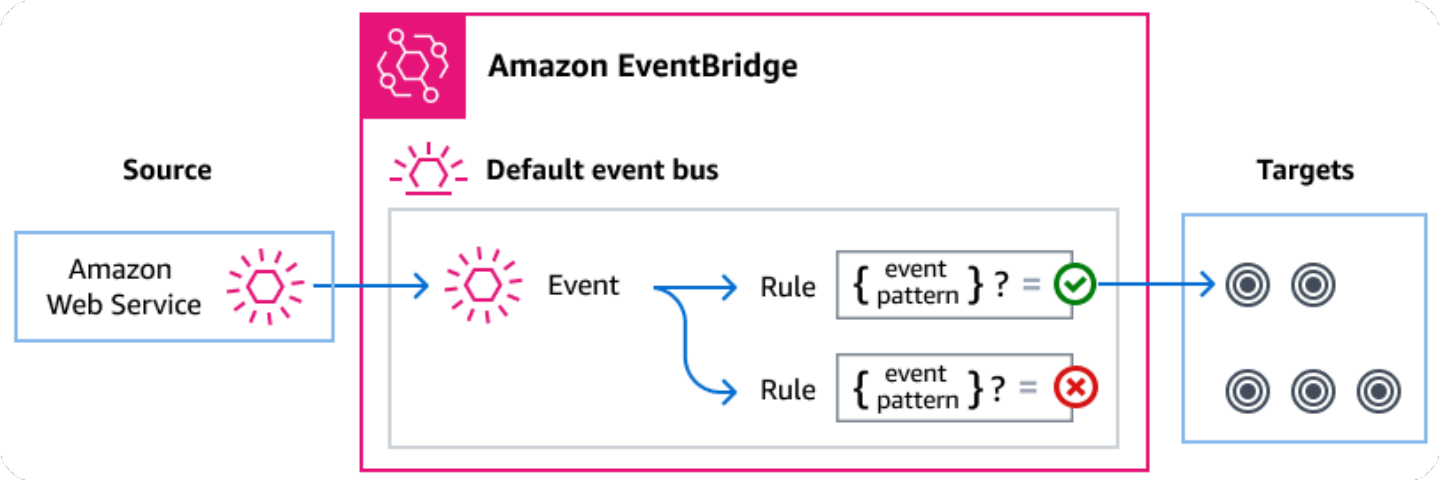
を使用した Deadline Cloud イベントの管理 Amazon EventBridge

Amazon EventBridge は、イベントを使用してアプリケーションコンポーネントを接続するサーバーレスサービスであり、スケーラブルなイベント駆動型アプリケーションを簡単に構築できます。イベント駆動型アーキテクチャとは、イベントの発信と応答によって連携する、疎結合のソフトウェアシステムを構築するスタイルです。イベントとは、リソースまたは環境で発生した変更を指します。

処理の流れ

多くの AWS サービスと同様に、Deadline Cloud はイベントを生成し、 EventBridge デフォルトのイベントバスに送信します。（デフォルトのイベントバスはすべての AWS アカウントで自動的にプロビジョニングされます）。イベントバスは、イベントを受信するルーターであり、ゼロ個以上の送信先やターゲットに配信します。イベントが受信されると、ユーザーがイベントバスに対して指定したルールによって評価されます。各ルールは、イベントがルールのイベントパターンに一致するかどう

かをチェックします。一致する場合、イベントバスはそのイベントを指定されたターゲットに送信します。



トピック

- Deadline Cloud イベント
- EventBridge ルールを使用した Deadline Cloud イベントの配信
- Deadline Cloud イベントの詳細リファレンス

Deadline Cloud イベント

Deadline Cloud は、次のイベントをデフォルトの EventBridge イベントバスに自動的に送信します。ルールのイベントパターンに一致するイベントは、ベストエフォートベースで指定されたターゲットに配信されます。イベントは順不同で配信される可能性があります。

詳細については、「Amazon EventBridge ユーザーガイド」の「EventBridge イベント」を参照してください。

イベントの詳細のタイプ	説明
予算しきい値に達しました	キューが割り当てられた予算の割合に達すると送信されます。
ジョブのライフサイクルステータスの変更	ジョブのライフサイクルステータスが変更されたときに送信されます。
ジョブ実行ステータスの変更	ジョブ内のタスクの全体的なステータスが変更されたときに送信されます。

イベントの詳細のタイプ	説明
ステップライフサイクルステータスの変更	ジョブ内のステップのライフサイクルステータスが変更されたときに送信されます。
ステップ実行ステータスの変更	ステップ内のタスクの全体的なステータスが変更されたときに送信されます。
タスク実行ステータスの変更	タスクのステータスが変更されたときに送信されます。

EventBridge ルールを使用した Deadline Cloud イベントの配信

EventBridge デフォルトのイベントバスで Deadline Cloud イベントをターゲットに送信するには、ルールを作成する必要があります。各ルールには、イベントバスで受信した各イベント EventBridge と一致するイベントパターンが含まれています。イベントデータが指定されたイベントパターンと一致する場合、 はそのイベントをルールのターゲット (複数可) に EventBridge 配信します。

イベントバスルールの詳細な作成方法については、「EventBridge ユーザーガイド」の「イベントに反応する Amazon EventBridge ルールの作成」を参照してください。

Deadline Cloud イベントに一致するイベントパターンの作成

各イベントパターンは JSON 形式のオブジェクトで、以下が含まれています。

- イベントを送信するサービスを識別する `source` 属性。Deadline Cloud イベントの場合、ソースは ですaws.deadline。
- (オプション): 照合するイベントタイプの配列を含む `detail-type` 属性。
- (オプション): 照合対象となるその他のイベントデータを含む `detail` 属性。

たとえば、次のイベントパターンは、Deadline Cloud farmIdに指定された のすべてのフリートサイズレコメンデーション変更イベントと一致します。

```
{
  "source": ["aws.deadline"],
  "detail-type": ["Fleet Size Recommendation Change"],
  "detail": {
     "farmId": "farm-12345678900000000000000000000000"
  }
```

```
}
```

詳細については、「EventBridge ユーザーガイド」の「Amazon EventBridge のイベントパターン」を参照してください。

Deadline Cloud イベントの詳細リファレンス

AWS サービスからのすべてのイベントには、イベントのソースである AWS サービス、イベントが生成された時刻、イベントが発生したアカウントとリージョンなど、イベントに関するメタデータを含む共通のフィールドセットがあります。これらの一般的なフィールドの定義については、「Amazon EventBridge ユーザーガイド」の「イベント構造リファレンス」を参照してください。

さらに、各イベントには、その特定のイベントに固有のデータを含む `detail` フィールドがあります。以下のリファレンスでは、さまざまな Deadline Cloud イベントの詳細フィールドを定義します。

EventBridge を使用して Deadline Cloud イベントを選択および管理する場合、次の点に注意してください。

- Deadline Cloud からのすべてのイベントの `source`フィールドは に設定されます`aws.deadline`。
- `detail-type` フィールドはイベントタイプを指定します。

  例えば、`Fleet Size Recommendation Change`。
- `detail` フィールドには、その特定のイベントに固有のデータが含まれます。

ルールが Deadline Cloud イベントと一致できるようにするイベントパターンの構築については、「Amazon EventBridge ユーザーガイド」の「イベントパターン」を参照してください。

イベントとその EventBridge 処理方法の詳細については、Amazon EventBridge 「 ユーザーガイド」のAmazon EventBridge 「 イベント」を参照してください。

トピック

- Budget Threshold Reached イベント
- フリートサイズのレコメンデーション変更イベント
- ジョブライフサイクルステータス変更イベント
- ジョブ実行ステータス変更イベント
- ステップライフサイクルステータス変更イベント

- ステップ実行ステータス変更イベント
- タスク実行ステータス変更イベント

Budget Threshold Reached イベント

Budget Threshold Reached イベントを使用して、使用された予算の割合をモニタリングできます。Deadline Cloud は、使用された割合が次のしきい値を超えたときにイベントを送信します。

- 10、20、30、40、50、60、70、75、80、85、90、95、96、97、98、99、100

Deadline Cloud が Budget Threshold Reached イベントを送信する頻度は、予算が制限に近づくにつれて増加します。これにより、制限に近づいた予算を注意深くモニタリングし、支出が過剰にならないように対策を講じることができます。独自の予算しきい値を設定することもできます。Deadline Cloud は、使用量がカスタムしきい値を超えるとイベントを送信します。

予算の量を変更すると、Deadline Cloud が次回 Budget Threshold Reached イベントを送信したときに、使用された予算の現在の割合に基づきます。たとえば、上限に達した 100 USD の予算に 50 USD を追加すると、次の Budget Threshold Reached イベントは、予算が 75% であることを示します。

以下は、`Budget Threshold Reached` イベントの詳細フィールドです。

Deadline Cloud イベントの特定の値が含まれているため、 `source` および `detail-type`フィールドは以下に含まれています。すべてのイベントに含まれる他のメタデータフィールドの定義については、Amazon EventBridge 「 ユーザーガイド」の「イベント構造のリファレンス」を参照してください。

```
{
    "version": "0",
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "detail-type": "Budget Threshold Reached",
    "source": "aws.deadline",
    "account": "111122223333",
    "time": "2017-12-22T18:43:48Z",
    "region": "aa-example-1",
    "resources": [],
    "detail": {
        "farmId": "farm-1234567890000000000000000000000",
        "budgetId": "budget-1234567890000000000000000000000",
        "thresholdInPercent": 0
```

```
    }
}
```

`detail-type`

イベントのタイプを示します。

このイベントの場合、この値は `Budget Threshold Reached` です。

`source`

イベントを発生させたサービスを識別します。Deadline Cloud イベントの場合、この値は ですaws.deadline。

`detail`

イベントに関する情報を含む JSON オブジェクト。このフィールドの内容は、イベントを生成するサービスによって決まります。

このイベントの場合、このデータには以下が含まれます。

`farmId`

ジョブを含むファームの識別子。

`budgetId`

しきい値に達した予算の識別子。

`thresholdInPercent`

使用された予算の割合。

フリートサイズのレコメンデーション変更イベント

イベントベースの自動スケーリングを使用するようにフリートを設定すると、Deadline Cloud はフリートの管理に使用できるイベントを送信します。これらの各イベントには、フリートの現在のサイズとリクエストされたサイズに関する情報が含まれています。EventBridge イベントと Lambda 関数の例を使用してイベントを処理する例については、「Deadline Cloud スケールレコメンデーション機能を使用して Amazon EC2 フリートを自動スケーリングする」を参照してください。

フリートサイズのレコメンデーション変更イベントは、以下が発生すると送信されます。

- 推奨されるフリートサイズが変更され、oldFleetSize が newFleetSize と異なる場合。

- 実際のフリートサイズが推奨フリートサイズと一致しないことをサービスが検出した場合。実際のフリートサイズは、GetFleet オペレーションレスポンスの workerCount から取得できます。GetFleet これは、アクティブな Amazon EC2 インスタンスが Deadline Cloud ワーカーとして登録できない場合に発生する可能性があります。

以下は、`Fleet Size Recommendation Change` イベントの詳細フィールドです。

Deadline Cloud イベントの特定の値が含まれているため、 `source` および `detail-type`フィールドは以下に含まれています。すべてのイベントに含まれる他のメタデータフィールドの定義については、Amazon EventBridge 「 ユーザーガイド」の「イベント構造のリファレンス」を参照してください。

```
{
    "version": "0",
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "detail-type": "Fleet Size Recommendation Change",
    "source": "aws.deadline",
    "account": "111122223333",
    "time": "2017-12-22T18:43:48Z",
    "region": "aa-example-1",
    "resources": [],
    "detail": {
        "farmId": "farm-1234567890000000000000000000000",
        "fleetId": "fleet-1234567890000000000000000000000",
        "oldFleetSize": 1,
        "newFleetSize": 5,
    }
}
```

`detail-type`

イベントのタイプを示します。

このイベントの場合、この値は `Fleet Size Recommendation Change` です。

`source`

イベントを発生させたサービスを識別します。Deadline Cloud イベントの場合、この値は ですaws.deadline。

`detail`

イベントに関する情報を含む JSON オブジェクト。このフィールドの内容は、イベントを生成するサービスによって決まります。

このイベントの場合、このデータには以下が含まれます。

`farmId`

ジョブを含むファームの識別子。

`fleetId`

サイズ変更が必要なフリートの識別子。

`oldFleetSize`

フリートの現在のサイズ。

`newFleetSize`

フリートに推奨される新しいサイズ。

ジョブライフサイクルステータス変更イベント

ジョブを作成または更新すると、Deadline Cloud はライフサイクルステータスを設定して、ユーザーが最後に開始したアクションのステータスを表示します。

ジョブライフサイクルステータスの変更イベントは、ジョブの作成時など、ライフサイクルステータスの変更に対して送信されます。

以下は、`Job Lifecycle Status Change` イベントの詳細フィールドです。

Deadline Cloud イベントの特定の値が含まれているため、 `source` および `detail-type`フィールドは以下に含まれています。すべてのイベントに含まれる他のメタデータフィールドの定義については、「 Amazon EventBridge ユーザーガイド」の「イベント構造のリファレンス」を参照してください。

```
{
    "version": "0",
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "detail-type": "Job Lifecycle Status Change",
    "source": "aws.deadline",
    "account": "111122223333",
```

```
    "time": "2017-12-22T18:43:48Z",
    "region": "aa-example-1",
    "resources": [],
    "detail": {
        "farmId": "farm-12345678900000000000000000000000",
        "queueId": "queue-12345678900000000000000000000000",
        "jobId": "job-12345678900000000000000000000000",
        "previousLifecycleStatus": "UPDATE_IN_PROGRESS",
        "lifecycleStatus": "UPDATE_SUCCEEDED"
    }
}
```

`detail-type`

イベントのタイプを示します。

このイベントの場合、この値は `Job Lifecycle Status Change` です。

`source`

イベントを発生させたサービスを識別します。Deadline Cloud イベントの場合、この値は です`aws.deadline`。

`detail`

イベントに関する情報を含む JSON オブジェクト。このフィールドの内容は、イベントを生成するサービスによって決まります。

このイベントの場合、このデータには以下が含まれます。

`farmId`

ジョブを含むファームの識別子。

`queueId`

ジョブを含むキューの識別子。

`jobId`

ジョブの識別子。

`previousLifecycleStatus`

ジョブが離れるライフサイクル状態。このフィールドは、ジョブを初めて送信するときには含まれません。

`lifecycleStatus`

ジョブが入るライフサイクル状態。

ジョブ実行ステータス変更イベント

ジョブは多くのタスクで構成されます。各タスクのステータスは です。すべてのタスクのステータスが結合され、ジョブの全体的なステータスが表示されます。詳細については、「Deadline Cloud ユーザーガイド」の「Deadline Cloud のジョブの状態」を参照してください。 AWS

