開発者ガイド

AWS Panorama

AWS Panorama: 開発者ガイド

Table of Contents

AWS Panorama とは?

AWS Panorama は、オンプレミスのカメラネットワークでコンピュータビジョンを利用できるようにするサービスです。AWS Panorama アプライアンスやその他の互換デバイスをデータセンターに設置してAWS Panorama で登録し、クラウドからコンピュータービジョンアプリケーションをデプロイします。AWS Panorama は既存のリアルタイムストリーミングプロトコル (RTSP) ネットワークカメラで動作します。アプライアンスは、AWS パートナーが提供する安全なコンピュータビジョンアプリケーション、または AWS Panorama アプリケーション SDK を使用して独自に構築したアプリケーションを実行します。

AWS Panorama アプライアンスは、機械学習のワークロードに最適化された強力なシステムオンモジュール (SOM) を使用するコンパクトなエッジアプライアンスです。アプライアンスは、複数のビデオストリームに対して複数のコンピュータービジョンモデルを並列実行し、結果をリアルタイムで出力できます。商業および産業環境での使用を想定して設計されており、防塵・液体保護 (IP-62) に対応しています。

AWS Panorama アプライアンスを使用すると、AWS クラウドに画像を送信しなくても、自己完結型のコンピュータービジョンアプリケーションをエッジで実行できます。AWS SDK を使用すると、他の AWS サービスと統合し、それらを使用してアプリケーションからのデータを時系列的に追跡できます。他の AWS サービスと統合することで、AWS Panorama を使用して次の処理ができます。

- トラフィックパターンの分析 — AWS SDK を使用して、Amazon DynamoDB の小売分析用のデータを記録します。サーバーレスアプリケーションを使用して、収集したデータを経時的に分析し、データの異常を検出し、将来の行動を予測します。
- 現場の安全に関する警告を受け取る — 産業現場の立ち入り禁止区域を監視します。アプリケーションが潜在的に危険な状況を検出したら、Amazon Simple Storage Service (Amazon S3) にイメージをアップロードし、Amazon Simple Notification Service (Amazon SNS) トピックに通知を送信します。
- 品質管理の向上 — 組立ラインの出力を監視して、要件に適合しない部品を特定します。適合しない部品の画像をテキストとバウンディングボックスで強調表示し、品質管理チームが確認できるようにモニターに表示します。
- トレーニングデータとテストデータの収集 — コンピュータービジョンモデルが識別できなかったオブジェクトや、モデルの推測に対する信頼性が境界線だったオブジェクトの画像をアップロードします。サーバーレスアプリケーションを使用して、タグ付けが必要な画像のキューを作成します。イメージにタグを付け、それらを使用して Amazon SageMaker でモデルを再トレーニングします。

AWS Panorama は他の AWS サービスを使用して、AWS Panorama アプライアンスの管理、モデルとコードへのアクセス、アプリケーションのデプロイを行います。AWS Panorama はユーザーが他のサービスとやり取りすることなくできる限り対応しますが、以下のサービスに関する知識は AWS Panorama の仕組みを理解するのに役立ちます。

- SageMaker — SageMaker を使用して、カメラやセンサーからトレーニングデータを収集し、機械学習モデルを構築して、コンピュータービジョン用にトレーニングすることができます。AWS Panorama はSageMaker Neo を使用してモデルをAWS Panorama アプライアンス上で実行するように最適化します。
- Amazon S3 — Amazon S3 Access Pointsを使用して、AWS Panorama アプライアンスにデプロイするアプリケーションコード、モデル、および構成ファイルをステージングします。
- AWS IoT — AWS Panorama は AWS IoT サービスを使用して AWS Panorama アプライアンスの状態を監視し、ソフトウェアアップデートを管理し、アプリケーションをデプロイします。AWS IoT を直接使用する必要はありません。

AWS Panorama アプライアンスを使い始めてサービスの詳細を確認するには、AWS Panorama の開始方法 に進んでください。

AWS Panorama の開始方法

AWS Panorama を始めるには、まずサービスの概念とこのガイドで使用されている用語について学んでください。その後、AWS Panorama コンソールを使用してAWS Panorama アプライアンスを登録し、アプリケーションを作成できます。約 1 時間で、デバイスの設定、ソフトウェアの更新、サンプルアプリケーションのデプロイを行うことができます。このセクションのチュートリアルを完了するには、AWS Panorama アプライアンス と、ローカル・ネットワーク経由でビデオをストリーミングするカメラを使用します。

> **Note**
>
> AWS Panorama アプライアンス を購入するには、AWS Panorama コンソールにアクセスしてください。

AWS Panorama サンプルアプリケーションは AWS Panorama 機能の使用方法を示しています。SageMaker でトレーニングされたモデルと、AWS Panorama Application SDK を使用して推論を実行してビデオを出力するサンプルコードが含まれています。サンプルアプリケーションには、開発とデプロイのワークフローをコマンドラインから自動化する方法を示す AWS CloudFormation テンプレートとスクリプトが含まれています。

この章の最後の 2 つのトピックでは、モデルとカメラの要件、およびAWS Panorama アプライアンスのハードウェア仕様について詳しく説明します。アプライアンスやカメラをまだ入手していない場合や、独自のコンピュータービジョンモデルを開発する計画がある場合は、まずこれらのトピックで詳細を確認してください。

