aws

デベロッパーガイド

Deadline クラウド

Copyright © 2026 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 © 2026 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
Farm リソース 2
ジョブ実行リソース 3
その他の重要な概念と用語 5
アーキテクチャガイダンス 8
ジョブソース 10
インタラクティブワークフロー 10
自動ワークフロー 10
ジョブの送信 10
DCC との統合送信者 11
カスタムジョブ定義 11
アプリケーション管理 11
サービスマネージドフリートの Deadline クラウドマネージド conda チャネル (SMF) 12
セルフマネージド conda チャネル 12
カスタムアプリケーション管理 12
アプリケーションのライセンス 13
サービスマネージドフリートと使用ベースのライセンス 13
カスタマーマネージドフリートと使用量ベースのライセンス 13
カスタムライセンス 13
アセットアクセス 14
ジョブアタッチメント 14
カスタムストレージアクセス 15
ジョブのモニタリングと出力管理 15
Deadline Cloud モニター 15
カスタムモニターアプリケーション 16
自動モニタリングソリューション 16
ワーカーインフラストラクチャ管理 16
サービスマネージドフリート 16
カスタマーマネージドフリート 16
アーキテクチャの例 17
従来のプロダクションスタジオ 17
クラウドの Studio 20

ECommerceオートメーション 21
Whitelabel/OEM/B2C カスタマー 24
Deadline Cloud ワークロードとは 27
ワークロードが本番環境からどのように発生するか 27
ワークロードの構成要素 28
ワークロードの移植性 29
入門 32
ファームを作成する 32
次のステップ 37
ワーカーエージェントを実行する 37
次のステップ 39
ジョブの送信 39
simple_job サンプルを送信する 40
パラメータを使用して送信する 43
simple_file_job ジョブを作成する 44
次の手順 47
添付ファイルを含むジョブの送信 48
ジョブアタッチメントのキューを設定する 49
ジョブアタッチメントを使用して送信する 51
ジョブアタッチメントの保存方法 53
次の手順 57
サービスマネージドフリートを追加する 57
次の手順 60
ファームリソースをクリーンアップする 60
ジョブを構築する 64
ジョブバンドル 65
ジョブテンプレートの要素 68
タスクチャンキング 71
パラメータ値の要素 73
アセット参照要素 76
ジョブでのファイルの使用 79
サンプルプロジェクトインフラストラクチャ 80
ストレージプロファイルとパスマッピング 82
ジョブアタッチメント 90
ジョブを使用したファイルの送信 91
ジョブからの出力ファイルの取得 102

依存ステップでのファイルの使用 106
ジョブのリソース制限を作成する 108
制限の停止と削除 110
制限を作成する 111
制限とキューを関連付ける 111
制限が必要なジョブを送信する 112
ジョブを送信する 114
ターミナルから 114
スクリプトから 115
アプリケーション内から 116
スケジュールジョブ 118
フリートの互換性を確認する 118
フリートスケーリング 120
セッション 120
ステップの依存関係 123
ジョブの変更 124
カスタマーマネージドフリート 130
CMF を作成する 130
ワーカーホストのセットアップ 136
Python 環境を設定する 137
ワーカーエージェントをインストールする 137
ワーカーエージェントを設定する 139
ジョブのユーザーとグループを作成する 140
ワーカーホストの保護 143
アクセスを管理する 145
アクセス許可の付与 146
アクセスの取り消し 147
ジョブ用のソフトウェアをインストールする 147
DCC アダプターをインストールする 148
認証情報の設定 148
ネットワークの設定 152
ワーカーホストをテストする 152
AMI の作成 155
インスタンスを準備する 155
の構築 AMI 157
フリートインフラストラクチャを作成する 158

フリートの自動スケーリング 163
フリートヘルスチェック 168
サービスマネージドフリート 169
VPC リソースを SMF に接続する 169
VPC リソースエンドポイントの仕組み 170
前提条件 170
VPC リソースエンドポイントを設定する 171
VPC リソースへのアクセス 171
認証とセキュリティ 172
技術的考慮事項 172
トラブルシューティング 172
ジョブアタッチメント 173
ファイルシステムモードを選択する 173
転送パフォーマンスの最適化 174
ジョブ出力をダウンロードする 175
ワーカーにカスタムソフトウェアをデプロイして設定する 176
デプロイ方法を選択する 176
キュー環境を使用してジョブを設定する 177
ジョブ環境を制御する 178
ジョブにアプリケーションを提供する 194
S3 を使用して conda チャネルを作成する 197
パッケージをローカルでビルドおよびテストする 198
Amazon S3 conda チャネルにパッケージを発行する 202
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 206
conda チャネルをキュー環境に追加する 207
アプリケーションまたはプラグインの conda パッケージを作成する 208
の conda ビルドレシピを作成する Blender 211
の conda レシピを作成する Maya 216
MtoA プラグインの conda レシピを作成する 219
Deadline Cloud でパッケージビルドを自動化する 221
ホスト設定スクリプト 225
トラブルシューティング 228
ソフトウェアライセンスの使用 232
SMF フリートをライセンスサーバーに接続する 232
ステップ 1: キュー環境を設定する 233
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 243

ステップ 3: CloudFormation テンプレートのセットアップ 244
CMF フリートをライセンスエンドポイントに接続する 254
ステップ 1: セキュリティグループを作成する 255
ステップ 2: ライセンスエンドポイントを設定する 256
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 257
ステップ 4: ライセンスエンドポイントを削除する 260
AI エージェントの使用 262
モニタリング 265
CloudTrail ログ 266
Deadline Cloud CloudTrail のデータイベント 268
Deadline Cloud CloudTrail の管理イベント 270
Deadline Cloud イベントの例 273
CloudWatch によるモニタリング 274
CloudWatch メトリクス 275
推奨アラーム 278
を使用したイベントの管理 EventBridge 279
Deadline Cloud イベント 280
Deadline Cloud イベントの送信 281
イベントの詳細リファレンス 282
セッション統計の集計データのクエリ 297
集約リクエストの開始 297
結果を取得する 298
userID を使用したユーザーメタデータの取得 299
ユーザー ID をマッピングするには 299
ID ストア ID の検索 300
ユーザーマッピングの検証 301
その他のリソース 301
セキュリティ 302
データ保護 303
保管中の暗号化 304
転送中の暗号化 304
キー管理 305
ネットワーク間トラフィックのプライバシー 315
オプトアウト 315
Identity and Access Management 316
オーディエンス 317

アイデンティティを使用した認証 317
ポリシーを使用したアクセスの管理 319
Deadline Cloud と IAM の連携方法 320
アイデンティティベースのポリシーの例 326
AWS マネージドポリシー 336
サービスロール 340
トラブルシューティング 354
コンプライアンス検証 356
耐障害性 356
インフラストラクチャセキュリティ 356
設定と脆弱性の分析 357
サービス間での不分別な代理処理の防止 358
AWS PrivateLink 359
考慮事項 359
Deadline Cloud エンドポイント 360
エンドポイントの作成 361
制限されたネットワーク環境 362
AWS 許可リストの API エンドポイント 362
許可リストのウェブドメイン 362
許可リストを作成する環境固有のエンドポイント 363
セキュリティのベストプラクティス 364
データ保護 364
IAM アクセス許可 365
ユーザーおよびグループとしてジョブを実行する 365
ネットワーク 366
ジョブデータ 366
ファーム構造 366
ジョブアタッチメントキュー 367
カスタムソフトウェアバケット 370
ワーカーホスト 371
ホスト設定スクリプト 372
ワークステーション 372
ダウンロードしたソフトウェアを検証する 373
ドキュメント履歴 380
..... cccIxxxi

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 アイデンティティセンター 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。

Farm リソース

この図は、Deadline Cloud ファームリソースがどのように連携するかを示しています。

ファーム

ファームには、ジョブの送信と実行に関連する他のすべてのリソースが含まれています。ファームは互いに独立しているため、本番環境の分離に役立ちます。

キュー

キューは、関連付けられたフリートでスケジューリングするためのジョブを保持します。ユーザーはジョブをキューに送信し、キュー内の優先度とステータスを管理できます。ジョブを実行するには、キューをキューとフリートの関連付けを持つフリートに関連付ける必要があり、キューを複数のフリートに関連付けることができます。

フリート

フリートには、実行中のジョブのコンピューティング容量が含まれています。フリートは、サービスマネージド型でもカスタマーマネージド型でもかまいません。サービスマネージドフリートは Deadline Cloud で実行され、自動スケーリング、ライセンス、ソフトウェアアクセスなどの組

み込み機能が含まれています。カスタマーマネージドフリートは、Amazon EC2 インスタンスやオンプレミスサーバーなどの独自のコンピューティングリソースで実行されます。

予算

予算は、ジョブアクティビティの支出しきい値を設定し、しきい値に達したときにジョブのスケジュールの停止などのアクションを実行できます。

キュー環境

キュー環境は、ワークロード環境をセットアップまたは削除するために各ワーカーで実行されるスクリプトを定義します。環境変数の設定、ソフトウェアのインストール、アセットストレージの設定に役立ちます。

ストレージプロファイル

ストレージプロファイルは、ホストとワークステーションのグループの設定であり、ファイルシステム上のデータがどこにあるかを に指示します。Deadline Cloud は、 から送信Windowsされ、 で実行されるジョブなど、異なる設定のホストでジョブを実行するときに、ストレージプロファイルを使用してパスをマッピングしますLinux。

[制限]

制限により、フローティングライセンスなどの共有リソースの使用状況を追跡し、ジョブ間での割り当て方法を制御できます。制限は、キューと制限の関連付けを持つキューに関連付けられます。

モニタリング

モニターは Deadline Cloud Monitor ウェブアプリケーションの URL を設定し、エンドユーザーがジョブをモニタリングおよび管理できるようにします。ブラウザまたは Deadline Cloud Monitor デスクトップアプリケーションからアクセスできます。

ジョブ実行リソース

この図は、Deadline Cloud ジョブリソースがどのように連携するかを示しています。

ジョブ

ジョブは、ユーザーが Deadline Cloud に送信して、使用可能なワーカーでスケジュールおよび実行するための一連の作業です。ジョブは 3D シーンをレンダリングしたり、シミュレーションを実行したりできます。ジョブは、ランタイム環境とプロセス、およびジョブ固有のパラメータ

を定義する再利用可能なジョブテンプレートから作成されます。ジョブには、実行する作業を定義するステップとタスクが含まれており、優先順位、最大ワーカー数、再試行設定で設定できます。

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。

Step

ステップは、タスクパラメータ値を除いて、多くの同じタスクを実行するためのテンプレートを提供するジョブの一部です。ステップには他のステップへの依存関係があるため、シーケンシャル実行パスまたは並列実行パスを使用して複雑なワークフローを作成できます。レンダリングジョブでは、ステップは多くの場合、フレームをレンダリングするためのコマンドを定義し、フレーム番号をタスクパラメータとして使用します。

タスク

タスクは Deadline Cloud の最小の作業単位です。タスクはステップの一部であり、ワーカーによって実行され、ジョブの一部として実行する必要がある個々のオペレーションを表します。タスクは特定のパラメータで設定でき、その機能と可用性に基づいてワーカーに割り当てられます。レンダリングジョブでは、タスクが 1 つのフレームをレンダリングすることがよくあります。

ワーカー

ワーカーはフリートの一部であり、ジョブからタスクを実行します。ワーカーは、GPU アクセラレーター、CPU アーキテクチャ、オペレーティングシステムなどの特定の機能で設定できます。サービスマネージドフリートでは、フリートがスケールアウトおよびスケールインすると、ワーカーが自動的に作成されます。

インスタンス

フリートは CPU リソースにインスタンスを使用します。インスタンスは Amazon EC2 パフォーマンスインスタンスです。Deadline Cloud はオンデマンドインスタンスとスポットインスタンスを使用します。

オンデマンドインスタンス

オンデマンドインスタンスの料金は 2 番目で、長期的なコミットメントはなく、中断されません。

スポットインスタンス

スポットインスタンスは、割引価格で使用できる予約されていない容量ですが、オンデマンドリクエストによって中断される可能性があります。

待機して保存する

待機と保存機能を使用すると、ジョブのスケジュールが遅れて低コストになり、オンデマンドリクエストとスポットリクエストによって中断される可能性があります。Wait and Save は、Deadline Cloud のサービスマネージドフリートでのみ使用できます。

Wait and Save は、Deadline Cloud AWS でのビジュアルコンピューティングワークロードの実行を管理するためのものです。詳細については、AWS 「サービス条件」を参照してください。

セッション

セッションは、ジョブに対するワーカーの作業のシーケンスを表します。1 つのセッション中に、ワーカーに複数のタスクが割り当てられ、タスクが順番に実行されます。多くの場合、セッションには、タスクアクションを実行する前に環境を設定し、アセットをロードするセットアップアクションがあります。

セッションアクション

セッションアクションは、環境の設定、タスクの実行、アセットの同期など、セッション中に実行される特定のオペレーションを表します。

その他の重要な概念と用語

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。

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。これを使用して、アクティビティを制限して予算内にとどまることもできます。

Deadline Cloud クライアントライブラリ

オープンソースの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 (CLI) 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。Deadline Cloud には、Autodesk Maya、Blender、Maxon Cinema 4D などの多くの DCCs との統合が組み込まれているため、DCC 内からジョブを送信し、事前設定されたソフトウェアとライセンスを使用してサービスマネージドフリートでレンダリングできます。

ジョブアタッチメント

ジョブアタッチメントは、テクスチャ、3D モデル、ライティングリグなどのジョブの一部としてアセットをアップロードおよびダウンロードする Deadline Cloud 機能です。ジョブアタッチメントは Amazon S3 に保存されるため、共有ネットワークストレージが不要になります。

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。

Deadline Cloud 送信者

Deadline Cloud 送信者は、ユーザーが DCC 内からジョブを簡単に送信できるようにする DCC のプラグインです。

ライセンスエンドポイント

ライセンスエンドポイントは、サードパーティー製品の Deadline Cloud の使用ベースのライセンスを VPC 内で利用できるようにします。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。ライセンスエンドポイントはファームに接続されておらず、個別に使用できます。

[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、目的、所有者、環境など、さまざまな方法で AWS リソースを分類できます。

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。

Deadline クラウドアーキテクチャガイダンス

このトピックでは、Deadline Cloud を使用してワークロード用の信頼性、安全性、効率、費用対効果の高いレンダーファームを設計および構築するためのガイダンスとベストプラクティスを提供します。このガイダンスを使用すると、安定して効率的なワークロードを構築し、イノベーションに注力しながらコストを削減し、さらにカスタマーエクスペリエンスを向上させることができます。

このコンテンツは、最高技術責任者 (CTO)、アーキテクト、デベロッパー、および運用チームのメンバーを対象としています。

end-to-endのレンダリングワークフローには、ジョブ生成、アセットアクセス、ジョブモニタリングなど、プロセスの複数のレイヤーでソリューションが必要です。Deadline Cloud は、レンダリングプロセスのレイヤーごとに複数のソリューションを提供します。各レイヤーで Deadline Cloud のオプションから選択することで、ユースケースに合ったワークフローを設計できます。

レイヤーごとに、ユースケースに最適なアプローチを決定する必要があります。これらは厳密なシナリオ定義ではなく、Deadline Cloud を使用する唯一の方法ではありません。代わりに、これらは Deadline Cloud がビジネスやワークフローにどのように適合するかを理解するのに役立つ大まかな概念のセットです。Deadline Cloud ワークロードは、ジョブソース、ジョブ送信、アプリケーション管理、アプリケーションライセンス、アセットアクセス、出力管理、ワーカーインフラストラクチャ管理のレイヤーに分割できます。

一般的に、1 つのレイヤーの任意のシナリオを別のレイヤーの他のシナリオとmix-and-matchできます。ただし、以下に指定する特定の組み合わせは除きます。

Job Source
Interactive Workflow
Automated Workflow
Job Submission
Integrated Submitter
Custom Job Definition
Application Management
Conda Application Management
Custom Application Management
Application Licensing
Deadline Cloud UBL
Custom Licensing
Asset Access
Job Attachments
Custom Storage
Job Monitoring
Deadline Cloud monitor
Custom Monitor Application
Automated Monitoring Solution
Worker Infrastructure

ジョブソース

ジョブソースは、新しいジョブが Deadline Cloud によってレンダリングされるシステムに入るアクセスポイントです。大まかに言うと、人間のインタラクティブ性と自動コンピュータシステムの 2 つの主なジョブソースがあります。

インタラクティブワークフロー

このシナリオでは、アーティストまたはその他のクリエイティブロールが Deadline Cloud ファームで処理される作業の主要なジェネレーターです。通常、これらのジョブからの出力は、大規模なプロジェクトまたはチームの主要なアーティファクトです。業界標準のデジタルコンテンツ作成 (DCC) ツールなどのソフトウェアを使用して作業を実行します。Deadline Cloud ファームにジョブを手動で送信し、後で確認するために出力を表示しています。ワークステーション自体は によって管理されません AWS。

ほとんどの場合、これらのアーティストはワークロードアプリケーションレイヤーとモニタリングレイヤーで Deadline Cloud 統合送信者と Deadline Cloud モニターを使用します。

自動ワークフロー

このシナリオでは、お客様が所有するプログラムシステムが Deadline Cloud ファームのジョブの主要なジェネレーターです。これは、3D モデルやスキャンから生成されたターンテーブルビデオなど、小売パイプラインのアセット生成である可能性があります。これは、スポーツ用のブロードキャストグラフィックスとプレイヤーカードの自動合成である可能性があります。このシナリオのテーマは、個々の が各ジョブを Deadline Cloud に手動で送信するのではなく、より大きなシステムの一部としてジョブが生成されることです。

自動ジョブでは、Deadline Cloud 統合送信者と Deadline Cloud モニターを使用するのは一般的ではありません。多くの場合、ジョブ定義はユーザーが作成したカスタムアプリケーション開発であり、ジョブ出力は承認と配布のためにデジタルアセット管理 (DAM) システムまたはメディアアセット管理 (MAM) システムに自動的に流れます。

ジョブの送信

ジョブは、OpenJobDescription テンプレートを使用して Deadline Cloud に送信されます。OpenJobDescription は、さまざまなスケジューリングシステムのデプロイ間で移植可能なバッチ処理ジョブを定義するための柔軟なオープン仕様です。ジョブ定義ファイルは、ジョブのパラメータ、ジョブのステップ、ジョブの入力に基づいてステップをパラメータ化する方法、および処理を実

行するワーカーで実行される実際のスクリプトを記述します。ワークロード送信のアイデアは、これらのジョブ定義の作成方法、作成者、送信方法です。

DCC との統合送信者

Deadline Cloud 統合送信者は、Deadline Cloud を業界標準の DCC またはソフトウェアパッケージと結び付けるソフトウェアの一部です。統合された送信者は、レンダリング、複合、またはその他のワークロードのデータと設定をジョブテンプレートに変換する方法を決定します。これは Deadline Cloud で理解できます。多くの統合送信者は、Deadline Cloud チームまたはソフトウェアパッケージの作成者によって作成および管理されますが、目的のアプリケーションにまだ存在しない場合は、独自の送信者を作成して管理できます。Deadline Cloud チームでサポートされている DCCs の有限セットがあります。

インタラクティブワークフローには通常、統合された送信者が含まれますが、必ずしもそうではありません。テンプレート化された自動ワークフローの場合、アーティストが DCC でテンプレートジョブを設定し、ジョブバンドルの 1 回限りのエクスポートを実行するのが一般的なワークフローです。このジョブバンドルは、パラメータ化された方法で Deadline Cloud でその特定の種類のジョブを実行する方法を定義します。このジョブバンドルは、自動化の目的で自動化ワークフローシナリオに統合できます。

カスタムジョブ定義

カスタムアプリケーションとワークフローでは、これらのジョブ定義の作成方法と Deadline Cloud への送信方法を完全に制御できます。たとえば、e コマースサイトでは、販売しているオブジェクトの 3D モデルをアップロードするよう販売者に求める場合があります。このアップロード後、e コマースプラットフォームは Deadline Cloud に送信するジョブ定義を動的に生成し、サイトで使用できる他の 3D オブジェクトと一致する共通の照明を使用して、共通のバックグラウンドでターンテーブルアニメーションを自動的に生成できます。e コマースプラットフォームの開発中、ソフトウェア開発者はジョブ定義を作成し、最終的に販売者が提供したパラメータを使用して e コマースプラットフォームに埋め込み、プラットフォームの製品アップロードワークフロー中にこのジョブを送信するようにプラットフォームをコーディングします。

Deadline Cloud は、github のサンプルリポジトリに多数のサンプルジョブ定義を提供します。

アプリケーション管理

ジョブが Deadline Cloud に送信され、ワーカーに割り当てられると、ジョブ定義のスクリプトがワーカーで実行されます。ほとんどの場合、このスクリプトはアプリケーションを呼び出して、レン

ダラー、複合、エンコード、フィルタリング、その他計算負荷の高いタスクなど、実際の処理を実行します。アプリケーション管理は、必要なバージョンのソフトウェアをワーカーが確実に利用できるようにする概念です。

任意のパッケージ管理システムを使用してアプリケーションを管理できますが、Deadline Cloud には conda パッケージを簡単に使用できるツールが多数用意されています。Conda は、オープンソース、クロスプラットフォーム、言語に依存しないパッケージマネージャーおよび環境管理システムです。

サービスマネージドフリートの Deadline クラウドマネージド conda チャネル (SMF)

サービスマネージドフリートを使用する場合、Deadline Cloud マネージド conda チャネルは自動的に設定され、ジョブで使用できるように設定されます。Deadline Cloud サービスは、この conda チャネルで多数のパートナー DCC アプリケーションとレンダリングを提供します。詳細については、Deadline Cloud ユーザーガイドの「キュー環境の作成」を参照してください。これらのパッケージは Deadline Cloud サービスによって自動的に最新の状態に保たれ、メンテナンスは必要ありません。この conda チャネルは、サービスマネージドフリートを使用する場合にのみ使用でき、カスタマーマネージドフリートを使用する場合は使用できません。

セルフマネージド conda チャネル

Deadline Cloud マネージド conda チャネルを使用できない場合は、Deadline Cloud フリートにアプリケーションをインストール、パッチ適用、管理する方法を決定する必要があります。1 つのオプションは、セットアップして維持する conda チャネルを作成することです。これは、Deadline クラウドマネージド conda チャネルと最も密接に相互運用されます。例えば、Deadline クラウドマネージド conda チャネルから DCC を使用できますが、特定の DCC プラグインを含む独自のパッケージを持ち込むことができます。このプロセスの詳細については、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。

カスタムアプリケーション管理

アプリケーション管理の場合、Deadline Cloud の要件は、ジョブスクリプトがワーカーで実行されるときにアプリケーションが PATH で利用可能であることです。

すでに Rez パッケージを構築して保守している場合は、キュー環境を使用して Rez リポジトリからアプリケーションをインストールできます。キュー環境の例はAWS 、Deadline Cloud GitHub 組織にあります。

存続期間の長いワーカーまたはシステムイメージを持つカスタマーマネージドフリートでアプリケーションをすでに管理している場合、アプリケーション管理にキュー環境は必要ありません。アプリケーションがジョブユーザーのパスに表示されていることを確認し、ジョブを送信します。

アプリケーションのライセンス

Deadline Cloud で一般的に実行されるワークロードの多くは、ソフトウェアベンダーからのソフトウェアライセンスが必要です。これらのアプリケーションは、多くの場合、シートごと、CPU ごと、またはホストごとにライセンスされます。Deadline Cloud でのサードパーティー製ソフトウェアの使用が、サードパーティーライセンス契約に従っていることを確認するのはお客様の責任です。オープンソースソフトウェア、カスタムソフトウェア、またはライセンスフリーソフトウェアを使用している場合は、このレイヤーを設定する必要はありません。Deadline Cloud はレンダーライセンスのみをサポートし、ワークステーションライセンスはサポートしていないことに注意してください。

サービスマネージドフリートと使用ベースのライセンス

Deadline Cloud のサービスマネージドフリートを使用する場合、サポートされているソフトウェアに対して使用量ベースのライセンス (UBL) が自動的に設定されます。サービスマネージドフリートで実行されるジョブには、サポートされているアプリケーション用に環境変数が自動的に設定され、Deadline Cloud ライセンスサーバーを使用するように指示されます。Deadline Cloud UBL を使用する場合、ライセンスされたアプリケーションを使用する時間数に対してのみ課金されます。

カスタマーマネージドフリートと使用量ベースのライセンス

Deadline Cloud 使用状況ベースのライセンス (UBL) は、サービスマネージドフリートを使用しない場合にも利用できます。このシナリオでは、Deadline Cloud ライセンスサーバーへのアクセスを提供する選択した VPC サブネットに IP アドレスを提供する Deadline Cloud ライセンスエンドポイントを設定します。ワーカーで適切なソフトウェア固有の環境変数を設定し、ワーカーからそれらのライセンスエンドポイント IP アドレスへのネットワーク接続を設定すると、ワーカーはサポートされているソフトウェアのライセンスをチェックアウトおよびチェックインできます。サービスマネージドフリートで UBL を使用する場合と同じライセンスに対して 1 時間あたりに課金されます。

カスタムライセンス

Deadline Cloud UBL でサポートされていないアプリケーションを使用するか、既存のライセンスがまだ有効である可能性があります。このシナリオでは、ワーカー (顧客管理またはサービス管理) か

らライセンスサーバーへのネットワークパスを設定する責任があります。カスタムライセンスの詳細については、「」を参照してください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。

アセットアクセス

ジョブがワーカーに送信され、アプリケーションが設定されたら、ジョブに必要なアセットデータにアクセスするようにワーカーを設定する必要があります。これは、3D データ、テクスチャデータ、アニメーションデータ、ビデオフレーム、またはジョブで使用されるその他の種類のデータです。

まず、データが現在保存されている場所について考えます。これは、ワークステーションのハードドライブ、ユーザーコラボレーションツール、ソースコントロール、オンプレミスまたはクラウド内の共有ファイルシステム、Amazon S3、またはその他の任意の場所にある場合があります。

次に、ワーカーがこのデータにアクセスするために必要なものを検討します。このデータは企業ネットワークでのみ利用できますか? データにアクセスするには、どのような ID または認証情報が必要ですか? データソースは、ジョブを処理する予定のワーカー数でジョブをサポートするようにスケーリングされていますか?