ジョブ実行ステータス変更イベントは、次の場合に送信されます。

- 結合された`taskRunStatus`フィールドが変更されます。
- ジョブが READY 状態にある場合を除き、ジョブは再キューに入れられます。

次の場合、ジョブ実行ステータス変更イベントは送信されません。

- ジョブが最初に作成されます。ジョブの作成をモニタリングするには、ジョブライフサイクルステータス変更イベントの変更をモニタリングします。
- ジョブの `taskRunStatusCounts`フィールドは変更されますが、結合されたジョブタスクの実行ステータスは変更されません。

以下は、`Job Run Status Change` イベントの詳細フィールドです。

Deadline Cloud イベントの特定の値が含まれているため、 `source` および `detail-type`フィールドは以下に含まれています。すべてのイベントに含まれる他のメタデータフィールドの定義については、「 Amazon EventBridge ユーザーガイド」の「イベント構造のリファレンス」を参照してください。

```
{
    "version": "0",
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "detail-type": "Job Run Status Change",
    "source": "aws.deadline",
    "account": "111122223333",
    "time": "2017-12-22T18:43:48Z",
    "region": "aa-example-1",
    "resources": [],
```

```
    "detail": {
        "farmId": "farm-12345678900000000000000000000000",
        "queueId": "queue-12345678900000000000000000000000",
        "jobId": "job-12345678900000000000000000000000",
        "previousTaskRunStatus": "RUNNING",
        "taskRunStatus": "SUCCEEDED",
        "taskRunStatusCounts": {
            "PENDING": 0,
            "READY": 0,
            "RUNNING": 0,
            "ASSIGNED": 0,
            "STARTING": 0,
            "SCHEDULED": 0,
            "INTERRUPTING": 0,
            "SUSPENDED": 0,
            "CANCELED": 0,
            "FAILED": 0,
            "SUCCEEDED": 20,
            "NOT_COMPATIBLE": 0
       }
    }
}
```

`detail-type`

イベントのタイプを示します。

このイベントの場合、この値は `Job Run Status Change` です。

`source`

イベントを発生させたサービスを識別します。Deadline Cloud イベントの場合、この値は です`aws.deadline`。

`detail`

イベントに関する情報を含む JSON オブジェクト。このフィールドの内容は、イベントを生成するサービスによって決まります。

このイベントの場合、このデータには以下が含まれます。

`farmId`

ジョブを含むファームの識別子。

`queueId`

ジョブを含むキューの識別子。

`jobId`

ジョブの識別子。

`previousTaskRunStatus`

ジョブが退出するタスク実行状態。

`taskRunStatus`

ジョブが入るタスク実行状態。

`taskRunStatusCounts`

各状態のジョブのタスクの数。

ステップライフサイクルステータス変更イベント

イベントを作成または更新すると、Deadline Cloud はジョブのライフサイクルステータスを設定し、ユーザーが最後に開始したアクションのステータスを記述します。

ステップライフサイクルステータス変更イベントは、次の場合に送信されます。

- ステップの更新が開始されます (UPDATE_IN_PROGRESS)。
- ステップの更新が正常に完了しました (UPDATE_SUCEEDED)。
- ステップの更新に失敗しました (UPDATE_FAILED)。

ステップが最初に作成されたときにイベントは送信されません。ステップの作成をモニタリングするには、ジョブライフサイクルステータス変更イベントの変更をモニタリングします。

以下は、`Step Lifecycle Status Change` イベントの詳細フィールドです。

Deadline Cloud イベントの特定の値が含まれているため、 フィールド`source`と `detail-type`フィールドは以下のとおりです。すべてのイベントに含まれる他のメタデータフィールドの定義については、Amazon EventBridge 「 ユーザーガイド」の「イベント構造のリファレンス」を参照してください。

```
{
    "version": "0",
```

```
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "detail-type": "Step Lifecycle Status Change",
    "source": "aws.deadline",
    "account": "111122223333",
    "time": "2017-12-22T18:43:48Z",
    "region": "aa-example-1",
    "resources": [],
    "detail": {
        "farmId": "farm-12345678900000000000000000000000",
        "queueId": "queue-12345678900000000000000000000000",
        "jobId": "job-12345678900000000000000000000000",
        "stepId": "step-12345678900000000000000000000000",
        "previousLifecycleStatus": "UPDATE_IN_PROGRESS",
        "lifecycleStatus": "UPDATE_SUCCEEDED"
    }
}
```

`detail-type`

イベントのタイプを示します。

このイベントの場合、この値は `Step Lifecycle Status Change` です。

`source`

イベントを発生させたサービスを識別します。Deadline Cloud イベントの場合、この値は です`aws.deadline`。

`detail`

イベントに関する情報を含む JSON オブジェクト。このフィールドの内容は、イベントを生成するサービスによって決まります。

このイベントの場合、このデータには以下が含まれます。

`farmId`

ジョブを含むファームの識別子。

`queueId`

ジョブを含むキューの識別子。

`jobId`

ジョブの識別子。

`stepId`

現在のジョブステップの識別子。

`previousLifecycleStatus`

ステップが離れるライフサイクル状態。

`lifecycleStatus`

ステップが入るライフサイクル状態。

ステップ実行ステータス変更イベント

ジョブの各ステップは、多くのタスクで構成されます。各タスクのステータスは です。タスクのステータスが結合され、ステップとジョブの全体的なステータスが表示されます。

ステップ実行ステータス変更イベントは、次の場合に送信されます。

- 組み合わせた `taskRunStatus`が変更されます。
- ステップが READY 状態にある場合を除き、ステップは再キューに入れられます。

以下の場合、イベントは送信されません。

- ステップが最初に作成されます。ステップの作成をモニタリングするには、ジョブライフサイクルステータス変更イベントの変更をモニタリングします。
- ステップは`taskRunStatusCounts`変更されますが、結合ステップタスクの実行ステータスは変更されません。

以下は、`Step Run Status Change` イベントの詳細フィールドです。

Deadline Cloud イベントの特定の値が含まれているため、 フィールド`source`と `detail-type`フィールドは以下のとおりです。すべてのイベントに含まれる他のメタデータフィールドの定義については、Amazon EventBridge 「 ユーザーガイド」の「イベント構造のリファレンス」を参照してください。

```
{
    "version": "0",
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "detail-type": "Step Run Status Change",
    "source": "aws.deadline",
```

```
    "account": "111122223333",
    "time": "2017-12-22T18:43:48Z",
    "region": "aa-example-1",
    "resources": [],
    "detail": {
        "farmId": "farm-12345678900000000000000000000000",
        "queueId": "queue-12345678900000000000000000000000",
        "jobId": "job-12345678900000000000000000000000",
        "stepId": "step-12345678900000000000000000000000",
        "previousTaskRunStatus": "RUNNING",
        "taskRunStatus": "SUCCEEDED",
        "taskRunStatusCounts": {
            "PENDING": 0,
            "READY": 0,
            "RUNNING": 0,
            "ASSIGNED": 0,
            "STARTING": 0,
            "SCHEDULED": 0,
            "INTERRUPTING": 0,
            "SUSPENDED": 0,
            "CANCELED": 0,
            "FAILED": 0,
            "SUCCEEDED": 20,
            "NOT_COMPATIBLE": 0
       }
    }
}
```

`detail-type`

イベントのタイプを示します。

このイベントの場合、この値は `Step Run Status Change` です。

`source`

イベントを発生させたサービスを識別します。Deadline Cloud イベントの場合、この値は です`aws.deadline`。

`detail`

イベントに関する情報を含む JSON オブジェクト。このフィールドの内容は、イベントを生成するサービスによって決まります。

このイベントの場合、このデータには以下が含まれます。

`farmId`

ジョブを含むファームの識別子。

`queueId`

ジョブを含むキューの識別子。

`jobId`

ジョブの識別子。

`stepId`

現在のジョブステップの識別子。

`previousTaskRunStatus`

ステップが離れる実行状態。

`taskRunStatus`

ステップが入る実行状態。

`taskRunStatusCounts`

各状態のステップのタスクの数。

タスク実行ステータス変更イベント

`runStatus` フィールドは、タスクの実行時に更新されます。イベントは、次の場合に送信されます。

- タスクの実行ステータスが変更されます。
- タスクが READY 状態にある場合を除き、タスクは再キューに入れられます。

以下の場合、イベントは送信されません。

- タスクが最初に作成されます。タスクの作成をモニタリングするには、ジョブライフサイクルステータス変更イベントの変更をモニタリングします。

以下は、`Task Run Status Change` イベントの詳細フィールドです。

Deadline Cloud イベントの特定の値が含まれているため、 フィールド`source`と `detail-type`フィールドは以下のとおりです。すべてのイベントに含まれる他のメタデータフィールドの定

義については、Amazon EventBridge 「 ユーザーガイド」の「イベント構造のリファレンス」を参照してください。

```
{
    "version": "0",
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "detail-type": "Task Run Status Change",
    "source": "aws.aws.deadline",
    "account": "111122223333",
    "time": "2017-12-22T18:43:48Z",
    "region": "aa-example-1",
    "resources": [],
    "detail": {
        "farmId": "farm-12345678900000000000000000000000",
        "queueId": "queue-12345678900000000000000000000000",
        "jobId": "job-12345678900000000000000000000000",
        "stepId": "step-12345678900000000000000000000000",
        "taskId": "task-12345678900000000000000000000000-0",
        "previousRunStatus": "RUNNING",
        "runStatus": "SUCCEEDED"
    }
}
```

`detail-type`

イベントのタイプを示します。

このイベントの場合、この値は `Fleet Size Recommendation Change` です。

`source`

イベントを発生させたサービスを識別します。Deadline Cloud イベントの場合、この値は ですaws.deadline。

`detail`

イベントに関する情報を含む JSON オブジェクト。このフィールドの内容は、イベントを生成するサービスによって決まります。

このイベントの場合、このデータには以下が含まれます。

`farmId`

ジョブを含むファームの識別子。

`queueId`

ジョブを含むキューの識別子。

`jobId`

ジョブの識別子。

stepId

現在のジョブステップの識別子。

`taskId`

実行中のタスクの識別子。

`previousRunStatus`

タスクが離れる実行状態。

`runStatus`

タスクが入力する実行ステータス。

のセキュリティ Deadline Cloud

のクラウドセキュリティが最優先事項 AWS です。お客様は AWS 、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャからメリットを得られます。

セキュリティは、 AWS お客様とお客様の間の責任共有です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティと説明しています。

- クラウドのセキュリティ – AWS は、 AWS のサービス で実行されるインフラストラクチャを保護する責任を担います AWS クラウド。 は、お客様が安全に使用できるサービス AWS も提供します。サードパーティーの監査者は、AWS コンプライアンスプログラムコンプライアンスプログラムの一環として、当社のセキュリティの有効性を定期的にテストおよび検証。が適用されるコンプライアンスプログラムの詳細については AWS Deadline Cloud、「コンプライアンスプログラムAWS のサービス による対象範囲内」および「コンプライアンスプログラム」を参照してください。
- クラウド内のセキュリティ – お客様の責任は、使用する によって決まり AWS のサービス ます。また、ユーザーは、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

このドキュメントは、 を使用する際の責任共有モデルの適用方法を理解するのに役立ちます Deadline Cloud。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達成する Deadline Cloud ように を設定する方法について説明します。また、 Deadline Cloud リソースのモニタリングや保護 AWS のサービス に役立つ他の の使用方法についても説明します。

トピック

- でのデータ保護 Deadline Cloud
- Deadline Cloud での Identity and Access Management
- のコンプライアンス検証 Deadline Cloud
- の耐障害性 Deadline Cloud
- Deadline Cloud のインフラストラクチャセキュリティ
- Deadline Cloud の設定と脆弱性の分析
- サービス間での不分別な代理処理の防止
- インターフェイスエンドポイント (AWS PrivateLink) AWS Deadline Cloud を使用した へのアクセス

- Deadline Cloud のセキュリティのベストプラクティス

でのデータ保護 Deadline Cloud

責任 AWS 共有モデル、 でのデータ保護に適用されます AWS Deadline Cloud。このモデルで説明されているように、 AWS はすべての を実行するグローバルインフラストラクチャを保護する責任があります AWS クラウド。ユーザーは、このインフラストラクチャでホストされるコンテンツに対する管理を維持する責任があります。また、使用する「 AWS のサービス 」のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、データプライバシーに関するよくある質問を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログに投稿された AWS 責任共有モデルおよび GDPR のブログ記事を参照してください。

データ保護の目的で、認証情報を保護し AWS アカウント 、 AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 が必須で、TLS 1.3 をお勧めします。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。CloudTrail 証跡を使用して AWS アクティビティをキャプチャする方法については、「 AWS CloudTrail ユーザーガイド」のCloudTrail 証跡の使用」を参照してください。
- AWS 暗号化ソリューションと、 内のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。
- Amazon Macie などの高度な管理されたセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API AWS を介して にアクセスするときに FIPS 140-3 検証済みの暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-3」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報を、タグ、または [名前] フィールドなどの自由形式のテキストフィールドに含めないことを強くお勧めします。これは、コンソール Deadline Cloud 、API、または SDK を使用して AWS CLIまたは他の AWS のサービス を操作する場合も同様です。 AWS SDKs タグ、または名前に使用される自由記述のテキストフィールドに入力したデータは、請求または診断ログに使用される場合があります。外部サーバーに URL を提供する場合、そ

のサーバーへのリクエストを検証できるように、認証情報を URL に含めないことを強くお勧めします。

Deadline Cloud ジョブテンプレートの名前フィールドに入力されたデータは、請求ログや診断ログに含まれている可能性があるため、機密情報や機密情報を含めないでください。

トピック

- 保管中の暗号化
- 転送中の暗号化
- キー管理
- ネットワーク間トラフィックのプライバシー
- オプトアウト

保管中の暗号化

AWS Deadline Cloud は、 AWS Key Management Service （ AWS KMS ）に保存されている暗号化キーを使用して保管時に暗号化することで、機密データを保護します。保管時の暗号化は、 AWS リージョン Deadline Cloud が利用可能なすべての で使用できます。

データの暗号化とは、ディスクに保存された機密データが、有効なキーがないユーザーやアプリケーションによって読み取れないことを意味します。有効なマネージドキーを持つ当事者のみがデータを復号できます。