トピック

- AWS Panorama の概念
- AWS Panorama アプライアンスのセットアップ
- AWS Panorama サンプルアプリケーションのデプロイ
- AWS Panorama アプリケーションの開発
- サポートされているコンピュータービジョンモデルとカメラ
- AWS Panorama アプライアンスの仕様
- Service Quotas

AWS Panorama の概念

AWS Panorama では、コンピュータビジョンアプリケーションを作成し、それを AWS Panorama アプライアンスまたは互換性のあるデバイスにデプロイして、ネットワークカメラからのビデオストリームを分析します。Python でアプリケーションコードを記述し、ドッカーを使用してアプリケーションコンテナを構築します。AWS Panorama アプリケーション CLI を使用して、機械学習モデルをローカルまたは Amazon Simple Storage Service (Amazon S3) からインポートします。アプリケーションは AWS Panorama アプリケーション SDK を使用してカメラからのビデオ入力を受信し、モデルと対話します。

概念

- AWS Panorama アプライアンス
- 互換性のあるデバイス
- アプリケーション
- ノード
- モデル

AWS Panorama アプライアンス

AWS Panorama アプライアンスはアプリケーションを実行するハードウェアです。アプライアンスの登録、ソフトウェアの更新、アプリケーションのデプロイを行うために AWS Panorama コンソールを使用します。AWS Panorama アプライアンスのソフトウェアは、カメラストリームに接続し、ビデオフレームをアプリケーションに送信し、付属のディスプレイにビデオ出力を表示します。

AWS Panorama アプライアンスは Nvidia Jetson AGX Xavier を搭載したエッジデバイスです。処理のために画像を AWS クラウドに送信する代わりに、最適化されたハードウェア上でアプリケーションをローカルに実行します。これにより、動画をリアルタイムで分析し、結果をローカルで処理できます。アプライアンスのステータスの報告、ログのアップロード、およびソフトウェアの更新とデプロイを行うには、インターネット接続が必要です。

詳細については、「AWS Panorama アプライアンスの管理」を参照してください。

互換性のあるデバイス

AWS Panorama アプライアンスに加えて、AWS Panorama はAWS パートナーの互換性のあるデバイスをサポートしています。互換性のあるデバイスは AWS Panorama アプライアンスと同じ機能を

サポートします。互換性のあるデバイスを AWS Panorama コンソールと API に登録して管理し、同じ方法でアプリケーションを構築してデプロイします。

- レノボ ThinkEdge® SE70 — Nvidia Jetson Xavier NX を搭載

このガイドの内容とサンプルアプリケーションは、AWS Panorama アプライアンスを使用して開発されています。お使いのデバイスの特定のハードウェアおよびソフトウェア機能の詳細については、製造元のドキュメントを参照してください。

アプリケーション

アプリケーションは AWS Panorama アプライアンス上で動作し、ビデオストリームでコンピュータビジョンタスクを実行します。Python コードと機械学習モデルを組み合わせてコンピュータビジョンアプリケーションを構築し、インターネット経由で AWS Panorama アプライアンスにデプロイできます。アプリケーションはディスプレイに動画を送信したり、AWS SDK を使用して結果を AWS のサービスに送信したりできます。

アプリケーションを構築してデプロイするには、AWS Panorama アプリケーション CLI を使用します。AWS Panorama アプリケーション CLI は、デフォルトのアプリケーションフォルダと構成ファイルの生成、ドッカーによるコンテナの構築、アセットのアップロードを行うコマンドラインツールです。1 つのデバイスで複数のアプリケーションを実行できます。

詳細については、「AWS Panorama アプリケーションを管理する」を参照してください。

ノード

アプリケーションは、入力、出力、モデル、コードを表すノードと呼ばれる複数のコンポーネントで構成されます。ノードは構成のみ (入力と出力) でも、アーティファクト (モデルとコード) を含むこともできます。アプリケーションのコードノードはノードパッケージにバンドルされており、Amazon S3 アクセスポイントにアップロードすると、AWS Panorama アプライアンスからアクセスできます。アプリケーションマニフェストは、ノード間の接続を定義する構成ファイルです。

詳細については、「アプリケーションノード」を参照してください。

モデル

コンピュータビジョンモデルは、画像を処理するようにトレーニングされた機械学習ネットワークです。コンピュータビジョンモデルは、分類、検出、セグメンテーション、追跡などのさまざまなタス

クを実行できます。コンピュータビジョンモデルは、画像を入力として受け取り、画像または画像内のオブジェクトに関する情報を出力します。

AWS Panorama は PyTorch、Apache MXNet、TensorFlow で構築されたモデルをサポートしています。Amazon SageMaker を使用してモデルを構築することも、お使いの開発環境で構築することもできます。詳細については、「???」を参照してください。

AWS Panorama アプライアンスのセットアップ

AWS Panorama アプライアンスまたは互換性のあるデバイスの使用を開始するには、AWS Panorama コンソールに登録し、ソフトウェアを更新します。セットアッププロセス中に、物理アプライアンスを表すアプライアンスリソースを AWS Panorama に作成し、USB ドライブを使用してファイルをアプライアンスにコピーします。アプライアンスはこれらの証明書と設定ファイルを使用して AWS Panorama サービスに接続します。次に、AWS Panorama コンソールを使用してアプライアンスのソフトウェアを更新し、カメラを登録します。