ジョブアタッチメント

アセットアクセスのメカニズムから始める最も簡単なのは、Deadline Cloud ジョブアタッチメントです。ジョブアタッチメントを使用してジョブを送信すると、ジョブに必要なデータが、ジョブに必要なファイルを指定するマニフェストファイルとともに Amazon S3 バケットにアップロードされます。ジョブアタッチメントを使用すると、複雑なネットワークや共有ストレージのセットアップは必要ありません。ファイルは 1 回だけアップロードされるため、それ以降のアップロードはより迅速に完了します。ワーカーがジョブの処理を完了すると、出力データが Amazon S3 にアップロードされ、アーティストまたは別のクライアントがダウンロードできるようになります。ジョブアタッチメントは、あらゆるサイズのフリートに合わせてスケールし、シンプルで高速にオンボードして使用できます。

ジョブアタッチメントは、すべての状況に最適なツールではありません。データがすでにオンになっている場合 AWS、ジョブアタッチメントは、関連する転送時間とストレージコストを含むデータのコピーを追加します。ジョブの添付ファイルでは、ジョブが送信時に必要なデータを完全に指定して、データをアップロードできるようにする必要があります。

ジョブアタッチメントを使用するには、Deadline Cloud キューにジョブアタッチメントバケットが関連付けられている必要があり、そのバケットへのアクセスを提供するためにキューロールを使用す

る必要があります。デフォルトでは、Deadline Cloud 統合送信者はすべてジョブアタッチメントをサポートします。Deadline Cloud 統合送信者を使用していない場合は、Deadline Cloud Python ライブラリを統合することで、ジョブアタッチメントをカスタムソフトウェアで使用できます。

カスタムストレージアクセス

ジョブアタッチメントを使用しない場合、ワーカーがジョブに必要なデータにアクセスできることを確認する責任があります。Deadline Cloud には、これをサポートし、ジョブをポータブルに保つためのツールが多数用意されています。アーティストやワーカー用の共有ネットワークストレージがすでにある場合、LucidLink などの外部サービスを使用する場合は、カスタムストレージソリューションを使用できます。

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のレイアウトを記述します。ストレージプロファイルを使用して、Windows ワークステーションを使用するアーティストがLinuxワーカーによって処理されるジョブを送信すると、Deadline Cloud は、ワーカーが設定したデータストレージにアクセスできるようにパスマッピングが行われるようにします。

Deadline Cloud のサービスマネージドフリートを使用する場合、ホスト設定スクリプトと VPC リソースエンドポイントにより、ワーカーは VPC で利用可能な共有ストレージやその他のサービスを直接マウントしてアクセスできます。

ジョブのモニタリングと出力管理

Deadline Cloud に送信されたジョブが正常に完了すると、ユーザーまたはプロセスは Deadline Cloud 外のビジネスワークフローで使用するジョブ出力をダウンロードします。ジョブの失敗後、ジョブログとモニタリング情報は問題を診断するのに役立ちます。

Deadline Cloud モニター

Deadline Cloud モニターアプリケーションは、ウェブおよびデスクトップで利用できます。このソリューションは、ストレージにジョブアタッチメントを使用するさまざまな DCCs のインタラクティブワークフローを使用するスタジオに最適です。モニターは、IAM Identity Center を使用する場合にのみサポートします。IAM Identity Center はワークフォースアイデンティティ製品であり、コンシューマーアイデンティティ (B2C) ソリューションではないため、多くの B2C シナリオには適していません。

カスタムモニターアプリケーション

ユーザーのモニタリングエクスペリエンスをカスタマイズする場合、B2C 製品を構築する場合、または Deadline Cloud を使用して高度に特殊なシステムを構築する場合は、カスタムモニタリングアプリケーションの作成を選択します。AWS Deadline Cloud API を使用して、ワークフロー全体のコンテキストと Deadline Cloud の概念を組み合わせて、このカスタムアプリケーションを作成できます。たとえば、B2C 製品には、ユーザーがセットアップする独自のプロジェクト概念があり、アプリケーションは Deadline Cloud ジョブを同じインターフェイスにネストできます。

自動モニタリングソリューション

シナリオによっては、Deadline Cloud 専用のモニタリングアプリケーションは必要ありません。このシナリオは、スポーツやニュースのブロードキャストグラフィックスなど、パイプライン内のアセットを自動的にレンダリングするために Deadline Cloud が使用される自動ワークフローで一般的です。このシナリオでは、Deadline Cloud API イベントと EventBridge イベントを使用して外部メディアアセット管理システムと統合し、承認を行い、プロセスの次のステップにデータを移動します。

ワーカーインフラストラクチャ管理

Deadline Cloud フリートは、Deadline Cloud キューに送信されたジョブを処理できるサーバー (ワーカー) のグループであり、Deadline Cloud ファームのコアインフラストラクチャです。

サービスマネージドフリート

サービスマネージドフリートでは、Deadline Cloud は、ワーカーホスト、オペレーティングシステム、ネットワーク、パッチ適用、自動スケーリング、およびレンダーファームを実行するその他の要素を担当します。必要なワーカーの最小数と最大数、およびアプリケーションに必要なシステム仕様を指定します。残りは Deadline Cloud が行います。サービスマネージドフリートは、Deadline Cloud マネージド conda チャネルを使用して業界 DCC アプリケーションを簡単に管理できる唯一のフリートオプションです。さらに、Deadline Cloud UBL 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。Wait and Save フリートは、低コストで遅延耐性のあるワークロードで、サービスマネージドフリートでのみ使用できます。

カスタマーマネージドフリート

ワーカーホストとその環境をより細かく制御する必要がある場合は、カスタマーマネージドフリートを使用します。カスタマーマネージドフリートは、Deadline Cloud をオンプレミスで使用する場合

に最適です。詳細については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を参照してください。

アーキテクチャの例

従来のプロダクションスタジオ

従来のプロダクションスタジオには、複数の物理的な場所からサービスレンダリングワークロードにまたがる、大量のコンピューティング、ストレージ、ネットワークインフラストラクチャが必要です。個々のソフトウェアパッケージとベンダーには、バージョニング、互換性、リソースの競合を解決する際に満たす必要がある固有のハードウェア、ソフトウェア、ネットワーク、ライセンス要件があります。

アーティストワークステーション、レンダーノード、ネットワークストレージ、ライセンスサーバー、ジョブキューイングシステム、モニタリングツール、アセット管理に個別のインフラストラクチャ要件があることが一般的です。スタジオは通常、複数のバージョンの DCC ツール、レンダラー、プラグイン、カスタムツールを維持しながら、レンダラーファーム全体で複雑なライセンス契約を管理する必要があります。開発、品質保証、本番環境を検討すると、スタジオインフラストラクチャが複雑になります。

サービスマネージドオプションを使用した一般的な Deadline Cloud デプロイは、以下を通じてこれらの課題の多くを解決または軽減します。

- 統合された DCC 送信者によるインタラクティブなワークフロージョブの送信
- Deadline クラウドマネージド conda チャネルによるアプリケーション管理
- サポートされているソフトウェア用に自動的に設定された使用状況ベースのライセンス
- ジョブアタッチメントによるアセット管理
- Deadline Cloud Monitor アプリケーションによるモニタリング
- サービスマネージドフリートによるインフラストラクチャ管理

このアプローチにより、アーティストは複雑なインフラストラクチャを管理することなく、使い慣れた DCC ツールからスケーラブルなクラウドレンダーファームにジョブを直接送信できます。このサービスは、ソフトウェアのデプロイ、ライセンス、データ転送、インフラストラクチャのスケーリングを自動的に処理します。アーティストはウェブインターフェイスまたはデスクトップアプリケーションを介してジョブをモニタリングでき、出力は自動的に Amazon S3 に保存されるため、簡単にアクセスできます。

この設定により、スタジオは開発環境と本番環境を数分で作成し、使用するコンピューティングとライセンスに対してのみ料金を支払うことができ、インフラストラクチャ管理ではなくクリエイティブな作業に集中できます。サービスマネージド型アプローチは、アーティストの使い慣れたワークフローを維持しながら、クラウドレンダリングを採用するための最速の道を提供します。

Interactive Workflow
Automated Workflow
Integrated Submitter
Custom Job Definition
Conda Application Management
Custom Application Management
Deadline Cloud UBL
Custom Licensing
Job Attachments
Custom Storage
Deadline Cloud monitor
Custom Monitor Application
Automated Monitoring Solution

クラウドの Studio

最新のビジュアルエフェクトとアニメーションスタジオは、アーティストワークステーションを含むパイプライン全体をクラウドに移行し始めています。このアプローチにより、オンプレミスインフラストラクチャが不要になり、グローバルコラボレーションが可能になり、インタラクティブな作業とレンダリングの両方をシームレスにスケーリングできます。ただし、クラウドリソースの管理、データへの低レイテンシーアクセスの確保、クラウドベースのワークステーションとレンダーファームの統合にも新しい課題が伴います。

一般的なクラウドネイティブスタジオでは、クラウドワークステーション、共有ストレージ、レンダリングインフラストラクチャ、およびこれらすべてのコンポーネントのソフトウェアデプロイを管理するための統一されたアプローチが必要です。従来のアプローチでは、パフォーマンス、コスト、柔軟性のバランスをとるのに苦労する、手動管理の複雑なシステムになることがよくあります。

クラウドネイティブスタジオの Deadline Cloud デプロイは、以下を使用して実装できます。

- クラウドワークステーション上の統合された DCC 送信者を介したインタラクティブなワークフロージョブの送信
- Deadline Cloud マネージド conda チャネルによるアプリケーション管理レンダーノード
- サポートされているソフトウェア用に自動的に設定された使用状況ベースのライセンス
- 共有プロジェクトデータに FSx for Windows File Server を使用したカスタムストレージアクセス
- Deadline Cloud Monitor アプリケーションによるモニタリング
- サービスマネージドフリートを使用したインフラストラクチャ管理

このアプローチにより、アーティストは高性能の共有ストレージに直接アクセスできるクラウドベースのワークステーションで作業し、Deadline Cloud ファームにシームレスにジョブを送信できます。スタジオは、同じ conda チャネルを使用してワークステーションとレンダーノードの両方にまたがるソフトウェアのデプロイを管理できるため、一貫性を確保し、メンテナンスのオーバーヘッドを削減できます。

この設定の主な利点は次のとおりです。

- どこからでもワークステーションにアクセスできるアーティストとのグローバルなコラボレーション
- ワークステーションとレンダーノード間で一貫したソフトウェア環境
- ワークステーションとレンダーノードの両方からアクセスできる高性能共有ストレージ

- インタラクティブコンピューティングリソースとバッチコンピューティングリソースの両方の柔軟なスケーリング
- クラウド内のすべてのスタジオインフラストラクチャの一元管理

このシナリオのストレージ設定には通常、以下が含まれます。

- クラウドワークステーションと Deadline Cloud ワーカーの両方がアクセスできるプロジェクトデータ用の FSx for Windows File Server
- ワークステーションとレンダーノード間のパスマッピングを管理するための Deadline Cloud のストレージプロファイル
- VPC リソースエンドポイントとホスト設定スクリプトを使用した Deadline Cloud ワーカーへの FSx 共有の直接マウント

このクラウドネイティブなアプローチにより、スタジオはオンプレミスインフラストラクチャを排除し、使い慣れたアーティストワークフローを維持しながら、あらゆる規模のプロジェクトを迅速にスケーリングできます。サービスマネージドリソースとカスタマーマネージドリソースを組み合わせて使用できる柔軟性を提供し、管理のしやすさと特定のパフォーマンス要件の両方を最適化します。

Deadline Cloud とともにクラウドワークステーションを活用することで、スタジオは、小規模なチームから大規模なプロダクションまでシームレスにスケールする、完全に統合されたグローバルにアクセス可能な本番稼働用パイプラインを実現できます。

ECommerceオートメーション

最新の e コマースプラットフォームでは、数百万の項目にわたる豊富な製品可視化を実現するために、大規模な自動アセット生成が必要です。従来のアプローチでは、大量の 3D モデルを標準化された製品メディアに処理するために多大なインフラストラクチャ投資が必要となり、多くの場合、処理バックログを作成するプロビジョニング不足のシステムまたはアイドル容量を持つプロビジョニング過剰のシステムのいずれかになります。

一般的な自動 e コマースワークフローでは、製品アップロード処理、3D モデル検証、レンダーファーム管理、出力処理、製品情報システムとの統合を処理する必要があります。これらのワークフローを管理するには、従来、一貫した品質を確保し、大規模なコスト効率を維持しながら、複数のレンダリングアプリケーション、コンピューティングリソース、データ処理パイプラインを調整する必要があります。

e コマース自動化用の Deadline Cloud デプロイは、以下を使用して実装できます。

- 既存の e コマース取り込みアプリケーションでのカスタム API 統合によるワークフロージョブの自動送信
- 標準化された製品の視覚化に合わせたカスタムジョブ定義
- Deadline クラウドマネージド conda チャネルによるアプリケーション管理
- サポートされているソフトウェア用に自動的に設定された使用状況ベースのライセンス
- アセット管理のための Amazon S3 の直接統合
- 既存の製品管理システムと統合されたカスタムモニタリングアプリケーション
- Elastic Scaling のサービスマネージドフリート

このアプローチにより、1 日あたり数千の製品を処理し、ターンテーブルアニメーションなどの標準化された製品視覚化を自動的に生成できます。サービスマネージドインフラストラクチャは、ワーカーの再利用と最適化されたアプリケーションデプロイを通じてコスト効率を維持しながら、変化する需要に合わせて自動的にスケーリングされます。

eCommerce
Interactive Workflow
Automated Workflow
Integrated Submitter
Custom Job Definition
Conda Application Management
Custom Application Management
Deadline Cloud UBL
Custom Licensing
Job Attachments
Custom Storage
Deadline Cloud monitor
Custom Monitor Application
Automated Monitoring Solution

Whitelabel/OEM/B2C カスタマー

従来のデジタルコンテンツ作成 (DCC) ソフトウェアでは、通常、ユーザーが独自のレンダリングインフラストラクチャを維持したり、ワークステーションでレンダリングをローカルに処理したりする必要があり、その結果、ハードウェアへの多大な投資や長い待機時間により、クリエイティブワークフローが中断されます。ソフトウェアベンダーにとって、クラウドレンダリング機能を提供するには、従来、複雑なインフラストラクチャと請求システムの構築と維持が必要でした。

B2C ソフトウェアに統合された Deadline Cloud デプロイにより、ユーザーの使い慣れたインターフェイス内で直接、シームレスなクラウドレンダリングが可能になります。この統合では、以下が組み合わされます。

- DCC アプリケーション内に埋め込まれたインタラクティブなワークフロージョブの送信
- Deadline レンダーアプリケーションデプロイ用のクラウドマネージド conda チャネル
- 使用状況ベースのライセンスが自動的に設定される
- ベンダーマネージドストレージを使用したジョブアタッチメントによるアセット管理
- DCC インターフェイスに直接統合されたカスタムモニタリング
- ユーザー間で共有されるサービスマネージドフリート

このアプローチにより、エンドユーザーは、アカウント、インフラストラクチャ、複雑なセットアップを管理することなく、ソフトウェア内からワンクリックでレンダリングをクラウドに送信できます。ソフトウェアベンダーは、以下のマルチテナント環境を維持します。

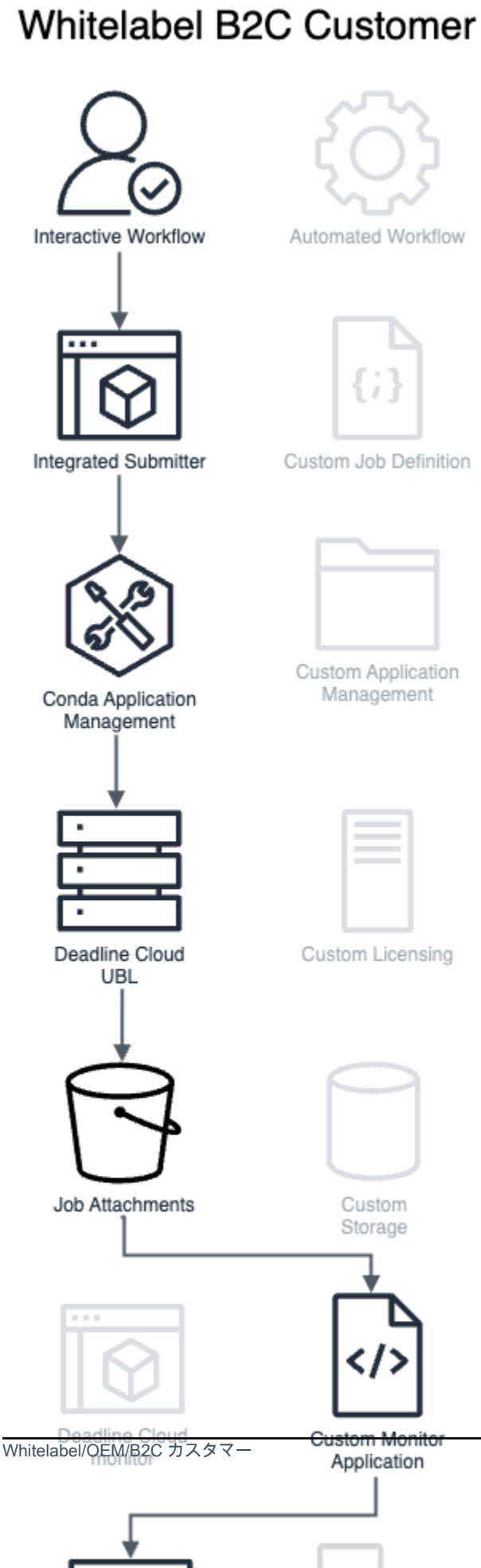
- ユーザーが既存のソフトウェア認証情報を使用して認証する
- ジョブはユーザーごとの専用キューに自動的にルーティングされます
- IAM 制御のストレージプレフィックスを使用してアセットを安全に分離する
- 請求はベンダーの既存のシステムを通じて処理されます。
- ジョブのステータスと出力は、ユーザーのアプリケーションに直接ストリーミングされます。

共有フリートアプローチは、ワーカーのウォームプールを維持し、起動時間を最小限に抑えながら、ユーザーベース全体でリソース使用率を最大化することで、最適なパフォーマンスを確保します。この設定により、ソフトウェアベンダーは、追加のセットアップやアカウントを必要とする個別のサービスではなく、シームレスな製品機能としてクラウドレンダリングを提供できます。

エンドユーザーには以下の利点があります。

- 使い慣れたインターフェイスからのワンクリック送信
- インフラストラクチャ管理なしPay-as-you-go
- 共有インフラストラクチャによるジョブの起動時間の短縮
- 完了したレンダリングの自動ダウンロードと整理
- すべてのプラットフォームで一貫したエクスペリエンス

この統合パターンにより、ソフトウェアベンダーは、アプリケーションにネイティブであると感じるシンプルでコンシューマーフレンドリーなエクスペリエンスを維持しながら、ユーザーベース全体にエンタープライズグレードのレンダリング機能を提供できます。


Whitelabel B2C Customer
Interactive Workflow
Automated Workflow
Integrated Submitter
Custom Job Definition
Conda Application Management
Custom Application Management
Deadline Cloud UBL
Custom Licensing
Job Attachments
Custom Storage
Deadline Cloud monitor
Custom Monitor Application
Automated Monitoring Solution

Deadline Cloud ワークロードとは

AWS Deadline Cloud を使用すると、クラウドでアプリケーションを実行するジョブを送信し、ビジネスにとって重要なコンテンツやインサイトを制作するためのデータを処理できます。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、製品カタログ画像、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アーティスティックまたはテクニカル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多くのタスクには、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が含まれます。

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画を処理します。
- 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。本番稼働中は、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、最終的な本番キャラクターに適用されると、簡単に 3 時間になる可能性があります。この進行により、GUI 内で寿命を迎えたワークロードが大きくなり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可能性がありますが、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、Deadline Cloud CLI を使用してユーザーが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、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や、 や Linux 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できます。

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。このアプローチにより、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ジョブを移植できます。

- ジョブの入力ファイル用のファイルパス参照を再配置し、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 がファイルをジョブに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、このように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。このアプローチは、 で実行ビットが設定されたスクリプトファイルの使用と比較して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、ログを正しくテーリングしません。この動作をオフにMSYS_NO_PATHCONV=1するには、 変数を設定します。
    - 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にこの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、ジョブに必要なアプリケーションパッケージ名CondaPackagesとバージョンを一覧表示するデフォルトのRezPackagesパラメータ値を使用して、ジョブで および/または パラメータを使用できます。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 のデベロッパー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します。

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、キュー ID を に追加する `~/.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と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。このトピックでは、ジョブバンドルを設定する方法の例を示します。

このセクションの手順を実行する前に、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
- 次の手順

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、ジョブで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、および dataFlow を追加しますparameterDefinitions。

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

c. 次のbashスクリプトコマンドを、 が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、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
  ```

次の手順

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、以下を試すことができます。

- 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 のデベロッパー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 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の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で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 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と同様の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を実行する前に、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
- 次の手順

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。

ジョブアタッチメントを使用して simple_file_job ジョブバンドルを送信するには

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hash-jobattachments.txt、存在しません。

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、ジョブが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 .
```

次のようなファイルのプロパティhash-jobattachments.txtが出力に表示されます。

```
{
     "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
```

次の手順

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 のデベロッパー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 のデベロッパー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。

デベロッパーファームにサービスマネージドフリートを追加するには

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
```

次の手順

サービスマネージドフリートを作成してテストしたら、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-abcdefgh01234567890123456789012id",
            "fleetId": "fleet-abcdefgh01234567890123456789012id",
            "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-abcdefgh01234567890123456789012id",
            "fleetId": "fleet-abcdefgh01234567890123456789012id",
            "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"
```

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。ジョブバンドルは、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ファイルを保存し、テキストエディタを使用してジョブバンドルを作成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、ジョブからの出力を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。

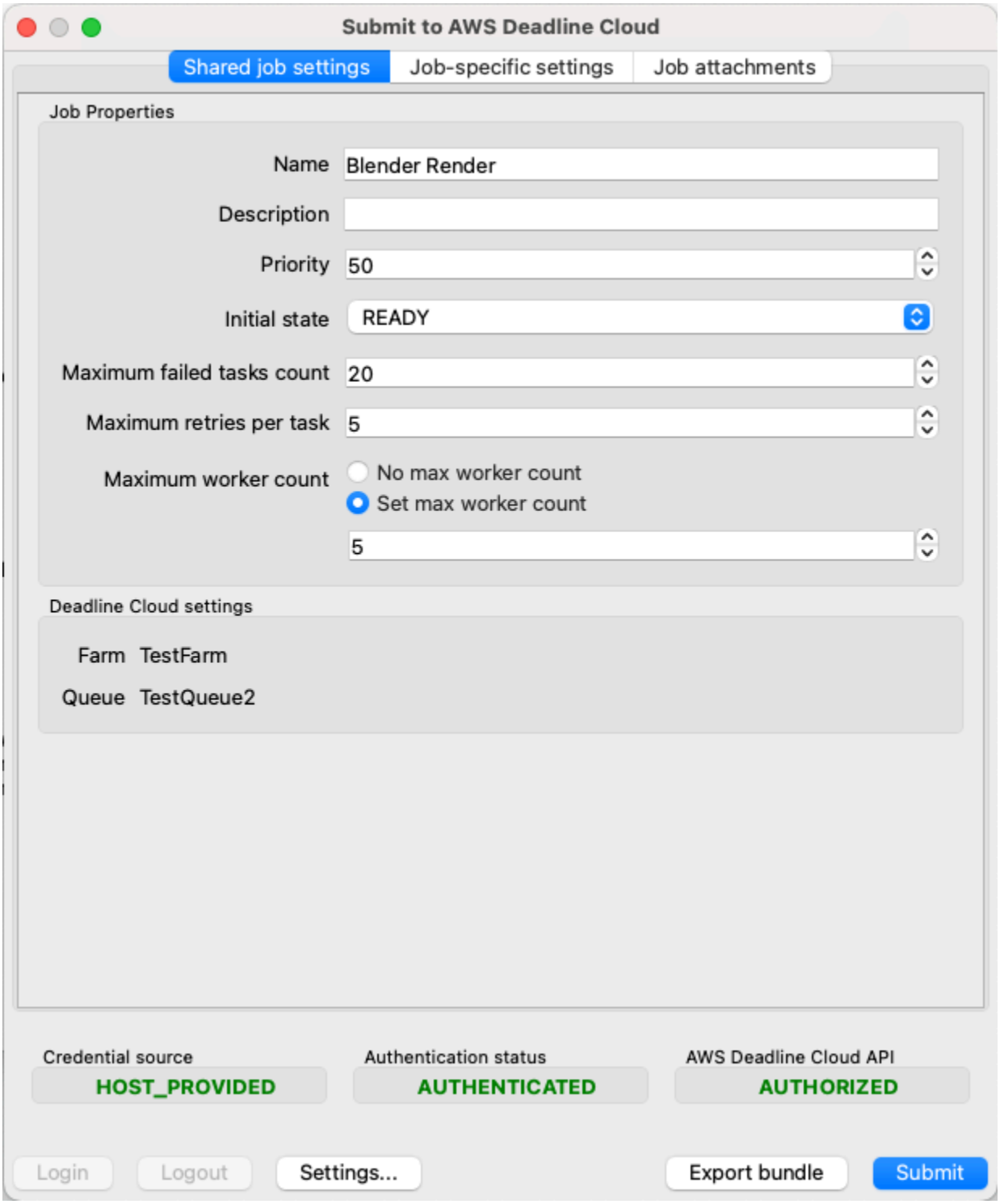
ジョブバンドルは、OpenJD ジョブテンプレート、ジョブを定義する他のファイル、ジョブの入力に必要なジョブ固有のファイルを含むディレクトリ構造です。ジョブを YAML ファイルまたは JSON ファイルとして定義するファイルを指定できます。

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送信にジョブバンドルを使用するか、グラフィカル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で次の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	使用可能
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_STEP_ID	タスクアクション
DEADLINE_SESSION_ID	すべてのアクション
DEADLINE_TASK_ID	タスクアクション
DEADLINE_SESSIONACTION_ID	すべてのアクション

トピック

- ジョブバンドルのジョブテンプレート要素
- ジョブテンプレートのタスクチャンキング
- ジョブバンドルのパラメータ値要素
- アセットがジョブバンドルの要素を参照する

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、Deadline Cloud ジョブの一部として実行されるランタイム環境と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、キューに適用された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。

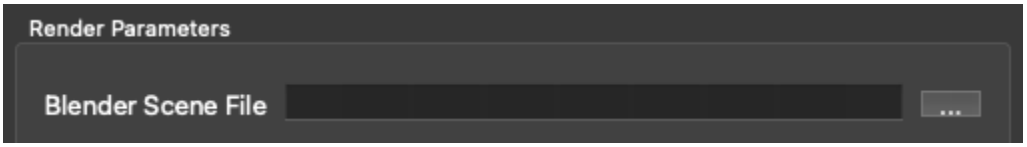
たとえば、`blender_render`サンプルのジョブテンプレートでは、入力パラメータをファイルパスBlenderSceneFileとして定義します。

```
- 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、 deadline bundle gui-submit コマンドを使用するコマンドラインと、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 リポジトリの gui_control_showcase サンプルを参照してください。

objectType および 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のサンプルが他にもあります。

ジョブテンプレートのタスクチャンキング

タスクチャンキングを使用すると、複数のタスクをチャンクと呼ばれる単一の作業単位にグループ化できます。たとえば、レンダリングジョブでは、Deadline Cloud はコマンド呼び出しごとに 1 フレームではなく、複数のフレームを一緒にディスパッチできることを意味します。これにより、各タスクでアプリケーションを起動するオーバーヘッドが軽減され、合計ジョブランタイムが短縮されます。詳細については、OpenJD Wiki の「一度に複数のフレームを実行する」を参照してください。

OpenJD は、ジョブテンプレートにオプション機能を追加する拡張機能をサポートしています。TASK_CHUNKING 拡張機能を追加することで、タスクチャンキングが有効になります。チャンキングを使用するには、 拡張機能をジョブテンプレートに追加し、 CHUNK[INT] タスクパラメータタイプを使用します。同じdeadline bundle submitコマンドを使用してチャンクされたジョブを送信します。たとえば、次のジョブテンプレートは 10 のチャンクでフレームをレンダリングします。