が保管中のデータの暗号化 AWS KMS にどのように Deadline Cloud 使用されるかについては、「」を参照してくださいキー管理。

転送中の暗号化

転送中のデータの場合、 AWS Deadline Cloud は Transport Layer Security (TLS) 1.2 または 1.3 を使用して、サービスとワーカー間で送信されるデータを暗号化します。TLS 1.2 が必須で、TLS 1.3 をお勧めします。さらに、Virtual Private Cloud (VPC) を使用する場合は、 AWS PrivateLink を使用して VPC と 間のプライベート接続を確立できます Deadline Cloud。

キー管理

新しいファームを作成するときは、次のいずれかのキーを選択してファームデータを暗号化できます。

- AWS 所有 KMS キー – ファームの作成時にキーを指定しない場合のデフォルトの暗号化タイプ。KMS キーは によって所有されています AWS Deadline Cloud。 AWS 所有キーを表示、管理、使用することはできません。ただし、データを暗号化するキーを保護するためにアクションを実行する必要はありません。詳細については、「 デAWS Key Management Service ベロッパーガイド」のAWS 「 所有キー」を参照してください。
- カスタマーマネージド KMS キー – ファームの作成時にカスタマーマネージドキーを指定します。ファーム内のすべてのコンテンツは KMS キーで暗号化されます。キーはアカウントに保存され、ユーザーが作成、所有、管理し、 AWS KMS 料金が適用されます。ユーザーは、KMS キーに関する完全なコントロール権を持ちます。次のようなタスクを実行できます。
  - キーポリシーの確立と維持
  - IAM ポリシーとグラントの策定と維持
  - キーポリシーの有効化と無効化
  - タグの追加
  - キーエイリアスの作成

  Deadline Cloud ファームで使用される顧客所有のキーを手動でローテーションすることはできません。キーの自動ローテーションがサポートされています。

  詳細については、「 AWS Key Management Service デベロッパーガイド」の「カスタマー所有キー」を参照してください。

  カスタマーマネージドキーを作成するには、「 AWS Key Management Service デベロッパーガイド」の「対称カスタマーマネージドキーの作成」の手順に従います。

AWS KMS 許可 Deadline Cloud の使用方法

Deadline Cloud には、カスタマーマネージドキーを使用するための許可が必要です。カスタマーマネージドキーで暗号化されたファームを作成すると、 は、指定した KMS キーへのアクセスを取得する`CreateGrant`リクエスト AWS KMS を に送信して、ユーザーに代わって許可 Deadline Cloud を作成します。

Deadline Cloud は複数の許可を使用します。各権限は、データを暗号化または復号 Deadline Cloud する必要がある の異なる部分によって使用されます。 Deadline Cloud また、 は権限を使用して、Amazon Simple Storage Service、Amazon Elastic Block Store、OpenSearch など、ユーザーに代わってデータを保存するために使用される他の AWS サービスへのアクセスを許可します。

がサービスマネージドフリート内のマシンを管理 Deadline Cloud できるようにする権限には、Deadline Cloud サービスプリンシパル`GranteePrincipal`の代わりに のアカウント番号とロールが含まれます。一般的ではありませんが、これは、ファームに指定されたカスタマーマネージド KMS キーを使用して、サービスマネージドフリートのワーカーの Amazon EBS ボリュームを暗号化するために必要です。

カスタマーマネージドキーポリシー

キーポリシーは、カスタマーマネージドキーへのアクセスを制御します。各キーには、キーを使用できるユーザーとその使用方法を決定するステートメントを含むキーポリシーが 1 つだけ必要です。カスタマーマネージドキーを作成するときに、キーポリシーを指定できます。詳細については、AWS Key Management Service デベロッパーガイドの「Managing access to customer managed keys」を参照してください。

CreateFarm の最小 IAM ポリシー

カスタマーマネージドキーを使用して コンソールまたは `CreateFarm` API オペレーションを使用してファームを作成するには、次の AWS KMS API オペレーションを許可する必要があります。

- `kms:CreateGrant` - カスタマーマネージドキーに許可を追加します。指定された AWS KMS キーへのコンソールアクセスを許可します。詳細については、 デAWS Key Management Service ベロッパーガイドの「許可の使用」を参照してください。
- `kms:Decrypt` – Deadline Cloud がファーム内のデータを復号できるようにします。
- `kms:DescribeKey` – がキー Deadline Cloud を検証できるように、カスタマーマネージドキーの詳細を提供します。
- `kms:GenerateDataKey` – が一意のデータキーを使用してデータを暗号化 Deadline Cloud できるようにします。

次のポリシーステートメントは、 `CreateFarm`オペレーションに必要なアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DeadlineCreateGrants",
            "Effect": "Allow",
            "Action": [
```

```
                "kms:Decrypt",
                "kms:GenerateDataKey",
                "kms:CreateGrant",
                "kms:DescribeKey"
            ],
            "Resource": "arn:aws::kms:us-west-2:111122223333:key/1234567890abcdef0",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "deadline.us-west-2.amazonaws.com"
                }
            }
        }
    ]
}
```

読み取り専用オペレーションの最小 IAM ポリシー

ファーム、キュー、フリートに関する情報の取得など、カスタマーマネージドキーを読み取り専用 Deadline Cloud オペレーションに使用するには。次の AWS KMS API オペレーションを許可する必要があります。

- `kms:Decrypt` – Deadline Cloud がファーム内のデータを復号できるようにします。
- `kms:DescribeKey` – がキー Deadline Cloud を検証できるように、カスタマーマネージドキーの詳細を提供します。

次のポリシーステートメントは、読み取り専用オペレーションに必要なアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DeadlineReadOnly",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey"
            ],
            "Resource": "arn:aws::kms:us-west-2:111122223333:key/a1b2c3d4-5678-90ab-
cdef-EXAMPLE11111",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "deadline.us-west-2.amazonaws.com"
```

```
                }
            }
        }
    ]
}
```

読み取り/書き込みオペレーションの最小 IAM ポリシー

ファーム、キュー、フリートの作成や更新などの読み取り/書き込み Deadline Cloud オペレーションにカスタマーマネージドキーを使用するには。次の AWS KMS API オペレーションを許可する必要があります。

- `kms:Decrypt` – Deadline Cloud がファーム内のデータを復号できるようにします。
- `kms:DescribeKey` – がキー Deadline Cloud を検証できるように、カスタマーマネージドキーの詳細を提供します。
- `kms:GenerateDataKey` – が一意のデータキーを使用してデータを暗号化 Deadline Cloud できるようにします。

次のポリシーステートメントは、 `CreateFarm`オペレーションに必要なアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DeadlineReadWrite",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey",
            ],
            "Resource": "arn:aws::kms:us-west-2:111122223333:key/a1b2c3d4-5678-90ab-
cdef-EXAMPLE11111",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "deadline.us-west-2.amazonaws.com"
                }
            }
        }
    ]
```