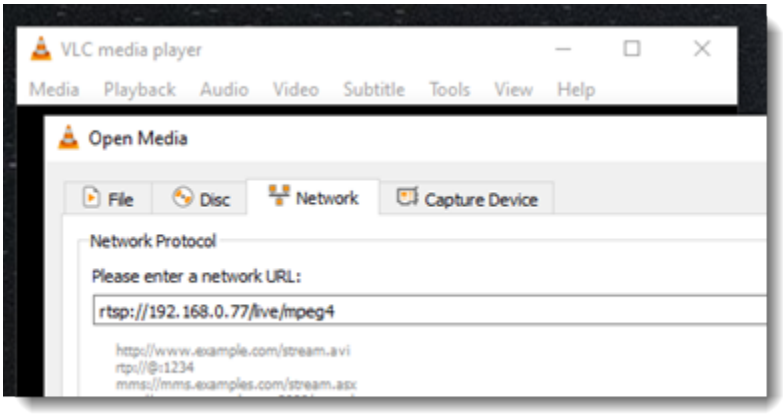
セクション

前提条件

このチュートリアルを実行するには、AWS Panorama アプライアンスまたは互換性のあるデバイスと、次のハードウェアが必要です：

- ディスプレイ — サンプルアプリケーション出力を表示するための HDMI 入力付きディスプレイ。
- USB ドライブ (AWS Panorama アプライアンスに付属) — 設定ファイルと証明書を含むアーカイブを AWS Panorama アプライアンスに転送するための、少なくとも 1 GB のストレージを備えた FAT32 フォーマットの USB 3.0 フラッシュメモリドライブ。
- カメラ — RTSP ビデオストリームを出力する IP カメラ。

カメラの製造元から提供されているツールと説明書を使用して、カメラの IP アドレスとストリームパスを特定します。VLC などの動画プレーヤーをネットワークメディアソースとして開くと、ストリーム URL を確認できます:



AWS Panorama コンソールは、他の AWS サービスを使用してアプリケーションコンポーネントの組み立て、権限の管理、設定の検証を行います。アプライアンスを登録してサンプルアプリケーションをデプロイするには、以下の権限が必要です:

- AWSPanoramaFullAccess — AWS Panorama、Amazon S3 の AWS Panorama アクセスポイント、AWS Secrets Manager のアプライアンス認証情報、Amazon CloudWatch のアプライアンスログへのフルアクセスを提供します。AWS Panorama のサービスにリンクされたロールを作成する権限が含まれます。
- AWS Identity and Access Management(IAM) — 初回実行時に、AWS Panorama サービスと AWS Panorama アプライアンスで使用されるロールを作成します。

IAM でロールを作成する権限がない場合は、管理者に AWS Panorama コンソールを開き、サービスロールを作成するためのプロンプトを受け入れてもらいます。

AWS Panorama アプライアンスの登録と設定

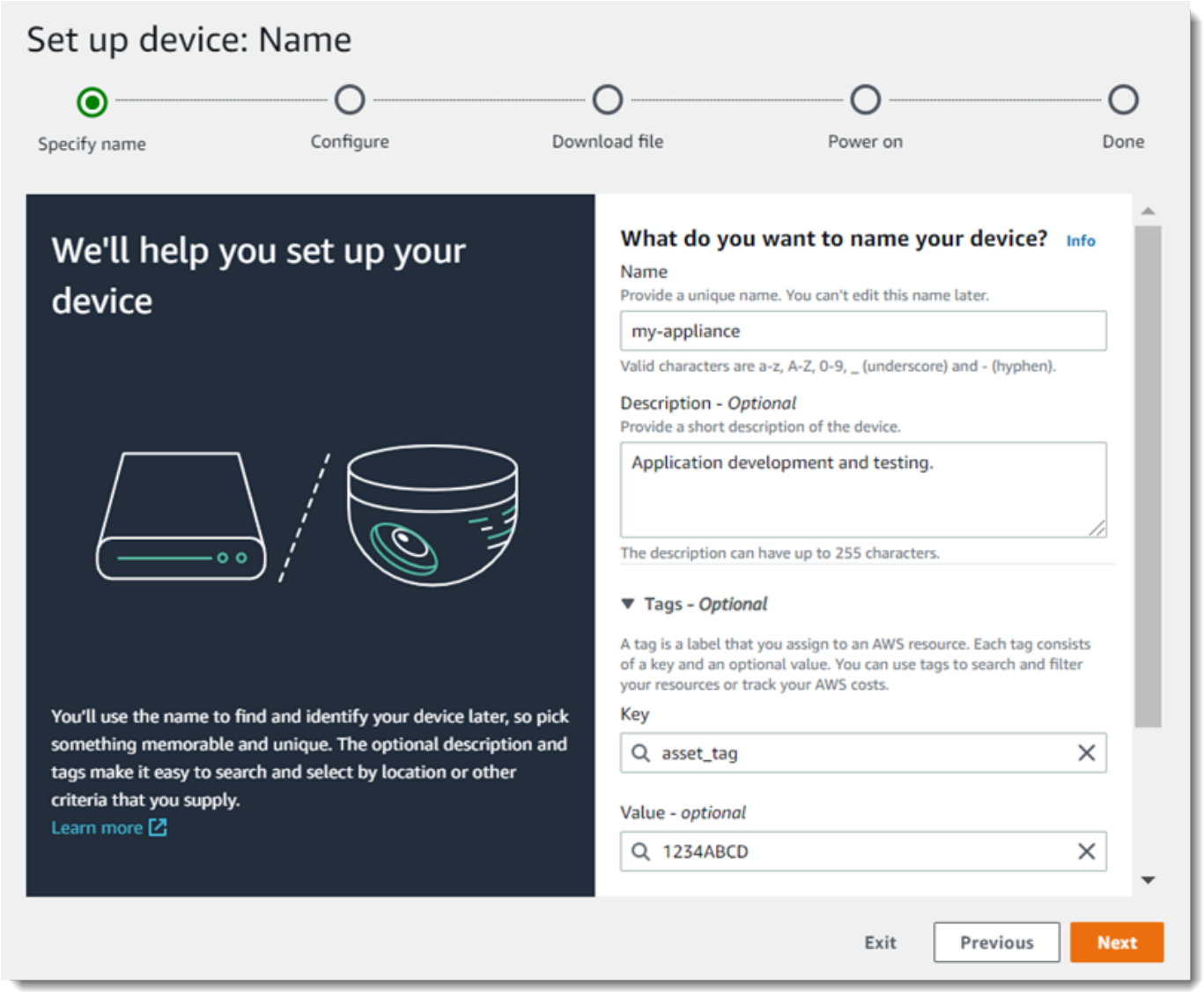
AWS Panorama アプライアンスは、ローカルネットワーク接続を介してネットワーク対応カメラに接続するハードウェアデバイスです。AWS Panorama Application SDK とコンピュータビジョンアプリケーションを実行するためのサポートソフトウェアを含む Linux ベースのオペレーティングシステムを使用しています。