```
specificationVersion: 'jobtemplate-2023-09'
extensions:
  - TASK_CHUNKING
name: Blender Render with Contiguous Chunking
parameterDefinitions:
  - name: BlenderSceneFile
    type: PATH
    objectType: FILE
    dataFlow: IN
  - name: Frames
    type: STRING
    default: "1-100"
  - name: OutputDir
    type: PATH
    objectType: DIRECTORY
    dataFlow: OUT
```

```
    default: "./output"
steps:
  - name: RenderBlender
    parameterSpace:
      taskParameterDefinitions:
        - name: Frame
          type: CHUNK[INT]
          range: "{{Param.Frames}}"
          chunks:
            defaultTaskCount: 10
            rangeConstraint: CONTIGUOUS
    script:
      actions:
        onRun:
          command: bash
          args: ["{{Task.File.Run}}"]
      embeddedFiles:
        - name: Run
          type: TEXT
          data: |
            set -xeuo pipefail

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

            # Parse the chunk range (e.g., "1-10") into start and end frames
            START_FRAME="$(echo '{{Task.Param.Frame}}' | cut -d- -f1)"
            END_FRAME="$(echo '{{Task.Param.Frame}}' | cut -d- -f2)"

            blender --background '{{Param.BlenderSceneFile}}' \
                    --render-output '{{Param.OutputDir}}/output_####' \
                    --render-format PNG \
                    --use-extension 1 \
                    -s "$START_FRAME" \
                    -e "$END_FRAME" \
                    --render-anim
```

この例では、Deadline Cloud は 100 フレームを 1-10、 11-20などのチャンクに分割します。{{Task.Param.Frame}} 変数は、 のような範囲式に展開されます1-10。rangeConstraint は に設定されているためCONTIGUOUS、範囲は常に start-end形式です。スクリプトはこの範囲を解析し、 の -sおよび -eオプションを使用して開始フレームと終了フレームを Blender に渡します--render-anim。

chunks プロパティは、次のフィールドをサポートしています。

- `defaultTaskCount` – (必須) 1 つのチャンクに結合するタスクの数。最大値は 150 です。
- `rangeConstraint` – (必須) の場合`CONTIGUOUS`、チャンクは常に のような連続した範囲です`1-10`。の場合`NONCONTIGUOUS`、チャンクは のような任意のセットにすることができます`1,3,7-10`。
- `targetRuntimeSeconds` – (オプション) 各チャンクの秒単位のターゲットランタイム。Deadline Cloud は、いくつかのチャンクが完了すると、このターゲットに近づくようにチャンクサイズを動的に調整できます。

連続チャンクと非連続チャンクの両方を含む基本および Blender の例を含むその他のタスクチャンキングの例については、GitHub の Deadline Cloud サンプルリポジトリの「タスクチャンキングサンプル」を参照してください。

> **カスタマーマネージドフリートの要件**
>
> タスクチャンキングには、互換性のあるワーカーエージェントバージョンが必要です。カスタマーマネージドフリートを使用する場合は、チャンキングでジョブを送信する前に、ワーカーエージェントが更新されていることを確認してください。サービスマネージドフリートは常に互換性のあるワーカーエージェントバージョンを使用します。

> **チャンクジョブの出力のダウンロード**
>
> チャンクジョブ内の 1 つのタスクの出力をダウンロードすると、Deadline Cloud はチャンク全体の出力をダウンロードします。例えば、フレーム 1 ~ 10 が一緒に処理された場合、フレーム 3 の出力のダウンロードにはすべてのフレーム 1 ~ 10 が含まれます。この機能には`deadline-cloud`バージョン 0.53.3 以降が必要です。

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。maxWorkerCount 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、ジョブバンドルが動作します。

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のない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
> ```

アセットがジョブバンドルの要素を参照する

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、入力ファイルとディレクトリ、および添付ファイルの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。

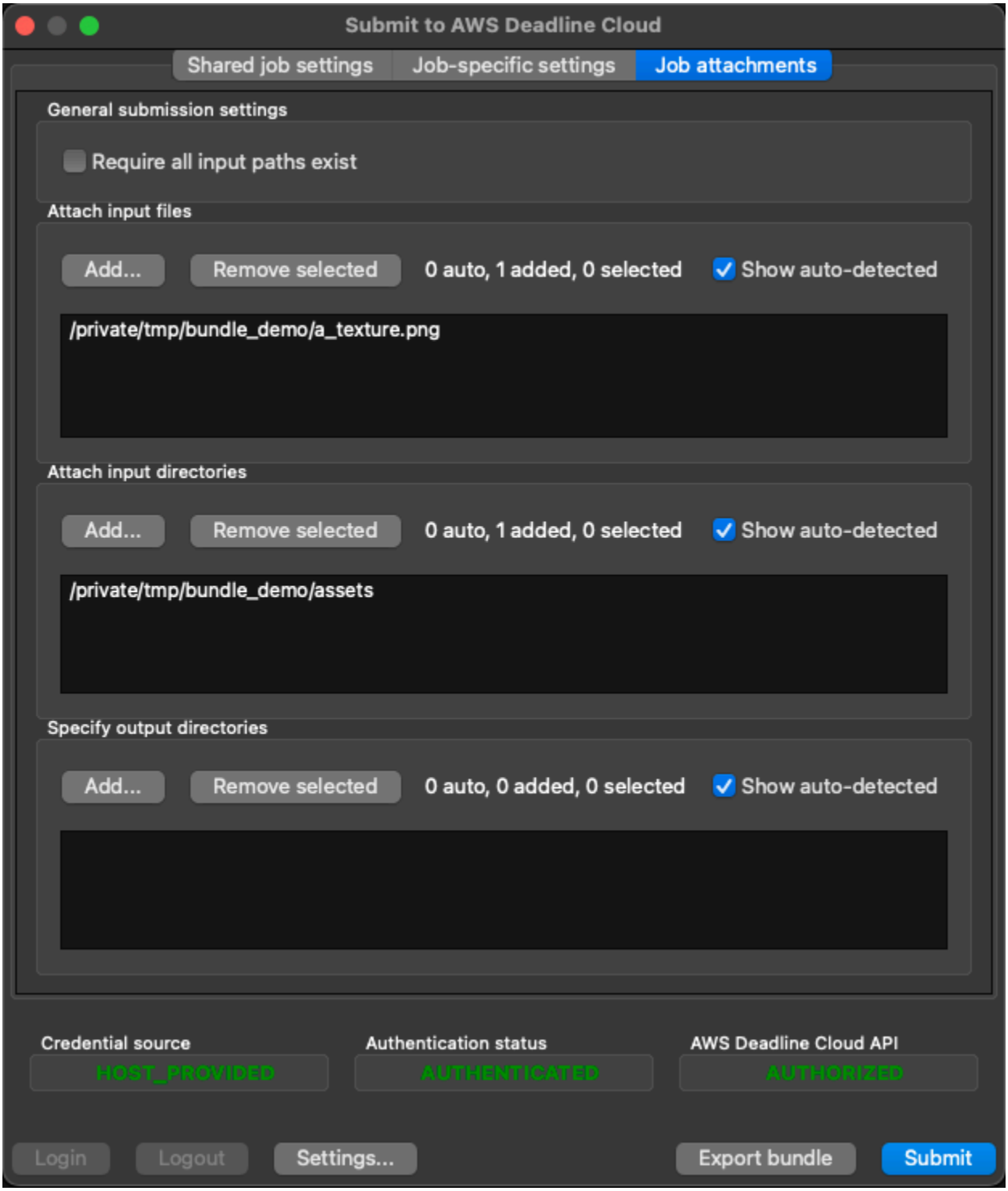
ストレージプロファイルの作成と使用の詳細については、「Deadline Cloud ユーザーガイド」の「Deadline Cloud の共有ストレージ」を参照してください。 AWS

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。このマッピングは、 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のに役立ちます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によっても使用されます。

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、 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の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 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されたジョブまたはホストで実行された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 に指示します。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をキューにいつでも送信できます。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。例えば、次のようになります。

```
$ 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が を定義している場合でも、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を実行する」の手順に従ってワーカーを実行します AWS CloudShell。以前にこれらの指示に従った場合は、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。読みやすいよう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てい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、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の場合にも、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のに最適な方法です。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にのみ存在す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。これらの方法のいずれか、または 3 つすべての組み合わせを使用できます。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するように、ジョブのバンドルのストレージプロファイルを指定できます。

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のファイルを使用可能にする方法を示します。

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、ハッシュ値は同じです。このコンテンツアドレス可能なストレージにより、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、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script.shアップロードされ、そのファイルの内容のハッシュが 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、ジョブを送信するときに 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について考慮されたファイルシステムの場所を確認できます。キューのストレージプロファイルWSAll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

ジョブアタッチメントを使用して Deadline Cloud が Amazon S3 にアップロードするファイルを使用するジョブには、ワーカーホストのファイルシステムを通じ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の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、名前がセッションの一時ディレクトリの「assetroot」で始まる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、「アサルート」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。

例えば、次のようになります。

```
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口グファイルには、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、およびそれらの出力ファイルをワークステーションで取得する方法を示します。

この例の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識別子から算出されます。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 Monitor 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、出力ファイルは代わりに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。

この例の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 つの異なる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、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 つのセッションの口グを確認します。

```
# 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 つの個別の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、同時に限られた数のワーカーにのみデータを提供できます。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を持つ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。制約のある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、ジョブテンプレートで指定しますが、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

キューと制限の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状態のままで、キューと制限間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、タスクの実行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、最終的にタスクの完了時に停止します。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 (Deadline Cloud コンソール) 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。プレフィックスの**amount.**後に金額名が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 (Deadline Cloud コンソール) 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、制限を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する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。送信者で制限の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応じて変わります。

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、タスクを実行してから、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。

サービスマネージドフリートワーカーでは、セッションディレクトリはセッション終了後に削除されますが、他のディレクトリはセッション間で保持されます。この動作により、複数のセッションで再利用できるデータのキャッシュ戦略を実装できます。セッション間でデータをキャッシュするには、ジョブを実行しているユーザーのホームディレクトリに保存します。たとえば、conda パッケージは、ワーカーの とWindowsワーカーの `C:\Users\job-user\.conda-pkgs`のジョブユーザーのホームディレクトリ`/home/job-user/.conda-pkgs`にキャッシュされますLinux。このデータは、ワーカーがシャットダウンするまで引き続き使用できます。

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。

ステップの依存関係には以下の制限があります。

- ステップあたりの依存関係 – ステップは、最大 128 個の他のステップに依存します。
- ステップあたりのコンシューマー – 1 つのステップに応じて、最大 32 の他のステップを指定できます。

```
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- ジョブを失敗としてマークする

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。最大再試行回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– ジョブの名前を変更する

ジョブの表示名を更新します。ジョブ名は最大 128 文字です。

```
aws deadline update-job \
--farm-id farmID \
--queue-id queueID \
--job-id jobID \
--name "New Job Name"
```

Example– ジョブの説明を変更する

ジョブの説明を更新します。説明は最大 2048 文字です。既存の説明を削除するには、空の文字列を渡します。

```
aws deadline update-job \
--farm-id farmID \
```

```
--queue-id queueID \
--job-id jobID \
--description "New Job Description"
```

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

ステータスを持つ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– ステップを一時停止する

、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、各フリート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。この推奨事項により、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の 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 。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "credentials.deadline.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn":
 "arn:aws:deadline:*:111122223333: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。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "deadline:AssumeFleetRoleForWorker",
                "deadline:UpdateWorker",
                "deadline:DeleteWorker",
                "deadline:UpdateWorkerSchedule",
                "deadline:BatchGetJobEntity",
                "deadline:AssumeQueueRoleForWorker"
            ],
            "Resource": "arn:aws:deadline:*:111122223333:*",
            "Condition": {
                "StringEquals": {
```

```
                    "aws:PrincipalAccount": "${aws:ResourceAccount}"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream"
            ],
            "Resource": "arn:aws:logs:*:*:*://deadline/*",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalAccount": "${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を担当します。

- 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

Server

- Windowsベースの Amazon Elastic Compute Cloud (Amazon EC2) インスタンス。Windows Server 2022をお勧めします。
- 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を 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の 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

Linux

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
```

Server

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 ワーカーホストの長い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 アプリケーション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` Linuxまたは `C:\ProgramData\Amazon\Deadline\Config\worker.toml`の 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**。

  Linux

  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
  ```

  Server

  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。この設定は、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ることを意味します。また、ジョブとワーカーエージェントユーザーとの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ます。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
```

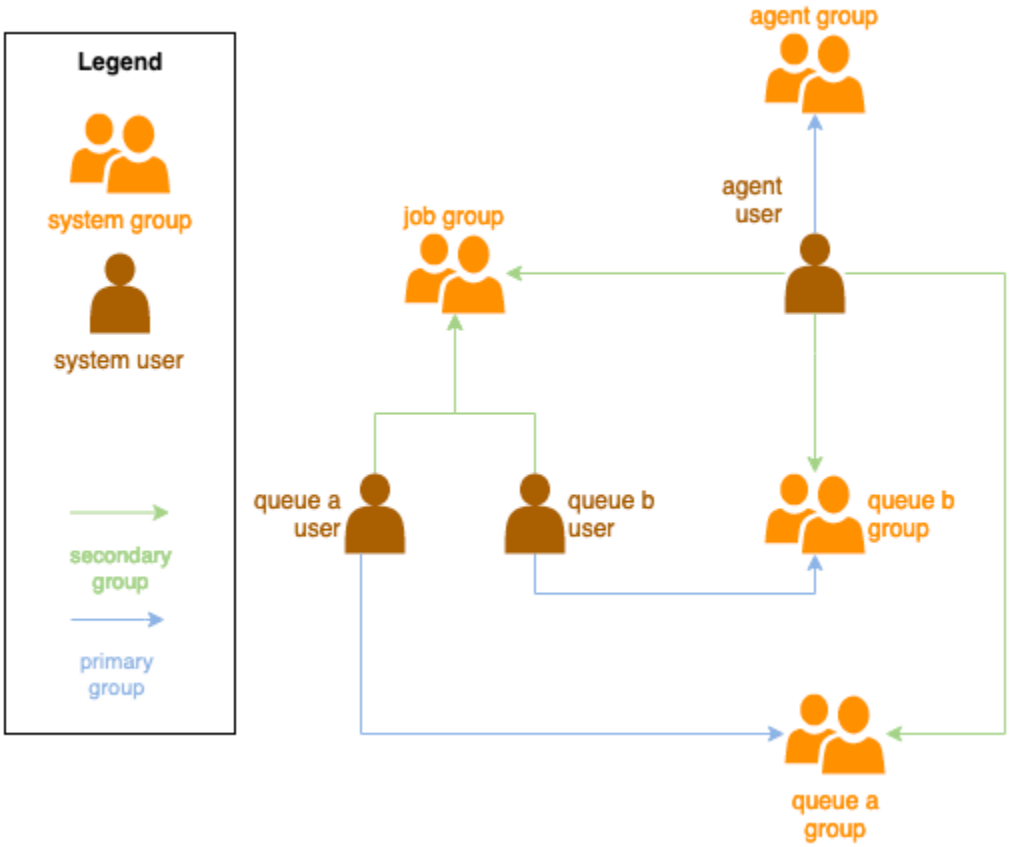
- ワーカーエージェントユーザーは、 としてsudoコマンドを実行できる必要がありますjobRunAsUser。次のコマンドを実行してエディタを開き、新しい sudoers ルールを作成します。

```
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、エージェントユーザーとフリートに関連付けられたキューのjobRunAsUserユーザーと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。

ワーカーホストの保護

ワーカーホストを設定するときは、セキュリティのベストプラクティスに従って機密情報を保護し、適切なアクセスコントロールを維持します。

ログフォルダのアクセス許可の設定

ワーカーエージェントは、ホスト設定スクリプトとジョブ実行の機密情報を含むログファイルを書き込みます。install-deadline-worker コマンドは、安全なアクセス許可を持つログディレクトリを作成します。インストール前にディレクトリを手動で作成する必要がある場合は、次の手順を使用して、サービスマネージドフリートで使用されるアクセス許可に一致します。

Linux

でログディレクトリのアクセス許可を設定するには Linux

1. ログディレクトリを作成します。

```
sudo mkdir -p /var/log/amazon/deadline
```

2. 所有者とグループをワーカーエージェントユーザーに設定します。

```
sudo chown -R deadline-worker:deadline-worker /var/log/amazon/deadline
```

3. アクセス許可を 750 に設定します。

```
sudo chmod -R 750 /var/log/amazon/deadline
```

これらのアクセス許可により、ワーカーエージェントのユーザーとグループのみがログファイルにアクセスできるようになり、ジョブユーザーやその他の権限のないユーザーが機密情報の可能性がある情報を読み取れなくなります。

Server

でログディレクトリのアクセス許可を設定するには Windows

1. 管理者 PowerShell ターミナルを開きます。
2. ログディレクトリを作成します。

```
New-Item -ItemType Directory -Force -Path "$env:PROGRAMDATA\Amazon\Deadline
\Logs"
```

3. ワーカーエージェントのユーザーと管理者のみを許可するように制限された ACLs を設定します。

```
$acl = Get-Acl "$env:PROGRAMDATA\Amazon\Deadline\Logs"
$acl.SetAccessRuleProtection($true, $false)
$acl.Access | ForEach-Object { $acl.RemoveAccessRule($_) }
$agentRule = New-Object
 System.Security.AccessControl.FileSystemAccessRule("deadline-worker",
 "FullControl", "ContainerInherit,ObjectInherit", "None", "Allow")
$adminRule = New-Object
 System.Security.AccessControl.FileSystemAccessRule("Administrators",
 "FullControl", "ContainerInherit,ObjectInherit", "None", "Allow")
$acl.AddAccessRule($agentRule)
$acl.AddAccessRule($adminRule)
Set-Acl "$env:PROGRAMDATA\Amazon\Deadline\Logs" $acl
```

これらのコマンドは、ログディレクトリへのアクセスをワーカーエージェントユーザーと管理者グループのみに制限し、ジョブユーザーやその他の権限のないユーザーが機密情報を読み取るのを防ぎます。

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想定されます。

```
{
    "password": "JOB_USER_PASSWORD"
}
```

ワーカーがキューの設定済み 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とフリートの関連付けが作成されると、フリートロールに付与されます。

- フリートとキュー間のキューと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より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」セクションで、 フォームのポリシーステートメントを追加します。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Principal" : {
        "AWS" : "arn:aws:iam::111122223333:role/FleetRole"
      },
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Resource" : "arn:aws:secretsmanager:us-
east-1:111122223333:secret:YourSecretName-ABC123"
    }
  ]
}
```

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より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。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Principal" : {
        "AWS" : "arn:aws:iam::111122223333:role/FleetRole"
      },
      "Action" : [
        "secretsmanager:GetSecretValue"
      ],
      "Resource" : "arn:aws:secretsmanager:us-
east-1:111122223333:secret:YourSecretName-ABC123"
    }
  ]
}
```

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を使用して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. 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。
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 deadline-cloud-for-3ds-max
```

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。

テストでは、 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て、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. ユーザーに名前を付けます。へのユーザーアクセスを許可する AWS マネジメントコンソール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] 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。

   JSON

```
{
  "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。 `%USERPROFILE\.aws\credentials`

  次のキーを置き換えます。

```
[default]
aws_access_key_id=ACCESS_KEY_ID
aws_secret_access_key=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 アイデンティティセン

ターときは、 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 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 CloudFormation コンソールを開きます。

   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「 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、操作するための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、推奨される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"
        }
    }
}
```

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 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、フリートが正常であり、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。ヘルスチェックにより、、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。

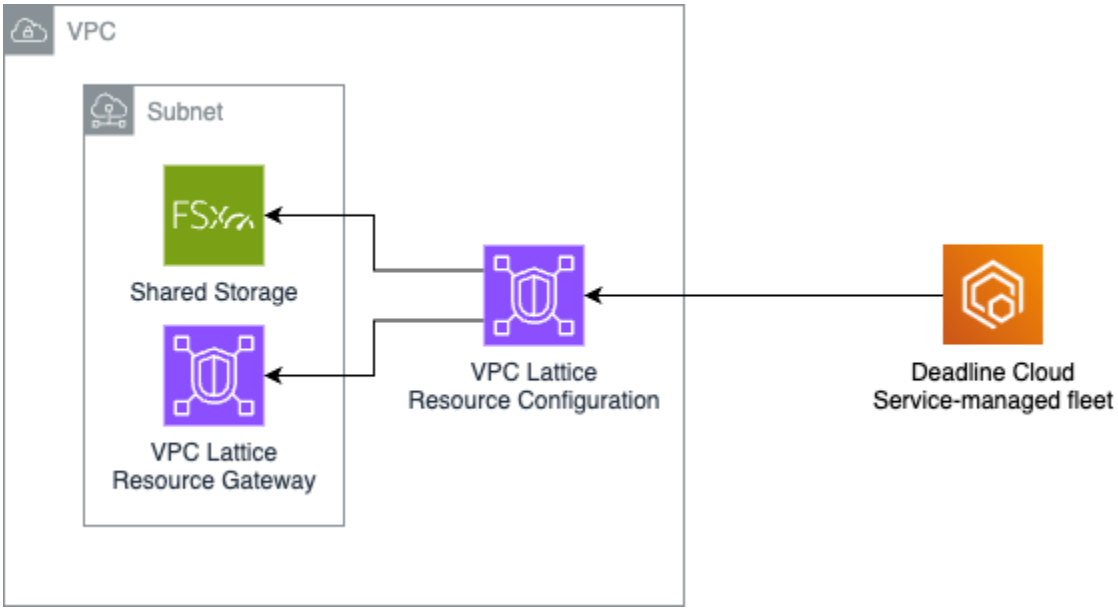
> **Note**
>
> ホスト設定スクリプトを使用してワーカーにカスタムソフトウェアをインストールするには、「」を参照してください管理者権限を持つホスト設定スクリプトを実行する。

トピック

- VPC リソースエンドポイントを使用して VPC リソースを SMF に接続する
- サービスマネージドフリートでジョブアタッチメントを使用する

VPC リソースエンドポイントを使用して VPC リソースを SMF に接続する

Deadline Cloud サービスマネージドフリート (SMF) の Amazon VPC リソースエンドポイントを使用すると、ネットワークファイルシステム (NFS)、ライセンスサーバー、データベースなどの VPC リソースを Deadline Cloud ワーカーに接続できます。この機能を使用すると、VPC 内の既存のインフラストラクチャと統合しながら、Deadline Cloud のフルマネージドプラットフォームを活用できます。



> **Tip**
>
> Amazon FSx クラスターをセットアップしてサービスマネージドフリートに接続するリファレンス CloudFormation テンプレートについては、GitHub の Deadline Cloud サンプルリポジトリの「smf_vpc_fsx」を参照してください。

VPC リソースエンドポイントの仕組み

VPC リソースエンドポイントは VPC Lattice を使用して、Deadline Cloud SMF ワーカーと VPC 内のリソースの間に安全な接続を作成します。接続は一方向です。つまり、ワーカーは VPC 内のリソースへの接続を確立し、データを前後に転送できますが、VPC 内のリソースはワーカーへの接続を確立できません。

VPC リソースを Deadline Cloud のサービスマネージドフリートに接続すると、ワーカーはプライベートドメイン名を使用して VPC 内のリソースにアクセスできます。さらに、ワーカーから VPC Lattice を介した VPC リソースへのトラフィックフロー、VPC 内のリソースには VPC Lattice リソースゲートウェイからのトラフィックが表示されます。

詳細については、VPC Lattice ユーザーガイドを参照してください。

前提条件

VPC リソースを Deadline Cloud サービスマネージドフリートに接続する前に、以下があることを確認してください。

- 接続するリソースを含む VPC を持つ AWS アカウント。
- VPC Lattice リソースを作成および管理するための IAM アクセス許可。
- 少なくとも 1 つのサービスマネージドフリートを持つ Deadline Cloud ファーム。
- アクセス可能にする VPC リソース (FSx、NFS、ライセンスサーバーなど) 。

VPC リソースエンドポイントを設定する

VPC リソースエンドポイントを設定するには、VPC Lattice と にリソースを作成しAWS RAM、それらのリソースを Deadline Cloud のフリートに接続する必要があります。SMF の VPC リソースエンドポイントを設定するには、次の手順を実行します。

1. VPC Lattice でリソースゲートウェイを作成するには、「VPC Lattice ユーザーガイド」の「リソースゲートウェイの作成」を参照してください。
2. VPC Lattice でリソース設定を作成するには、「VPC Lattice ユーザーガイド」の「リソース設定の作成」を参照してください。
3. リソースを Deadline Cloud フリートと共有するには、 でリソース共有を作成します AWS RAM。手順については、「リソース共有の作成」を参照してください。

   リソース共有の作成中に、プリンシパル でドロップダウンからサービスプリンシパルを選択し、 と入力します**fleets.deadline.amazonaws.com**。
4. リソース設定を Deadline Cloud フリートに接続するには、次の手順を実行します。

   a. まだの場合は、Deadline Cloud コンソールを開きます。

   b. ナビゲーションペインで、ファームを選択し、ファームを選択します。

   c. フリートタブを選択し、フリートを選択します。

   d. [Configurations (設定)] タブを選択します。

   e. VPC リソースエンドポイントで、編集を選択します。

   f. 作成したリソース設定を選択し、変更の保存を選択します。

VPC リソースへのアクセス

VPC リソースをフリートに接続すると、ワーカーは次の形式のプライベートドメイン名を使用してリソースにアクセスできます。 `<resource_config_id>.resource-endpoints.deadline.<region>.amazonaws.com`

このドメインはプライベートであり、ワーカーのみがアクセスできます (インターネットやワークステーションからはアクセスできません)。

ワーカーに VPC リソースへのアクセスをマウントまたは設定するには、ホスト設定スクリプトを使用します。ホスト設定スクリプトは、ワーカーの起動時に管理者権限で実行されるため、ファイルシステムのマウント、ネットワーク設定の設定、その他のセットアップタスクを実行できます。

認証とセキュリティ

認証が必要なリソースの場合は、認証情報を AWS Secrets Manager に安全に保存し、ホスト設定スクリプトまたはジョブスクリプトからシークレットにアクセスし、アクセスを制御するための適切なファイルシステムのアクセス許可を実装します。複数のフリート間でリソースを共有する場合は、セキュリティへの影響を考慮してください。たとえば、2 つのフリートが同じ共有ストレージに接続されている場合、1 つのフリートで実行されるジョブは、他のフリートから作成されたアセットにアクセスできる可能性があります。

技術的考慮事項

VPC リソースエンドポイントを使用する場合は、次の点を考慮してください。

- 接続は、ワーカーから VPC リソースにのみ開始でき、VPC リソースからワーカーには開始できません。
- 確立されると、リソース設定が切断されていても、接続はリセットされるまで保持されます。
- VPC Lattice 接続は、追加料金なしでアベイラビリティーゾーン間の接続を自動的に処理します。リソースゲートウェイは VPC リソースとアベイラビリティーゾーンを共有する必要があるため、すべてのアベイラビリティーゾーンにまたがるようにリソースゲートウェイを設定することをお勧めします。
- VPC リソースエンドポイントを通過するトラフィックは、ネットワークアドレス変換 (NAT) を使用します。NAT は、すべてのユースケースと互換性があるわけではありません。たとえば、Microsoft Active Directory (AD) は NAT 経由で接続できません。

VPC Lattice クォータの詳細については、「VPC Lattice のクォータ」を参照してください。

トラブルシューティング

VPC リソースエンドポイントで問題が発生した場合は、以下を確認してください。

- 「mount.nfs: access denied by server while mounting」などのエラーメッセージが表示された場合は、NFS ボリュームのクライアント設定を更新する必要がある場合があります。
- Amazon EC2 インスタンスまたは VPC AWS CloudShell でテストして、リソース設定の設定を確認します。
- シンプルな CLI ジョブで Deadline Cloud 接続をテストします。詳細については、GitHub の「Deadline Cloud サンプル」を参照してください。
- 接続に障害が発生した場合は、リソースゲートウェイのセキュリティグループの設定を確認します。
- VPC アクセスログを有効にして接続をモニタリングします。

サービスマネージドフリートでジョブアタッチメントを使用する

ジョブアタッチメントは、Amazon Simple Storage Service (Amazon S3) を使用して、ワークステーションと Deadline Cloud ワーカー間でファイルを転送します。ジョブアタッチメントを単独で、または共有ストレージと一緒に使用して、ジョブスクリプト、設定ファイル、ローカルに保存されているプロジェクトアセットなど、他のジョブと共有されていないジョブに補助データをアタッチできます。

ジョブアタッチメントの仕組みについては、Deadline Cloud ユーザーガイドの「ジョブアタッチメント」を参照してください。ジョブバンドルで入出力ファイルを指定する方法の詳細については、「」を参照してくださいジョブアタッチメントを使用してファイルを共有する。

ファイルシステムモードを選択する

添付ファイルを含むジョブを送信する場合、 `fileSystem`プロパティを設定することで、ワーカーが Amazon S3 からファイルをロードする方法を選択できます。

- COPIED (デフォルト) – タスクを開始する前に、すべてのファイルをローカルディスクにダウンロードします。すべてのタスクでほとんどの入力ファイルが必要な場合に最適です。
- VIRTUAL – ファイルをオンデマンドでダウンロードする仮想ファイルシステムをマウントします。タスクが入力ファイルのサブセットのみを必要とする場合に最適です。Linux SMF ワーカーでのみ使用できます。

> **Important**
>
> VIRTUAL モードのキャッシュはメモリ消費量を増やす可能性があり、すべてのワークロードに最適化されているわけではありません。本番稼働用ジョブを実行する前にワークロードをテストすることをお勧めします。

ファイルシステムモードの設定の詳細については、「Deadline Cloud ユーザーガイド」の「仮想ファイルシステム」を参照してください。

転送パフォーマンスの最適化

Amazon S3 から SMF ワーカーにファイルを同期する速度は、フリートの Amazon Elastic Block Store (Amazon EBS) ボリューム設定によって異なります。デフォルトでは、SMF ワーカーはベースラインパフォーマンス設定で gp3 Amazon EBS ボリュームを使用します。入力ファイルが大きいワークロードや小さいファイルが多いワークロードでは、Amazon EBS スループットと IOPS 設定を増やすことで転送速度を向上させることができます。これらの設定は、 AWS Command Line Interface () を使用して更新できますAWS CLI。

スループット (MiB/秒)

ボリュームとの間でデータを読み書きできるレート。gp3 ボリュームのデフォルトは 125 MiB/秒、最大は 1,000 MiB/秒です。大規模なシーケンシャルファイル転送の場合は を増やします。

IOPS

1 秒あたりの入出力オペレーション。デフォルトは 3,000 IOPS、gp3 ボリュームの最大は 16,000 IOPS です。多くの小さなファイルを転送する場合は を増やします。

> **Note**
>
> Amazon EBS スループットと IOPS を増やすと、フリートのコストが増加します。料金の詳細については、「Deadline Cloud の料金」を参照してください。

を使用して既存のフリートの Amazon EBS 設定を更新するには AWS CLI

- 次のコマンドを実行します。