```
}
```

暗号化キーのモニタリング

Deadline Cloud ファームで AWS KMS カスタマーマネージドキーを使用する場合、 AWS CloudTrailまたは Amazon CloudWatch Logs を使用して、 が Deadline Cloud 送信するリクエストを追跡できます AWS KMS。

許可の CloudTrail イベント

次の CloudTrail イベント例は、通常、`CreateFarm`、、または `CreateFleet`オペレーションを呼び出すときに`CreateMonitor`、許可が作成されたときに発生します。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:SampleUser01",
        "arn": "arn:aws::sts::111122223333:assumed-role/Admin/SampleUser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws::iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2024-04-23T02:05:26Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "deadline.amazonaws.com"
    },
    "eventTime": "2024-04-23T02:05:35Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "deadline.amazonaws.com",
    "userAgent": "deadline.amazonaws.com",
```

```
    "requestParameters": {
        "operations": [
            "CreateGrant",
            "Decrypt",
            "DescribeKey",
            "Encrypt",
            "GenerateDataKey"
        ],
        "constraints": {
            "encryptionContextSubset": {
                "aws:deadline:farmId": "farm-abcdef1234567890987654321fedcba",
                "aws:deadline:accountId": "111122223333"
            }
        },
        "granteePrincipal": "deadline.amazonaws.com",
        "keyId": "arn:aws::kms:us-west-2:111122223333:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
        "retiringPrincipal": "deadline.amazonaws.com"
    },
    "responseElements": {
        "grantId": "6bbe819394822a400fe5e3a75d0e9ef16c1733143fff0c1fc00dc7ac282a18a0",
        "keyId": "arn:aws::kms:us-west-2:111122223333:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "readOnly": false,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws::kms:us-west-2:111122223333:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE44444"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

復号用の CloudTrail イベント

次の CloudTrail イベント例は、カスタマーマネージド KMS キーを使用して値を復号するときに発生します。


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:SampleUser01",
        "arn": "arn:aws::sts::111122223333:assumed-role/SampleRole/SampleUser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws::iam::111122223333:role/SampleRole",
                "accountId": "111122223333",
                "userName": "SampleRole"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2024-04-23T18:46:51Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "deadline.amazonaws.com"
    },
    "eventTime": "2024-04-23T18:51:44Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "deadline.amazonaws.com",
    "userAgent": "deadline.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:deadline:farmId": "farm-abcdef1234567890098765432 1fedcba",
            "aws:deadline:accountId": "111122223333",
            "aws-crypto-public-key": "AotL+SAMPLEVALUEiOMEXAMPLEaaqNOTREALaGTESTONLY
+p/5H+EuKd4Q=="
        },
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
```

```
        "keyId": "arn:aws::kms:us-west-2:111122223333:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
    },
    "responseElements": null,
    "requestID": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeffffff",
    "eventID": "ffffffff-eeee-dddd-cccc-bbbbbbaaaaaa",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws::kms:us-west-2:111122223333:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

暗号化用の CloudTrail イベント

次の CloudTrail イベント例は、カスタマーマネージド KMS キーを使用して値を暗号化するときに発生します。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:SampleUser01",
        "arn": "arn:aws::sts::111122223333:assumed-role/SampleRole/SampleUser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws::iam::111122223333:role/SampleRole",
                "accountId": "111122223333",
                "userName": "SampleRole"
            },
            "webIdFederationData": {},
```

```
            "attributes": {
                "creationDate": "2024-04-23T18:46:51Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "deadline.amazonaws.com"
    },
    "eventTime": "2024-04-23T18:52:40Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "deadline.amazonaws.com",
    "userAgent": "deadline.amazonaws.com",
    "requestParameters": {
        "numberOfBytes": 32,
        "encryptionContext": {
            "aws:deadline:farmId": "farm-abcdef1234567890098765432 1fedcba",
            "aws:deadline:accountId": "111122223333",
            "aws-crypto-public-key": "AotL+SAMPLEVALUEiOMEXAMPLEaaqNOTREALaGTESTONLY
+p/5H+EuKd4Q=="
        },
        "keyId": "arn:aws::kms:us-
west-2:111122223333:key/abcdef12-3456-7890-0987-654321fedcba"
    },
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws::kms:us-west-2:111122223333:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE33333"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

カスタマーマネージド KMS キーの削除

AWS Key Management Service（AWS KMS) でカスタマーマネージド KMS キーを削除すると、破壊的であり、潜在的に危険です。これにより、キーマテリアルとキーに関連付けられているすべてのメタデータが削除され、元に戻すことはできません。カスタマーマネージド KMS キーを削除すると、そのキーで暗号化されたデータを復号できなくなります。これは、データが回復不能になることを意味します。

そのため、 AWS KMS は KMS キーを削除する前に最大 30 日間の待機期間をお客様に付与します。デフォルトの待機時間は、30 日です。

待機期間について

カスタマーマネージド KMS キーを削除することは破壊的で潜在的に危険であるため、7～30 日間の待機期間を設定する必要があります。デフォルトの待機時間は、30 日です。

ただし、実際の待機期間は、スケジュールした期間よりも最大 24 時間長くなる場合があります。キーが削除される実際の日時を取得するには、 DescribeKeyオペレーションを使用します。また、[General configuration] (一般的な設定) セクションのキーの詳細ページにある AWS KMS コンソールでは、削除のためにスケジュールされた日付を確認することが可能です。タイムゾーンに注意してください。

削除の待機期間中は、カスタマーマネージドキーのステータスおよびキーの状態が削除保留中になります。

- 削除保留中のカスタマーマネージド KMS キーは、暗号化オペレーションに使用することはできません。
- AWS KMS は、削除保留中のカスタマーマネージド KMS キーのバッキングキーをローテーションしません。

カスタマーマネージド KMS キーの削除の詳細については、 AWS Key Management Service デベロッパーガイドの「カスタマーマスターキーの削除」を参照してください。

ネットワーク間トラフィックのプライバシー

AWS Deadline Cloud はAmazon Virtual Private Cloud (Amazon VPC) をサポートして接続を保護します。Amazon VPC は、Virtual Private Cloud (VPC) のセキュリティを強化、モニタリングするために使用できる機能を提供します。

VPC 内で実行される Amazon Elastic Compute Cloud (Amazon EC2) インスタンスを使用して、カスタマーマネージドフリート (CMF) を設定できます。使用する Amazon VPC エンドポイントをデプロイすることで AWS PrivateLink、CMF のワーカーと Deadline Cloud エンドポイント間のトラフィックは VPC 内に留まります。さらに、インスタンスへのインターネットアクセスを制限するように VPC を設定できます。

サービスマネージドフリートでは、ワーカーはインターネットからアクセスできませんが、インターネットアクセスがあり、インターネット経由で Deadline Cloud サービスに接続できます。

オプトアウト

AWS Deadline Cloud は、開発と改善に役立つ特定の運用情報を収集します Deadline Cloud。収集されたデータには、 AWS アカウント ID やユーザー ID などの情報が含まれているため、 に問題がある場合は正しく識別できます Deadline Cloud。また、リソース IDs (該当する場合は FarmID または QueueID)、製品名 (JobAttachments、WorkerAgent など) 、製品バージョンなどの Deadline Cloud 特定の情報を収集します。

アプリケーション設定を使用して、このデータ収集をオプトアウトできます。クライアントワークステーションとフリートワーカー Deadline Cloudの両方とやり取りする各コンピュータは、個別にオプトアウトする必要があります。

Deadline Cloud モニター - デスクトップ

Deadline Cloud モニター - デスクトップは、クラッシュが発生したときやアプリケーションが開かれたときなどの運用情報を収集し、アプリケーションに問題が発生したときの把握に役立ちます。この運用情報の収集をオプトアウトするには、設定ページに移動し、データ収集をオンにして Deadline Cloud Monitor のパフォーマンスを測定します。

オプトアウトすると、デスクトップモニターは運用データを送信しなくなります。以前に収集されたデータは保持され、引き続きサービスの改善に使用される可能性があります。詳細については、データプライバシーのよくある質問を参照してください。

AWS Deadline Cloud CLI とツール

AWS Deadline Cloud CLI、送信者、ワーカーエージェントはすべて、クラッシュが発生したときやジョブが送信されたときなどの運用情報を収集し、これらのアプリケーションで問題が発生したときの把握に役立ちます。この運用情報の収集をオプトアウトするには、次のいずれかの方法を使用します。

- ターミナルで、 と入力します**`deadline config set telemetry.opt_out true`**。

これにより、現在のユーザーとして実行されているときに CLI、送信者、ワーカーエージェントがオプトアウトされます。

- Deadline Cloud ワーカーエージェントをインストールするときは、**`--telemetry-opt-out`**コマンドライン引数を追加します。例えば、 **`./install.sh --farm-id $FARM_ID --fleet-id $FLEET_ID --telemetry-opt-out`**。
- ワーカーエージェント、CLI、または送信者を実行する前に、環境変数を設定します。**`DEADLINE_CLOUD_TELEMETRY_OPT_OUT=true`**

オプトアウトすると、 Deadline Cloud ツールは運用データを送信しなくなります。以前に収集されたデータは保持され、引き続きサービスの改善に使用される可能性があります。詳細については、データプライバシーのよくある質問を参照してください。

Deadline Cloud での Identity and Access Management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を認証 (サインイン) し、誰に Deadline Cloud リソースの使用を許可する (アクセス許可を付与する) かを制御します。IAM は、追加料金なしで使用できる AWS のサービス です。

トピック

- 対象者
- アイデンティティを使用した認証
- ポリシーを使用したアクセスの管理
- Deadline Cloud と IAM の連携方法
- Deadline Cloud のアイデンティティベースのポリシーの例
- AWS Deadline Cloud の マネージドポリシー
- AWS Deadline Cloud のアイデンティティとアクセスのトラブルシューティング

対象者

AWS Identity and Access Management (IAM) の使用方法は、Deadline Cloud で行う作業によって異なります。

サービスユーザー – Deadline Cloud サービスを使用してジョブを実行する場合、管理者から必要な認証情報とアクセス許可が提供されます。さらに多くの Deadline Cloud 機能を使用して作業を行う場合は、追加のアクセス許可が必要になることがあります。アクセスの管理方法を理解すると、管理者に適切なアクセス許可をリクエストするのに役に立ちます。Deadline Cloud の機能にアクセスできない場合は、「」を参照してください[AWS Deadline Cloud のアイデンティティとアクセスのトラブルシューティング](#)。

サービス管理者 – 社内の Deadline Cloud リソースを担当している場合は、通常、Deadline Cloud へのフルアクセスがあります。サービスユーザーがどの Deadline Cloud 機能やリソースにアクセスするかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。会社が Deadline Cloud で IAM を使用する方法の詳細については、「」を参照してください[Deadline Cloud と IAM の連携方法](#)。

IAM 管理者 – IAM 管理者は、Deadline Cloud へのアクセスを管理するポリシーの作成方法の詳細について確認する場合があります。IAM で使用できる Deadline Cloud アイデンティティベースのポリシーの例を表示するには、「」を参照してください[Deadline Cloud のアイデンティティベースのポリシーの例](#)。

アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。として、IAM ユーザーとして AWS アカウントのルートユーザー、または IAM ロールを引き受けることによって認証 (サインイン AWS) される必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。 AWS IAM Identity Center（ IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook 認証情報は、フェデレーション ID の例です。フェデレーティッド ID としてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーション AWS を使用して にアクセスすると、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、「 AWS サインイン ユーザーガイド」の[「へのサインイン AWS アカウント](#)方法」を参照してください。

AWS プログラムで にアクセスする場合、 はソフトウェア開発キット (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、認証情報を使用してリクエストを暗号化して署名します。AWS ツールを使用しない場合は、自分でリクエストに署名する必要があります。リクエストに自分

で署名する推奨方法の使用については、「IAM ユーザーガイド」の「API リクエストに対するAWS Signature Version 4」を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。たとえば、 では、アカウントのセキュリティを高めるために多要素認証 (MFA) を使用する AWS ことをお勧めします。詳細については、「AWS IAM Identity Center ユーザーガイド」の「多要素認証」および「IAM ユーザーガイド」の「IAM のAWS 多要素認証」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、アカウント内のすべての およびリソースへの AWS のサービス 完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることでアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

フェデレーティッドアイデンティティ

ベストプラクティスとして、管理者アクセスを必要とするユーザーを含む人間のユーザーに、ID プロバイダーとのフェデレーションを使用して一時的な認証情報 AWS のサービス を使用して にアクセスすることを要求します。

フェデレーティッド ID は、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダー、AWS Directory Service、アイデンティティセンターディレクトリ、または ID ソースを介して提供された認証情報 AWS のサービス を使用して にアクセスするユーザーです。フェデレーティッド ID がにアクセスすると AWS アカウント、ロールを引き受け、ロールは一時的な認証情報を提供します。

アクセスを一元管理する場合は、 AWS IAM Identity Centerを使用することをお勧めします。IAM Identity Center でユーザーとグループを作成するか、独自の ID ソースのユーザーとグループのセットに接続して同期し、すべての AWS アカウント とアプリケーションで使用できます。IAM Identity Center の詳細については、「AWS IAM Identity Center ユーザーガイド」の「What is IAM Identity Center?」(IAM Identity Center とは) を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、単一のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内の ID です。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保

有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、「IAM ユーザーガイド」の「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する許可を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時認証情報が提供されます。詳細については、「IAM ユーザーガイド」の「IAM ユーザーに関するユースケース」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内の ID です。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。で IAM ロールを一時的に引き受けるには AWS Management Console、ユーザーから IAM ロール (コンソール) に切り替えることができます。ロールを引き受けるには、 または AWS API オペレーションを AWS CLI 呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「ロールを引き受けるための各種方法」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッド ID に許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーティッド ID が認証されると、その ID はロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションのロールについては、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールを作成する」を参照してください。IAM Identity Center を使用する場合は、許可セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。アクセス許可セットの詳細については、「AWS IAM Identity Center User Guide」の「Permission sets」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。

- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、 (プロキシとしてロールを使用する代わりに) リソースに直接ポリシーをアタッチできます。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。
- クロスサービスアクセス — 一部の は他の の機能 AWS のサービス を使用します AWS のサービス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの許可、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用してアクションを実行すると AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストをリクエストする と組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。
  - サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除することができます。詳細については、「IAM ユーザーガイド」の「AWS のサービスに許可を委任するロールを作成する」を参照してください。
  - サービスにリンクされたロール – サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスリンクロールのアクセス許可を表示できますが、編集することはできません。
- Amazon EC2 で実行されているアプリケーション – IAM ロールを使用して、EC2 インスタンスで実行され、 AWS CLI または AWS API リクエストを行うアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。EC2 インスタンスに AWS ロールを割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロ

ファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、「IAM ユーザーガイド」の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して許可を付与する」を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、ID AWS またはリソースにアタッチします。ポリシーは AWS 、アイデンティティまたはリソースに関連付けられているときにアクセス許可を定義する のオブジェクトです。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うときに、これらのポリシー AWS を評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き受けることができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの許可を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。そのポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS API からロール情報を取得できます。

アイデンティティベースのポリシー

アイデンティティベースポリシーは、IAM ユーザーグループ、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースポリシーの作成方法については、「IAM ユーザーガイド」の「カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する」を参照してください。

アイデンティティベースのポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、 内の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシー

が含まれます。マネージドポリシーまたはインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「管理ポリシーとインラインポリシーのいずれかを選択する」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM の AWS マネージドポリシーを使用できません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、および Amazon VPC は AWS WAF、ACLs。ACL の詳細については、「Amazon Simple Storage Service デベロッパーガイド」の「アクセスコントロールリスト (ACL) の概要」を参照してください。

その他のポリシータイプ

AWS は、一般的でない追加のポリシータイプをサポートしています。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になりま

す。アクセス許可の境界の詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可の境界」を参照してください。

- サービスコントロールポリシー (SCPs) – SCPs は、 の組織または組織単位 (OU) の最大アクセス許可を指定する JSON ポリシーです AWS Organizations。 AWS Organizations は、ビジネスが所有する複数の AWS アカウント をグループ化して一元管理するためのサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP は、各 を含むメンバーアカウントのエンティティのアクセス許可を制限します AWS アカウントのルートユーザー。Organizations と SCP の詳細については、「AWS Organizations ユーザーガイド」の「サービスコントロールポリシー (SCP)」を参照してください。
- リソースコントロールポリシー (RCP) – RCP は、所有する各リソースにアタッチされた IAM ポリシーを更新することなく、アカウント内のリソースに利用可能な最大数のアクセス許可を設定するために使用できる JSON ポリシーです。RCP は、メンバーアカウントのリソースのアクセス許可を制限し、組織に属しているかどうかにかかわらず AWS アカウントのルートユーザー、 を含む ID の有効なアクセス許可に影響を与える可能性があります。RCP をサポートする のリストを含む Organizations と RCP の詳細については、AWS Organizations RCPs「リソースコントロールポリシー (RCPs」を参照してください。 AWS のサービス
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。が複数のポリシータイプが関与する場合にリクエストを許可するかどうか AWS を決定する方法については、「IAM ユーザーガイド」の「ポリシー評価ロジック」を参照してください。

Deadline Cloud と IAM の連携方法

IAM を使用して Deadline Cloud へのアクセスを管理する前に、Deadline Cloud で使用できる IAM 機能について説明します。

AWS Deadline Cloud で使用できる IAM 機能

IAM 機能	Deadline Cloud のサポート
アイデンティティベースポリシー	はい
リソースベースのポリシー	いいえ
ポリシーアクション	はい
ポリシーリソース	はい
ポリシー条件キー (サービス固有)	はい
ACL	いいえ
ABAC (ポリシー内のタグ)	あり
一時的な認証情報	あり
転送アクセスセッション (FAS)	あり
サービスロール	はい
サービスリンクロール	いいえ

Deadline Cloud およびその他の がほとんどの IAM 機能と AWS のサービス 連携する方法の概要については、IAM ユーザーガイドのAWS 「IAM と連携する のサービス」を参照してください。

Deadline Cloud のアイデンティティベースのポリシー

アイデンティティベースのポリシーのサポート: あり

アイデンティティベースポリシーは、IAM ユーザーグループ、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。ID ベースのポリシーの作成方法については、「IAM ユーザーガイド」の「カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。プリンシパルは、それが添付されている

ユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシーで使用できるすべての要素について学ぶには、「IAM ユーザーガイド」の「IAM JSON ポリシーの要素のリファレンス」を参照してください。

Deadline Cloud のアイデンティティベースのポリシーの例

Deadline Cloud アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいDeadline Cloud のアイデンティティベースのポリシーの例。

Deadline Cloud 内のリソースベースのポリシー

リソースベースのポリシーのサポート: なし

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

クロスアカウントアクセスを有効にするには、アカウント全体、または別のアカウントの IAM エンティティをリソースベースのポリシーのプリンシパルとして指定します。リソースベースのポリシーにクロスアカウントのプリンシパルを追加しても、信頼関係は半分しか確立されない点に注意してください。プリンシパルとリソースが異なる場合 AWS アカウント、信頼されたアカウントの IAM 管理者は、プリンシパルエンティティ (ユーザーまたはロール) にリソースへのアクセス許可も付与する必要があります。IAM 管理者は、アイデンティティベースのポリシーをエンティティにアタッチすることで権限を付与します。ただし、リソースベースのポリシーで、同じアカウントのプリンシパルへのアクセス権が付与されている場合は、アイデンティティベースのポリシーをさらに付与する必要はありません。詳細については、「IAM ユーザーガイド」の「IAM でのクロスアカウントリソースアクセス」を参照してください。

Deadline Cloud のポリシーアクション

ポリシーアクションのサポート:あり

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

JSON ポリシーの `Action` 要素にはポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションの名前は通常、関連付けられた AWS API オペレーションと同じです。一致する API オペレーションのない許可のみのアクションなど、いくつかの例外があります。また、ポリシーに複数のアクションが必要なオペレーションもあります。これらの追加アクションは依存アクションと呼ばれます。

このアクションは関連付けられたオペレーションを実行するためのアクセス許可を付与するポリシーで使用されます。

Deadline Cloud アクションのリストを確認するには、「サービス認可リファレンス」の「Deadline Cloud AWS で定義されるアクション」を参照してください。

Deadline Cloud のポリシーアクションは、アクションの前に次のプレフィックスを使用します。

```
awsdeadlinecloud
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

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

Deadline Cloud アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいDeadline Cloud のアイデンティティベースのポリシーの例。

Deadline Cloud のポリシーリソース

ポリシーリソースのサポート: あり

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

`Resource` JSON ポリシー要素はアクションが適用されるオブジェクトを指定します。ステートメントには`Resource` または `NotResource` 要素を含める必要があります。ベストプラクティスとし

て、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの許可と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

Deadline Cloud リソースタイプとその ARNs「Deadline Cloud AWS で定義されるリソース」を参照してください。 各リソースの ARN を指定できるアクションについては、「Deadline Cloud AWS で定義されるアクション」を参照してください。

Deadline Cloud アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいDeadline Cloud のアイデンティティベースのポリシーの例。

Deadline Cloud のポリシー条件キー

サービス固有のポリシー条件キーのサポート: あり

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

`Condition` 要素 (または `Condition` ブロック) を使用すると、ステートメントが有効な条件を指定できます。`Condition` 要素はオプションです。イコールや未満などの 条件演算子 を使用して条件式を作成して、ポリシーの条件とリクエスト内の値を一致させることができます。

1 つのステートメントに複数の `Condition` 要素を指定する場合、または 1 つの `Condition` 要素に複数のキーを指定する場合、 AWS では `AND` 論理演算子を使用してそれらを評価します。1 つの条件キーに複数の値を指定すると、 は論理`OR`オペレーションを使用して条件 AWS を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。詳細については、「IAM ユーザーガイド」の「IAM ポリシーの要素: 変数およびタグ」を参照してください。

AWS は、グローバル条件キーとサービス固有の条件キーをサポートしています。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイド」のAWS 「グローバル条件コンテキストキー」を参照してください。

Deadline Cloud 条件キーのリストを確認するには、「サービス認可リファレンス」のAWS 「Deadline Cloud の条件キー」を参照してください。条件キーを使用できるアクションとリソースについては、「Deadline Cloud AWS で定義されるアクション」を参照してください。

Deadline Cloud アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいDeadline Cloud のアイデンティティベースのポリシーの例。

Deadline Cloud ACLs

ACL のサポート: なし

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Deadline Cloud での ABAC

ABAC (ポリシー内のタグ) のサポート: あり

属性ベースのアクセス制御 (ABAC) は、属性に基づいてアクセス許可を定義する認可戦略です。では AWS、これらの属性はタグと呼ばれます。タグは、IAM エンティティ (ユーザーまたはロール) および多くの AWS リソースにアタッチできます。エンティティとリソースのタグ付けは、ABAC の最初の手順です。その後、プリンシパルのタグがアクセスしようとしているリソースのタグと一致した場合にオペレーションを許可するように ABAC ポリシーをします。