AWS に接続してアプライアンスを管理し、アプリケーションをデプロイする場合、アプライアンスはデバイス証明書を使用します。AWS Panorama コンソールを使用してプロビジョニング証明書を生成します。アプライアンスはこの一時証明書を使用して初期設定を完了し、永久デバイス証明書をダウンロードします。

> **Important**
>
> この手順で生成するプロビジョニング証明書は 5 分間だけ有効です。この時間内に登録プロセスを完了しない場合は、最初からやり直す必要があります。

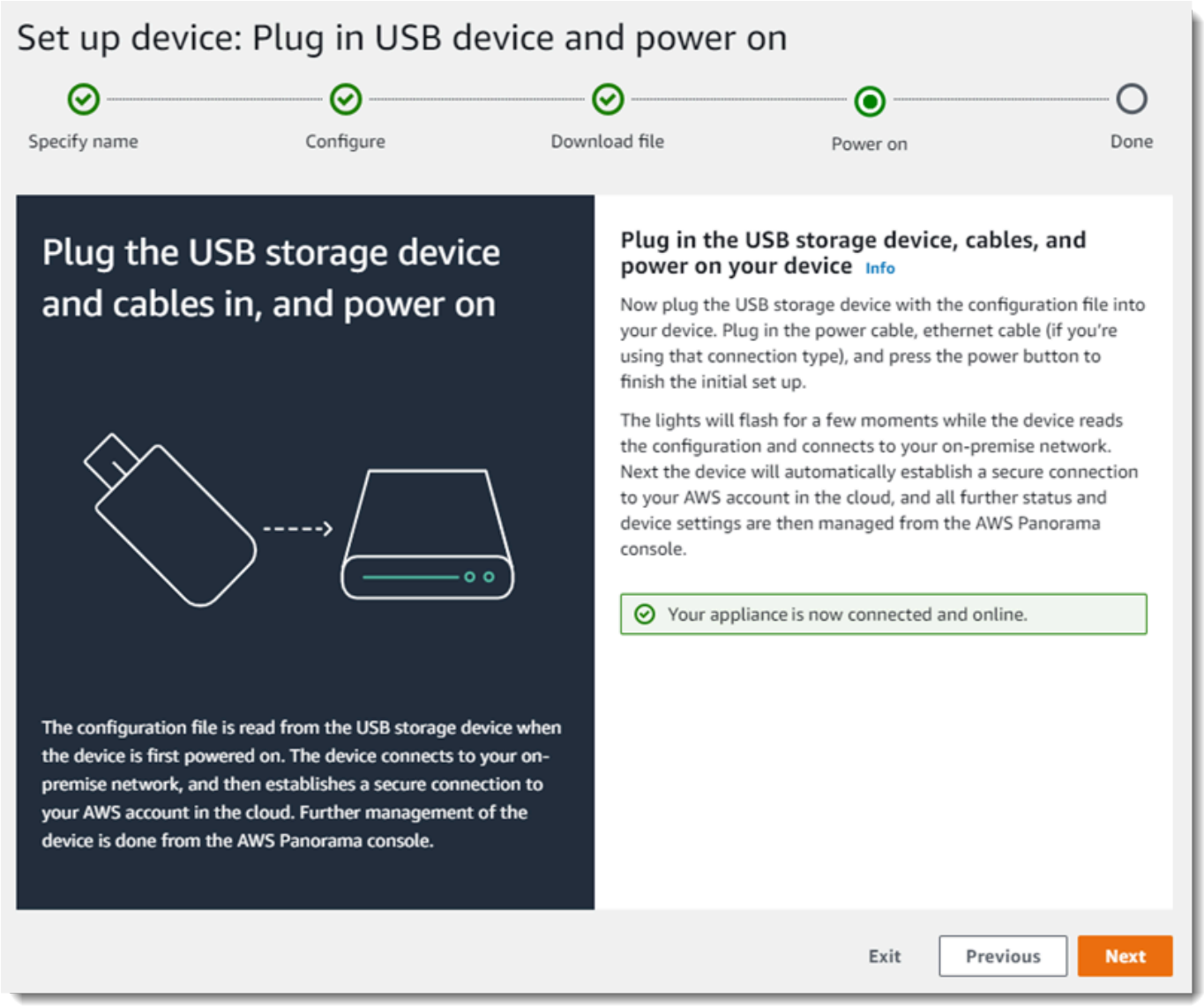
アプライアンスを登録する

1. コンピュータに USB ドライブを接続します。ネットワークと電源ケーブルを接続してアプライアンスを準備します。アプライアンスの電源が入り、USB ドライブが接続されるのを待ちます。
2. AWS Panorama コンソールの「はじめに」ページを開きます。
3. デバイスを追加 を選びます。
4. セットアップを開始 を選択します。
5. AWS Panorama でアプライアンスを表すデバイスリソースの名前と説明を入力します。次へ を選択します。



6. IP アドレス、NTP サーバー、または DNS 設定を手動で割り当てる必要がある場合は、「ネットワークの詳細設定」を選択します。それ以外の場合は、次へ を選択します。
7. アーカイブをダウンロード を選択します。次へ をクリックします。
8. 設定アーカイブを USB ドライブのルートディレクトリにコピーします。
9. USB ドライブを、アプライアンスの前面、HDMI ポートの横にある USB 3.0 ポートに接続します。