```
aws deadline update-fleet \
```

```
  --farm-id farm-0123456789abcdef0 \
  --fleet-id fleet-0123456789abcdef0 \
  --configuration '{
    "serviceManagedEc2": {
      "instanceCapabilities": {
        "vCpuCount": {"min": 4},
        "memoryMiB": {"min": 8192},
        "osFamily": "linux",
        "cpuArchitectureType": "x86_64",
        "rootEbsVolume": {
          "sizeGiB": 250,
          "iops": 6000,
          "throughputMiB": 500
        }
      },
      "instanceMarketOptions": {"type": "spot"}
    }
  }'
```

ジョブ出力をダウンロードする

ジョブが完了したら、Deadline Cloud CLI または AWS Deadline Cloud Monitor (Deadline Cloud Monitor) を使用して出力ファイルをダウンロードします。

```
deadline job download-output --job-id job-0123456789abcdef0
```

ジョブ完了時の出力の自動ダウンロードについては、Deadline Cloud ユーザーガイドの「自動ダウンロード」を参照してください。

ワーカーにカスタムソフトウェアをデプロイして設定する

AWS Deadline Cloud には、ワーカーにカスタムソフトウェア、プラグイン、ツールをデプロイおよび設定するための複数の方法が用意されています。選択する方法は、管理者権限が必要かどうか、ソフトウェアが変更される頻度、ソフトウェアをすべてのジョブで使用できるか特定のジョブのみで使用できるかなど、要件によって異なります。

デプロイ方法を選択する

次の表を使用して、ユースケースに適したデプロイ方法を選択します。

条件	キュー環境	ホスト設定スクリプト	カスタム conda パッケージ
必要な管理者権限	いいえ	あり	なし
実行時	セッション開始	ワーカーの起動	セッション開始
スコープ	キューまたはジョブごと	フリート内のすべてのワーカー	キューまたはジョブごと
ジョブの送信によって制御可能	はい	なし	はい
セットアップの複雑さ	低	中	高
次の用途に適しています	シンプルなプラグイン、スクリプト、環境変数	システムドライバー、Docker、ストレージマウント	依存関係を持つ複雑なアプリケーション

クイックディシジョンガイド:

- 管理者権限またはルート権限が必要ですか? ホスト設定スクリプトを使用します。
- 管理者権限のないシンプルなプラグインまたはスクリプト キュー環境を使用します。
- バージョン管理が必要な複雑なアプリケーションですか? カスタム conda パッケージを作成します。

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で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。これらは、 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` – ジョブにインストールする conda パッケージ一致仕様のスペース区切りリスト。たとえば、Blender 送信者は `blender=3.6`を Blender 3.6 のレンダリング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。

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として、キュー、ジョブ、ステップがあります。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化された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。キュー用に 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。このアプローチにより、 環境をステップで作成してテストし、ジョブに対して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

多くのアプリケーションとフレームワークでは、環境変数を使用して機能設定、ログ記録レベル、表示設定を制御します。Open Job Description (OpenJD) 環境を使用して、スコープ内のすべてのタスクコマンドが継承する環境変数を設定できます。

環境変数スコープ

AWS Deadline Cloud は、キューにアタッチしたキュー環境から環境変数を適用します。ジョブテンプレート内で、OpenJD 環境を使用してジョブレベルとステップレベルで環境変数を定義することもできます。より狭いスコープで定義された変数は、より広いスコープから同じ名前の変数を上書きします。

- キュー環境 – Deadline Cloud のキューにアタッチするテンプレート。変数は、キューに送信されたすべてのジョブに適用されます。固定値のvariablesマップで変数を設定することも、動的値のスクリプトを使用することもできます。
- ジョブ環境 – ジョブテンプレートの jobEnvironments で定義されます。変数は、ジョブ内のすべてのステップとタスクに適用されます。ジョブレベルの変数は、キューレベルの変数を同じ名前で上書きします。
- ステップ環境 – ジョブテンプレートの stepEnvironments で定義されます。変数は、そのステップのタスクにのみ適用されます。ステップレベルの変数は、ジョブレベルまたはキューレベルの変数を同じ名前で上書きします。

キュー環境で変数を設定する

キュー環境の環境変数は、固定値のvariablesマップを使用するか、動的値のonEnterアクションscriptで を使用して設定できます。

次のキュー環境テンプレートは、variablesマップを使用してQT_QPA_PLATFORM変数を に設定しoffscreen、Qt Framework を使用するアプリケーションが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実行できるようにします。

```
specificationVersion: 'environment-2023-09'
environment:
  name: QtOffscreen
  variables:
    QT_QPA_PLATFORM: offscreen
```

仮想環境の変更PATHやアクティブ化などの動的な値の場合は、stdout openjd_env: *VAR*=*value*の形式で行を出力するスクリプトを使用します。openjd_env: プレフィックスは必須です。プレフィックスなしで echo、export、またはその他のシェルメカニズムを使用しても、ジョブやタスクに変数は伝達されません。

次のキュー環境テンプレートは、スクリプトを使用して QT_QPA_PLATFORM変数を設定します。

```
specificationVersion: 'environment-2023-09'
environment:
  name: QtOffscreen
  script:
    actions:
      onEnter:
        command: bash
```

```
        args:
        - "{{Env.File.Enter}}"
    embeddedFiles:
    - name: Enter
      type: TEXT
      data: |
        #!/bin/env bash
        set -euo pipefail
        echo "openjd_env: QT_QPA_PLATFORM=offscreen"
```

キュー環境にアタッチするには、Deadline Cloud コンソールまたは を使用します AWS CLI。詳細については、 AWS Deadline Cloud ユーザーガイドの「キュー環境の作成」を参照してください。次の AWS CLI コマンドは、テンプレートファイルからキュー環境を作成します。

```
aws deadline create-queue-environment \
    --farm-id FARM_ID \
    --queue-id QUEUE_ID \
    --priority 1 \
    --template-type YAML \
    --template file://my-queue-env.yaml
```

conda 仮想環境の作成やアクティブ化などのより複雑な例については、GitHub の Deadline Cloud キュー環境のサンプルを参照してください。

ジョブテンプレートでの変数の設定

ジョブテンプレートで、 jobEnvironmentsまたは stepEnvironmentsエントリにvariablesマップを追加します。各エントリはキーと値のペアで、キーは変数名、値は変数値です。

次のジョブテンプレートは、 QT_QPA_PLATFORM環境変数を に設定しoffscreen、Qt Framework を使用するアプリケーションが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実行できるようにします。

```
specificationVersion: 'jobtemplate-2023-09'
name: MyJob
jobEnvironments:
- name: JobEnv
  variables:
    QT_QPA_PLATFORM: offscreen
```

1 つの環境定義で複数の変数を設定できます。

```
jobEnvironments:
- name: JobEnv
  variables:
    JOB_VERBOSITY: MEDIUM
    JOB_PROJECT_ID: my-project-id
    JOB_ENDPOINT_URL: https://my-host-name/my/path
    QT_QPA_PLATFORM: offscreen
```

{{Param.*ParameterName*}} 構文を使用して、変数値のジョブパラメータを参照できます。

```
jobEnvironments:
- name: JobEnv
  variables:
    JOB_EXAMPLE_PARAM: "{{Param.ExampleParam}}"
```

特定のステップのジョブレベルの変数を上書きするには、同じ変数名でstepEnvironmentsエントリを定義します。次の例ではJOB_PROJECT_ID、ジョブレベルで 値を で定義しproject-12、ステップレベルで 値を で上書きしますstep-project-12。ステップのタスクは、ステップレベルの値を使用します。

```
specificationVersion: 'jobtemplate-2023-09'
name: MyJob
jobEnvironments:
- name: JobEnv
  variables:
    JOB_PROJECT_ID: project-12
steps:
- name: MyStep
  stepEnvironments:
  - name: StepEnv
    variables:
      JOB_PROJECT_ID: step-project-12
```

試す: 環境変数サンプルの実行

Deadline Cloud サンプルリポジトリには、環境変数の設定と表示を示すジョブバンドルが含まれています。サンプルジョブテンプレートは、ジョブレベルとステップレベルの両方で変数を定義し、マージされた結果を出力するタスクを実行します。サンプルを実行し、結果を検査するには、次の手順に従います。

前提条件

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 AWS Deadline Cloud モニターがない場合は、「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
cd deadline-cloud-samples/job_bundles
```

サンプルの実行

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
```

2. Deadline Cloud モニターで、新しいジョブを選択して進行状況をモニタリングします。キューに関連付けられたLinuxフリートにワーカーが使用可能になると、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を選択します。

セッションアクションとその定義の比較

ログビューには 3 つのセッションアクションが表示されます。テキストエディタでファイル job_env_vars/template.yaml を開き、各アクションをジョブテンプレートの定義と比較します。

1. JobEnv セッション起動アクションを選択します。ログ出力には、設定されているジョブレベルの環境変数が表示されます。

```
Setting: JOB_VERBOSITY=MEDIUM
Setting: JOB_EXAMPLE_PARAM=An example parameter value
Setting: JOB_PROJECT_ID=project-12
Setting: JOB_ENDPOINT_URL=https://internal-host-name/some/path
Setting: QT_QPA_PLATFORM=offscreen
```

ジョブテンプレートの次の行は、この環境を定義します。

```
jobEnvironments:
- name: JobEnv
  variables:
    JOB_VERBOSITY: MEDIUM
    JOB_EXAMPLE_PARAM: "{{Param.ExampleParam}}"
    JOB_PROJECT_ID: project-12
    JOB_ENDPOINT_URL: https://internal-host-name/some/path
    QT_QPA_PLATFORM: offscreen
```

2. StepEnv セッションの起動アクションを選択します。ログ出力には、オーバーライドされた を含むステップレベルの変数が表示されますJOB_PROJECT_ID。

```
Setting: STEP_VERBOSITY=HIGH
Setting: JOB_PROJECT_ID=step-project-12
```

ジョブテンプレートの次の行は、この環境を定義します。

```
stepEnvironments:
- name: StepEnv
  variables:
    STEP_VERBOSITY: HIGH
    JOB_PROJECT_ID: step-project-12
```

3. タスク実行セッションアクションを選択します。ログ出力には、タスクで使用できるマージされた環境変数が表示されます。がステップレベルの値 JOB_PROJECT_IDを使用することに注意してくださいstep-project-12。

```
Environment variables starting with JOB_*:
JOB_ENDPOINT_URL=https://internal-host-name/some/path
JOB_EXAMPLE_PARAM='An example parameter value'
JOB_PROJECT_ID=step-project-12
JOB_VERBOSITY=MEDIUM

Environment variables starting with STEP_*:
STEP_VERBOSITY=HIGH
```

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、ディレクトリパスを毎回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では変数を変更できないため、代わりにスクリプトを実行して変更します。スクリプトがモノをセットアップして を変更すると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、RandomSleepCommandの起動とタスク実行の 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 RandomSleepCommand 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 17:25:32-07:00
 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、バックグラウンドデーモンプロセスとしてアプリケーションを 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 パターンを備えた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 アプリケーションでは、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、 コンソールでサービスマネージドフリートに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 つの 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、指定されたすべての環境変数が設定された conda 環境を使用できます。

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。

期限クラウドチャネルBlenderから取得する

次の例は、deadline-cloudconda 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 チャネルによって提供され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「デフォルトのキュー環境」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、 コンソールでサービスマネージドフリートに使用され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 つの 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

ジョブが `deadline-cloud`または `conda-forge`チャネルで利用できないアプリケーションを実行する必要がある場合は、カスタム conda チャネルをホストして独自のパッケージを提供できます。AWS Deadline Cloud (Deadline Cloud) コンソールでキューを作成すると、コンソールはデフォルトで conda キュー環境を追加します。パッケージをジョブで使用できるようにするには、カスタムチャネルをキュー環境に追加します。

conda チャネルは、ファイルシステムや Amazon Simple Storage Service (Amazon S3) バケットなど、さまざまな方法でホストできる静的ホストコンテンツです。Deadline Cloud ファームがアセットに共有ファイルシステムを使用している場合は、その任意のパスをチャネル名として使用できます。 AWS Identity and Access Management (IAM) アクセス許可を使用して、より広範なアクセスのために Amazon S3 バケットでチャネルをホストできます。

パッケージをローカルで構築してテストし、チャネルに公開できます。パッケージをローカルに構築すると、インフラストラクチャのセットアップなしでパッケージビルドレシピの反復処理を簡単に開始できます。Deadline Cloud パッケージ構築キューを使用してパッケージを構築し、チャネルに公開することもできます。パッケージ構築キューは、複数のオペレーティングシステムとアクセラレーター設定のパッケージのメンテナンスを簡素化します。バージョンを更新し、パッケージビルドの完全なセットをどこからでも送信できます。

スタジオと Deadline Cloud ファームのチャネルは、複数の方法で設定できます。1 つの Amazon S3 チャネルを持ち、それを使用するようにすべてのワークステーションとファームホストを設定できます。(AWS DataSync DataSync) を使用して、複数のチャネルとミラーリングを設定することもできます。例えば、Deadline Cloud パッケージ構築キューは、ワークステーションとオンプレミスファームホストのオンプレミスでミラーリングされる Amazon S3 チャネルに発行できます。

トピック

- パッケージをローカルでビルドおよびテストする
- Amazon S3 conda チャネルにパッケージを発行する
- 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
- アプリケーションまたはプラグインの 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
- Deadline Cloud でパッケージビルドを自動化する

パッケージをローカルでビルドおよびテストする

Amazon S3 にパッケージを発行したり、Deadline Cloud ファームで CI/CD オートメーションを設定する前に、ローカルファイルシステムチャネルを使用してワークステーションで conda パッケージを構築およびテストできます。このアプローチにより、レシピをローカルで迅速に反復処理し、パッケージを検証できます。

`rattler-build publish` コマンドはレシピを構築し、結果のパッケージをチャネルにコピーし、1 つのステップでチャネルのインデックスを作成します。ローカルファイルシステムディレクトリをターゲットにすると、ディレクトリが存在しない場合、 によってチャネルが自動的に`rattler-build`作成および初期化されます。

次の手順では、 Blenderの Deadline Cloud サンプルリポジトリの 4.5 サンプルレシピを使用しますGitHub。サンプルリポジトリとは異なるレシピを置き換えるか、独自のレシピを使用できます。

前提条件

開始する前に、ワークステーションに次のツールをインストールします。

- pixi – パッケージのインストール`rattler-build`とテストに使用するパッケージマネージャー。pixi.sh から pixi をインストールします。
- rattler-build – Deadline Cloud conda レシピで使用されるパッケージビルドツール。pixi をインストールしたら、次のコマンドを実行して をインストールします`rattler-build`。

```
pixi global install rattler-build
```

- git – サンプルリポジトリのクローン作成に必要です。ではWindows、 の git Windows bash は、Windowsサンプルレシピの一部に必要なシェルも提供します。

パッケージの構築とローカルチャネルへの発行

この手順では、Deadline Cloud サンプルリポジトリのクローンを作成し、 `rattler-build publish` を使用してパッケージを構築し、ローカルファイルシステムチャネルに公開します。

パッケージを構築してローカルチャネルに公開するには

1. Deadline Cloud サンプル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
```

2. `conda_recipes` ディレクトリを変更します。

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

3. 次のコマンドを実行して 4.5 Blender レシピを構築し、パッケージをローカルチャネルディレクトリに発行します。

   Linux と でmacOS、次のコマンドを実行します。

```
rattler-build publish blender-4.5/recipe/recipe.yaml \
    --to file://$HOME/my-conda-channel
```

   Windows (cmd) で、次のコマンドを実行します。

```
rattler-build publish blender-4.5/recipe/recipe.yaml ^
    --to file://%USERPROFILE%/my-conda-channel
```

   `rattler-build publish` コマンドは次のアクションを実行します。

   - レシピからパッケージを構築します。
   - ディレクトリが存在しない場合は、チャネルディレクトリを作成します。
   - パッケージファイルをチャネルにコピーします。
   - チャネルのインデックスを作成して、パッケージマネージャーがパッケージを検索できるようにします。

パッケージレシピが conda-forge などの特定のチャネルからのパッケージに依存している場合は、 コマンド-c conda-forgeに を追加します。

レシピを変更した後にパッケージを再構築するには、 --build-number=+1を追加してビルド番号を自動的に増分します。

```
rattler-build publish blender-4.5/recipe/recipe.yaml \
    --to file://$HOME/my-conda-channel \
    --build-number=+1
```

の詳細についてはrattler-build publish、rattler-build 公開ドキュメントを参照してください。

ビルドのデバッグ

ビルドが失敗した場合、 は調査できるようにビルドディレクトリrattler-buildを保持します。次のコマンドを実行して、ビルド中のすべての環境変数をセットアップして、ビルド環境でインタラクティブシェルを開きます。

```
rattler-build debug shell
```

デバッグシェルから、ファイルの変更、個々のビルドコマンドの実行、依存関係の追加を行って問題を分離できます。詳細については、rattler-build ドキュメントの「ビルドのデバッグ」を参照してください。

パッケージのテスト

パッケージを構築して公開したら、一時的な pixi プロジェクトを作成します。プロジェクトを使用してローカルチャネルからパッケージをインストールし、正しく動作することを確認します。

パッケージをテストするには

1. 一時テストディレクトリを作成し、ローカルチャネルを使用して pixi プロジェクトを初期化します。

   Linux および でmacOS、次のコマンドを実行します。

   ```
   mkdir package-test-env
   ```

```
cd package-test-env
pixi init --channel file://$HOME/my-conda-channel
```

Windows (cmd) で、次のコマンドを実行します。

```
mkdir package-test-env
cd package-test-env
pixi init --channel file://%USERPROFILE%/my-conda-channel
```

2. パッケージをプロジェクトに追加します。

```
pixi add blender=4.5
```

3. パッケージが正しく動作することを確認します。

```
pixi run blender --version
```

パッケージに満足したら、Deadline Cloud ワーカーがパッケージをインストールできるように、パッケージを Amazon S3 conda チャネルに発行できます。「パッケージを S3 conda チャネルに発行する」を参照してください。

クリーンアップ

テスト後、テストプロジェクトとローカルチャネルを削除できます。

テストリソースをクリーンアップするには

1. テストプロジェクトディレクトリを削除します。

   Linux と でmacOS、次のコマンドを実行します。

```
rm -rf package-test-env
```

   Windows (cmd) で、次のコマンドを実行します。

```
rmdir /s /q package-test-env
```

2. ローカル conda チャネルディレクトリを削除します。

   Linux と でmacOS、次のコマンドを実行します。

```
rm -rf $HOME/my-conda-channel
```

Windows (cmd) で、次のコマンドを実行します。

```
rmdir /s /q %USERPROFILE%\my-conda-channel
```

3. (オプション) ビルドされたパッケージファイルを含むrattler-build出力ディレクトリを削除します。

   Linux と でmacOS、次のコマンドを実行します。

   ```
   rm -rf deadline-cloud-samples/conda_recipes/output
   ```

   Windows (cmd) で、次のコマンドを実行します。

   ```
   rmdir /s /q deadline-cloud-samples\conda_recipes\output
   ```

Amazon S3 conda チャネルにパッケージを発行する

Conda パッケージを Amazon Simple Storage Service (Amazon S3) バケットに発行して、Deadline Cloud (Deadline Cloud) AWS ワーカーが実行中のジョブ用にインストールできるようにします。`rattler-build publish` コマンドは、ローカルファイルシステムチャネルと同じ方法で Amazon S3 で動作します。コマンドはレシピを構築して結果を公開したり、既に構築したパッケージファイルを公開したりできます。どちらの場合も、 コマンドはパッケージをバケットにアップロードし、1 ステップでチャネルのインデックスを作成します。

`rattler-build publish` コマンドは標準の認証情報チェーン AWS を使用して で認証されるため、任意の AWS ツールのように AWS 設定を使用します。認証情報の設定の詳細については、AWS Command Line Interface (AWS CLI) ユーザーガイドの「設定と認証情報ファイルの設定」を参照してください。

前提条件

Amazon S3 にパッケージを発行する前に、以下の前提条件を満たしてください。

- pixi と rattler-build – pixi.sh から pixi をインストールし、 をインストールしますrattler-build。

```
pixi global install rattler-build
```

- git – サンプルリポジトリのクローン作成に必要です。ではWindows、 の git Windows bash は、Windowsサンプルレシピの一部に必要なシェルも提供します。
- Amazon S3 バケット – conda チャネルとして使用する Amazon S3 バケット。Deadline Cloud ファームのジョブアタッチメントバケットを使用するか、別のバケットを作成できます。
- AWS 認証情報 – `aws configure` コマンドまたは `aws login` コマンドを使用して、ワークステーションで認証情報を設定します。詳細については、AWS Command Line Interface ユーザーガイドの「AWS CLIのセットアップ」を参照してください。
- IAM アクセス許可 – (オプション) 認証情報のアクセス許可の範囲を減らすには、Amazon S3 バケットと使用するチャネルプレフィックス (例:) に対して以下のアクセス許可のみを付与する (IAM) ポリシーを使用できます AWS Identity and Access Management `/Conda/*`。
  - `s3:GetObject`
  - `s3:PutObject`
  - `s3:DeleteObject`
  - `s3:ListBucket`
  - `s3:GetBucketLocation`

Amazon S3 チャネルへのパッケージの発行

`s3://` ターゲット`rattler-build publish`で を使用して、Amazon S3 conda チャネルにパッケージを発行します。チャネルがバケットに存在しない場合、 はチャネルを自動的に`rattler-build`初期化します。開始する前に、前提条件を満たしていることを確認してください。

次の例では、 の Deadline Cloud サンプルリポジトリから Blender 4.5 サンプルレシピを公開します GitHub。 https://github.com/aws-deadline/deadline-cloud-samplesサンプルリポジトリとは異なるレシピを置き換えるか、独自のレシピを使用できます。

Amazon S3 チャネルにパッケージを発行するには

1. Deadline Cloud サンプル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
   ```

2. `conda_recipes` ディレクトリを変更します。

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

3. 以下のコマンドを実行してください。*amzn-s3-demo-bucket* をバケット名に置き換えます。

   Linux と でmacOS、次のコマンドを実行します。

   ```
   rattler-build publish blender-4.5/recipe/recipe.yaml \
       --to s3://amzn-s3-demo-bucket/Conda/Default
   ```

   Windows (cmd) で、次のコマンドを実行します。

   ```
   rattler-build publish blender-4.5/recipe/recipe.yaml ^
       --to s3://amzn-s3-demo-bucket/Conda/Default
   ```

   /Conda/Default プレフィックスは、バケット内のチャネルを整理します。別のプレフィックスを使用できますが、プレフィックスはチャネルを参照するすべてのコマンドとキュー設定で一貫している必要があります。

更新されたパッケージを再構築して公開するには、 --build-number=+1を追加してビルド番号を自動的に増分します。