ABAC は、急成長する環境やポリシー管理が煩雑になる状況で役立ちます。

タグに基づいてアクセスを管理するには、`aws:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値はありです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

ABAC の詳細については、「IAM ユーザーガイド」の「ABAC 認可でアクセス許可を定義する」を参照してください。ABAC をセットアップする手順を説明するチュートリアルについては、「IAM

ユーザーガイド」の「属性ベースのアクセスコントロール (ABAC) を使用する」を参照してください。

Deadline Cloud での一時的な認証情報の使用

一時的な認証情報のサポート: あり

一部の AWS のサービス は、一時的な認証情報を使用してサインインすると機能しません。一時的な認証情報と AWS のサービス 連携する などの詳細については、IAM ユーザーガイドの「IAM AWS のサービス と連携する 」を参照してください。

ユーザー名とパスワード以外の AWS Management Console 方法で にサインインする場合は、一時的な認証情報を使用します。たとえば、会社のシングルサインオン (SSO) リンク AWS を使用してにアクセスすると、そのプロセスによって一時的な認証情報が自動的に作成されます。また、ユーザーとしてコンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えに関する詳細については、「IAM ユーザーガイド」の「ユーザーから IAM ロールに切り替える (コンソール)」を参照してください。

一時的な認証情報は、 AWS CLI または AWS API を使用して手動で作成できます。その後、これらの一時的な認証情報を使用してアクセスすることができます AWS。長期的なアクセスキーを使用する代わりに、一時的な認証情報を動的に生成 AWS することをお勧めします。詳細については、「IAM の一時的セキュリティ認証情報」を参照してください。

Deadline Cloud の転送アクセスセッション

転送アクセスセッション (FAS) のサポート: あり

IAM ユーザーまたはロールを使用して でアクションを実行すると AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストをリクエストする と組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

Deadline Cloud のサービスロール

サービスロールのサポート: あり

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービスに許可を委任するロールを作成する」を参照してください。

> **Warning**
>
> サービスロールのアクセス許可を変更すると、Deadline Cloud の機能が破損する可能性があります。Deadline Cloud が指示する場合にのみ、サービスロールを編集します。

Deadline Cloud のサービスにリンクされたロール

サービスにリンクされたロールのサポート: なし

サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、 サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールのアクセス許可を表示できますが、編集することはできません。

サービスにリンクされたロールの作成または管理の詳細については、「IAM と提携するAWS のサービス」を参照してください。表の「サービスリンクロール」列に Yes と記載されたサービスを見つけます。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[はい] リンクを選択します。

Deadline Cloud のアイデンティティベースのポリシーの例

デフォルトでは、ユーザーとロールには Deadline Cloud リソースを作成または変更するアクセス許可はありません。また、、 AWS Command Line Interface （ AWS CLI ） AWS Management Console、または AWS API を使用してタスクを実行することはできません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

これらサンプルの JSON ポリシードキュメントを使用して、IAM アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーを作成する (コンソール)」を参照してください。

各リソースタイプの ARNsAWS 「Deadline Cloud のアクション、リソース、および条件キー」を参照してください。

トピック

- ポリシーに関するベストプラクティス
- Deadline Cloud コンソールの使用
- キューにジョブを送信するポリシー
- ライセンスエンドポイントの作成を許可するポリシー
- 特定のファームキューのモニタリングを許可するポリシー

ポリシーに関するベストプラクティス

ID ベースのポリシーは、アカウント内の Deadline Cloud リソースを作成、アクセス、または削除できるかどうかを決定します。これらのアクションを実行すると、 AWS アカウントに料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーを開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードにアクセス許可の付与を開始するには、多くの一般的なユースケースにアクセス許可を付与するAWS 管理ポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義することで、アクセス許可をさらに減らすことをお勧めします。詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」または「ジョブ機能のAWS マネージドポリシー」を参照してください。
- 最小特権を適用する – IAM ポリシーで許可を設定する場合は、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、「IAM ユーザーガイド」の「IAM でのポリシーとアクセス許可」を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。条件を使用して、サービスアクションが などの特定の を通じて使用されている場合に AWS のサービス、サービスアクションへのアクセスを許可することもできます AWS CloudFormation。詳細については、「IAM ユーザーガイド」の「IAM JSON ポリシー要素:条件」を参照してください。
- IAM Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサ

ポートします。詳細については、「IAM ユーザーガイド」の「IAM Access Analyzer でポリシーを検証する」を参照してください。

- 多要素認証 (MFA) を要求する – で IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、セキュリティを強化するために MFA を有効にします。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、「IAM ユーザーガイド」の「MFA を使用した安全な API アクセス」を参照してください。

IAM でのベストプラクティスの詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティのベストプラクティス」を参照してください。

Deadline Cloud コンソールの使用

AWS Deadline Cloud コンソールにアクセスするには、最小限のアクセス許可のセットが必要です。これらのアクセス許可により、 の Deadline Cloud リソースの詳細を一覧表示および表示できます AWS アカウント。最小限必要な許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) に対してコンソールが意図したとおりに機能しません。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスが許可されます。

ユーザーとロールが引き続き Deadline Cloud コンソールを使用できるようにするには、エンティティに Deadline Cloud *ConsoleAccess*または *ReadOnly* AWS マネージドポリシーもアタッチします。詳細については、「IAM ユーザーガイド」の「ユーザーへのアクセス許可の追加」を参照してください。

キューにジョブを送信するポリシー

この例では、特定のファーム内の特定のキューにジョブを送信するアクセス許可を付与するスコープダウンポリシーを作成します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SubmitJobsFarmAndQueue",
```

```
            "Effect": "Allow",
            "Action": "deadline:CreateJob",
            "Resource": "arn:aws:deadline:REGION:ACCOUNT_ID:farm/FARM_A/queue/QUEUE_B/
job/*"
        }
    ]
}
```

ライセンスエンドポイントの作成を許可するポリシー

この例では、ライセンスエンドポイントを作成および管理するために必要なアクセス許可を付与するスコープダウンポリシーを作成します。このポリシーを使用して、ファームに関連付けられた VPC のライセンスエンドポイントを作成します。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "CreateLicenseEndpoint",
        "Effect": "Allow",
        "Action": [
            "deadline:CreateLicenseEndpoint",
            "deadline:DeleteLicenseEndpoint",
            "deadline:GetLicenseEndpoint",
            "deadline:ListLicenseEndpoints",
            "deadline:PutMeteredProduct",
            "deadline:DeleteMeteredProduct",
            "deadline:ListMeteredProducts",
            "deadline:ListAvailableMeteredProducts",
            "ec2:CreateVpcEndpoint",
            "ec2:DescribeVpcEndpoints",
            "ec2:DeleteVpcEndpoints"
        ],
        "Resource": "*"
    }]
}
```

特定のファームキューのモニタリングを許可するポリシー

この例では、特定のファームの特定のキュー内のジョブをモニタリングするアクセス許可を付与するスコープダウンポリシーを作成します。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "MonitorJobsFarmAndQueue",
        "Effect": "Allow",
        "Action": [
            "deadline:SearchJobs",
            "deadline:ListJobs",
            "deadline:GetJob",
            "deadline:SearchSteps",
            "deadline:ListSteps",
            "deadline:ListStepConsumers",
            "deadline:ListStepDependencies",
            "deadline:GetStep",
            "deadline:SearchTasks",
            "deadline:ListTasks",
            "deadline:GetTask",
            "deadline:ListSessions",
            "deadline:GetSession",
            "deadline:ListSessionActions",
            "deadline:GetSessionAction"
        ],
        "Resource": [
            "arn:aws:deadline:REGION:123456789012:farm/FARM_A/queue/QUEUE_B",
            "arn:aws:deadline:REGION:123456789012:farm/FARM_A/queue/QUEUE_B/*"
        ]
    }]
}
```

AWS Deadline Cloud の マネージドポリシー

AWS 管理ポリシーは、 によって作成および管理されるスタンドアロンポリシーです AWS。 AWS 管理ポリシーは、ユーザー、グループ、ロールにアクセス許可の割り当てを開始できるように、多くの一般的なユースケースにアクセス許可を付与するように設計されています。

AWS 管理ポリシーは、すべての AWS お客様が使用できるため、特定のユースケースに対して最小特権のアクセス許可を付与しない場合があることに注意してください。ユースケースに固有のカスタマー管理ポリシーを定義して、アクセス許可を絞り込むことをお勧めします。

AWS 管理ポリシーで定義されているアクセス許可は変更できません。が AWS 管理ポリシーで定義されたアクセス許可 AWS を更新すると、ポリシーがアタッチされているすべてのプリンシパル ID

(ユーザー、グループ、ロール) に影響します。 AWS は、新しい が起動されるか、新しい API オペレーション AWS のサービス が既存のサービスで使用できるようになったときに、 AWS 管理ポリシーを更新する可能性が高くなります。

詳細については「IAM ユーザーガイド」の「AWS マネージドポリシー」を参照してください。

AWS 管理ポリシー: AWSDeadlineCloud-FleetWorker

`AWSDeadlineCloud-FleetWorker` ポリシーを (IAM) ID に AWS Identity and Access Management アタッチできます。

このポリシーは、このフリートのワーカーに、サービスへの接続とサービスからのタスクの受信に必要なアクセス許可を付与します。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `deadline` – プリンシパルがフリート内のワーカーを管理できるようにします。

ポリシーの詳細の JSON リストについては、AWS AWSDeadlineCloud-FleetWorker」を参照してください。

AWS 管理ポリシー: AWSDeadlineCloud-WorkerHost

`AWSDeadlineCloud-WorkerHost` ポリシーを IAM アイデンティティにアタッチできます。

このポリシーは、最初に サービスに接続するために必要なアクセス許可を付与します。Amazon Elastic Compute Cloud (Amazon EC2) インスタンスプロファイルとして使用できます。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `deadline` – ワーカーの作成、ワーカーのフリートロールの引き受け、ワーカーへのタグの適用をユーザーに許可します

ポリシーの詳細の JSON リストについては、AWS AWSDeadlineCloud-WorkerHost」を参照してください。

AWS 管理ポリシー: AWSDeadlineCloud-UserAccessFarms

`AWSDeadlineCloud-UserAccessFarms` ポリシーを IAM アイデンティティにアタッチできます。

このポリシーにより、ユーザーは自分がメンバーであるファームとそのメンバーシップレベルに基づいてファームデータにアクセスできます。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `deadline` – ファームデータへのアクセスをユーザーに許可します。
- `ec2` – ユーザーが Amazon EC2 インスタンスタイプの詳細を表示できるようにします。
- `identitystore` – ユーザーがユーザー名とグループ名を表示できるようにします。

ポリシーの詳細の JSON リストについては、AWSDeadlineCloud-UserAccessFarms」を参照してください。

AWS 管理ポリシー: AWSDeadlineCloud-UserAccessFleets

`AWSDeadlineCloud-UserAccessFleets` ポリシーを IAM アイデンティティにアタッチできます。

このポリシーにより、ユーザーは自分がメンバーであるファームとそのメンバーシップレベルに基づいてフリートデータにアクセスできます。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `deadline` – ファームデータへのアクセスをユーザーに許可します。
- `ec2` – ユーザーが Amazon EC2 インスタンスタイプの詳細を表示できるようにします。
- `identitystore` – ユーザーがユーザー名とグループ名を表示できるようにします。

ポリシーの詳細の JSON リストについては、AWS AWSDeadlineCloud-UserAccessFleets」を参照してください。

AWS 管理ポリシー: AWSDeadlineCloud-UserAccessJobs

`AWSDeadlineCloud-UserAccessJobs` ポリシーを IAM アイデンティティにアタッチできます。

このポリシーにより、ユーザーは自分がメンバーであるファームとそのメンバーシップレベルに基づいてジョブデータにアクセスできます。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `deadline` – ファームデータへのアクセスをユーザーに許可します。
- `ec2` – ユーザーが Amazon EC2 インスタンスタイプの詳細を表示できるようにします。
- `identitystore` – ユーザーがユーザー名とグループ名を表示できるようにします。

ポリシーの詳細の JSON リストについては、AWS AWSDeadlineCloud-UserAccessJobs」を参照してください。

AWS 管理ポリシー: AWSDeadlineCloud-UserAccessQueues

`AWSDeadlineCloud-UserAccessQueues` ポリシーを IAM アイデンティティにアタッチできます。

このポリシーにより、ユーザーは自分がメンバーであるファームとそのメンバーシップレベルに基づいてキューデータにアクセスできます。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `deadline` – ファームデータへのアクセスをユーザーに許可します。
- `ec2` – ユーザーが Amazon EC2 インスタンスタイプの詳細を表示できるようにします。
- `identitystore` – ユーザーがユーザー名とグループ名を表示できるようにします。

ポリシーの詳細の JSON リストについては、AWS AWSDeadlineCloud-UserAccessQueues」を参照してください。

AWS マネージドポリシーの Deadline Cloud 更新

このサービスがこれらの変更の追跡を開始してからの Deadline Cloud の AWS マネージドポリシーの更新に関する詳細を表示します。このページの変更に関する自動アラートについては、Deadline Cloud Document 履歴ページの RSS フィードにサブスクライブしてください。

変更	説明	日付
AWSDeadlineCloud-WorkerHost – 変更	Deadline Cloud に新しいアクション `deadline:TagResource` と が追加され`deadline:ListTagsForResource` 、フリートのワーカーに関連付けられたタグを追加および表示できます。	2025 年 5 月 30 日
AWSDeadlineCloud-UserAccessFarms – 変更 AWSDeadlineCloud-UserAccessJobs – 変更 AWSDeadlineCloud-UserAccessQueues – 変更	Deadline Cloud に新しいアクション `deadline:GetJobTemplate` と が追加され`deadline:ListJobParameterDefinitions` 、ジョブを再送信できるようになりました。	2024 年 10 月 7 日
Deadline Cloud が変更の追跡を開始しました	Deadline Cloud は、 AWS マネージドポリシーの変更の追跡を開始しました。	2024 年 4 月 2 日

AWS Deadline Cloud のアイデンティティとアクセスのトラブルシューティング

次の情報は、Deadline Cloud と IAM の使用時に発生する可能性がある一般的な問題の診断と修復に役立ちます。

トピック

- Deadline Cloud でアクションを実行する権限がない
- iam:PassRole を実行する権限がありません
- 自分の 以外のユーザーに Deadline Cloud リソース AWS アカウント へのアクセスを許可したい

Deadline Cloud でアクションを実行する権限がない

アクションを実行する権限がないというエラーが表示された場合は、そのアクションを実行できるようにポリシーを更新する必要があります。

次のエラー例は、mateojackson IAM ユーザーがコンソールを使用して、ある *my-example-widget* リソースに関する詳細情報を表示しようとしたことを想定して、その際に必要なawsdeadlinecloud:*GetWidget* アクセス許可を持っていない場合に発生するものです。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awsdeadlinecloud:GetWidget on resource: my-example-widget
```