   USB ドライブを接続すると、アプライアンスは設定アーカイブとネットワーク設定ファイルをアプライアンス自身にコピーし、AWS クラウドに接続します。接続が完了すると、アプライアンスのステータスライトが緑色から青色に変わり、その後緑色に戻ります。
10. 続行するには、次へ を選択します。



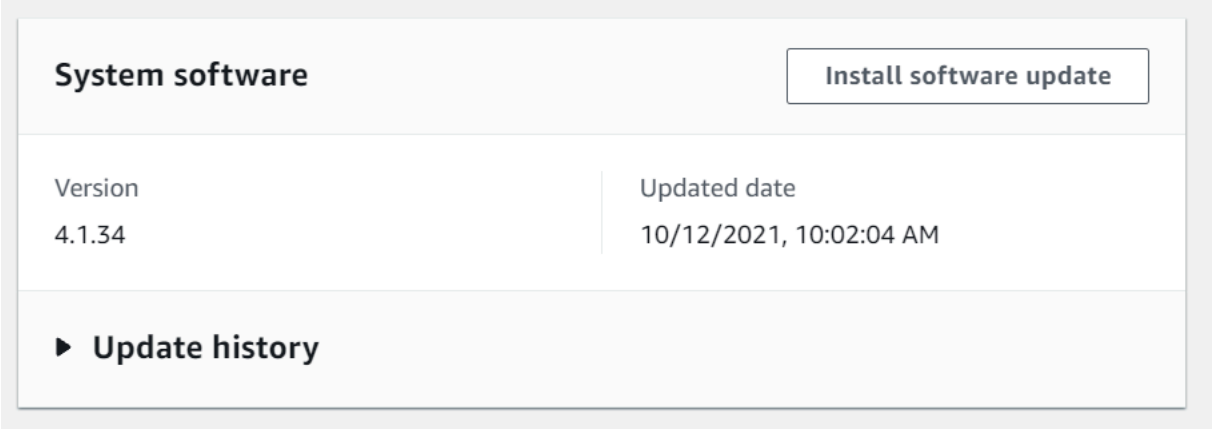
11. 完了 をクリックします。

アプライアンスソフトウェアをアップグレードする

AWS Panorama アプライアンスには、Linux オペレーティングシステム、AWS Panorama アプリケーション SDK、サポートするコンピュータビジョンライブラリとフレームワークなど、いくつかのソフトウェアコンポーネントがあります。アプライアンスで最新の機能やアプリケーションを確実に使用できるようにするには、セットアップ後、および更新が可能になるたびにソフトウェアをアップグレードしてください。

アプライアンスソフトウェアをアップグレードする

1. AWS Panorama コンソールの [デバイス] ページを開きます。
2. アプライアンスを選択します。
3. 設定を選択する。
4. [システムのソフトウェア] で [ソフトウェアのインストール] を選択します。



5. 新しいバージョンを選択し、[インストール] を選択します。

> **Important**
> 続行する前に、USB ドライブをアプライアンスから取り外し、フォーマットして内容を削除します。設定アーカイブには機密データが含まれており、自動的には削除されません。