```
rattler-build publish blender-4.5/recipe/recipe.yaml \
    --to s3://amzn-s3-demo-bucket/Conda/Default \
    --build-number=+1
```

パッケージレシピが conda-forge などの特定のチャネルのパッケージに依存している場合は、 コマンド-c conda-forgeに を追加します。

ローカルビルドのファイルなど、既にビルドしたパッケージ.condaファイルを発行することもできます。*amzn-s3-demo-bucket* をバケット名に置き換えます。

```
rattler-build publish output/linux-64/blender-4.5.0-hb0f4dca_0.conda \
    --to s3://amzn-s3-demo-bucket/Conda/Default
```

パッケージのテスト

パッケージを公開したら、一時 pixi プロジェクトを作成して、パッケージが正しく動作することを確認します。プロジェクトは、Amazon S3 チャネルからパッケージをインストールします。

パッケージをテストするには

1. 一時的なテストディレクトリを作成し、Amazon S3 チャネルを使用して pixi プロジェクトを初期化します。*amzn-s3-demo-bucket* をバケット名に置き換えます。

```
mkdir package-test-env
cd package-test-env
pixi init --channel s3://amzn-s3-demo-bucket/Conda/Default
```

2. パッケージをプロジェクトに追加します。

```
pixi add blender=4.5
```

3. パッケージが正しく動作することを確認します。

```
pixi run blender --version
```

クリーンアップ

テストが完了したら、テストプロジェクトディレクトリを削除します。

テストリソースをクリーンアップするには

- テストプロジェクトディレクトリを削除します。

  Linux と でmacOS、次のコマンドを実行します。

```
rm -rf package-test-env
```

  Windows (cmd) で、次のコマンドを実行します。

```
rmdir /s /q package-test-env
```

ビルドのデバッグ

ビルドが失敗した場合、 は調査できるようにビルドディレクトリ`rattler-build`を保持します。次のコマンドを実行して、ビルド中のすべての環境変数をセットアップして、ビルド環境でインタラクティブシェルを開きます。

```
rattler-build debug shell
```

デバッグシェルから、ファイルの変更、個々のビルドコマンドの実行、依存関係の追加を行って問題を分離できます。詳細については、rattler-build ドキュメントの「ビルドのデバッグ」を参照してください。

他のプラットフォーム用のパッケージの構築

`rattler-build publish` コマンドは、コマンドを実行するワークステーションのオペレーティングシステム用のパッケージを構築します。Deadline Cloud フリートがワークステーションとは異なるオペレーティングシステムを使用している場合、またはパッケージに他のホスト要件がある場合は、次のオプションがあります。

- ターゲットオペレーティングシステムに一致するホスト`rattler-build publish`で を実行します。たとえば、実行中の Amazon Elastic Compute Cloud (Amazon EC2) インスタンスLinuxを使用して、Linuxフリートのパッケージを構築します。
- Deadline Cloud パッケージ構築キューを使用して、ターゲットプラットフォームでのビルドを自動化します。「パッケージ構築キューを作成する」を参照してください。
- (アドバンスト) クロスコンパイルを使用して、ワークステーションとは異なるプラットフォームのパッケージを構築します。詳細については、rattler-build ドキュメントの「クロスコンパイル」を参照してください。

次の手順

Amazon S3 conda チャネルにパッケージを発行したら、チャネルを使用するように Deadline Cloud キューを設定します。

- カスタム conda パッケージの本番稼働用キューのアクセス許可を設定する – 本番稼働用キューに Amazon S3 conda チャネルへの読み取り専用アクセスを許可します。
- conda チャネルをキュー環境に追加する – Amazon S3 conda チャネルからパッケージをインストールするようにキュー環境を設定します。

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フィールドを提供します。

本番キューの 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。次の手順に従ってください。

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. 新しく作成した 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 の柔軟なチャネル優先度を有効にします。Blender 4.2 が新しいチャネルとdeadline-cloudチャネルの両方にあるblender=4.2かどうかをリクエストするジョブの場合、パッケージはチャネルリストの最初のチャネルからプルされます。指定されたパッケージバージョンが最初のチャネルで見つからない場合、後続のチャネルはパッケージバージョンの順にチェックされます。

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。

アプリケーションまたはプラグインの conda パッケージを作成する

conda パッケージは、任意の言語で記述されたソフトウェアの圧縮アーカイブです。Conda はさまざまなオペレーティングシステムとアーキテクチャの組み合わせをサポートしているため、Blender、Maya、 などの完全なアプリケーションを Python やその他の言語用のライブラリ Nukeとともにパッケージ化できます。conda パッケージの詳細については、conda ドキュメントの「パッケージ」を参照してください。

conda パッケージを使用するには、仮想環境にインストールします。conda 仮想環境には、パッケージがインストールされているプレフィックスディレクトリがあります。パッケージのインストールでは、サポートされている場合にファイルのハードリンクまたは再リンクを使用するため、同じパッケージで複数の環境を作成しても、ディスク容量が大幅に増えることはありません。仮想環境を使用するには、仮想環境をアクティブ化して環境変数を設定します。アクティベーションは、パッケージが提供するスクリプトを実行し、各パッケージに PATH やその他の環境変数を変更する機会を与えます。Conda パッケージには通常アプリケーションまたはライブラリが含まれていますが、柔軟なアクティベーションにより、共有ファイルシステムにインストールされているアプリケーションを指すこともできます。

カスタムパッケージの作成には 3 つの段階があります。1 つのレシピにはビルド手順が含まれており、1 つのパッケージはビルドされたアーティファクト (.conda または .tar.bz2 ファイル) であり、1 つのチャネルはインストール用のパッケージをホストします。rattler-build publish コマンドは 3 つのステップをすべて処理します。レシピをパッケージに構築してチャネルに発行することも、パッケージアーティファクトを直接使用して公開することもできます。

conda-forge コミュニティは、さまざまなオープンソースソフトウェアのパッケージレシピを維持し、パッケージアーティファクトをconda-forgeチャネルでホストします。をパッケージソースconda-forgeとして含めるようにキューを設定し、実行する conda-forge パッケージに依存するカスタムパッケージを構築できます。の場合Linux、conda-forge は CUDA サポートを含む完全なコンパイラツールチェーンをホストし、一貫したコンパイルとリンクのオプションが選択されます。conda-forge パッケージを独自のレシピの依存関係として使用したり、同じ環境のカスタムパッケージと一緒にインストールしたりできます。

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。

> **Note**
>
> 大規模なアプリケーションでは、ソースアーカイブ、抽出されたファイル、ビルド出力に数十 GB の空きディスク容量が必要になる場合があります。パッケージビルド出力に十分な空き容量があるディスクを使用していることを確認してください。

アプリケーションをパッケージ化する

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. 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`$CONDA_PREFIX/opt/<application-name>`。これにより、 `bin`や などの標準プレフィックスディレクトリから分離されます`lib`。
2. シンボリックリンクまたは起動スクリプトを に追加`$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。たとえば、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. アプリケーションがプラグインをサポートしている場合は、プラグインパッケージが仮想環境でアプリケーションと統合するために従うべき明確な規則を指定します。たとえば、Maya2026 サンプルレシピは、この規則をMayaプラグインに文書化しています。
5. 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。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には、conda チャネルにプライベート Amazon S3 バケットを使用することをお勧めします。

`deadline-cloud` チャネル内のパッケージのサンプルレシピは、 の Deadline Cloud サンプルリポジトリで入手できますGitHub。

プラグインをパッケージ化する

アプリケーションプラグインは、独自の conda パッケージとしてパッケージ化できます。プラグインパッケージを作成するときは、次のガイドラインに従ってください。

- ビルドレシピ に、ビルド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。
- 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。

アダプターパッケージ

一部の Deadline Cloud アプリケーション統合では、アプリケーションインターフェイスを拡張するアダプターを使用して、ジョブテンプレートの記述を簡素化します。アダプターは、バックグラウンドデーモンの実行、ステータスのレポート、パスマッピングの適用をサポートするコマンドラインインターフェイスです。詳細については、「」の「Open Job Description Adaptor Runtime」を参照してくださいGitHub。例えば、 の deadline-cloud-for-maya GitHubには、統合されたジョブ送信 GUI と、サービスマネージドフリートのmaya-openjdパッケージとして利用可能なMayaアダプターが含まれています。

Deadline Cloud 送信者 GUIs からのジョブ送信には、ジョブを実行するための仮想環境に含める conda パッケージを指定するCondaPackagesパラメータ値が含まれています。のCondaPackagesパラメータ値Mayaは通常 `maya=2025.* maya-openjd=0.15.* maya-mtoa`のようになります。プラグインパッケージの代替エントリが含まれている場合があります。キュー環境がジョブを実行するための conda 仮想環境を設定すると、これらのパッケージ名とバージョンの制約が互換性があるように解決され、実行に必要なすべての依存関係パッケージが追加されます。各アダプターとプラグインパッケージは、 のバージョン、Python のバージョンMaya、その他の依存関係など、互換性のあるものを指定します。

の maya-openjd レシピなどのサンプルを使用して独自のアダプターパッケージを構築するには GitHub、conda-forge が提供する Python やその他の依存関係用のパッケージをビルドできます。依存関係を満たすには、まず期限と openjd-adaptor-runtime レシピを構築する必要があります。

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 conda ビルドレシピのサンプルを作成しました。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、最も一般的な設定と一致するように、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/recipe.yaml にあります。ファイルが YAML を生成するテンプレートである方法など、詳細については、conda build meta.yaml ドキュメントも参照してください。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、パッケージが任意のディレクトリプレフィックスの conda 仮想環境にインストールされた場合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、バイナリでアプリケーションを再パッケージ化する場合は、それらを変更する必要があります。

パッケージ化してい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、依存関係が実行ファイルまたはそれを必要とする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、絶対的なパスはありません。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に、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 互換シェルの 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 のインストール時に 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。このスクリプトには、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が記載されています。

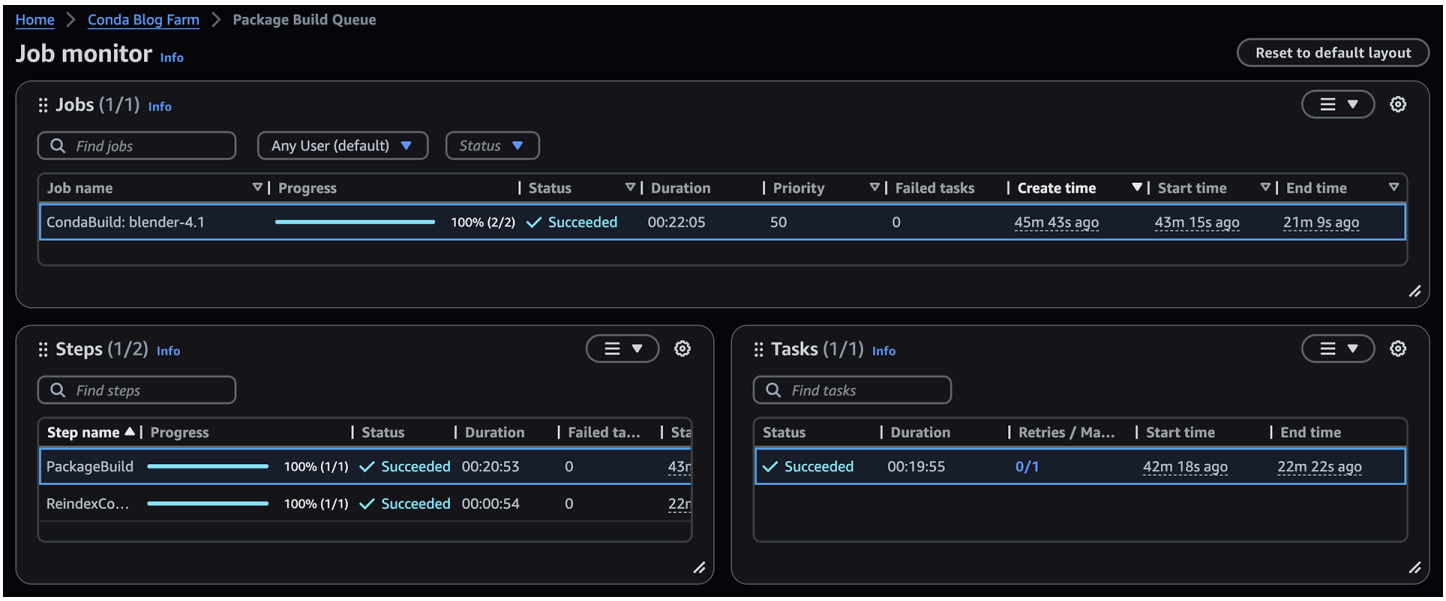
```
./submit-package-job blender-4.2/
```

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 つのステップが表示されます。パッケージを構築し、インデックスを再作成します。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、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モニタに表示されます。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パッケージを構築し、結果を Amazon S3 にアップロードします。

アクションが実行されると、構造化された形式でログが 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セットアップ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。

先ほどダウンロードした 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 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またはプラグインの 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、 が適用される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、プラグインは、含まれている方法でアプリケーションと統合する必要があり、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。これは、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の設定から分離し、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いずれかに .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ビルドと Windowsビルドの両方を指定し、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、新しく起動した 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、アプリケーションが必要とする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に対して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無効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 などのユーティリティ変数を設定する方法`MAYA_LOCATION`と、ソフトウェア`MAYA_VERSION`が実行中の を理解するために使用できる Maya 。

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 Windowsrecipe/build_win.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。

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、 パッケージ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。

- 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。
- 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に`$PREFIX/usr/autodesk/maya$MAYA_VERSION/modules`、仮想環境のMayaモジュールパス を設定します。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、パッケージの構築`maya`時と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、 パッケージ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 レンダー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および 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-forge チャネルを `CondaChannels` conda キュー環境のデフォルトリストに追加して、`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 でパッケージビルドを自動化する

CI/CD ワークフローの場合、または複数のオペレーティングシステムのパッケージを構築する必要がある場合は、Deadline Cloud パッケージ構築キューを作成できます。キューはフリートでビルドジョブをスケジュールし、パッケージをビルドして Amazon Simple Storage Service (Amazon S3) conda チャネルに発行します。これにより、必要なすべての設定でソフトウェアリリースの継続的なパッケージビルドの維持が簡素化されます。

(AWS CloudFormation CloudFormation) テンプレートを使用するか、Deadline Cloud コンソールから手動でパッケージ構築キューを作成できます。 CloudFormation テンプレートは、本番稼働用キューとパッケージ構築キューが既に設定されている完全なファームをデプロイします。コンソールからキューを作成すると、個々の設定をより詳細に制御できます。

を使用してパッケージ構築キューを作成する CloudFormation

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 ファームを作成できます。テンプレートは、プライベート Amazon S3 conda チャネルを使用して、本稼働キューとパッケージ構築キューを設定します。

テンプレートをデプロイする前に、ジョブアタッチメントと conda チャネルを保持する Amazon S3 バケットを作成します。Amazon S3 コンソールからバケットを作成できます。テンプレートをデプロイするときは、バケット名が必要です。

CloudFormation テンプレートをデプロイするには

1. の Deadline Cloud サンプルリポジトリから deadline-cloud-starter-farm-template.yaml テンプレートをダウンロードしますGitHub。
2. CloudFormation コンソールから、スタックの作成を選択し、新しいリソース (標準) を使用します。
3. テンプレートファイルをアップロードするオプションを選択し、`deadline-cloud-starter-farm-template.yaml`ファイルをアップロードします。
4. などのスタックの名前を入力し**`StarterFarm`**、ジョブアタッチメントと conda チャネルの Amazon S3 バケットの名前を指定します。
5. CloudFormation コンソールの手順に従って、スタックの作成を完了します。

テンプレートパラメータとカスタマイズオプションの詳細については、 の Deadline Cloud サンプルリポジトリのスターターファーム README を参照してくださいGitHub。

コンソールからパッケージ構築キューを作成する

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 Amazon 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か、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

パッケージ構築キューがキューの Amazon 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。このロールには、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. アクセス許可タブで、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": "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"
    }
   }
  },
```

パッケージビルドジョブを送信する

パッケージ構築キューを作成し、キューのアクセス許可を設定したら、ジョブを送信して conda パッケージを構築できます。の Deadline Cloud サンプルリポジトリのsubmit-package-jobスクリプトは、conda レシピのビルドジョブGitHubを送信します。

また、以下も必要になります。

- ワークステーションにインストールされている Deadline Cloud CLI。
- アクティブな AWS Deadline Cloud Monitor (Deadline Cloud Monitor) ログインセッション。
- Deadline Cloud サンプルリポジトリのクローン。

パッケージビルドジョブを送信するには

1. 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. サンプルリポジトリの conda_recipes ディレクトリに変更します。

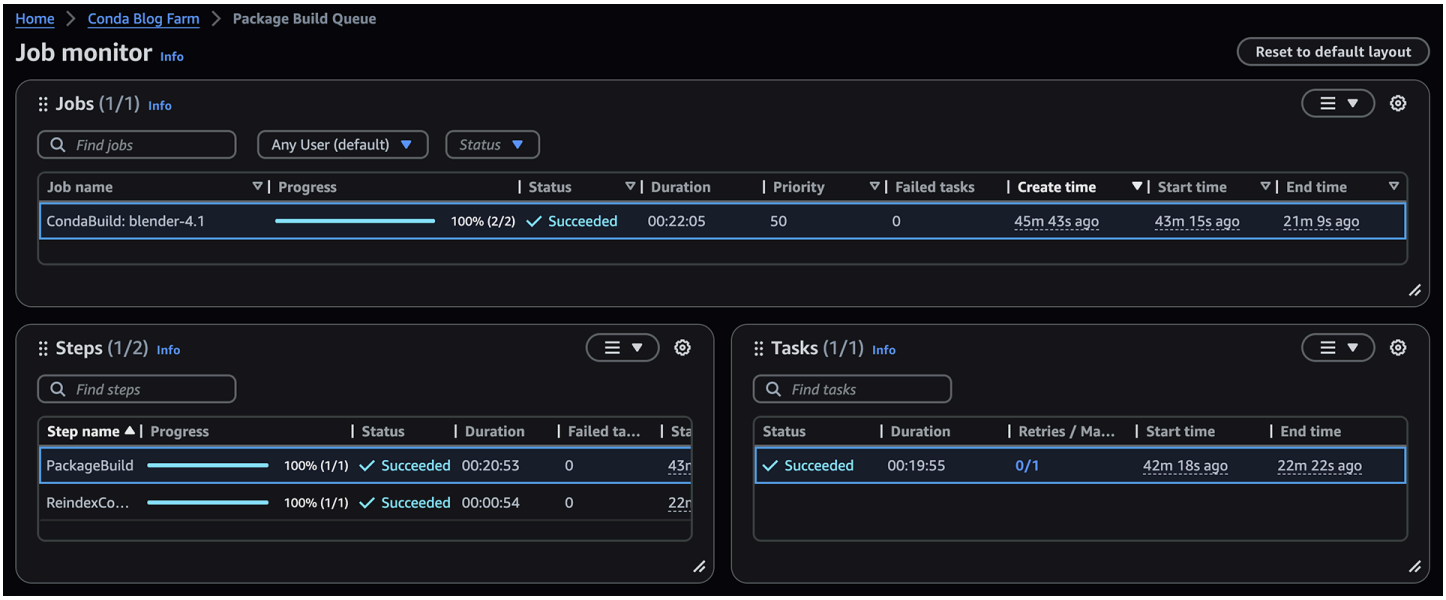
```
cd deadline-cloud-samples/conda_recipes
```

3. レシピディレクトリを使用してsubmit-package-jobスクリプトを実行します。次の例では、4.5 Blender レシピを構築します。

```
./submit-package-job blender-4.5/
```

   レシピにまだダウンロードしていないソースアーカイブが必要な場合、スクリプトはダウンロード手順を提供します。アーカイブをダウンロードし、スクリプトを再度実行します。

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



モニターには、パッケージの構築と conda チャネルのインデックス再作成の 2 つのステップが表示されます。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、モニターにセッションアクションが表示されます。

- アタッチメントの同期 – 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 – キュー環境アクション。ビルドジョブは conda パッケージを指定しないため、このアクションはすぐに終了します。
- CondaBuild Env の起動 – 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使用して conda 仮想環境を作成します。
- タスク実行 – パッケージを構築し、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、Amazon CloudWatch (CloudWatch) にログが送信されます。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、環境の設定と削除に関する追加のログを表示します。

管理者権限を持つホスト設定スクリプトを実行する

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 (sudo では Linux、 では 管理者Windows) 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、昇格されたアクセス許可で実行されます。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お客様の責任です。
> ホスト設定スクリプトを使用する場合は、フリートの状態をモニタリングする責任があります。

ホスト設定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
- などのサードパーティーのクラウドストレージソリューションをインストールしますLucidLink。チュートリアルについては、「 for M&E ブログ」の「Deadline Cloud のサービスマネージドフリートスクリプトを使用して LucidLink をセットアップする」を参照してください。 AWS

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 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

Linux

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
```

Server

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きます。
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します。

> **Tip**
>
> ホスト設定スクリプトをテストするときは、フリートの最大ワーカー数を 1 に設定して、スクリプトで反復しながら複数のワーカーを起動しないようにします。

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で、 という名前のストリームを探します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。

ホスト設定スクリプトの実行をモニタリングする

ホスト設定スクリプトを使用すると、Deadline Cloud ワーカーを完全に制御できます。任意のソフトウェアパッケージをインストールしたり、オペレーティングシステムパラメータを再設定したり、共有ファイルシステムをマウントしたりできます。この高度な機能と Deadline Cloud の数千のワーカーにスケールする機能により、設定スクリプトが正常に実行されるか失敗したかをモニタリングできます。

ホスト設定スクリプトの実行をモニタリングするには、次のソリューションをお勧めします。

CloudWatch Logs のモニタリング

すべてのフリートホスト設定ログは、フリートの CloudWatch ロググループ、特にワーカーの CloudWatch ログストリームにストリーミングされます。たとえば、 `/aws/deadline/farm-123456789012/fleet-777788889999`はファーム 123456789012、フリート のロググループです777788889999。

各ワーカーは、 などの専用ログストリームをプロビジョニングしますworker-123456789012。ホスト設定ログには、ホスト設定スクリプトの実行やホスト設定スクリプトの実行完了、終了コード: 0 などのログバナーが含まれます。スクリプトの終了コードは、完成したバナーに含まれ、CloudWatch ツールを使用してクエリできます。

CloudWatch Logs Insights

CloudWatch Logs Insights は、ログ情報を分析するための高度な機能を提供します。たとえば、次の Log Insights クエリは、ホスト設定の終了コードを時間でソートして解析します。

```
fields @timestamp, @message, @logStream, @log
| filter @message like /Finished running Host Configuration Script/
| parse @message /exit code: (?<exit_code>\d+)/
| display @timestamp, exit_code
| sort @timestamp desc
```

CloudWatch Logs Insights の詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「CloudWatch Logs Insights で口グデータを分析する」を参照してください。

ワーカーエージェントの構造化ログ記録

Deadline Cloud のワーカーエージェントは、構造化された JSON ログを CloudWatch に発行します。ワーカーエージェントは、ワーカーの状態を分析するためのさまざまな構造化ログを提供しま

す。詳細については、GitHub の「Deadline Cloud ワーカーエージェントのログ記録」を参照してください。

構造化ログの属性は、Log Insights のフィールドに解凍されます。この CloudWatch 機能を使用して、ホスト設定の起動失敗をカウントおよび分析できます。たとえば、カウントクエリとビンクエリを使用して、障害が発生する頻度を判断できます。

```
fields @timestamp, @message, @logStream, @log
| sort @timestamp desc
| filter message like /Worker Agent host configuration failed with exit code/
| stats count(*) by exit_code, bin(1h)
```

メトリクスとアラームの CloudWatch メトリクスフィルター

CloudWatch メトリクスフィルターを設定して、ログから CloudWatch メトリクスを生成できます。メトリクスフィルターを使用すると、ホスト設定スクリプトの実行をモニタリングするためのアラームとダッシュボードを作成できます。

メトリックスフィルタを作成するには

1. CloudWatch コンソールを開きます。
2. ナビゲーションペインで、ログ、ロググループを選択します。
3. フリートのロググループを選択します。
4. [Create metric filter] (メトリクスフィルターの作成) を選択します。
5. 次のいずれかを使用してフィルターパターンを定義します。
   - 成功メトリクスの場合:

     ```
     {$.message = "*Worker Agent host configuration succeeded.*"}
     ```

   - 失敗メトリクスの場合:

     ```
     {$.exit_code != 0 && $.message = "*Worker Agent host configuration failed with
      exit code*"}
     ```

6. Next を選択して、次の値を持つメトリクスを作成します。
   - メトリクス名前空間: メトリクス名前空間 (例: **MyDeadlineFarm**)
   - メトリクス名: リクエストされたメトリクス名 (例: **host_config_failure**)

- メトリクス値: **1** (各インスタンスは 1 のカウントです)
- デフォルト値: 空のままにします
- 単位: **Count**

メトリクスフィルターを作成したら、ホスト設定の障害率の上昇に対してアクションを実行するように標準の CloudWatch アラームを設定したり、メトリクスを CloudWatch ダッシュボードに追加して day-to-dayやモニタリングを行うことができます。

詳細については、Amazon CloudWatch Logs ユーザーガイド」の「フィルタとパターン構文」を参照してください。

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する時間数に基づいて追跡および請求します。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を提供しません。
- 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。以下の例では、ライセンスサーバーがライセンスを提供できない場合、Deadline Cloud の使用ベースのライセンスが使用されます。ライセンスを使い切った後に使用量ベースのライセンスを使用しないパイプラインまたは製品に適用されないセクションを削除します。

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: 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で AWS インスタンスが設定されてい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。

ライセンス環境変数の設定については、「」を参照してくださいステップ 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。カスタムライセンスサーバーのセットアップでは、ライセンスサーバーアドレスは Amazon VPC エンドポイントではなく localhost のままになります。

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。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Action": [
                "ssm:StartSession"
            ],
            "Resource": [
                "arn:aws:ssm:us-east-1::document/AWS-
StartPortForwardingSession",
                "arn:aws:ec2:us-east-1:111122223333: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。

キューに新しいキュー環境を追加するには

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: "2701,2702,7075,2703,6101,1715,1716,1717,7054,7055,30304"
   default: "2701,2702,7075,2703,6101,1715,1716,1717,7054,7055,30304"
environment:
 name: BYOL License Forwarding
 variables:
  example_LICENSE: 2701@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/
windows/SessionManagerPlugin.zip" -o "{{Session.WorkingDirectory}}/ssm-
plugin.zip"
      powershell -Command "Expand-Archive -Path '{{Session.WorkingDirectory}}/
ssm-plugin.zip' -DestinationPath '{{Session.WorkingDirectory}}/ssm-plugin'
 -Force; Expand-Archive -Path '{{Session.WorkingDirectory}}/ssm-plugin/
package.zip' -DestinationPath '{{Session.WorkingDirectory}}/ssm-plugin/package'
 -Force"
      conda activate
      python "{{Env.File.StartSession}}" "{{Session.WorkingDirectory}}/ssm-
plugin/package/bin/session-manager-plugin.exe"
   - 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
      import os
      import tempfile

      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)}")

      # Enabling UBL after the BYOL has run out requires prepending the BYOL
 configuration to the existing license setup
      # Remove the sections that do not apply to your pipeline, or you do not
 want to use UBL after exhausting the BYOL licenses.
      # The port numbers used may not match what your license server is serving.

      # Arnold
      os.environ["ADSKFLEX_LICENSE_FILE"] = f"2701@localhost;
{os.environ.get('ADSKFLEX_LICENSE_FILE', '')}"
      print(f"openjd_env:
 ADSKFLEX_LICENSE_FILE={os.environ['ADSKFLEX_LICENSE_FILE']}")

      # Cinema4D
      os.environ["g_licenseServerRLM"] = f"localhost:7057;
{os.environ.get('g_licenseServerRLM', '')}"
      print(f"openjd_env:
 g_licenseServerRLM={os.environ['g_licenseServerRLM']}")

      # Nuke
      os.environ["foundry_LICENSE"] = f"6101@localhost;
{os.environ.get('foundry_LICENSE', '')}"
      print(f"openjd_env: foundry_LICENSE={os.environ['foundry_LICENSE']}")

      # SideFX
      os.environ["SESI_LMHOST"] = f"localhost:1715;
{os.environ.get('SESI_LMHOST', '')}"
      print(f"openjd_env: SESI_LMHOST={os.environ['SESI_LMHOST']}")

      # Redshift and Red Giant
```

```
      os.environ["redshift_LICENSE"] = f"7054@localhost;7055@localhost;
{os.environ.get('redshift_LICENSE', '')}"
      print(f"openjd_env: redshift_LICENSE={os.environ['redshift_LICENSE']}")

      # V-Ray doesn't support multiple license servers in a single environment
 variable
      # See https://documentation.chaos.com/space/LIC5/125050770/Sharing+a
+License+Configuration+in+a+Network
      vray_license = os.environ.get('VRAY_AUTH_CLIENT_SETTINGS', '')
      xml_content = """<VRLClient>
        <LicServer>
          <Host>localhost</Host>
          <Port>30304</Port>"""

      if vray_license and vray_license.startswith('licset://'):
          server_parts = vray_license.removeprefix('licset://').split(':')
          if len(server_parts) >= 2:
              xml_content += f"""
          <Host1>{server_parts[0]}</Host1>
          <Port1>{server_parts[1]}</Port1>"""

      xml_content += """
          <User></User>
          <Pass></Pass>
        </LicServer>
      </VRLClient>"""

      temp_dir = tempfile.gettempdir()
      xml_path = os.path.join(temp_dir, 'vrlclient.xml')

      with open(xml_path, 'w') as f:
          f.write(xml_content)

      os.environ["VRAY_AUTH_CLIENT_FILE_PATH"] = temp_dir
      print(f"openjd_env:
 VRAY_AUTH_CLIENT_FILE_PATH={os.environ['VRAY_AUTH_CLIENT_FILE_PATH']}")

      # Clear the existing VRAY_AUTH_CLIENT_SETTINGS so only the vrlclient.xml
 file is used.
      os.environ["VRAY_AUTH_CLIENT_SETTINGS"] = ''
      print(f"openjd_env:
 VRAY_AUTH_CLIENT_SETTINGS={os.environ['VRAY_AUTH_CLIENT_SETTINGS']}")

      # Print out the created xml file's contents
```

```
      print(f"V-Ray configuration file: {xml_path}")
      with open(xml_path, 'r') as f:
          print(f"{f.read()}")
```