この場合、awsdeadlinecloud:*GetWidget* アクションを使用して *my-example-widget*リソースへのアクセスを許可するように、mateojackson ユーザーのポリシーを更新する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン認証情報を提供した担当者が管理者です。

iam:PassRole を実行する権限がありません

iam:PassRole アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して Deadline Cloud にロールを渡すことができるようにする必要があります。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、既存のロールをそのサービスに渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

次の例のエラーは、 という IAM ユーザーがコンソールを使用して Deadline Cloud でアクションを実行marymajorしようとすると発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。メアリーには、ロールをサービスに渡す許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに `iam:PassRole` アクションの実行を許可する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン資格情報を提供した担当者が管理者です。

自分の 以外のユーザーに Deadline Cloud リソース AWS アカウント へのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- Deadline Cloud がこれらの機能をサポートしているかどうかを確認するには、「」を参照してくださいDeadline Cloud と IAM の連携方法。
- 所有 AWS アカウント している のリソースへのアクセスを提供する方法については、「IAM ユーザーガイド」の「所有 AWS アカウント している別の の IAM ユーザーへのアクセスを提供する」を参照してください。
- リソースへのアクセスをサードパーティーに提供する方法については AWS アカウント、IAM ユーザーガイドの「サードパーティー AWS アカウント が所有する へのアクセスを提供する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、「IAM ユーザーガイド」の「外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可」を参照してください。
- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用方法の違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。

のコンプライアンス検証 Deadline Cloud

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、AWS のサービス 「コンプライアンスプログラムによる範囲内」を参照して、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS 「 Compliance ProgramsAssurance」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「Downloading Reports in AWS Artifact」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。 は、コンプライアンスに役立つ以下のリソース AWS を提供します。

- セキュリティのコンプライアンスとガバナンス – これらのソリューション実装ガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスの機能をデプロイする手順を示します。
- HIPAA 対応サービスのリファレンス – HIPAA 対応サービスの一覧が提供されています。すべての AWS のサービス が HIPAA の対象となるわけではありません。
- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- AWS カスタマーコンプライアンスガイド – コンプライアンスの観点から責任共有モデルを理解します。このガイドでは、複数のフレームワーク (米国国立標準技術研究所 (NIST)、Payment Card Industry Security Standards Council (PCI)、国際標準化機構 (ISO) など) にわたるセキュリティコントロールを保護および AWS のサービス マッピングするためのベストプラクティスをまとめています。
- 「 デベロッパーガイド」の「ルールによるリソースの評価」 – この AWS Config サービスは、リソース設定が内部プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。 AWS Config
- AWS Security Hub – これにより AWS のサービス 、 内のセキュリティ状態を包括的に把握できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールの一覧については、Security Hub のコントロールリファレンスを参照してください。
- Amazon GuardDuty – 不審なアクティビティや悪意のあるアクティビティがないか環境をモニタリングすることで AWS アカウント、、ワークロード、コンテナ、データに対する潜在的な脅威 AWS のサービス を検出します。GuardDuty を使用すると、特定のコンプライアンスフレームワークで義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件に対応できます。
- AWS Audit Manager – これにより AWS のサービス 、 AWS 使用状況を継続的に監査し、リスクの管理方法と規制や業界標準への準拠を簡素化できます。

の耐障害性 Deadline Cloud

AWS グローバルインフラストラクチャは、 AWS リージョン およびアベイラビリティーゾーンを中心に構築されています。 は、低レイテンシー、高スループット、高度に冗長なネットワークで接続された、物理的に分離および分離された複数のアベイラビリティーゾーン AWS リージョン を提供します。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性が高く、フォールトトレラントで、スケーラブルです。

AWS リージョン およびアベイラビリティーゾーンの詳細については、AWS 「 グローバルインフラストラクチャ」を参照してください。

AWS Deadline Cloud は、ジョブアタッチメント S3 バケットに保存されているデータをバックアップしません。SAmazon S3S3 バックアップメカニズムを使用して、ジョブアタッチメントデータのバックアップを有効にできますAWS Backup。

Deadline Cloud のインフラストラクチャセキュリティ

マネージドサービスである AWS Deadline Cloud は、 AWS グローバルネットワークセキュリティで保護されています。 AWS セキュリティサービスと がインフラストラクチャ AWS を保護する方法については、AWS 「 クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには、「Security Pillar AWS Well-Architected Framework」の「Infrastructure Protection」を参照してください。

AWS 公開された API コールを使用して、ネットワーク経由で Deadline Cloud にアクセスします。クライアントは以下をサポートする必要があります。

- Transport Layer Security (TLS)。TLS 1.2 が必須で、TLS 1.3 をお勧めします。
- DHE (楕円ディフィー・ヘルマン鍵共有) や ECDHE (楕円曲線ディフィー・ヘルマン鍵共有) などの完全前方秘匿性 (PFS) による暗号スイート。これらのモードはJava 7 以降など、ほとんどの最新システムでサポートされています。

また、リクエストにはアクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。またはAWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

Deadline Cloud は、 AWS PrivateLink Virtual Private Cloud (VPC) エンドポイントポリシーの使用をサポートしていません。エンドポイントへのフルアクセスを許可する AWS PrivateLink デフォルトのポリシーを使用します。詳細については、 AWS PrivateLink ユーザーガイドの「デフォルトのエンドポイントポリシー」を参照してください。

Deadline Cloud の設定と脆弱性の分析

AWS は、ゲストオペレーティングシステム (OS) やデータベースのパッチ適用、ファイアウォール設定、ディザスタリカバリなどの基本的なセキュリティタスクを処理します。これらの手順は適切な第三者によって確認され、証明されています。詳細については、以下のリソース を参照してください。

- 責任共有モデル
- Amazon Web Services: セキュリティプロセスの概要 (ホワイトペーパー)

AWS Deadline Cloud は、サービスマネージドフリートまたはカスタマーマネージドフリートのタスクを管理します。

- サービスマネージドフリートの場合、Deadline Cloud はゲストオペレーティングシステムを管理します。
- カスタマーマネージドフリートの場合は、オペレーティングシステムを管理する責任があります。

AWS Deadline Cloud の設定と脆弱性の分析の詳細については、「」を参照してください。

- Deadline Cloud のセキュリティのベストプラクティス

サービス間での不分別な代理処理の防止

混乱した代理問題は、アクションを実行するためのアクセス許可を持たないエンティティが、より特権のあるエンティティにアクションの実行を強制できてしまう場合に生じる、セキュリティ上の問題です。では AWS、サービス間のなりすましにより、混乱した代理問題が発生する可能性があります。サービス間でのなりすましは、1 つのサービス (呼び出し元サービス) が、別のサービス (呼び出し対象サービス) を呼び出すときに発生する可能性があります。呼び出し元サービスは、本来ならアクセスすることが許可されるべきではない方法でその許可を使用して、別のお客様のリソースに対する処理を実行するように操作される場合があります。これを防ぐため、 AWS では、アカウントのリ

ソースへのアクセス権が付与されたサービスプリンシパルで、すべてのサービスのデータを保護するために役立つツールを提供しています。

リソースポリシーで aws:SourceArnおよび aws:SourceAccount グローバル条件コンテキストキーを使用して、 がリソースに別のサービス AWS Deadline Cloud に付与するアクセス許可を制限することをお勧めします。クロスサービスアクセスにリソースを 1 つだけ関連付けたい場合は、`aws:SourceArn` を使用します。そのアカウント内のリソースをクロスサービスの使用に関連付けることを許可する場合は、`aws:SourceAccount` を使用します。

「混乱した代理」問題から保護するための最も効果的な方法は、リソースの完全なAmazon リソースネーム (ARN)を指定しながら、グローバル条件コンテキストキー `aws:SourceArn` を使用することです。リソースの完全な ARN が不明な場合や、複数のリソースを指定する場合には、グローバルコンテキスト条件キー `aws:SourceArn` で、ARN の未知部分を示すためにワイルドカード文字 (*) を使用します。例えば、`arn:aws:awsdeadlinecloud:*:123456789012:*`。

`aws:SourceArn` の値に Amazon S3 バケット ARN などのアカウント ID が含まれていない場合は、両方のグローバル条件コンテキストキーを使用して、アクセス許可を制限する必要があります。

次の例は、 で `aws:SourceArn`および `aws:SourceAccount` グローバル条件コンテキストキーを使用して、混乱した代理問題 Deadline Cloud を防ぐ方法を示しています。

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "awsdeadlinecloud.amazonaws.com"
    },
    "Action": "awsdeadlinecloud:ActionName",
    "Resource": [
      "*"
    ],
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:awsdeadlinecloud:*:123456789012:*"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
```

```
}
```

インターフェイスエンドポイント (AWS PrivateLink) AWS Deadline Cloud を使用した へのアクセス

を使用して AWS PrivateLink 、VPC と の間にプライベート接続を作成できます AWS Deadline Cloud。インターネットゲートウェイ、NAT デバイス、VPN 接続、または AWS Direct Connect 接続を使用せずに、VPC 内にある Deadline Cloud かのように にアクセスできます。VPC のインスタンスは、パブリック IP アドレスがなくても Deadline Cloudにアクセスできます。

このプライベート接続を確立するには、 AWS PrivateLinkを利用したインターフェイスエンドポイントを作成します。インターフェイスエンドポイントに対して有効にする各サブネットにエンドポイントネットワークインターフェイスを作成します。これらは、 Deadline Cloud宛てのトラフィックのエントリポイントとして機能するリクエスタ管理型ネットワークインターフェイスです。

Deadline Cloud には、デュアルスタックのエンドポイントも用意されています。デュアルスタックエンドポイントは、IPv6 および IPv4 経由のリクエストをサポートします。

詳細については「 AWS PrivateLink Guide (AWS PrivateLink ガイド)」の「Access an AWS のサービス using an interface VPC endpoint (インターフェイス VPC エンドポイントを使用して にアクセスする)」を参照してください。

に関する考慮事項 Deadline Cloud

のインターフェイスエンドポイントを設定する前に Deadline Cloud、「 AWS PrivateLink ガイド」の「インターフェイス VPC エンドポイントを使用して AWS サービスにアクセスする」を参照してください。

Deadline Cloud は、インターフェイスエンドポイントを介したすべての API アクションの呼び出しをサポートしています。

デフォルトでは、 へのフルアクセス Deadline Cloud はインターフェイスエンドポイントを介して許可されます。または、セキュリティグループをエンドポイントネットワークインターフェイスに関連付けて、インターフェイスエンドポイント Deadline Cloud 経由で へのトラフィックを制御することもできます。

Deadline Cloud は VPC エンドポイントポリシーもサポートしています。詳細については、『AWS PrivateLink ガイド』の「Control access to VPC endpoints using endpoint policies (エンドポイントポリシーを使用して VPC エンドポイントへのアクセスをコントロールする)」を参照してください。

Deadline Cloud エンドポイント

Deadline Cloud は 4 つのエンドポイントを使用してサービスにアクセスします AWS PrivateLink 。2 つは IPv4 用、2 つは IPv6 用です。

ワーカーは`scheduling.deadline.region.amazonaws.com`エンドポイントを使用して、キューからタスクを取得し、進捗状況を報告し Deadline Cloud、タスク出力を送り返します。カスタマーマネージドフリートを使用している場合、管理オペレーションを使用しない限り、作成する必要があるのはスケジューリングエンドポイントだけです。たとえば、ジョブがより多くのジョブを作成する場合は、管理エンドポイントを有効にして `CreateJob`オペレーションを呼び出す必要があります。

Deadline Cloud モニターは を使用して、キューとフリートの作成と変更、ジョブ、ステップ、タスクのリストの取得など、ファーム内のリソースmanagement.deadline.*region*.amazonaws.comを管理します。

Deadline Cloud には、次の AWS サービスエンドポイントのエンドポイントも必要です。

- Deadline Cloud は AWS STS を使用してワーカーを認証し、ワーカーがジョブアセットにアクセスできるようにします。詳細については AWS STS、「 AWS Identity and Access Management ユーザーガイド」の「IAM の一時的なセキュリティ認証情報」を参照してください。
- インターネット接続のないサブネットにカスタマーマネージドフリートを設定する場合は、ワーカーがログを書き込めるように、Amazon CloudWatch Logs の VPC エンドポイントを作成する必要があります。詳細については、「Amazon CloudWatch によるモニタリング」を参照してください。
- ジョブアタッチメントを使用する場合は、ワーカーがアタッチメントにアクセスできるように、Amazon Simple Storage Service (Amazon S3) の VPC エンドポイントを作成する必要があります。詳細については、「 のジョブアタッチメント Deadline Cloud」を参照してください。

のエンドポイントを作成する Deadline Cloud

Amazon VPC コンソールまたは AWS Command Line Interface () Deadline Cloud を使用して、 のインターフェイスエンドポイントを作成できますAWS CLI。詳細については、「AWS PrivateLink ガイド」の「インターフェイスエンドポイントを作成」を参照してください。

次のサービス名 Deadline Cloud を使用して、 の管理エンドポイントとスケジューリングエンドポイントを作成します。*region* をデプロイした AWS リージョン に置き換えます Deadline Cloud。

```
com.amazonaws.region.deadline.management
```

```
com.amazonaws.region.deadline.scheduling
```

Deadline Cloud はデュアルスタックのエンドポイントをサポートしています。

インターフェイスエンドポイントのプライベート DNS を有効にすると、デフォルトのリージョン DNS 名 Deadline Cloud を使用して に API リクエストを行うことができます。たとえば、ワーカーオペレーション`scheduling.deadline.us-east-1.amazonaws.com`の場合は 、その他すべてのオペレーション`management.deadline.us-east-1.amazonaws.com`の場合は です。

また、次のサービス名 AWS STS を使用して のエンドポイントを作成する必要があります。

```
com.amazonaws.region.sts
```

カスタマーマネージドフリートがインターネット接続のないサブネット上にある場合は、次のサービス名を使用して CloudWatch Logs エンドポイントを作成する必要があります。

```
com.amazonaws.region.logs
```

ジョブアタッチメントを使用してファイルを転送する場合は、次のサービス名を使用して Amazon S3 エンドポイントを作成する必要があります。