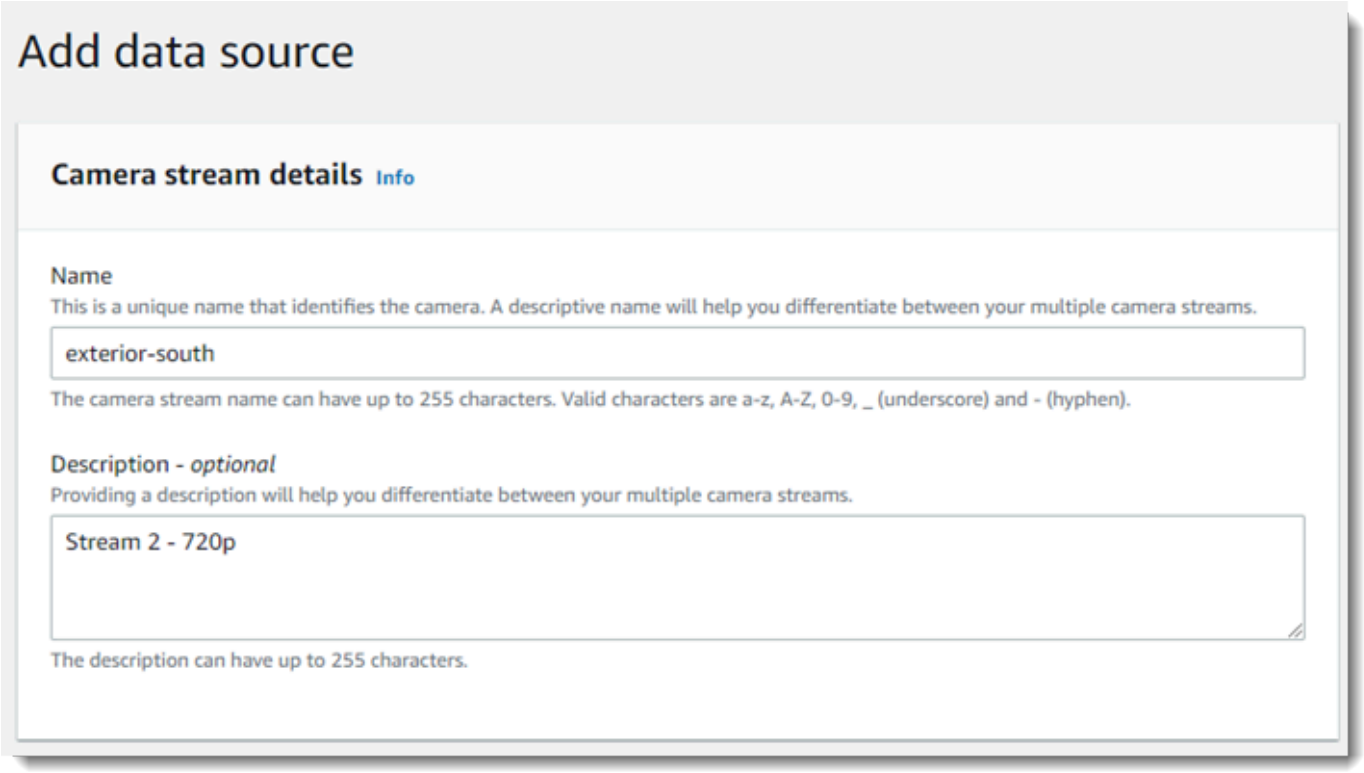
アップグレードのプロセスは完了までに 30 分以上かかる場合があります。その進行状況は、AWS Panorama コンソールまたは接続されたモニターでモニタリングできます。プロセスが完了すると、アプライアンスは再起動します。

カメラストリームを追加します

次に、カメラストリームを AWS Panorama コンソールに登録します。

カメラストリームを登録する

1. AWS Panorama コンソールの [データソース] ページを開きます。
2. データソースを追加する を選択する。



3. 以下を設定します。

   - 名前 — カメラストリームの名前。
   - 説明 — カメラ、その場所、その他の詳細に関する簡単な説明。
   - RTSP URL — カメラの IP アドレスとストリームへのパスを指定する URL。例えば、`rtsp://192.168.0.77/live/mpeg4/` などです。
   - 認証情報 — カメラストリームがパスワードで保護されている場合は、ユーザーネームとパスワードを指定します。
4. 保存を選択します。

AWS Panorama はカメラの認証情報を AWS Secrets Manager に安全に保管します。複数のアプリケーションが同じカメラストリームを同時に処理できます。

次のステップ

セットアップ中にエラーが発生した場合は、「トラブルシューティング」 を参照してください。

サンプルアプリケーションをデプロイするには、次のトピックに進んでください。

AWS Panorama サンプルアプリケーションのデプロイ

AWS Panorama Appliance または互換デバイスをセットアップし、ソフトウェアをアップグレードしたら、サンプルアプリケーションをデプロイします。以下のセクションでは、AWS Panorama アプリケーション CLI を使用してサンプルアプリケーションをインポートし、AWS Panorama コンソールでデプロイします。

サンプルアプリケーションでは、機械学習モデルを使用して、ネットワークカメラからのビデオフレーム内のオブジェクトを分類します。AWS Panorama アプリケーション SDK を使用してモデルの読み込み、画像の取得、モデルの実行を行います。次に、アプリケーションは結果を元の動画の上に重ねて表示し、接続されたディスプレイに出力します。

小売店では、人の往来パターンを分析することでトラフィック量を予測できます。この分析を他のデータと組み合わせることで、休日やその他のイベント前後の人員配置ニーズの増加を計画したり、広告や販売促進の効果を測定したり、ディスプレイの配置や在庫管理を最適化したりできます。

セクション

前提条件

このチュートリアルの手順に従うには、コマンドを実行するコマンドラインターミナルまたはシェルが必要になります。コード一覧では、コマンドは、該当する場合、プロンプト記号 ($) と現在のディレクトリの名前が前に付けられています。

```
~/panorama-project$ this is a command
this is output
```

コマンドが長い場合は、エスケープ文字 (\) を使用してコマンドを複数行に分割します。

Linux および macOS では、任意のシェルとパッケージマネージャーを使用します。Windows 10 では、Linux 用の Windows サブシステムをインストールして、Windows 統合バージョンの Ubuntu および Bash を入手できます。Windows での開発環境の設定方法については、Windows 開発環境でのセットアップ を参照してください。