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: "2701,2702,7075,2703,6101,1715,1716,1717,7054,7055,30304"
   default: "2701,2702,7075,2703,6101,1715,1716,1717,7054,7055,30304"
environment:
 name: BYOL License Forwarding
 variables:
  example_LICENSE: 2701@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
      import os
      import tempfile

      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)}")

      # Enabling UBL after the BYOL has run out requires prepending the BYOL
 configuration to the existing license setup
      # Remove the sections that do not apply to your pipeline, or you do not
 want to use UBL after exhausting the BYOL licenses.
      # The port numbers used may not match what your license server is serving.

      # Arnold
      os.environ["ADSKFLEX_LICENSE_FILE"] = f"2701@localhost:
{os.environ.get('ADSKFLEX_LICENSE_FILE', '')}"
      print(f"openjd_env:
 ADSKFLEX_LICENSE_FILE={os.environ['ADSKFLEX_LICENSE_FILE']}")

      # Nuke
      os.environ["foundry_LICENSE"] = f"6101@localhost:
{os.environ.get('foundry_LICENSE', '')}"
      print(f"openjd_env: foundry_LICENSE={os.environ['foundry_LICENSE']}")

      # SideFX
      os.environ["SESI_LMHOST"] = f"localhost:1715;
{os.environ.get('SESI_LMHOST', '')}"
      print(f"openjd_env: SESI_LMHOST={os.environ['SESI_LMHOST']}")

      # Redshift and Red Giant
      os.environ["redshift_LICENSE"] = f"7054@localhost:7055@localhost:
{os.environ.get('redshift_LICENSE', '')}"
      print(f"openjd_env: redshift_LICENSE={os.environ['redshift_LICENSE']}")

      # V-Ray doesn't support multiple license servers in a single environment
 variable
      # See https://documentation.chaos.com/space/LIC5/125050770/Sharing+a
+License+Configuration+in+a+Network
      vray_license = os.environ.get('VRAY_AUTH_CLIENT_SETTINGS', '')
      xml_content = """<VRLClient>
```

```
        <LicServer>
          <Host>localhost</Host>
          <Port>30304</Port>"""

    if vray_license and vray_license.startswith('licset://'):
        server_parts = vray_license.removeprefix('licset://').split(':')
        if len(server_parts) >= 2:
            xml_content += f"""
        <Host1>{server_parts[0]}</Host1>
        <Port1>{server_parts[1]}</Port1>"""

    xml_content += """
        <User></User>
        <Pass></Pass>
      </LicServer>
    </VRLClient>"""

    temp_dir = tempfile.gettempdir()
    xml_path = os.path.join(temp_dir, 'vrlclient.xml')

    with open(xml_path, 'w') as f:
        f.write(xml_content)

    os.environ["VRAY_AUTH_CLIENT_FILE_PATH"] = temp_dir
    print(f"openjd_env:
VRAY_AUTH_CLIENT_FILE_PATH={os.environ['VRAY_AUTH_CLIENT_FILE_PATH']}")

    # Clear the existing VRAY_AUTH_CLIENT_SETTINGS so only the vrlclient.xml
file is used.
    os.environ["VRAY_AUTH_CLIENT_SETTINGS"] = ''
    print(f"openjd_env:
VRAY_AUTH_CLIENT_SETTINGS={os.environ['VRAY_AUTH_CLIENT_SETTINGS']}")

    # Print out the created xml file's contents
    print(f"V-Ray configuration file: {xml_path}")
    with open(xml_path, 'r') as f:
        print(f"{f.read()}")
```

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)

- Bring your own license (BYOL) を使い切った後に使用量ベースのライセンス (UBL) を使用する場合は、ライセンスサーバーに対してポートが正しいことを確認してください。BYOL を使い切った後に UBL を使用しない場合は、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 に設定したままにすることも、複数のキュー環境間で優先順位が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。

   キュー環境を設定すると、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、以下の手順に従います。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: CloudFormation テンプレートのセットアップ

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る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 CloudFormation テンプレートを使用します。

```
AWSTemplateFormatVersion: 2010-09-09
Description: "Create &ADC; 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. &ADC; SMF packages are
```

```
            installed from the "deadline-cloud" channel that is configured by
 &ADC;.

            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: ""
        - 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:
 "2701,2702,7075,2703,6101,1715,1716,1717,7054,7055,30304"
          default: "2701,2702,7075,2703,6101,1715,1716,1717,7054,7055,30304"
        environment:
        name: BYOL License Forwarding
        variables:
          example_LICENSE: 2701@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
```

```
            import os
            import tempfile

            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)}")

            # Enabling UBL after the "bring your own license" (BYOL) has run out
requires prepending the BYOL configuration to the existing license setup
            # Remove the sections that do not apply to your pipeline, or you do
not want to use UBL after exhausting the BYOL licenses.
            # The port numbers used may not match what your license server is
serving.

            # Arnold
```

```
            os.environ["ADSKFLEX_LICENSE_FILE"] = f"2701@localhost:
{os.environ.get('ADSKFLEX_LICENSE_FILE', '')}"
            print(f"openjd_env:
 ADSKFLEX_LICENSE_FILE={os.environ['ADSKFLEX_LICENSE_FILE']}")

            # Nuke
            os.environ["foundry_LICENSE"] = f"6101@localhost:
{os.environ.get('foundry_LICENSE', '')}"
            print(f"openjd_env: foundry_LICENSE={os.environ['foundry_LICENSE']}")

            # SideFX
            os.environ["SESI_LMHOST"] = f"localhost:1715;
{os.environ.get('SESI_LMHOST', '')}"
            print(f"openjd_env: SESI_LMHOST={os.environ['SESI_LMHOST']}")

            # Redshift and Red Giant
            os.environ["redshift_LICENSE"] = f"7054@localhost:7055@localhost:
{os.environ.get('redshift_LICENSE', '')}"
            print(f"openjd_env:
 redshift_LICENSE={os.environ['redshift_LICENSE']}")

            # V-Ray doesn't support multiple license servers in a single
 environment variable
            # See https://documentation.chaos.com/space/LIC5/125050770/Sharing+a
+License+Configuration+in+a+Network
            vray_license = os.environ.get('VRAY_AUTH_CLIENT_SETTINGS', '')
            xml_content = """<VRLClient>
              <LicServer>
                <Host>localhost</Host>
                <Port>30304</Port>"""

            if vray_license and vray_license.startswith('licset://'):
                server_parts = vray_license.removeprefix('licset://').split(':')
                if len(server_parts) >= 2:
                    xml_content += f"""
                <Host1>{server_parts[0]}</Host1>
                <Port1>{server_parts[1]}</Port1>"""

            xml_content += """
                <User></User>
                <Pass></Pass>
              </LicServer>
            </VRLClient>"""
```

```
            temp_dir = tempfile.gettempdir()
            xml_path = os.path.join(temp_dir, 'vrlclient.xml')

            with open(xml_path, 'w') as f:
                f.write(xml_content)

            os.environ["VRAY_AUTH_CLIENT_FILE_PATH"] = temp_dir
            print(f"openjd_env:
VRAY_AUTH_CLIENT_FILE_PATH={os.environ['VRAY_AUTH_CLIENT_FILE_PATH']}")

            # Clear the existing VRAY_AUTH_CLIENT_SETTINGS so only the
vrlclient.xml file is used.
            os.environ["VRAY_AUTH_CLIENT_SETTINGS"] = ''
            print(f"openjd_env:
VRAY_AUTH_CLIENT_SETTINGS={os.environ['VRAY_AUTH_CLIENT_SETTINGS']}")

            # Print out the created xml file's contents
            print(f"V-Ray configuration file: {xml_path}")
            with open(xml_path, 'r') as f:
                print(f"{f.read()}")
```

3. 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)
   - テンプレートで **example_LICENSE: 2700@localhost**を置き換えてライセンス環境変数を追加する

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。ライセンスサーバーは、サービスマネージドフリートで自動的に設定されます。以下のセットアップは、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
- Cinema 4D – 7057、TCP、IPv4, IPv6
- KeyShot – 2703、TCP、IPv4, IPv6
- Foundry Nuke – 6101、TCP、IPv4, IPv6
- Red Giant – 7055、TCP、IPV4
- Redshift – 7054、TCP、IPv4, IPv6
- SideFX Houdini、Mantra、および Karma – 1715 - 1717、TCP、IPv4, IPv6
- VRay – 30304、TCP、IPV4

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、以下を完了します。
   - [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

> **Note**
>
> Autodesk Maya と Arnold は一緒に、または個別に使用できます。Autodesk Maya にはポート 2702、Arnold にはポート 2701 を使用します。

Autodesk Maya の場合、環境変数`ADSKFLEX_LICENSE_FILE`を次のように設定します。

```
2702@VPC_Endpoint_DNS_Name
```

Arnold の場合、環境変数`ADSKFLEX_LICENSE_FILE`を次のように設定します。

```
2701@VPC_Endpoint_DNS_Name
```

Autodesk Maya と Arnold の場合、 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– 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pip install deadline-cloud-for-keyshot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– Red Giant

環境変数redshift_LICENSEを次のように設定します。

```
7055@VPC_Endpoint_DNS_Name
```

Red Giant と Redshift のredshift_LICENSE環境変数は同じであることに注意してください。両方のアプリケーションを使用する場合は、環境変数を次のように設定できます。

```
7054@VPC_Endpoint_DNS_Name:7055@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。

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、After Effects と Red Giant がインストールされていることを確認します。次に、次のようなコマンドを使用してプロジェクトをレンダリングできます。

```
C:\Program Files\Adobe\Adobe After Effects 2025\Support Files\aerender.exe -comp "Comp
 1" -project
          C:\Users\MyUser\myAfterEffectsProjectUsingRedGiant.aep -output
            C:\Users\MyUser\myMovieWithRedGiant.mp4
```

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ます。

```
/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動作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()"
```

Example– VRay

環境変数VRAY_AUTH_CLIENT_SETTINGSを次のように設定します。

```
licset://VPC_Endpoint_DNS_Name:30304
```

環境変数VRAY_AUTH_CLIENT_FILE_PATHを次のように設定します。

```
/null
```

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、次のようなコマンドを使用して VRay でイメージをレンダリングします。

```
/usr/Chaos/V-Ray/bin/vray -sceneFile=/root/my_scene.vrscene -display=0
```

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。

Deadline Cloud での AI エージェントの使用

AI エージェントを使用して、Deadline Cloud でジョブバンドルの記述、conda パッケージの開発、ジョブのトラブルシューティングを行います。このトピックでは、AI エージェントとは何か、AI エージェントを効果的に使用するための重要なポイント、エージェントが Deadline Cloud を理解するのに役立つリソースについて説明します。

AI エージェントは、大規模言語モデル (LLM) を使用してタスクを自律的に実行するソフトウェアツールです。AI エージェントは、フィードバックに基づいてファイルの読み取りと書き込み、コマンドの実行、ソリューションの反復を行うことができます。例としては、Kiro や IDE 統合アシスタントなどのコマンドラインツールなどがあります。

AI エージェントを使用するための重要なポイント

次のキーポイントは、Deadline Cloud で AI エージェントを使用すると、より良い結果を得るのに役立ちます。

- グラウンディングを提供する – AI エージェントは、関連するドキュメント、仕様、例にアクセスできる場合に最高のパフォーマンスを発揮します。エージェントを特定のドキュメントページに向け、既存のサンプルコードをリファレンスとして共有し、関連するオープンソースリポジトリをローカルワークスペースにクローンし、サードパーティーアプリケーションのドキュメントを提供することで、グラウンディングを提供できます。
- 成功基準を指定する – エージェントの期待される成果と技術要件を定義します。たとえば、ジョブバンドルの開発をエージェントに依頼する場合は、ジョブの入力、パラメータ、および期待される出力を指定します。仕様が不明な場合は、まずエージェントにオプションを提案してもらい、次に要件を一緒に絞り込みます。
- フィードバックループを有効にする – AI エージェントは、ソリューションをテストしてフィードバックを受け取ることができるときに、より効果的に反復処理を行います。最初の試行で有効なソリューションを期待する代わりに、エージェントにソリューションを実行して結果を確認する機能を与えます。このアプローチは、エージェントがステータスの更新、ログ、検証エラーにアクセスできる場合に適しています。たとえば、ジョブバンドルを開発するときは、エージェントがジョブを送信してログを確認できるようにします。
- 反復を想定する – 適切なコンテキストであっても、エージェントは軌道から外れたり、環境と一致しない仮定を立てることができます。エージェントがタスクにどのようにアプローチするかを観察し、その過程でガイダンスを提供します。エージェントが問題が発生した場合は、欠落しているコンテキストを追加し、特定のログファイルをポイントしてエラーを検出し、検出時に要件を絞り込み、エージェントが避けるべきことを明示的に記述するための負の要件を追加します。

エージェントコンテキストのリソース

以下のリソースは、AI エージェントが Deadline Cloud の概念を理解し、正確な出力を生成するのに役立ちます。

- Deadline Cloud Model Context Protocol (MCP) サーバー – Model Context Protocol をサポートするエージェントの場合、deadline-cloud リポジトリには、ジョブを操作するための MCP サーバーを含む Deadline Cloud クライアントが含まれています。
- AWSドキュメント MCP サーバー – MCP をサポートするエージェントの場合、Deadline Cloud ユーザーガイドやデベロッパーガイドなど、ドキュメントに直接アクセスできるようにAWSドキュメント AWS MCP サーバーを設定します。
- Open Job Description 仕様 – GitHub の Open Job Description 仕様は、ジョブテンプレートのスキーマを定義します。エージェントがジョブテンプレートの構造と構文を理解する必要がある場合は、このリポジトリを参照してください。
- deadline-cloud-samples – deadline-cloud-samples リポジトリには、一般的なアプリケーションとユースケースのサンプルジョブバンドル、conda レシピ、CloudFormationテンプレートが含まれています。
- aws-deadline GitHub 組織 – aws-deadline GitHub 組織には、他の統合の例として使用できる多くのサードパーティーアプリケーション用のリファレンスプラグインが含まれています。

プロンプトの例: ジョブバンドルの記述

次のプロンプト例は、AI エージェントを使用して、AI イメージを生成するための LoRA (Low-Rank Adaptation) アダプターをトレーニングするジョブバンドルを作成する方法を示しています。このプロンプトは、前述の重要なポイントを示しています。関連するリポジトリを指し示し、ジョブバンドル出力の成功基準を定義し、反復開発のフィードバックループの概要を示します。

```
Write a pair of job bundles for Deadline Cloud that use the diffusers Python library to
 train a LoRA adapter on a set of images and then generate images from it.

Requirements:
- The training job takes a set of JPEG images as input, uses an image description,
 LoRA rank, learning rate, batch size, and number of training steps as parameters, and
 outputs a `.safetensors` file.
- The generation job takes the `.safetensors` file as input and the number of images
 to generate, then outputs JPEG images. The jobs use Stable Diffusion 1.5 as the base
 model.
- The jobs run `diffusers` as a Python script. Install the necessary packages using
 conda by setting the job parameters:
```

```
    - `CondaChannels`: `conda-forge`
    - `CondaPackages`: list of conda packages to install

For context, clone the following repositories to your workspace and review their
 documentation and code:
- OpenJobDescription specification: https://github.com/OpenJobDescription/openjd-
specifications/blob/mainline/wiki/2023-09-Template-Schemas.md
- Deadline Cloud sample job bundles: https://github.com/aws-deadline/deadline-cloud-
samples/tree/mainline/job_bundles
- diffusers library: https://github.com/huggingface/diffusers

Read through the provided context before you start. To develop a job bundle, iterate
 with the following steps until the submitted job succeeds. If a step fails, update the
 job bundle and restart the loop:
1. Create a job bundle.
2. Validate the job template syntax: `openjd check`
3. Submit the job to Deadline Cloud: `deadline bundle submit`
4. Wait for the job to complete: `deadline job wait`
5. View the job status and logs: `deadline job logs`
6. Download the job output: `deadline job download-output`

To verify the training and generation jobs work together, iterate with the following
 steps until the generation job produces images that resemble the dog in the training
 data:
1. Develop and submit a training job using the training images in `./exdog`
2. Wait for the job to succeed then download its output.
3. Develop and submit a generation job using the LoRA adapter from the training job.
4. Wait for the job to succeed then download its output.
5. Inspect the generated images. If they resemble the dog in the training data, you're
 done. Otherwise, review the job template, job parameters, and job logs to identify and
 fix the issue.
```

AWSDeadline Cloud のモニタリング

モニタリングは、AWSDeadline 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。呼び出し元の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。

CloudTrail は、アカウントを作成する AWS アカウント と 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 マネジメントコンソール 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口グ記録しません。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 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 マネジメントコンソール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 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
- delete-limit
- 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
- モニターの取得
- 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
- list-fleets
- 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
- tag-resource
- untag-resource
- 予算の更新
- update-farm
- 更新フリート
- 更新制限
- モニターの更新
- update-queue
- 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処理します。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 の実行として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て CloudWatch `GetLogEvents` の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 マネジメントコンソール、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 マネジメントコンソールまたは 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	Unit
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。

ライセンスメトリクス

AWS/DeadlineCloud 名前空間には、ライセンス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	Unit
LicensesInUse	使用中のライセンスセッションの数。	カウント

次のディメンションを使用して、ライセンス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
FleetId	このディメンションを使用して、指定されたサービスマネージドフリートにデータをフィルタリングします。カスタマーマネージドフリートの場合は、代わりに LicenseEndpointId ディメンションを使用します。
LicenseEndpointId	このディメンションを使用して、指定されたライセンスエンドポイントにデータをフィルタリングします。
製品	このディメンションを使用して、指定された従量制製品に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	Unit
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	カウント

メトリクス	説明	Unit
	Cloud はMaxCountメトリクスを出力しません。	

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、リクエストしたデータを指定された制限までフィルタリングします。

推奨アラーム

CloudWatch を使用すると、メトリクスを監視し、しきい値を超えたときに通知を送信するアラームを作成したり、別のアクションを実行したりできます。CloudWatch アラームの設定の詳細については、Amazon CloudWatch ユーザーガイド」を参照してください。

次の Deadline Cloud メトリクスのアラームを設定することをお勧めします。

LicensesInUse

ディメンション: FleetId、LicenseEndpointId

アラームの説明: このアラームは、サービスマネージドフリートまたはライセンスエンドポイントのアクティブなライセンスセッションがアカウントクォータに近づいていることを検出します。この場合、ライセンスセッションのアカウントクォータを引き上げることができます。現在のクォータを確認し、Service Quotas を使用して引き上げをリクエストします。詳細については、Service Quotas ユーザーガイド」を参照してください。

目的: クォータ制限に達する前に使用状況をモニタリングすることで、ライセンスチェックアウトの失敗を防止します。

統計: Maximum

推奨されるしきい値: ライセンスセッションクォータの 90%

しきい値の根拠: しきい値をクォータの割合に設定し、制限に達する前にアクションを実行できるようにします。

期間: 1 分

アラームを発生させるデータポイント数: 1

評価期間数: 1

比較演算子: GREATER_THAN_THRESHOLD

その他のリソース

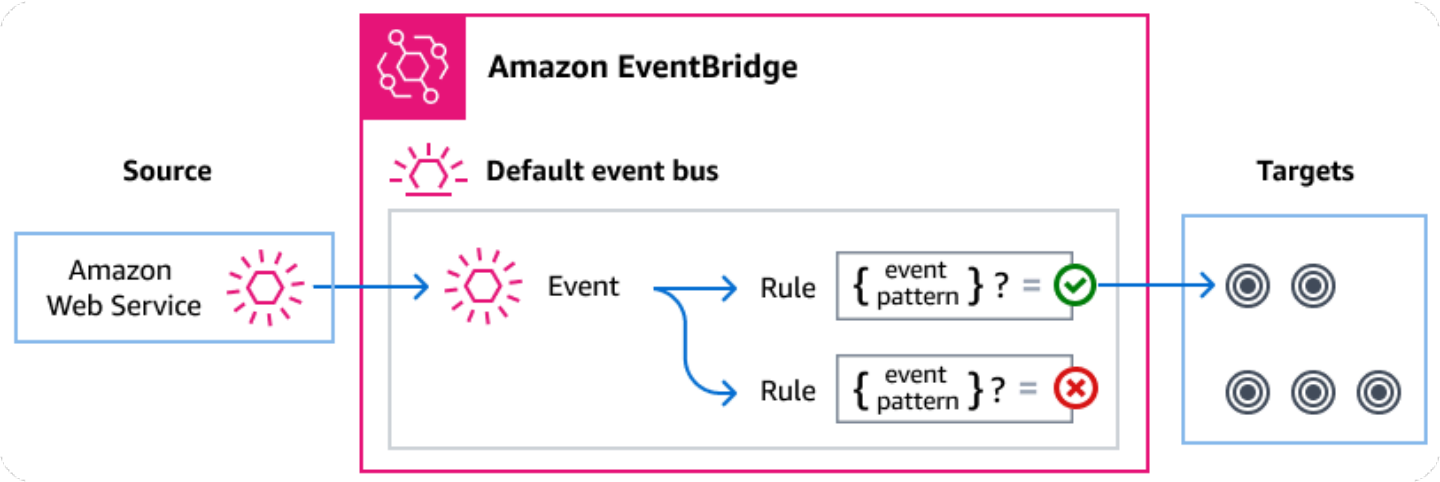
- Amazon CloudWatch ユーザーガイド
- Service Quotas User Guide

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、イベントを受信して、ゼロ個以上の送信先 (ターゲット) に配信するルーターです。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に変更があった場合に送信されます。
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。
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。

イベントの詳細のタイプ	説明
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「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。この頻度により、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に合格したときに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、 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": "Budget Threshold Reached",
    "source": "aws.deadline",
    "account": "111122223333",
    "time": "2017-12-22T18:43:48Z",
    "region": "aa-example-1",
    "resources": [],
    "detail": {
        "farmId": "farm-12345678900000000000000000000000",
        "budgetId": "budget-12345678900000000000000000000000",
        "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 `aws.deadline` です。

`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、 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": "Fleet Size Recommendation Change",
    "source": "aws.deadline",
    "account": "111122223333",
    "time": "2017-12-22T18:43:48Z",
    "region": "aa-example-1",
    "resources": [],
    "detail": {
        "farmId": "farm-12345678900000000000000000000000",
        "fleetId": "fleet-12345678900000000000000000000000",
        "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、ジョブの作成時を含め、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、 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": "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`

ジョブが退出す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、 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": "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記述します。

ステップライフサイクルステータスの変更イベントは、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_SUCCEEDED)。
- 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は、ステップとジョブの全体的なステータスを示すために結合されます。

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 `aws.deadline` です。

`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 `aws.deadline` です。

`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。

を使用したセッション統計の集計データのクエリ AWS CLI

コストを追跡したり、リソースの使用状況を分析したり、どのユーザーが最も多くリソースを消費しているかを特定したりするには、 AWS Command Line Interface (AWS CLI) を使用して AWS Deadline Cloud (Deadline Cloud) ファームの集約セッション統計をクエリできます。セッション統計 API は、キュー、フリート、インスタンスタイプ、ユーザーなどのさまざまなディメンション別にグループ化できるコスト、ランタイム、使用状況に関するデータを提供します。

セッション統計のクエリは非同期プロセスです。まず集約リクエストを開始し、次に集約 ID を使用して結果を取得します。

集約リクエストの開始

集約リクエストを開始するには、 `start-sessions-statistics-aggregation` コマンドを実行します。次の の例では、特定のキューのユーザー ID で統計をグループ化します。########テキストを自分の情報に置き換えます。

```
aws deadline start-sessions-statistics-aggregation \
    --farm-id farm-id \
    --resource-ids '{"queueIds":["queue-id"]}' \
    --start-time 2025-11-24T10:00:00Z \
    --end-time 2025-11-25T18:00:00Z \
    --group-by '["USER_ID"]' \
    --period HOURLY \
    --statistics '["SUM"]' \
    --timezone UTC-08:00 \
    --region region-name
```