```
com.amazonaws.region.s3
```

Deadline Cloud のセキュリティのベストプラクティス

AWS Deadline Cloud (Deadline Cloud) には、独自のセキュリティポリシーを開発および実装する際に考慮すべき多くのセキュリティ機能が用意されています。以下のベストプラクティスは一般的なガイドラインであり、完全なセキュリティソリューションを説明するものではありません。これらのベストプラクティスはお客様の環境に適切ではないか、十分ではない場合があるため、これらは指示ではなく、有用な考慮事項と見なしてください。

> **Note**
>
> 多くのセキュリティトピックの重要性の詳細については、「責任共有モデル」を参照してください。

データ保護

データ保護の目的で、 AWS Identity and Access Management (IAM) を使用して AWS アカウント 認証情報を保護し、個々のアカウントを設定することをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 が必須で、TLS 1.3 をお勧めします。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。
- AWS 暗号化ソリューションと、 内のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。
- Amazon S3 (Amazon Simple Storage Service) に保存されている個人情報の発見と保護を支援する Amazon Macie などのアドバンストマネージドセキュリティサービスを使用します。
- コマンドラインインターフェイスまたは API を使用して AWS にアクセスするときに FIPS 140-2 検証済みの暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-2」 を参照してください。

顧客のアカウント番号などの機密の識別情報は、[Name] (名前)フィールドなどの自由形式のフィールドに配置しないことを強くお勧めします。これは、コンソール、API、 AWS CLIまたは SDK AWS のサービス を使用して AWS Deadline Cloud または他の を使用する場合も同様です。 AWS SDKs Deadline Cloud または他のサービスに入力したデータは、診断ログに取り込まれる可能性があります。外部サーバーへの URL を指定するときは、そのサーバーへのリクエストを検証するための認証情報を URL に含めないでください。

AWS Identity and Access Management アクセス許可

ユーザー、 AWS Identity and Access Management (IAM) ロール、およびユーザーに最小権限を付与して、 AWS リソースへのアクセスを管理します。 AWS アクセス認証情報を作成、配布、ローテーション、および取り消すための認証情報管理ポリシーと手順を確立します。詳細については、「IAM ユーザーガイド」の「IAM のベストプラクティス」を参照してください。

ユーザーおよびグループとしてジョブを実行する

Deadline Cloud でキュー機能を使用する場合、OS ユーザーがキューのジョブに対する最小特権のアクセス許可を持つように、オペレーティングシステム (OS) ユーザーとそのプライマリグループを指定するのがベストプラクティスです。

「ユーザーとして実行」 (およびグループ) を指定すると、キューに送信されたジョブのプロセスは、その OS ユーザーを使用して実行され、そのユーザーの関連する OS アクセス許可を継承します。

フリートとキューの設定を組み合わせて、セキュリティ体制を確立します。キュー側では、「ユーザーとして実行されるジョブ」と IAM ロールを指定して、キューのジョブに OS と AWS アクセス許可を使用できます。フリートは、特定のキューに関連付けられているときにキュー内でジョブを実行するインフラストラクチャ (ワーカーホスト、ネットワーク、マウントされた共有ストレージ) を定義します。ワーカーホストで使用可能なデータは、1 つ以上の関連付けられたキューのジョブによってアクセスされる必要があります。ユーザーまたはグループを指定すると、ジョブ内のデータを他のキュー、インストールされている他のソフトウェア、またはワーカーホストにアクセスできる他のユーザーから保護できます。キューにユーザーがない場合、キューはエージェントユーザーとして実行され、任意のキューユーザーを偽装 (sudo) できます。このようにして、ユーザーのないキューは権限を別のキューにエスカレートできます。

ネットワーク

トラフィックが傍受またはリダイレクトされないようにするには、ネットワークトラフィックをルーティングする方法と場所を保護することが重要です。

ネットワーク環境は、次の方法で保護することをお勧めします。

- Amazon Virtual Private Cloud (Amazon VPC) サブネットルートテーブルを保護して、IP レイヤートラフィックのルーティング方法を制御します。
- ファームまたはワークステーションのセットアップで Amazon Route 53 (Route 53) を DNS プロバイダーとして使用している場合は、Route 53 API への安全なアクセスを確保します。
- オンプレミスのワークステーションやその他のデータセンターを使用して AWS の外部で Deadline Cloud に接続する場合は、オンプレミスのネットワークインフラストラクチャを保護します。これには、ルーター、スイッチ、その他のネットワークデバイスの DNS サーバーとルートテーブルが含まれます。

ジョブとジョブデータ

Deadline Cloud ジョブは、ワーカーホストのセッション内で実行されます。各セッションはワーカーホストで 1 つ以上のプロセスを実行します。通常、出力を生成するにはデータを入力する必要があります。

このデータを保護するには、キューを使用してオペレーティングシステムユーザーを設定できます。ワーカーエージェントは、キュー OS ユーザーを使用してセッションサブプロセスを実行します。これらのサブプロセスは、キュー OS ユーザーのアクセス許可を継承します。

これらのサブプロセスアクセスのデータへのアクセスを保護するために、ベストプラクティスに従うことをお勧めします。詳細については、責任共有モデルを参照してください。

ファーム構造

Deadline Cloud フリートとキューは、さまざまな方法で配置できます。ただし、特定の配置にはセキュリティ上の影響があります。

ファームは、フリート、キュー、ストレージプロファイルなど、他のファームと Deadline Cloud リソースを共有できないため、最も安全な境界の 1 つです。ただし、ファーム内で外部 AWS リソースを共有できるため、セキュリティの境界が侵害されます。

適切な設定を使用して、同じファーム内のキュー間にセキュリティ境界を確立することもできます。

次のベストプラクティスに従って、同じファームに安全なキューを作成します。

- フリートを同じセキュリティ境界内のキューにのみ関連付けます。次の点に注意してください:
  - ワーカーホストでジョブが実行された後、一時ディレクトリやキューユーザーのホームディレクトリなどにデータが残される可能性があります。
  - ジョブの送信先のキューに関係なく、同じ OS ユーザーがサービス所有のフリートワーカーホストですべてのジョブを実行します。
  - ジョブがワーカーホストで実行されているプロセスを離れ、他のキューのジョブが実行中の他のプロセスを監視できる場合があります。
- 同じセキュリティ境界内のキューのみが、ジョブアタッチメントの Amazon S3 バケットを共有していることを確認します。
- 同じセキュリティ境界内のキューのみが OS ユーザーを共有していることを確認します。
- ファームに統合されている他の AWS リソースを境界に保護します。

ジョブアタッチメントキュー

ジョブアタッチメントは、Amazon S3 バケットを使用するキューに関連付けられています。

- ジョブアタッチメントは、Amazon S3 バケットのルートプレフィックスとの間で書き込みおよび読み取りを行います。`CreateQueue` API コールでこのルートプレフィックスを指定します。
- バケットには対応する がありQueue Role、キューユーザーにバケットとルートプレフィックスへのアクセスを許可するロールを指定します。キューを作成するときは、ジョブアタッチメントバケットとルートプレフィックスとともに `Queue Role` Amazon リソースネーム (ARN) を指定します。
- `AssumeQueueRoleForRead`、、および `AssumeQueueRoleForWorker` API オペレーションへの認可された呼び出しはAssumeQueueRoleForUser、 の一時的なセキュリティ認証情報のセットを返しますQueue Role。

キューを作成し、Amazon S3 バケットとルートプレフィックスを再利用すると、情報が権限のない当事者に開示されるリスクがあります。例えば、QueueA と QueueB は同じバケットとルートプレフィックスを共有します。安全なワークフローでは、ArtistA は QueueA にアクセスできますが、QueueB にはアクセスできません。ただし、複数のキューがバケットを共有する場合、ArtistA は QueueB データ内のデータにアクセスできます。 QueueA

コンソールは、デフォルトで安全なキューを設定します。キューが共通のセキュリティ境界の一部でない限り、Amazon S3 バケットとルートプレフィックスの個別の組み合わせがあることを確認します。

キューを分離するには、バケットとルートプレフィックスへのキューアクセスのみを許可するQueue Roleように を設定する必要があります。次の例では、各########をリソース固有の情報に置き換えます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:ListBucket",
        "s3:GetBucketLocation"
      ],
      "Effect": "Allow",
```

```
      "Resource": [
        "arn:aws:s3:::JOB_ATTACHMENTS_BUCKET_NAME",
        "arn:aws:s3:::JOB_ATTACHMENTS_BUCKET_NAME/JOB_ATTACHMENTS_ROOT_PREFIX/*"
      ],
      "Condition": {
        "StringEquals": { "aws:ResourceAccount": "ACCOUNT_ID" }
      }
    },
    {
      "Action": ["logs:GetLogEvents"],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:REGION:ACCOUNT_ID:log-group:/aws/deadline/FARM_ID/*"
    }
  ]
}
```

また、ロールに信頼ポリシーを設定する必要があります。次の例では、########テキストをリソース固有の情報に置き換えます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": ["sts:AssumeRole"],
      "Effect": "Allow",
      "Principal": { "Service": "deadline.amazonaws.com" },
      "Condition": {
        "StringEquals": { "aws:SourceAccount": "ACCOUNT_ID" },
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:deadline:REGION:ACCOUNT_ID:farm/FARM_ID"
        }
      }
    },
    {
      "Action": ["sts:AssumeRole"],
      "Effect": "Allow",
      "Principal": { "Service": "credentials.deadline.amazonaws.com" },
      "Condition": {
        "StringEquals": { "aws:SourceAccount": "ACCOUNT_ID" },
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:deadline:REGION:ACCOUNT_ID:farm/FARM_ID"
        }
      }
```

```
    }
  ]
}
```

カスタムソフトウェア Amazon S3 バケット

に次のステートメントを追加してQueue Role、Amazon S3 バケット内のカスタムソフトウェアにアクセスできます。次の例では、*Software_BUCKET_NAME* を S3 バケットの名前に置き換えます。

```
"Statement": [
    {
        "Action": [
            "s3:GetObject",
            "s3:ListBucket"
        ],
        "Effect": "Allow",
        "Resource": [
            "arn:aws:s3:::SOFTWARE_BUCKET_NAME",
            "arn:aws:s3:::SOFTWARE_BUCKET_NAME/*"
        ]
    }
]
```

Amazon S3 セキュリティのベストプラクティスの詳細については、Amazon Simple Storage Service ユーザーガイドの「Amazon Amazon S3 のセキュリティのベストプラクティス」を参照してください。

ワーカーホスト

ワーカーホストを保護して、各ユーザーが割り当てられたロールに対してのみオペレーションを実行できるようにします。

ワーカーホストを保護するには、次のベストプラクティスをお勧めします。

- ホスト設定スクリプトを使用すると、ワーカーのセキュリティとオペレーションが変更される可能性があります。設定が正しくないと、ワーカーが不安定になったり、動作が停止したりする可能性があります。このような障害をデバッグするのはお客様の責任です。
- これらのキューに送信されたジョブが同じセキュリティ境界内にある場合を除き、複数のキューで同じjobRunAsUser値を使用しないでください。

- ワーカーエージェントが実行する OS ユーザーの名前jobRunAsUserにキューを設定しないでください。
- 目的のキューワークロードに必要な最小特権の OS アクセス許可をキューユーザーに付与します。エージェントプログラムファイルやその他の共有ソフトウェアを操作するためのファイルシステムの書き込みアクセス許可がないことを確認します。
- のルートユーザーLinuxと Administrator がWindowsアカウントを所有し、ワーカーエージェントプログラムファイルを変更できることを確認します。
- Linux ワーカーホストでは、ワーカーエージェントユーザーがキューユーザーとしてプロセスを起動/etc/sudoersできるようにするumaskオーバーライドを で設定することを検討してください。この設定は、他のユーザーがキューに書き込まれたファイルにアクセスできないようにするのに役立ちます。
- 信頼できる個人にワーカーホストへの最小特権アクセスを付与します。
- ローカル DNS オーバーライド設定ファイル (/etc/hosts Linuxおよび C:\Windows\system32\etc\hosts) へのアクセス許可を制限Windowsし、ワークステーションとワーカーホストオペレーティングシステムにテーブルをルーティングします。
- ワークステーションとワーカーホストオペレーティングシステムの DNS 設定へのアクセス許可を制限します。
- オペレーティングシステムとインストールされているすべてのソフトウェアに定期的にパッチを適用します。このアプローチには、送信者、アダプター、ワーカーエージェント、OpenJDパッケージなど、Deadline Cloud で特に使用されるソフトウェアが含まれます。
- Windows キュー に強力なパスワードを使用しますjobRunAsUser。
- キュー のパスワードを定期的にローテーションしますjobRunAsUser。
- Windows パスワードシークレットへの最小特権アクセスを確保し、未使用のシークレットを削除します。
- キューに、今後実行するスケジュールコマンドのjobRunAsUserアクセス許可を与えないでください。
  - でLinux、これらのアカウントによる cronおよび へのアクセスを拒否しますat。
  - でWindows、これらのアカウントによるWindowsタスクスケジューラへのアクセスを拒否します。

> **Note**
>
> オペレーティングシステムとインストール済みソフトウェアに定期的にパッチを適用する重要性の詳細については、「 責任共有モデル」を参照してください。

ホスト設定スクリプト

- ホスト設定スクリプトを使用すると、ワーカーのセキュリティとオペレーションが変更される可能性があります。設定が正しくないと、ワーカーが不安定になったり、動作が停止したりする可能性があります。このような障害をデバッグするのはお客様の責任です。

ワークステーション

Deadline Cloud にアクセスできるワークステーションを保護することが重要です。このアプローチは、Deadline Cloud に送信するジョブが、 に請求される任意のワークロードを実行できないようにするのに役立ちます AWS アカウント。

アーティストワークステーションを保護するために、次のベストプラクティスをお勧めします。詳細については、 責任共有モデルを参照してください。

- Deadline Cloud を含む AWS、 へのアクセスを提供する永続的な認証情報を保護します。詳細については、「 IAM ユーザーガイド」の「IAM ユーザーのアクセスキーの管理」を参照してください。
- 信頼できる安全なソフトウェアのみをインストールします。
- ユーザーは ID プロバイダーとフェデレーションし、一時的な認証情報 AWS を使用して にアクセスする必要があります。
- Deadline Cloud 送信者プログラムファイルに対する安全なアクセス許可を使用して、改ざんを防止します。
- 信頼できる個人にアーティストワークステーションへの最小特権アクセスを付与します。
- Deadline Cloud Monitor を通じて取得した送信者とアダプターのみを使用してください。
- ローカル DNS オーバーライド設定ファイル (`/etc/hosts` Linuxおよび 、 `C:\Windows\system32\etc\hosts`) へのアクセス許可を制限WindowsしmacOS、ワークステーションとワーカーホストオペレーティングシステムでテーブルをルーティングします。
- ワークステーションとワーカーホストオペレーティングシステム`/etc/resolve.conf`のアクセス許可を に制限します。

- オペレーティングシステムとインストールされているすべてのソフトウェアに定期的にパッチを適用します。このアプローチには、送信者、アダプター、ワーカーエージェント、OpenJDパッケージなど、Deadline Cloud で特に使用されるソフトウェアが含まれます。

ダウンロードしたソフトウェアの信頼性を検証する

インストーラをダウンロードした後、ファイルの改ざんから保護するために、ソフトウェアの信頼性を確認します。この手順は、 Windowsおよび Linuxシステムの両方で機能します。

Windows

ダウンロードしたファイルの真正性を検証するには、次の手順を実行します。

1. 次のコマンドで、 を、検証するファイル*file*に置き換えます。例えば、 ***C:\PATH\TO\MY\DeadlineCloudSubmitter-windows-x64-installer.exe*** 。また、 を、インストールされている SignTool SDK のバージョン*signtool-sdk-version*に置き換えます。例えば、**10.0.22000.0**。

   ```
   "C:\Program Files (x86)\Windows Kits\10\bin\signtool-sdk-version\x86\signtool.exe" verify /vfile
   ```

2. たとえば、次のコマンドを実行して、Deadline Cloud 送信者インストーラファイルを確認できます。

   ```
   "C:\Program Files (x86)\Windows Kits\10\bin\10.0.22000.0\x86\signtool.exe" verify /v DeadlineCloudSubmitter-windows-x64-installer.exe
   ```

リナックス

ダウンロードしたファイルの真正性を検証するには、 gpg コマンドラインツールを使用します。

1. 次のコマンドを実行してOpenPGPキーをインポートします。

   ```
    gpg --import --armor <<EOF
   -----BEGIN PGP PUBLIC KEY BLOCK-----