Python を使用して AWS Panorama アプリケーションを開発し、Python のパッケージマネージャーである pip でツールをインストールします。まだ Python がインストールされていない場合は、最新バージョンをインストールしてください。Python 3 がありピップはない場合は、オペレーティングシステムのパッケージマネージャーでピップをインストールするか、ピップに付属の新しいバージョンの Python をインストールしてください。

このチュートリアルでは、ドッカーを使用してアプリケーションコードを実行するコンテナを構築します。Docker は、Docker ウェブサイトの「Docker の入手」からインストールします。

このチュートリアルでは、AWS Panorama アプリケーション CLI を使用してサンプルアプリケーションのインポート、パッケージのビルド、アーティファクトのアップロードを行います。AWS Panorama アプリケーション CLI は、AWS Command Line Interface (AWS CLI) を使用してサービス API オペレーションを呼び出します。AWS CLI をすでにインストールしている場合、最新バージョンにアップグレードします。AWS Panorama アプリケーション CLI およびAWS CLI をインストールするには、pip を使用してください。

```
$ pip3 install --upgrade awscli panoramacli
```

サンプルアプリケーションをダウンロードし、ワークスペースに抽出します。

- サンプルアプリケーション — aws-panorama-サンプル.zip

サンプルアプリケーションをインポートする

サンプルアプリケーションをインポートしてアカウントで使用するには、AWS Panorama アプリケーション CLI を使用します。アプリケーションのフォルダとマニフェストには、プレースホルダーのアカウント番号への参照が含まれています。これらをアカウント番号で更新するには、`panorama-cli import-application` コマンドを実行します。

```
aws-panorama-sample$ panorama-cli import-application
```

packages ディレクトリ内のSAMPLE_CODE パッケージには、アプリケーションのベースイメージ、`panorama-application` を使用するドッカーファイルなど、アプリケーションのコード

と構成が含まれています。アプライアンス上で動作するアプリケーションコンテナを構築するには、panorama-cli build-container コマンドを使用します。

```
aws-panorama-sample$ ACCOUNT_ID=$(aws sts get-caller-identity --output text --query
 'Account')
aws-panorama-sample$ panorama-cli build-container --container-asset-name code_asset --
package-path packages/${ACCOUNT_ID}-SAMPLE_CODE-1.0
```

AWS Panorama アプリケーション CLI の最後のステップは、アプリケーションのコードとモデルノードを登録し、サービスが提供する Amazon S3 アクセスポイントにアセットをアップロードすることです。アセットには、コードのコンテナイメージ、モデル、およびそれぞれの記述子ファイルが含まれます。ノードを登録してアセットをアップロードするには、panorama-cli package-application コマンドを実行します。

```
aws-panorama-sample$ panorama-cli package-application
Uploading package model
Registered model with patch version
 bc9c58bd6f83743f26aa347dc86bfc3dd2451b18f964a6de2cc4570cb6f891f9
Uploading package code
Registered code with patch version
 11fd7001cb31ea63df6aaed297d600a5ecf641a987044a0c273c78ceb3d5d806
```

アプリケーションのデプロイ

AWS Panorama コンソールを使用して、アプリケーションをアプライアンスにデプロイします。

アプリケーションをデプロイするには

1. AWS Panorama コンソールの [デプロイされたアプリケーション] ページを開きます。
2. [アプリケーションをデプロイ] を選択します。
3. アプリケーションマニフェスト、graphs/aws-panorama-sample/graph.json の内容をテキストエディタに貼り付けます。[Next] (次へ) をクリックします。
4. [アプリケーション名] にはaws-panorama-sampleと入力します。
5. [デプロイに進む] を選択します。
6. [デプロイを開始] を選択します。
7. ロールを選択せずに [次へ] を選択します。
8. [デバイスを選択]を選択し、次にアプライアンスを選択します。[Next] (次へ) をクリックします。

9. 「データソースを選択」 ステップで、「入力を表示」を選択し、カメラストリームをデータソースとして追加します。[Next] (次へ) をクリックします。
10. [設定)] ステップで、[次へ] を選択します。
11. [デプロイ] を選択してから、[完了] を選択します。
12. デプロイされたアプリケーションのリストで [aws-panorama-サンプル] を選択します。

このページを更新して更新を確認するか、次のスクリプトを使用してコマンドラインからデプロイを監視してください。

Example モニター-デプロイメント.sh

```
while true; do
  aws panorama list-application-instances --query 'ApplicationInstances[?Name==`aws-
panorama-sample`]'
  sleep 10
done
```

```
[
    {
        "Name": "aws-panorama-sample",
        "ApplicationInstanceId": "applicationInstance-x264exmpl33gq5pchc2ekoi6uu",
        "DefaultRuntimeContextDeviceName": "my-appliance",
        "Status": "DEPLOYMENT_PENDING",
        "HealthStatus": "NOT_AVAILABLE",
        "StatusDescription": "Deployment Workflow has been scheduled.",
        "CreatedTime": 1630010747.443,
        "Arn": "arn:aws:panorama:us-west-2:123456789012:applicationInstance/
applicationInstance-x264exmpl33gq5pchc2ekoi6uu",
        "Tags": {}
    }
]
[
    {
        "Name": "aws-panorama-sample",
        "ApplicationInstanceId": "applicationInstance-x264exmpl33gq5pchc2ekoi6uu",
        "DefaultRuntimeContextDeviceName": "my-appliance",
        "Status": "DEPLOYMENT_PENDING",
        "HealthStatus": "NOT_AVAILABLE",
        "StatusDescription": "Deployment Workflow has completed data validation.",
```

```
        "CreatedTime": 1630010747.443,
        "Arn": "arn:aws:panorama:us-west-2:123456789012:applicationInstance/
applicationInstance-x264exmpl33gq5pchc2ekoi6uu",
        "Tags": {}
    }
]
...
```