統計は、、QUEUE_ID、FLEET_ID、JOB_IDINSTANCE_TYPE、 などの他のディメンションでグループ化できますLICENSE_PRODUCT。使用可能なすべてのパラメータの詳細については、 AWS CLI 「 コマンドリファレンス」のstart-sessions-statistics-aggregation」を参照してください。

レスポンスには集約 ID が含まれます。

```
{
    "aggregationId": "92b35143f2d04641979bc9b777232f38"
}
```

結果を取得する

集計 ID を使用して get-sessions-statistics-aggregation コマンドを実行して、結果を取得します。########テキストを自分の情報に置き換えます。

```
aws deadline get-sessions-statistics-aggregation \
    --farm-id farm-id \
    --aggregation-id aggregation-id \
    --region region-name
```

次の例は、統計をユーザー ID でグループ化する場合のレスポンスを示しています。userId フィールドには、ユーザーを識別するためにユーザー名にマッピングする必要がある UUID が含まれています。

```
{
    "statistics": [
        {
            "userId": "f9c1f3f0-1031-70dc-4d25-30d7225b04a0",
            "count": 1,
            "costInUsd": {
                "sum": 0.0
            },
            "runtimeInSeconds": {
                "sum": 53.773
            },
            "aggregationStartTime": "2025-11-24T22:00:00Z",
            "aggregationEndTime": "2025-11-24T23:00:00Z"
        }
    ],
    "status": "COMPLETED"
}
```

に関連付けられているユーザー名を確認するにはuserId、「」を参照してくださいthe section called “userID を使用したユーザーメタデータの取得”。

API の詳細については、「Deadline Cloud API リファレンス」を参照してください。

トピック

- the section called “userID を使用したユーザーメタデータの取得”

ID ストアで userID を使用してユーザーメタデータと属性を取得する

> **Note**
>
> この手順は、同じユーザー ID 形式を使用する SearchJobs API によって返される `createdBy`フィールドにも適用されます。

セッション統計の `userId`フィールドには、次のいずれかの値が含まれます。

- (AWS Identity and Access Management IAM) ロール ARN。例: `arn:aws:sts::123456789012:assumed-role/Admin/user-Isengard`。
- IAM Identity Center ユーザー ID (UUID)。例: `f9c1f3f0-1031-70dc-4d25-30d7225b04a0`。

IAM ロール ARNs、ユーザー名は ARN 自体に表示されます。IAM Identity Center ユーザー IDs の場合は、IAM Identity Center Identity Store API を使用してユーザー名を検索できます。

IAM Identity Center ユーザー ID に関連付けられたユーザー名を特定するには、次の手順を使用します。開始する前に、IAM Identity Center 設定から Identity Store ID を取得します。詳細については、「the section called “ID ストア ID の検索”」を参照してください。

ユーザー ID をマッピングするには

1. 次のコマンドを実行し、*`IdentityStoreId`* を Identity Store ID に、*`userUUID`* をセッション統計レスポンス`userId`の に置き換えます。

```
aws identitystore describe-user \
    --identity-store-id IdentityStoreId \
    --user-id userUUID
```

2. ユーザー名を含むレスポンスを確認します。

```
{
    "UserName": "jdoe",
    "UserId": "f9c1f3f0-1031-70dc-4d25-30d7225b04a0",
    "Name": {
        "FamilyName": "Doe",
```

```
        "GivenName": "Jane"
    },
    "DisplayName": "Jane Doe",
    "Emails": [{
        "Value": "jdoe@example.com",
        "Type": "work",
        "Primary": true
    }],
    "IdentityStoreId": "d-xxxxxxxxxx"
}
```

ID ストア ID の検索

ユーザー IDsユーザー名にマッピングするには、Identity Store ID が必要です。Identity Store ID は、IAM Identity Center コンソールまたは を使用して確認できます AWS CLI。

コンソール

コンソールを使用して Identity Store ID を検索するには、次の手順を使用します。

1. AWS マネジメントコンソールにサインインし、IAM Identity Center コンソールを開きます。
2. ナビゲーションペインで [設定] を選択します。
3. IAM Identity Center Identity Store ID 値をコピーします。形式は d-xxxxxxxxxx です。

AWS CLI

region-*name* を IAM Identity Center インスタンスが設定されているリージョンに置き換えて、次のコマンドを実行します。

```
aws sso-admin list-instances --region region-name
```

レスポンスには が含まれますIdentityStoreId。

```
{
    "Instances": [
        {
            "CreatedDate": "2025-11-19T15:45:55.160000-08:00",
            "IdentityStoreId": "d-xxxxxxxxxx",
            "InstanceArn": "arn:aws:sso:::instance/ssoins-xxxxxxxxxxxxxxxx",
```

```
            "OwnerAccountId": "123456789012",
            "Status": "ACTIVE"
        }
    ]
}
```

ユーザーマッピングの検証

ユーザー ID をユーザー名にマッピングしたら、IAM Identity Center コンソールでユーザー ID が想定ユーザーと一致することを確認できます。ユーザーマッピングを確認するには、次の手順を使用します。

1. AWS マネジメントコンソールにサインインし、IAM Identity Center コンソールを開きます。
2. ナビゲーションペインで [ユーザー] を選択します。
3. AWS CLI レスポンスからユーザー名を選択します。
4. 全般情報セクションで、ユーザー ID がセッション統計`userId`の と一致することを確認します。

その他のリソース

- IAM Identity Center ユーザーガイド
- IAM Identity Center Identity Store API リファレンス

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を活用できます。

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 を設定する方法を示します。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

- 制限されたネットワーク環境
- 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 アイデンティティセンターまたは 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 は必須ですが、TLS 1.3 を推奨します。
- 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、API、または SDK を使用して AWS CLI Deadline Cloud 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中のデータを暗号化することで、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。

Deadline Cloud は、サービスマネージドフリートワーカーインスタンスが終了すると、Amazon Elastic Block Store ボリュームを削除します。

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 は必須ですが、TLS 1.3 を推奨します。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
  - タグを追加する
  - 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。

Deadline Cloud が AWS KMS 許可を使用する方法

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 キーにアクセス AWS KMS するためのCreateGrantリクエストを 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。これは一般的ではありませんが、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。

JSON

```
{
    "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。

JSON

```
{
    "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。

JSON

```
{
    "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 に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-abcdef12345678900987654321fedcba",
                "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。キーを削除すると、データは回復できなくなります。

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 サービスに接続します。各サービスマネージドフリートは独自の独立したネットワークで実行され、ワーカーインスタンスは個々のお客様専用のままです。

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、 に問題がある場合に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

- オーディエンス
- 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
- サービスロール
- 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

オーディエンス

AWS Identity and Access Management (IAM) の使用方法は、ロールによって異なります。

- サービスユーザー - 機能にアクセスできない場合は、管理者にアクセス許可をリクエストします (「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」を参照)。
- サービス管理者 - ユーザーアクセスを決定し、アクセス許可リクエストを送信します (「Deadline Cloud と IAM の連携方法」を参照)
- IAM 管理者 - アクセスを管理するためのポリシーを作成します (「Deadline Cloud のアイデンティティベースのポリシーの例」を参照)

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。、IAM ユーザー 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で認証される必要があります。

(AWS IAM アイデンティティセンター IAM Identity Center)、シングルサインオン認証、Google/Facebook 認証情報などの ID ソースからの認証情報を使用して、フェデレーティッド ID としてサインインできます。サインインの詳細については、「AWS サインイン ユーザーガイド」の「AWS アカウントにサインインする方法」を参照してください。

プログラムによるアクセスの場合、 は SDK と CLI AWS を提供してリクエストを暗号化して署名します。詳細については、「IAM ユーザーガイド」の「API リクエストに対するAWS 署名バージョン 4」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、まず、すべての AWS のサービス および リソースへの完全なアクセス権を持つ AWS アカウント root ユーザーと呼ばれる 1 つのサインイン ID から始めます。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。ルートユーザー認証情報を必要とするタスクについては、「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。

フェデレーテッドアイデンティティ

ベストプラクティスとして、人間のユーザーが一時的な認証情報 AWS のサービス を使用して にアクセスするには、ID プロバイダーとのフェデレーションを使用する必要があります。

フェデレーティッド ID は、エンタープライズディレクトリ、ウェブ ID プロバイダー、または ID Directory Service ソースの認証情報 AWS のサービス を使用して にアクセスするユーザーです。フェデレーテッドアイデンティティは、一時的な認証情報を提供するロールを引き受けます。

アクセスを一元管理する場合は、 AWS IAM アイデンティティセンターをお勧めします。詳細については、「AWS IAM アイデンティティセンター ユーザーガイド」の「IAM アイデンティティセンターとは」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、特定の個人やアプリケーションに対する特定のアクセス許可を持つアイデンティティです。長期認証情報を持つ IAM ユーザーの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。詳細については、IAM ユーザーガイドの「ID プロバイダーとのフェデレーションを使用して にアクセスする必要がある AWS」を参照してください。

IAM グループは、IAM ユーザーの集合を指定し、大量のユーザーに対するアクセス許可の管理を容易にします。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を持つアイデンティであり、一時的な認証情報を提供します。ユーザーから IAM ロール (コンソール) に切り替えるか、 または API オペレーションを呼び出すこ

とで、ロールを引き受けることができます。 AWS CLI AWS 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 ロールは、フェデレーションユーザーアクセス、一時的な IAM ユーザーのアクセス許可、クロスアカウントアクセス、クロスサービスアクセス、および Amazon EC2 で実行するアプリケーションに役立ちます。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。

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、ID またはリソースに関連付けられたときにアクセス許可を定義します。 は、プリンシパル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。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、ポリシーを使用して、どのプリンシパルがどのリソースに対して、どのような条件でアクションを実行できるかを定義することで、誰が何にアクセスできるかを指定します。

デフォルトでは、ユーザーやロールにアクセス許可はありません。IAM 管理者は IAM ポリシーを作成してロールに追加し、このロールをユーザーが引き受けられるようにします。IAM ポリシーは、オペレーションの実行方法を問わず、アクセス許可を定義します。

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のポリシーは、アイデンティティ (ユーザー、グループ、またはロール) にアタッチできる JSON アクセス許可ポリシードキュメントです。これらのポリシーは、アイデンティティがどのリソースに対してどのような条件下でどのようなアクションを実行できるかを制御します。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、インラインポリシー (単一の ID に直接埋め込む) または管理ポリシー (複数の ID にアタッチされたスタンドアロンポリシー) にすることができます。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。例としては、IAM ロール信頼ポリシーや Amazon S3 バケットポリシーなどがあります。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。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。

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。

その他のポリシータイプ

AWS は、より一般的なポリシータイプによって付与されるアクセス許可の最大数を設定できる追加のポリシータイプをサポートしています。

- アクセス許可の境界 – アイデンティティベースのポリシーで IAM エンティティに付与することのできるアクセス許可の数の上限を設定します。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 (SCP) - AWS Organizations内の組織または組織単位の最大のアクセス許可を指定します。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」を参照してください。
- リソースコントロールポリシー (RCP) – は、アカウント内のリソースで利用できる最大数のアクセス許可を定義します。詳細については、「AWS Organizations ユーザーガイド」の「リソースコントロールポリシー (RCP)」を参照してください。
- セッションポリシー – ロールまたはフェデレーションユーザーの一時セッションを作成する際にパラメータとして渡される高度なポリシーです。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アクセス許可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のポリシー	あり
リソースベースのポリシー	なし
ポリシーアクション	あり
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	なし
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	あり
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、ユーザーグループ、ロールなど、アイデンティティにアタッチできる JSON 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。

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。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コントロールできます。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、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、全体のアカウント、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。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。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。

```
deadline
```

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": [
      "deadline:action1",
      "deadline: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。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。リソースレベルのアクセス許可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 要素は、定義された基準に基づいてステートメントが実行される時期を指定します。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。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「Deadline Cloud AWS 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アクセス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、タグと呼ばれる属性に基づいてアクセス許可を定義する認可戦略です。IAM エンティティと AWS リソースにタグをアタッチし、プリンシパルのタグがリソースのタグと一致するときにオペレーションを許可するように ABAC ポリシーを設計できます。

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

一時的な認証情報は AWS 、リソースへの短期的なアクセスを提供し、フェデレーションまたはスイッチロールの使用時に自動的に作成されます。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 ユーザーガイド」の「IAM の一時的な認証情報」および「AWS のサービス と IAM との連携」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

転送アクセスセッション (FAS) は、 を呼び出すプリンシパルのアクセス許可と 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 を使用します。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がありません。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。

これらのサンプルの 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
- コンソールにアクセスするためのポリシー
- 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、アカウント内で 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では、 AWS アカウントに費用が発生する場合があります。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 ポリシーでアクセス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。たとえば、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 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 アクセスアナライザー 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 アクセスアナライザー 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、MFA をオンにしてセキュリティを強化します。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。

コンソールにアクセスするためのポリシー

Deadline Cloud コンソールのすべての機能へのアクセスを許可するには、フルアクセスを付与するユーザーまたはロールにこの ID ポリシーをアタッチします。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Sid": "EC2InstanceTypeSelection",
        "Effect": "Allow",
        "Action": [
            "ec2:DescribeInstanceTypeOfferings",
            "ec2:DescribeInstanceTypes",
            "ec2:GetInstanceTypesFromInstanceRequirements",
            "pricing:GetProducts"
        ],
        "Resource": ["*"]
    },
    {
        "Sid": "VPCResourceSelection",
        "Effect": "Allow",
        "Action": [
            "ec2:DescribeVpcs",
            "ec2:DescribeSubnets",
            "ec2:DescribeSecurityGroups"
        ],
        "Resource": ["*"]
    },
```

```
    {
        "Sid": "ViewVpcLatticeResources",
        "Effect": "Allow",
        "Action": [
            "vpc-lattice:ListResourceConfigurations",
            "vpc-lattice:GetResourceConfiguration",
            "vpc-lattice:GetResourceGateway"
        ],
        "Resource": ["*"]
    },
    {
        "Sid": "ManageVpcEndpointsViaDeadline",
        "Effect": "Allow",
        "Action": [
            "ec2:CreateVpcEndpoint",
            "ec2:DescribeVpcEndpoints",
            "ec2:DeleteVpcEndpoints",
        "ec2:CreateTags"
        ],
        "Resource": ["*"],
        "Condition": {
        "StringEquals": { "aws:CalledViaFirst": "deadline.amazonaws.com" }
        }
    },
    {
        "Sid": "ChooseJobAttachmentsBucket",
        "Effect": "Allow",
        "Action": ["s3:GetBucketLocation", "s3:ListAllMyBuckets"],
        "Resource": "*"
    },
    {
        "Sid": "CreateDeadlineCloudLogGroups",
        "Effect": "Allow",
        "Action": ["logs:CreateLogGroup"],
        "Resource": "arn:aws:logs:*:*:log-group:/aws/deadline/*",
        "Condition": {
        "StringLike": { "aws:CalledViaFirst": "deadline.amazonaws.com" }
        }
    },
    {
        "Sid": "ValidateDependencies",
        "Effect": "Allow",
        "Action": ["s3:ListBucket"],
        "Resource": "*",
```

```
        "Condition": {
        "StringLike": { "aws:CalledViaFirst": "deadline.amazonaws.com" }
        }
    },
    {
        "Sid": "RoleSelection",
        "Effect": "Allow",
        "Action": ["iam:GetRole", "iam:ListRoles"],
        "Resource": "*"
    },
    {
        "Sid": "PassRoleToDeadlineCloud",
        "Effect": "Allow",
        "Action": ["iam:PassRole"],
        "Condition": {
        "StringLike": { "iam:PassedToService": "deadline.amazonaws.com" }
    },
        "Resource": "*"
    },
    {
        "Sid": "KMSKeySelection",
        "Effect": "Allow",
        "Action": ["kms:ListKeys", "kms:ListAliases"],
        "Resource": "*"
    },
    {
        "Sid": "IdentityStoreReadOnly",
        "Effect": "Allow",
        "Action": [
            "identitystore:DescribeUser",
            "identitystore:DescribeGroup",
            "identitystore:ListGroups",
            "identitystore:ListUsers",
            "identitystore:IsMemberInGroups",
            "identitystore:ListGroupMemberships",
            "identitystore:ListGroupMembershipsForMember",
            "identitystore:GetGroupMembershipId"
    ],
        "Resource": "*"
    },
    {
        "Sid": "OrganizationAndIdentityCenterIdentification",
        "Effect": "Allow",
        "Action": [
```

```
            "sso:ListDirectoryAssociations",
            "organizations:DescribeAccount",
            "organizations:DescribeOrganization",
            "sso:DescribeRegisteredRegions",
            "sso:GetManagedApplicationInstance",
            "sso:GetSharedSsoConfiguration",
            "sso:ListInstances",
            "sso:GetApplicationAssignmentConfiguration",
            "sso:GetSSOStatus",
            "sso:ListRegions",
            "sso:DescribeRegion"
    ],
        "Resource": "*"
    },
    {
        "Sid": "ManagedDeadlineCloudIDCApplication",
        "Effect": "Allow",
        "Action": [
            "sso:CreateApplication",
            "sso:PutApplicationAssignmentConfiguration",
            "sso:PutApplicationAuthenticationMethod",
            "sso:PutApplicationGrant",
            "sso:DeleteApplication",
            "sso:UpdateApplication"
    ],
        "Resource": "*",
        "Condition": {
        "StringLike": { "aws:CalledViaFirst": "deadline.amazonaws.com" }
        }
    },
    {
        "Sid": "ChooseSecret",
        "Effect": "Allow",
        "Action": ["secretsmanager:ListSecrets"],
        "Resource": "*"
    },
    {
        "Sid": "DeadlineMembershipActions",
        "Effect": "Allow",
        "Action": [
            "deadline:AssociateMemberToFarm",
            "deadline:AssociateMemberToFleet",
            "deadline:AssociateMemberToQueue",
            "deadline:AssociateMemberToJob",
```

```
            "deadline:DisassociateMemberFromFarm",
            "deadline:DisassociateMemberFromFleet",
            "deadline:DisassociateMemberFromQueue",
            "deadline:DisassociateMemberFromJob",
            "deadline:ListFarmMembers",
            "deadline:ListFleetMembers",
            "deadline:ListQueueMembers",
            "deadline:ListJobMembers"
    ],
        "Resource": ["*"]
    },
    {
        "Sid": "DeadlineControlPlaneActions",
        "Effect": "Allow",
        "Action": [
            "deadline:CreateMonitor",
            "deadline:GetMonitor",
            "deadline:UpdateMonitor",
            "deadline:DeleteMonitor",
            "deadline:ListMonitors",
            "deadline:CreateFarm",
            "deadline:GetFarm",
            "deadline:UpdateFarm",
            "deadline:DeleteFarm",
            "deadline:ListFarms",
            "deadline:CreateQueue",
            "deadline:GetQueue",
            "deadline:UpdateQueue",
            "deadline:DeleteQueue",
            "deadline:ListQueues",
            "deadline:CreateFleet",
            "deadline:GetFleet",
            "deadline:UpdateFleet",
            "deadline:DeleteFleet",
            "deadline:ListFleets",
            "deadline:ListWorkers",
            "deadline:CreateQueueFleetAssociation",
            "deadline:GetQueueFleetAssociation",
            "deadline:UpdateQueueFleetAssociation",
            "deadline:DeleteQueueFleetAssociation",
            "deadline:ListQueueFleetAssociations",
            "deadline:CreateQueueEnvironment",
            "deadline:GetQueueEnvironment",
            "deadline:UpdateQueueEnvironment",
```

```
            "deadline:DeleteQueueEnvironment",
            "deadline:ListQueueEnvironments",
            "deadline:CreateLimit",
            "deadline:GetLimit",
            "deadline:UpdateLimit",
            "deadline:DeleteLimit",
            "deadline:ListLimits",
            "deadline:CreateQueueLimitAssociation",
            "deadline:GetQueueLimitAssociation",
            "deadline:DeleteQueueLimitAssociation",
            "deadline:UpdateQueueLimitAssociation",
            "deadline:ListQueueLimitAssociations",
            "deadline:CreateStorageProfile",
            "deadline:GetStorageProfile",
            "deadline:UpdateStorageProfile",
            "deadline:DeleteStorageProfile",
            "deadline:ListStorageProfiles",
            "deadline:ListStorageProfilesForQueue",
            "deadline:ListBudgets",
            "deadline:TagResource",
            "deadline:UntagResource",
            "deadline:ListTagsForResource",
            "deadline:CreateLicenseEndpoint",
            "deadline:GetLicenseEndpoint",
            "deadline:DeleteLicenseEndpoint",
            "deadline:ListLicenseEndpoints",
            "deadline:ListAvailableMeteredProducts",
            "deadline:ListMeteredProducts",
            "deadline:PutMeteredProduct",
            "deadline:DeleteMeteredProduct"
        ],
        "Resource": ["*"]
      }]
}
```

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。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "SubmitJobsFarmAndQueue",
            "Effect": "Allow",
            "Action": "deadline:CreateJob",
            "Resource": "arn:aws:deadline:us-east-1:111122223333: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。

JSON

```
{
    "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": [
            "arn:aws:deadline:*:111122223333:*",
            "arn:aws:ec2:*:111122223333:vpc-endpoint/*"
        ]
    }]
}
```

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。

JSON

```
{
    "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:us-east-1:123456789012:farm/FARM_A/queue/QUEUE_B",
            "arn:aws:deadline:us-east-1:123456789012: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、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るため、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 マネージド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 マネージド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、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。
- `kms` – ユーザーが AWS Key Management Service (IAM Identity Center AWS KMS) インスタンスの AWS IAM アイデンティティセンター () カスタマーマネージドキーを設定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、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-UserAccessFarms – 変更	Deadline Cloud に新しいアクションが追加され`kms:Decrypt` 、IAM Identity Center インスタンスで AWS KMS カスタマーマネージドキーを使用できるようになりました。	2025 年 12 月 22 日
AWSDeadlineCloud-WorkerHost – 変更	Deadline Cloud に新しいアクション `deadline:`	2025 年 5 月 30 日

変更	説明	日付
	TagResource と が追加されdeadline:ListTagsForResource 、フリート内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。	
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 が 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

サービスロール

Deadline Cloud が IAM サービスロールを使用する方法

Deadline Cloud は自動的に IAM ロールを引き受け、ワーカー、ジョブ、および Deadline Cloud モニターに一時的な認証情報を提供します。このアプローチにより、ロールベースのアクセスコントロールを通じてセキュリティを維持しながら、手動の認証情報管理が不要になります。

モニター、フリート、キューを作成するときは、Deadline Cloud がユーザーに代わって引き受ける IAM ロールを指定します。その後、ワーカーと Deadline Cloud モニターは、アクセスするためにこれらのロールから一時的な認証情報を受け取ります AWS のサービス。

フリートロール

Deadline Cloud ワーカーが作業を受け取り、その作業の進行状況をレポートするために必要なアクセス許可を付与するようにフリートロールを設定します。

通常、このロールを自分で設定する必要はありません。このロールは、Deadline Cloud コンソールで作成して、必要なアクセス許可を含めることができます。トラブルシューティングのためのこのロールの詳細を理解するには、次のガイドを使用します。

プログラムでフリートを作成または更新する場合は、 `CreateFleet` または `UpdateFleet` API オペレーションを使用してフリートロール ARN を指定します。

フリートロールの動作

フリートロールは、ワーカーに以下のアクセス許可を付与します。

- 新しい作業を受け取り、進行中の作業の進捗状況を Deadline Cloud サービスに報告する
- ワーカーのライフサイクルとステータスを管理する
- ワーカーログのログイベントを Amazon CloudWatch Logs に記録する

フリートロールの信頼ポリシーを設定する

フリートロールは Deadline Cloud サービスを信頼し、特定のファームに限定する必要があります。

ベストプラクティスとして、信頼ポリシーには混乱した代理保護のセキュリティ条件を含める必要があります。混乱した代理の保護の詳細については、「Deadline Cloud ユーザーガイド」の「混乱した代理」を参照してください。

- `aws:SourceAccount` は、同じ のリソースのみがこのロールを引き受け AWS アカウント ることができるようにします。
- `aws:SourceArn` は、ロールの引き受けを特定の Deadline Cloud ファームに制限します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDeadlineCredentialsService",
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Principal": {
        "Service": "credentials.deadline.amazonaws.com"
      },
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "YOUR_ACCOUNT_ID"
```

```
        },
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:deadline:REGION:YOUR_ACCOUNT_ID:farm/YOUR_FARM_ID"
        }
      }
    }
  ]
}
```

フリートロールのアクセス許可をアタッチする

フリートロールに次の AWS 管理ポリシーをアタッチします。

AWSDeadlineCloud-FleetWorker

この管理ポリシーは、以下のアクセス許可を提供します。

- deadline:AssumeFleetRoleForWorker - ワーカーが認証情報を更新できるようにします。
- deadline:UpdateWorker - ワーカーがステータスを更新できるようにします (終了時に STOPPED など)。
- deadline:UpdateWorkerSchedule - 作業の取得と進行状況の報告用。
- deadline:BatchGetJobEntity - ジョブ情報を取得する場合。
- deadline:AssumeQueueRoleForWorker - ジョブの実行中にキューロールの認証情報にアクセスする場合。

暗号化されたファームに KMS アクセス許可を追加する

ファームが KMS キーを使用して作成された場合は、これらのアクセス許可をフリートロールに追加して、ワーカーがファーム内の暗号化されたデータにアクセスできるようにします。

KMS アクセス許可は、ファームに KMS キーが関連付けられている場合にのみ必要です。kms:ViaService 条件は 形式を使用する必要がありますdeadline.*{region}*.amazonaws.com。

フリートを作成すると、そのフリートの CloudWatch Logs ロググループが作成されます。ワーカーのアクセス許可は、Deadline Cloud サービスによって使用され、その特定のワーカー専用のログストリームを作成します。ワーカーをセットアップして実行すると、ワーカーはこれらのアクセス許可を使用してログイベントを CloudWatch Logs に直接送信します。

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CreateLogStream",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream"
      ],
      "Resource": "arn:aws:logs:REGION:YOUR_ACCOUNT_ID:log-group:/aws/
deadline/YOUR_FARM_ID/*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": [
            "deadline.REGION.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid": "ManageLogEvents",
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents",
        "logs:GetLogEvents"
      ],
      "Resource": "arn:aws:logs:REGION:YOUR_ACCOUNT_ID:log-group:/aws/
deadline/YOUR_FARM_ID/*"
    },
    {
      "Sid": "ManageKmsKey",
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "kms:DescribeKey",
        "kms:GenerateDataKey"
      ],
      "Resource": "YOUR_FARM_KMS_KEY_ARN",
      "Condition": {
        "StringEquals": {
          "kms:ViaService": "deadline.REGION.amazonaws.com"
        }
      }
    }
  ]
```