   mQINBGX6GQsBEADduUtJgqSXI+q76O6fsFwEYKmbnlyL0xKvlq32EZuyv0otZo5L
   le4m5Gg52AzrvPvDiUTLooAlvYeozaYyirIGsK08Ydz0Ftdjroiuh/mw9JSJDJRI
   rnRn5yKet1JFezkjopA3pjsTBP6lW/mb1bDBDEwwwtH0x9lV7A03FJ9T7Uzu/qSh
   ```

```
qO/UYdkafro3cPASvkqgDt2tCvURfBcUCAjZVFcLZcVD5iwXacxvKsxxS/e7kuVV
I1+VGT8Hj8XzWYhjCZxOLZk/fvpYPMyEEujN0fYUp6RtMIXve0C9awwMCy5nBG2J
eE2Ol5DsCpTaBd4Fdr3LWcSs8JFA/YfP9auL3NczOozPoVJt+fw8CBlVIXO0J7l5
hvHDjcC+5v0wxqAlMG6+f/SX7CT8FXK+L3iOJ5gBYUNXqHSxUdv8kt76/KVmQa1B
Akl+MPKpMq+lhw++S3G/lXqwWaDNQbRRw7dSZHymQVXvPp1nsqc3hV7KlOM+6s6g
1g4mvFY4lf6DhptwZLWyQXU8rBQpojvQfiSmDFrFPWFi5BexesuVnkGIolQoklKx
AVUSdJPVEJCteyy7td4FPhBaSqT5vW3+ANbr9b/uoRYWJvn17dN0cc9HuRh/Ai+I
nkfECo2WUDLZ0fEKGjGyFX+todWvJXjvc5kmE9Ty5vJp+M9Vvb8jd6t+mwARAQAB
tCxBV1MgRGVhZGxpbmUgQ2xvdWQgPGF3cy1kZWFkbGluZUBhbWF6b24uY29tPokC
VwQTAQgAQRYhBLhAwIwpqQeWoHH6pfbNPOa3bzzvBQJl+hkLAxsvBAUJA8JnAAUL
CQgHAgIiAgYVCgkICwIDFgIBAh4HAheAAAoJEPbNPOa3bzzvKswQAJXzKSAY8sY8
F6Eas2oYwIDDdDurs8FiEnFghjUEO6MTt9AykF/jw+CQg2UzFtEyObHBymhgmhXE
3buVeom96tgM3ZDfZu+sxi5pGX6oAQnZ6riztN+VpkpQmLgwtMGpSMLl3KLwnv2k
WK8mrR/fPMkfdaewB7A6RIUYiW33GAL4KfMIs8/vIwIJw99NxHpZQVoU6dFpuDtE
1OuxGcCqGJ7mAmo6H/YawSNp2Ns80gyqIKYo7o3LJ+WRroIRlQyctq8gnR9JvYXX
42ASqLq5+OXKo4qh81blXKYqtc176BbbSNFjWnzIQgKDgNiHFZCdcOVgqDhwO15r
NICbqqwwNLj/Fr2kecYx180Ktpl0jOOw5IOyh3bf3MVGWnYRdjvA1v+/CO+55N4g
z0kf50Lcdu5RtqV10XBCifn28pecqPaSdYcssYSRl5DLiFktGbNzTGcZZwITTKQc
af8PPdTGtnnb6P+cdbW3bt9MVtN5/dgSHLThnS8MPEuNCtkTnpXshuVuBGgwBMdb
qUC+HjqvhZzbwns8dr5WI+6HWNBFgGANn6ageYl58vVp0UkuNP8wcWjRARciHXZx
ku6W2jPTHDWGNrBQO2Fx7fd2QYJheIPPAShHcfJO+xgWCof45D0vAxAJ8gGg9Eq+
gFWhsx4NSHn2gh1gDZ41Ou/4exJ1lwPM
=uVaX
-----END PGP PUBLIC KEY BLOCK-----
EOF
```

2. OpenPGP キーを信頼するかどうかを決定します。上記のキーを信頼するかどうかを決定する際に考慮すべき要素には、次のようなものがあります。

   - このウェブサイトから GPG キーを取得するために使用したインターネット接続は安全です。
   - このウェブサイトにアクセスするデバイスは安全です。
   - AWS は、このウェブサイトでOpenPGPパブリックキーのホスティングを保護するための対策を講じています。

3. OpenPGP キーを信頼する場合は、次の例gpgのように で信頼するようにキーを編集します。

```
$ gpg --edit-key 0xB840C08C29A90796A071FAA5F6CD3CE6B76F3CEF

    gpg (GnuPG) 2.0.22; Copyright (C) 2013 Free Software Foundation, Inc.
    This is free software: you are free to change and redistribute it.
    There is NO WARRANTY, to the extent permitted by law.
```

```
   pub  4096R/4BF0B8D2  created: 2023-06-23  expires: 2025-06-22  usage: SCEA
                        trust: unknown       validity: unknown
   [ unknown] (1). AWS Deadline Cloud example@example.com

   gpg> trust
   pub  4096R/4BF0B8D2  created: 2023-06-23  expires: 2025-06-22  usage: SCEA
                        trust: unknown       validity: unknown
   [ unknown] (1). AWS Deadline Cloud aws-deadline@amazon.com

   Please decide how far you trust this user to correctly verify other users'
keys
   (by looking at passports, checking fingerprints from different sources,
etc.)

     1 = I don't know or won't say
     2 = I do NOT trust
     3 = I trust marginally
     4 = I trust fully
     5 = I trust ultimately
     m = back to the main menu

   Your decision? 5
   Do you really want to set this key to ultimate trust? (y/N) y

   pub  4096R/4BF0B8D2  created: 2023-06-23  expires: 2025-06-22  usage: SCEA
                        trust: ultimate      validity: unknown
   [ unknown] (1). AWS Deadline Cloud aws-deadline@amazon.com
   Please note that the shown key validity is not necessarily correct
   unless you restart the program.

   gpg> quit
```

4. Deadline Cloud 送信者インストーラを検証する

   Deadline Cloud 送信者インストーラを確認するには、次の手順を実行します。

   a. Deadline Cloud コンソールのダウンロードページに戻り、Deadline Cloud 送信者インストーラの署名ファイルをダウンロードします。

   b. 以下を実行して、Deadline Cloud 送信者インストーラの署名を確認します。

```
gpg --verify ./DeadlineCloudSubmitter-linux-x64-installer.run.sig ./
DeadlineCloudSubmitter-linux-x64-installer.run
```

5. Deadline Cloud モニターを確認する

> **Note**
>
> 署名ファイルまたはプラットフォーム固有の方法を使用して、Deadline Cloud モニターのダウンロードを確認できます。プラットフォーム固有の方法については、Linux (Debian)タブ、 Linux (RPM) タブ、またはダウンロードしたファイルタイプに基づく Linux (AppImage) タブを参照してください。

署名ファイルを使用して Deadline Cloud Monitor デスクトップアプリケーションを検証するには、次の手順を実行します。

a. Deadline Cloud コンソールのダウンロードページに戻り、対応する .sig ファイルをダウンロードして、 を実行します。

   .deb の場合：

   ```
   gpg --verify ./deadline-cloud-monitor_<APP_VERSION>_amd64.deb.sig ./
   deadline-cloud-monitor_<APP_VERSION>_amd64.deb
   ```

   .rpm の場合：

   ```
   gpg --verify ./deadline-cloud-monitor_<APP_VERSION>_x86_64.deb.sig ./
   deadline-cloud-monitor_<APP_VERSION>_x86_64.rpm
   ```

   .AppImage の場合：

   ```
   gpg --verify ./deadline-cloud-monitor_<APP_VERSION>_amd64.AppImage.sig ./
   deadline-cloud-monitor_<APP_VERSION>_amd64.AppImage
   ```

b. 出力が次のようになっていることを確認します。

   ```
   gpg: Signature made Mon Apr 1 21:10:14 2024 UTC

   gpg: using RSA key B840C08C29A90796A071FAA5F6CD3CE6B7
   ```

出力に というフレーズが含まれている場合`Good signature from "AWS Deadline Cloud"`、署名が正常に検証され、Deadline Cloud モニターのインストールスクリプトを実行できます。

リナックス (AppImage)

Linux .AppImage バイナリを使用するパッケージを確認するには、まず Linuxタブのステップ 1～3 を完了してから、次の手順を実行します。

1. GitHub の AppImageUpdate ページから、 validate-x86_64.AppImage ファイルをダウンロードします。
2. ファイルをダウンロードした後、実行権限を追加するには、次のコマンドを実行します。

```
chmod a+x ./validate-x86_64.AppImage
```

3. 実行アクセス許可を追加するには、次のコマンドを実行します。

```
chmod a+x ./deadline-cloud-monitor_<APP_VERSION>_amd64.AppImage
```

4. Deadline Cloud モニターの署名を確認するには、次のコマンドを実行します。

```
./validate-x86_64.AppImage ./deadline-cloud-monitor_<APP_VERSION>_amd64.AppImage
```

出力に というフレーズが含まれている場合`Validation successful`、署名が正常に検証され、Deadline Cloud モニターのインストールスクリプトを安全に実行できることを意味します。

リナックス (Debian)

Linux .deb バイナリを使用するパッケージを確認するには、まず Linuxタブのステップ 1～3 を完了します。

dpkg は、ほとんどのdebianベースのLinuxディストリビューションのコアパッケージ管理ツールです。ツールを使用して .deb ファイルを検証できます。

1. Deadline Cloud コンソールのダウンロードページから、Deadline Cloud モニター .deb ファイルをダウンロードします。
2. *<APP_VERSION>* を、検証する .deb ファイルのバージョンに置き換えます。

```
dpkg-sig --verify deadline-cloud-monitor_<APP_VERSION>_amd64.deb
```

3. 出力は次のようになります。

```
ProcessingLinux deadline-cloud-monitor_<APP_VERSION>_amd64.deb...
GOODSIG _gpgbuilder B840C08C29A90796A071FAA5F6CD3C 171200
```

4. .deb ファイルを検証するには、 `GOODSIG` が出力に存在することを確認します。

リナックス (RPM)

Linux .rpm バイナリを使用するパッケージを確認するには、まず Linuxタブのステップ 1 ~ 3 を完了します。

1. Deadline Cloud コンソールのダウンロードページから、Deadline Cloud Monitor .rpm ファイルをダウンロードします。
2. *<APP_VERSION>* を、検証する .rpm ファイルのバージョンに置き換えます。

```
gpg --export --armor "Deadline Cloud" > key.pub
sudo rpm --import key.pub
rpm -K deadline-cloud-monitor-<APP_VERSION>-1.x86_64.rpm
```

3. 出力は次のようになります。

```
deadline-cloud-monitor-deadline-cloud-
monitor-<APP_VERSION>-1.x86_64.rpm-1.x86_64.rpm: digests signatures OK
```

4. .rpm ファイルを検証するには、 `digests signatures OK`が出力にあることを確認します。

ドキュメント履歴

次の表に、AWS Deadline Cloud デベロッパーガイドの各リリースにおける重要な変更点を示します。

変更	説明	日付
ワーカーホスト設定スクリプトを追加	昇格された権限で実行されるワーカーホスト設定スクリプトを使用するためのドキュメントを追加しました。詳細については、「管理者としてスクリプトを実行してワーカーを設定する」を参照してください。	2025 年 5 月 12 日
の使用に関するセキュリティセクションを更新しました AWS PrivateLink	AWS PrivateLink と新しいデュアルスタックエンドポイントを使用して Deadline Cloud にアクセスする手順を更新しました。詳細については、「インターフェイスエンドポイントを使用して Deadline Cloud にアクセスする」を参照してください。	2025 年 3 月 17 日
カスタマーマネージドフリート認証情報の更新	フリートの保護に関する詳細情報を提供するために、カスタマーマネージドフリートの認証情報を作成する手順を更新しました。詳細については、「AWS 認証情報の設定」を参照してください。	2025 年 2 月 10 日
ユーザーガイドのコンテンツを再編成しました	デベロッパー向けのコンテンツを ユーザーガイドから デベ	2025 年 1 月 6 日

	ロッパーガイドに移動しました。 • カスタマーマネージドフリートを作成する手順をユーザーガイドから新しいカスタマーマネージドフリートの章に移動しました。 • 使用ベースのライセンスと、サービスおよびカスタマーマネージドフリートでの独自のライセンスの使用に関する情報を含む新しいソフトウェアライセンスの使用の章を作成しました。 • CloudTrail、CloudWatch、EventBridge によるモニタリングの詳細をユーザーガイドからモニタリングの章に移動しました。	
conda パッケージを作成する	アプリケーションの conda パッケージを作成する方法に関する情報を追加しました。詳細については、「conda パッケージを作成する」を参照してください。	2024 年 8 月 29 日
新しいガイド	これは Deadline Cloud デベロッパーガイドの最初のリリースです。	2024 年 7 月 26 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。