```
}
```

フリートロールの変更

フリートロールのアクセス許可はカスタマイズできません。説明されているアクセス許可は常に必須であり、アクセス許可を追加しても効果はありません。

カスタマーマネージドフリートホストロール

Amazon EC2 インスタンスまたはオンプレミスホストでカスタマーマネージドフリートを使用する場合は、WorkerHost ロールを設定します。

WorkerHost ロールの動作

WorkerHost ロールは、カスタマーマネージドフリートホストでワーカーをブートストラップします。ホストが以下を行うために必要な最小限のアクセス許可を提供します。

- Deadline Cloud でワーカーを作成する
- フリートロールを引き受けて運用認証情報を取得する
- フリートタグを使用してワーカーにタグを付ける (タグ伝達が有効になっている場合)

WorkerHost ロールのアクセス許可を設定する

次の AWS 管理ポリシーを WorkerHost ロールにアタッチします。

AWSDeadlineCloud-WorkerHost

この管理ポリシーは、以下のアクセス許可を提供します。

- `deadline:CreateWorker` - ホストが新しいワーカーを登録できるようにします。
- `deadline:AssumeFleetRoleForWorker` - ホストがフリートロールを引き受けることを許可します。
- `deadline:TagResource` - 作成中のワーカーのタグ付けを許可します (有効になっている場合) 。
- `deadline:ListTagsForResource` - 伝播用のフリートタグの読み取りを許可します。

ブートストラッププロセスを理解する

WorkerHost ロールは、ワーカーの初回起動時にのみ使用されます。

1. ワーカーエージェントは、WorkerHost 認証情報を使用してホストで起動します。
2. を呼び出しdeadline:CreateWorkerて Deadline Cloud に登録します。
3. 次に、 を呼び出しdeadline:AssumeFleetRoleForWorkerてフリートロールの認証情報を取得します。
4. この時点から、ワーカーはすべてのオペレーションにフリートロール認証情報のみを使用します。

WorkerHost ロールは、ワーカーの実行開始後は使用されません。このポリシーは、サービスマネージドフリートには必要ありません。サービスマネージドフリートでは、ブートストラップが自動的に実行されます。

キューロール

キューロールは、タスクを処理するときにワーカーによって引き受けられます。このロールは、タスクを完了するために必要なアクセス許可を提供します。

プログラムでキューを作成または更新する場合は、 CreateQueue または UpdateQueue API オペレーションを使用してキューロール ARN を指定します。

キューロールの信頼ポリシーを設定する

キューロールは Deadline Cloud サービスを信頼する必要があります。

ベストプラクティスとして、信頼ポリシーには混乱した代理保護のセキュリティ条件を含める必要があります。混乱した代理の保護の詳細については、「Deadline Cloud ユーザーガイド」の「混乱した代理」を参照してください。

- aws:SourceAccount は、同じ のリソースのみがこのロールを引き受け AWS アカウント ることができるようにします。
- aws:SourceArn は、ロールの引き受けを特定の Deadline Cloud ファームに制限します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "credentials.deadline.amazonaws.com",
```

```
          "deadline.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "YOUR_ACCOUNT_ID"
        },
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:deadline:us-west-2:123456789012:farm/{farm-id}"
        }
      }
    }
  ]
}
```

キューロールのアクセス許可を理解する

キューロールは 1 つの管理ポリシーを使用しません。代わりに、コンソールでキューを設定すると、Deadline Cloud は設定に基づいてキューのカスタムポリシーを作成します。

この自動作成されたポリシーは、以下へのアクセスを提供します。

ジョブアタッチメント

ジョブの入出力ファイル用に指定された Amazon S3 バケットへの読み取りおよび書き込みアクセス:

```
{
  "Effect": "Allow",
  "Action": [
    "s3:GetObject",
    "s3:PutObject",
    "s3:ListBucket",
    "s3:GetBucketLocation"
  ],
  "Resource": [
    "arn:aws:s3:::YOUR_JOB_ATTACHMENTS_BUCKET",
    "arn:aws:s3:::YOUR_JOB_ATTACHMENTS_BUCKET/YOUR_PREFIX/*"
  ],
  "Condition": {
    "StringEquals": {
      "aws:ResourceAccount": "YOUR_ACCOUNT_ID"
    }
  }
```

```
}
```

ジョブのログ

このキュー内のジョブの CloudWatch Logs への読み取りアクセス。各キューには独自のロググループがあり、各セッションには独自のログストリームがあります。

```
{
  "Effect": "Allow",
  "Action": [
    "logs:GetLogEvents"
  ],
  "Resource": "arn:aws:logs:REGION:YOUR_ACCOUNT_ID:log-group:/aws/
deadline/YOUR_FARM_ID/*"
}
```

サードパーティー製ソフトウェア

Deadline Cloud でサポートされているサードパーティーソフトウェア (Maya、Blender など) をダウンロードするアクセス:

```
{
  "Effect": "Allow",
  "Action": [
    "s3:ListBucket",
    "s3:GetObject"
  ],
  "Resource": "*",
  "Condition": {
    "ArnLike": {
      "s3:DataAccessPointArn": "arn:aws:s3:*:*:accesspoint/deadline-software-*"
    },
    "StringEquals": {
      "s3:AccessPointNetworkOrigin": "VPC"
    }
  }
}
```

ジョブのアクセス許可を追加する

ジョブがアクセスする必要がある AWS のサービス のキューロールにアクセス許可を追加します。OpenJobDescription ステップスクリプトを記述すると、 AWS CLI と SDK はキューロールの認

証情報を自動的に使用します。これを使用して、ジョブの完了に必要な追加サービスにアクセスします。

ユースケースの例を以下に示します。

- カスタムデータを取得するための
- カスタムライセンスサーバーにトンネルする SSM アクセス許可
- カスタムメトリクスを出力するための CloudWatch
- 動的ワークフロー用の新しいジョブを作成する Deadline Cloud アクセス許可

キューロールの認証情報の使用方法

Deadline Cloud は、キューロールの認証情報を以下に提供します。

- ジョブ実行中のワーカー
- Deadline Cloud CLI を介したユーザーと、ジョブの添付ファイルやログを操作する際のモニタリング

Deadline Cloud は、キューごとに個別の CloudWatch Logs ロググループを作成します。ジョブはキューロールの認証情報を使用して、キューのロググループにログを書き込みます。Deadline Cloud CLI とモニターは、キューロール (経由`deadline:AssumeQueueRoleForRead`) を使用してキューのロググループからジョブログを読み込みます。Deadline Cloud CLI とモニターは、キューロール (経由`deadline:AssumeQueueRoleForUser`) を使用してジョブアタッチメントデータをアップロードまたはダウンロードします。

ロールをモニタリングする

Deadline Cloud モニターウェブおよびデスクトップアプリケーションに Deadline Cloud リソースへのアクセスを許可するようにモニターロールを設定します。

プログラムでモニターを作成または更新する場合は、 `CreateMonitor` または `UpdateMonitor` API オペレーションを使用してモニターロール ARN を指定します。

モニターロールの動作

モニターロールにより、Deadline Cloud モニターはエンドユーザーに以下へのアクセスを提供できます。

- Deadline Cloud Integrated Submitters、CLI、モニターに必要な基本機能

- エンドユーザー向けのカスタム機能

モニターロールの信頼ポリシーを設定する

モニターロールは Deadline Cloud サービスを信頼する必要があります。

ベストプラクティスとして、信頼ポリシーには混乱した代理保護のセキュリティ条件を含める必要があります。混乱した代理の保護の詳細については、「Deadline Cloud ユーザーガイド」の「混乱した代理」を参照してください。

aws:SourceAccount は、同じ のリソースのみがこのロールを引き受け AWS アカウント ることができるようにします。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.deadline.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "YOUR_ACCOUNT_ID"
        }
      }
    }
  ]
}
```

モニターロールのアクセス許可をアタッチする

基本オペレーションのために、以下の AWS 管理ポリシーをすべてモニターロールにアタッチします。

- AWSDeadlineCloud-UserAccessFarms
- AWSDeadlineCloud-UserAccessFleets
- AWSDeadlineCloud-UserAccessJobs
- AWSDeadlineCloud-UserAccessQueues

モニターロールの仕組み

Deadline Cloud モニターを使用する場合、サービスユーザーは AWS IAM アイデンティティセンター (IAM Identity Center) を使用してサインインし、モニターロールが引き受けられます。引き受けたロール認証情報は、ファーム、フリート、キュー、その他の情報のリストなど、モニター UI を表示するためにモニターアプリケーションによって使用されます。

Deadline Cloud モニターデスクトップアプリケーションを使用する場合、これらの認証情報は、エンドユーザーから提供されたプロファイル名に対応する名前付き AWS 認証情報プロファイルを使用してワークステーションでさらに利用可能になります。名前付きプロファイルの詳細については、AWS SDK およびツールリファレンスガイドを参照してください。

この名前付きプロファイルは、Deadline CLI と送信者が Deadline Cloud リソースにアクセスする方法です。

高度なユースケースに合わせてモニターロールをカスタマイズする

モニターロールをカスタマイズして、各アクセスレベル (ビューワー、コントリビューター、マネージャー、所有者) でユーザーが実行できる操作を変更したり、高度なワークフローのアクセス許可を追加したりできます。

アクセスレベルのアクセス許可のカスタマイズ

モニターロールにアタッチされた 4 つの AWS 管理ポリシーは、各アクセスレベルが実行できる操作を制御します。モニターロールにカスタムポリシーを追加して、 `deadline:MembershipLevel`条件キーを使用して特定のアクセスレベルのアクセス許可を付与または制限できます。

たとえば、コントリビューターがジョブ (通常はマネージャーと所有者に制限されます) を更新およびキャンセルできるようにするには、次のようなポリシーを追加します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "deadline:UpdateJob",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "deadline:MembershipLevel": "CONTRIBUTOR"
        }
```

```
      }
    }
  ]
}
```

このポリシーを使用すると、コントリビューターはジョブを送信することに加えて、ジョブを更新およびキャンセルできます。

高度なワークフローに対するアクセス許可の追加

カスタム IAM ポリシーをモニターロールに追加して、すべてのモニターユーザーに追加のアクセス許可を付与できます。これは、ユーザーが標準の Deadline Cloud 機能 AWS のサービス を超えて にアクセスする必要がある高度なスクリプトワークフローに役立ちます。

モニターロールを変更するときは、次のガイドラインに従ってください。

- 管理ポリシーを削除しないでください。これらのポリシーを削除すると、モニター機能が壊れます。

Deadline Cloud Monitor がモニターロールの認証情報を使用する方法

Deadline Cloud Monitor は、認証時にモニターロールの認証情報を自動的に取得します。この機能を使用すると、デスクトップアプリケーションは、標準のウェブブラウザで使用できる以上の拡張モニタリング機能を提供できます。

Deadline Cloud Monitor でログインすると、 AWS CLI または他の AWS ツールで使用できるプロファイルが自動的に作成されます。このプロファイルはモニターロールの認証情報を使用し、モニターロールのアクセス許可 AWS のサービス に基づいて へのプログラムによるアクセスを許可します。

Deadline Cloud 送信者は同じように動作します。Deadline Cloud モニターによって作成されたプロファイルを使用して、適切なロールのアクセス許可 AWS のサービス で にアクセスします。

Deadline Cloud ロールの高度なカスタマイズ

Deadline Cloud ロールを追加のアクセス許可で拡張して、基本的なレンダリングワークフロー以外の高度なユースケースを実現できます。このアプローチでは、Deadline Cloud のアクセス管理システムを活用して、キューメンバーシップ AWS のサービス に基づいて追加の へのアクセスを制御します。

とのチームコラボレーション AWS CodeCommit

キューロールにアクセス AWS CodeCommit 許可を追加して、プロジェクトリポジトリでチームコラボレーションを有効にします。このアプローチでは、追加のユースケースに Deadline Cloud のアクセス管理システムを使用します。特定のキューにアクセスできるユーザーのみがこれらのアクセス AWS CodeCommit 許可を受け取るため、Deadline Cloud キューメンバーシップを通じてプロジェクトごとのリポジトリアクセスを管理できます。

これは、アーティストがレンダリングワークフローの一部として AWS CodeCommit リポジトリに保存されているプロジェクト固有のアセット、スクリプト、または設定ファイルにアクセスする必要があるシナリオに役立ちます。

キューロールにアクセス AWS CodeCommit 許可を追加する

アクセスを有効にするには、キューロールに次の AWS CodeCommit アクセス許可を追加します。

```
{
  "Effect": "Allow",
  "Action": [
    "codecommit:GitPull",
    "codecommit:GitPush",
    "codecommit:GetRepository",
    "codecommit:ListRepositories"
  ],
  "Resource": "arn:aws:codecommit:REGION:YOUR_ACCOUNT_ID:PROJECT_REPOSITORY"
}
```

アーティストワークステーションで認証情報プロバイダーを設定する

AWS CodeCommit アクセスに Deadline Cloud キュー認証情報を使用するように各アーティストワークステーションを設定します。この設定はワークステーションごとに 1 回行われます。

認証情報プロバイダーを設定するには

1. 認証情報プロバイダープロファイルを設定ファイル (~/.aws/config) AWS に追加します。

   ```
   [profile queue-codecommit]
   credential_process = deadline queue export-credentials --farm-id farm-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX --queue-id queue-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
   ```

2. AWS CodeCommit リポジトリにこのプロファイルを使用するように Git を設定します。

```
git config --global credential.https://git-codecommit.REGION.amazonaws.com.helper
 '!aws codecommit credential-helper --profile queue-codecommit $@'
git config --global credential.https://git-
codecommit.REGION.amazonaws.com.UseHttpPath true
```

farm-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX および *queue-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX* を実際のファーム ID とキュー IDs。*REGION* を自分の AWS リージョン (例:) に置き換えますus-west-2。

キュー認証情報 AWS CodeCommit での の使用

設定すると、Git オペレーションは AWS CodeCommit リポジトリにアクセスするときにキューロール認証情報を自動的に使用します。deadline queue export-credentials コマンドは、次のような一時的な認証情報を返します。

```
{
  "Version": 1,
  "AccessKeyId": "ASIA...",
  "SecretAccessKey": "...",
  "SessionToken": "...",
  "Expiration": "2025-11-10T23:02:23+00:00"
}
```

これらの認証情報は必要に応じて自動的に更新され、Git オペレーションはシームレスに機能します。

```
git clone https://git-codecommit.REGION.amazonaws.com/v1/repos/PROJECT_REPOSITORY
git pull
git push
```

アーティストは、個別の AWS CodeCommit 認証情報を必要とせずに、キューのアクセス許可を使用してプロジェクトリポジトリにアクセスできるようになりました。特定のキューにアクセスできるユーザーのみが関連付けられたリポジトリにアクセスできるため、Deadline Cloud のキューメンバーシップシステムを通じてきめ細かなアクセスコントロールが可能になります。

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正に役立ちます。

トピック

- Deadline Cloud でアクションを実行する権限がありません
- 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ありません

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 deadline:*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:
 deadline:GetWidget on resource: my-example-widget
```

この場合、deadline:*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アクセス許可が必要です。

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アクセス許可が必要です。Mary には、ロールをサービスに渡すアクセス許可がありません。

```
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、「コンプライアンスAWS のサービス プログラムによる対象範囲内」の「コンプライアンス」を参照し、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS 「コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「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。を使用する際のコンプライアンス責任の詳細については AWS のサービス、AWS 「 セキュリティドキュメント」を参照してください。

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ー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、フォールトトレランス、および拡張性が優れています。

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環境を AWS 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。クライアントは次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。

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:deadline:*:*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。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "deadline.amazonaws.com"
```

```
    },
    "Action": "deadline:CreateFarm",
    "Resource": [
      "*"
    ],
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:deadline:*:111122223333:*"
      },
      "StringEquals": {
        "aws:SourceAccount": "111122223333"
      }
    }
  }
}
```

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 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 内のインスタンスは Deadline Cloudにアクセスするためにパブリック IP アドレスを必要としません。

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 ガイド」の「Access AWS のサービス through AWS PrivateLink」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 を介して 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が `CreateJob`オペレーションを呼び出すことができるようにする必要があります。

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を管理します`management.deadline.region.amazonaws.com`。

AWS SDKsと CLI は、 `management`および `scheduling`プレフィックスをエンドポイントに自動的に追加します。この動作を無効にする場合は、「 SDK およびツールリファレンスガイド」の「ホストプレフィックスインジェクション」セクションを参照してください。 AWS SDKs

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
```

制限されたネットワーク環境

Deadline Cloud には、アーティストや他のユーザーがローカルワークステーションで使用するツールが用意されています。これらのツールを実行するには、 AWS API エンドポイントとウェブエンドポイントにアクセスする必要があります。次世代ファイアウォール (NGFW) や Secure Web Gateway (SWG) などのウェブコンテンツフィルタリングソリューションを使用して特定の AWS ドメインまたは URL エンドポイントへのアクセスをフィルタリングする場合は、ウェブコンテンツフィルタリングソリューションの許可リストに次のドメインまたは URL エンドポイントを追加する必要があります。

AWS 許可リストの API エンドポイント

、モニター AWS マネジメントコンソール、CLI、統合送信者などの Deadline Cloud クライアントツールには、Deadline Cloud に加えて AWS APIs へのアクセスが必要です。これらのエンドポイントは IPv4 のみをサポートします。

- scheduling.deadline.*[Region]*.amazonaws.com
- management.deadline.*[Region]*.amazonaws.com
- logs.*[Region]*.amazonaws.com
- ec2.*[Region]*.amazonaws.com
- s3.*[Region]*.amazonaws.com
- sts.*[Region]*.amazonaws.com
- identitystore.*[Region]*.amazonaws.com

許可リストのウェブドメイン

Deadline Cloud モニターを操作するには、次のドメインにアクセスする必要があります。

AWS サインインの許可リストドメインの詳細については、AWS 「 サインインユーザーガイド」の「許可リストに追加するドメイン」を参照してください。

- downloads.deadlinecloud.amazonaws.com
- d2ev1rdnjzhmnr.cloudfront.net
- prod.log.shortbread.aws.dev

- prod.tools.shortbread.aws.dev
- prod.log.shortbread.analytics.console.aws.a2z.com
- prod.tools.shortbread.analytics.console.aws.a2z.com
- global.help-panel.docs.aws.a2z.com
- *[Region]*.signin.aws
- *[Region]*.signin.aws.amazon.com
- sso.*[Region]*.amazonaws.com
- portal.sso.*[Region]*.amazonaws.com
- oidc.*[Region]*.amazonaws.com
- assets.sso-portal.*[Region]*.amazonaws.com

Deadline Cloud 送信者は、GUI 依存関係をダウンロードするために次のドメインにアクセスする必要があります。

- pypi.python.org
- pypi.org
- pythonhosted.org
- files.pythonhosted.org

許可リストを作成する環境固有のエンドポイント

これらのドメインは、Deadline Cloud の特定の設定によって異なります。追加の Deadline Cloud モニターまたはキューが作成された場合は、追加のドメインを許可リストに登録する必要があります。

- *[Directory ID or alias]*.awsapps.com

  このドメインは IAM Identity Center のセットアップに関連付けられており、同じ IAM Identity Center インスタンスを使用するこのすべてのセットアップで同じである必要があります。正確な値は、IAM Identity Center コンソールの設定 → AWS アクセスポータル URL のエンタープライズ管理者が見つけることができます。
- *[Monitor alias]*.*[Region]*.deadlinecloud.amazonaws.com

  このドメインは、Deadline Cloud での Monitor のセットアップ用です。アーティストは、ブラウザまたは Deadline Cloud Monitor アプリケーションにこのリンクを入力します。Deadline Cloud

が将来的に追加のアカウントまたはリージョンに設定されている場合、このドメインは変更されます。この値は、Dashboard → Monitor overview → Monitor details → URL の Deadline Cloud コンソールで確認できます。

- *[Bucket name]*.*[Region]*.s3.amazonaws.com

  これは、Deadline Cloud キューで使用されるジョブアタッチメントバケットのドメインです。各キューには、独自のジョブアタッチメントバケットを設定できます。正確なバケット名は、Deadline Cloud コンソールの Queues → Queue details → Job attachments にあります。ジョブアタッチメントの詳細については、キューのドキュメント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

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セキュリティ機能が多数用意されています。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 は必須ですが、TLS 1.3 を推奨します。
- 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。この推奨事項には、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 を使用する場合も含まれます。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。ワーカーホストで利用可能なデータは、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。

JSON

```
{
    "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": "111122223333"
                }
            }
        },
        {
            "Action": [
                "logs:GetLogEvents"
            ],
            "Effect": "Allow",
```

```
            "Resource": "arn:aws:logs:us-east-1:111122223333: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。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "sts:AssumeRole"
            ],
            "Effect": "Allow",
            "Principal": {
                "Service": "deadline.amazonaws.com"
            },
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:deadline:us-
east-1:111122223333:farm/FARM_ID"
                }
            }
        },
        {
            "Action": [
                "sts:AssumeRole"
            ],
            "Effect": "Allow",
            "Principal": {
                "Service": "credentials.deadline.amazonaws.com"
            },
            "Condition": {
```

```
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:deadline:us-
east-1:111122223333: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、*BUCKET_ACCOUNT_OWNER* をバケットを所有する AWS アカウント ID 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/*"
        ],
        "Condition": {
         "StringEquals": {
            "aws:ResourceAccount": "BUCKET_ACCOUNT_OWNER"
         }
      }
    }
]
```

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 が所有するアカウントのみがWindows独自のものであり、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更新します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には、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 など 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`/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でソフトウェアの信頼性を検証し、ファイルの改ざんから保護します。この手順は、 WindowsおよびLinuxシステムの両方で機能します。

Server

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*file*で、 を、検証するファイルに置き換えます。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
   ```

Linux

ダウンロードしたファイルの真正性を確認するには、 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-----

mQINBGlANDUBEACg6zffjN43gqe5ryPhk+wQM10rEdvmItw4WPWaVsN+/at/OIJw
MGCagSYXcgR+jKbsHQOQoEQdo5SrxxHjpKTEs3KQhGvf+ehrU1Ac7koXKIBWtes+
BI9F0slRECz0nXTOy/cd/90RXjpF07mreTLIKNIbybULfad82nYykpITjFr5XRGj
/shYkucxRQZdwkgkIYyV25pPICPd2RsX+Zua85jV8mCqVffDfRXvgcPe3+ofClj/
2CE8UfUIqO8Csua4YEkSqr3aeoTOEFT4kuQR5nFXVzorOEkQtO3gB35KNWKMlIOU
2vA+wyoL7nWSii4yfYtW3EZ+3gq6HxvnT9Zs8MC53uTOiOdamASXecYREwGmY/io
6n5XTEA/35LNbl4A756vSTZ7h4VFJAN5BpuqxstI1D7ou94skoSmcPoC/iniTvY9
kZylU5OCH/nifMAHM2a5jrQel80cW4oko9eyc8ENQpSy15JElFOKFf7D/4tcZJLF
F0VBTXbhfvq3dPfoq94IWt7p54Ovwj0S//CEu3jZYbNl2QC/3YiHE2H2XyGCQbq6
2MjcuxLnEapoRIqfbi8GPtCWVPzm28WGyKIDofWICczzeJFFJnvzrY3wRG64ibKJ
bR/uedwua1UuiC482V1FD5ffmzSSs8ktTp9hgj7RGDXlc9NTcF1jHxG9hwARAQAB
tCxBV1MgRGVhZGxpbmUgQ2xvdWQgPGF3cy1kZWFkbGluZUBhbWF6b24uY29tPokC
VwQTAQgAQRYhBJmXd7So2csyehiIYsg71N18bhtjBQJpQDQ1AhsvBQkDwmcABQsJ
CAcCAiICBhUKCQgLAgQWAgMBAh4HAheAAAoJEMg71N18bhtjk2UP/3h4KlEzZO/7
BxRmkbixuo1QuqOGvA6tXbSWaM8QH5jglcvL12PZLALklLT4v82uCsLR1lF8/Tch
cCl0SZEOFIS+XxAaw1Xfai6jlyLhabOwKF2ylq5eJlLcw1lh2nAArDRb4fLD0m1g
Dfqetq/XEpyXpOSkWxGRV4RlUdjQfytxrmcUnsT5/fk5f9VDdblu6K/lEmwfyYjB
lXv0uUCkqPot0SmbvOh3PY3Hi3n54ncy8NfTeV+TUvSe3C1s1zNl8aqHoTxJB/eU
kp+LFZ9m+igpSYnKeglKnytylH3KGCjTHglT/QXnI1wNTqmj1kFBVwtt/y1mtnA+
CPIUHP1CtbKsHaLtpp4llBm5TVtPN/Wqqicn5QLl4khg7R4K+V2aaA4ubY6p1tG9
0fFhN5tTnHDSKWMfmb83wfh5Zkcg85c3egjoit+wgGQRAQVqbznx7NqAHs9VoDIu
SPcAr+C329AOBzod4gyNGH7Ah5DkMITo4O4+axnAU9yhFOHcMJmTIask/fNg1Aum
OqYPMUwcgv1GZjLaTJyfGGC1xALsYR0KHnwIehD06MHR/Z98bGkcV8+Y0q8UPsd1
VN1fc1rjCJh/AT3w6owvG4DaEwspseSjzHv16mW4e2N6Uu23SPzgQsJ5qYN2g8D+
P7N9LGDfP8DaYc5JM9mlyFmYI2Q94ufL
=rY5l
-----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 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。

      署名ファイル (.sig) をダウンロードする

   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 Monitor インストーラに対応する署名ファイルをダウンロードします。

      - .deb 署名ファイルをダウンロードする
      - .rpm 署名ファイルをダウンロードする
      - .AppImage 署名ファイルをダウンロードする

   b. 署名を確認します。

      .deb の場合:

```
gpg --verify ./deadline-cloud-monitor_amd64.deb.sig ./deadline-cloud-
monitor_amd64.deb
```

      .rpm の場合:

```
gpg --verify ./deadline-cloud-monitor.x86_64.rpm.sig ./deadline-cloud-
monitor.x86_64.rpm
```

.AppImage の場合:

```
gpg --verify ./deadline-cloud-monitor_amd64.AppImage.sig ./deadline-cloud-
monitor_amd64.AppImage
```

c. 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 Monitor のインストールスクリプトを実行できます。

履歴キー

```
-----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+VpkpQmLgwtMGpSML13KLwnv2k
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
```

Linux (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。

Linux (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 Monitor .deb ファイルをダウンロードします。

   Deadline Cloud Monitor のダウンロード (.deb)

2. .deb ファイルを確認します。

```
dpkg-sig --verify deadline-cloud-monitor_amd64.deb
```

3. 出力は次のようになります。

```
Processing deadline-cloud-monitor_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。

Linux (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 Monitor .rpm ファイルをダウンロードします。

   Deadline Cloud Monitor のダウンロード (.rpm)

2. .rpm ファイルを確認します。

```
gpg --export --armor "Deadline Cloud" > key.pub
sudo rpm --import key.pub
rpm -K deadline-cloud-monitor.x86_64.rpm
```

3. 出力は次のようになります。

```
deadline-cloud-monitor.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

AWSDeadline Cloud の更新については、Deadline Cloud リリースノートを参照してください。

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。