アプリケーションが起動しない場合は、Amazon CloudWatch Logs のアプリケーションログとデバイスログを確認してください。

出力の表示

デプロイが完了すると、アプリケーションはビデオストリームの処理を開始し、CloudWatch にログを送信します。

CloudWatch Logs でログを表示する

1. CloudWatch Logs コンソールの [[ロググループ] ページ] を開きます。
2. 次のグループで AWS Panorama アプリケーションログとアプライアンスログを検索します:

   - デバイスログ — /aws/panorama/devices/*device-id*
   - アプリケーションログ – /aws/panorama/devices/*device-id*/applications/*instance-id*

```
2022-08-26 17:43:39 INFO     INITIALIZING APPLICATION
2022-08-26 17:43:39 INFO     ## ENVIRONMENT VARIABLES
{'PATH': '/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin', 'TERM':
 'xterm', 'container': 'podman'...}
2022-08-26 17:43:39 INFO     Configuring parameters.
2022-08-26 17:43:39 INFO     Configuring AWS SDK for Python.
2022-08-26 17:43:39 INFO     Initialization complete.
2022-08-26 17:43:39 INFO     PROCESSING STREAMS
2022-08-26 17:46:19 INFO     epoch length: 160.183 s (0.936 FPS)
2022-08-26 17:46:19 INFO     avg inference time: 805.597 ms
2022-08-26 17:46:19 INFO     max inference time: 120023.984 ms
2022-08-26 17:46:19 INFO     avg frame processing time: 1065.129 ms
2022-08-26 17:46:19 INFO     max frame processing time: 149813.972 ms
2022-08-26 17:46:29 INFO     epoch length: 10.562 s (14.202 FPS)
2022-08-26 17:46:29 INFO     avg inference time: 7.185 ms
```

```
2022-08-26 17:46:29 INFO     max inference time: 15.693 ms
2022-08-26 17:46:29 INFO     avg frame processing time: 66.561 ms
2022-08-26 17:46:29 INFO     max frame processing time: 123.774 ms
```

アプリケーションのビデオ出力を表示するには、HDMI ケーブルでアプライアンスをモニターに接続します。デフォルトでは、信頼度が 20% を超える分類結果がアプリケーションに表示されます。

Example スクイーズネット_クラス.json

```
["tench", "goldfish", "great white shark", "tiger shark",
"hammerhead", "electric ray", "stingray", "cock", "hen", "ostrich",
"brambling", "goldfinch", "house finch", "junco", "indigo bunting",
"robin", "bulbul", "jay", "magpie", "chickadee", "water ouzel",
"kite", "bald eagle", "vulture", "great grey owl",
"European fire salamander", "common newt", "eft",
"spotted salamander", "axolotl", "bullfrog", "tree frog",
...
```

サンプルモデルには、多くの動物、食べ物、一般的なオブジェクトを含む 1000 のクラスがあります。キーボードやコーヒーマグにカメラを向けてみてください。

簡単にするために、サンプルアプリケーションでは軽量の分類モデルを使用しています。このモデルは、各クラスの確率を含む 1 つの配列を出力します。実際のアプリケーションでは、多次元出力のオブジェクト検出モデルがより頻繁に使用されます。より複雑なモデルを使用するサンプルアプリケーションについては、サンプルアプリケーション、スクリプト、テンプレート を参照してください。

SDK for Python を有効にする

サンプルアプリケーションは AWS SDK for Python (Boto) を使用して Amazon CloudWatch にメトリクスをアップロードします。この機能を有効にするには、メトリックスを送信する権限をアプリケーションに付与するロールを作成し、そのロールをアタッチした状態でアプリケーションを再デプロイします。

サンプルアプリケーションには、必要な権限を持つロールを作成する AWS CloudFormation テンプレートが含まれています。ロールを作成するには、`aws cloudformation deploy` コマンドを使用します。

```
$ aws cloudformation deploy --template-file aws-panorama-sample.yml --stack-name aws-panorama-sample-runtime --capabilities CAPABILITY_NAMED_IAM
```

アプリケーションを再デプロイするには

1. AWS Panorama コンソールの [デプロイされたアプリケーション] ページを開きます。
2. アプリケーションを選択します。
3. [Replace (置換)] を選択します。
4. アプリケーションを実行するには、手順に従います。[IAM ロールを指定] で、作成したロールを選択します。名前が `aws-panorama-sample-runtime` で始まるものです。
5. デプロイが完了したら、CloudWatch コンソールを開き、`AWSPanoramaApplication` 名前空間のメトリックスを表示します。150 フレームごとに、アプリケーションはフレーム処理と推論時間のメトリクスを記録してアップロードします。

クリーンアップ

サンプルアプリケーションの操作が終了したら、AWS Panorama コンソールを使用してアプライアンスから削除できます。

アプライアンスからアプリケーションを削除するには

1. AWS Panorama コンソールの [デプロイされたアプリケーション] ページを開きます。
2. アプリケーションを選択します。
3. [デバイスから削除] を選択します。

次のステップ

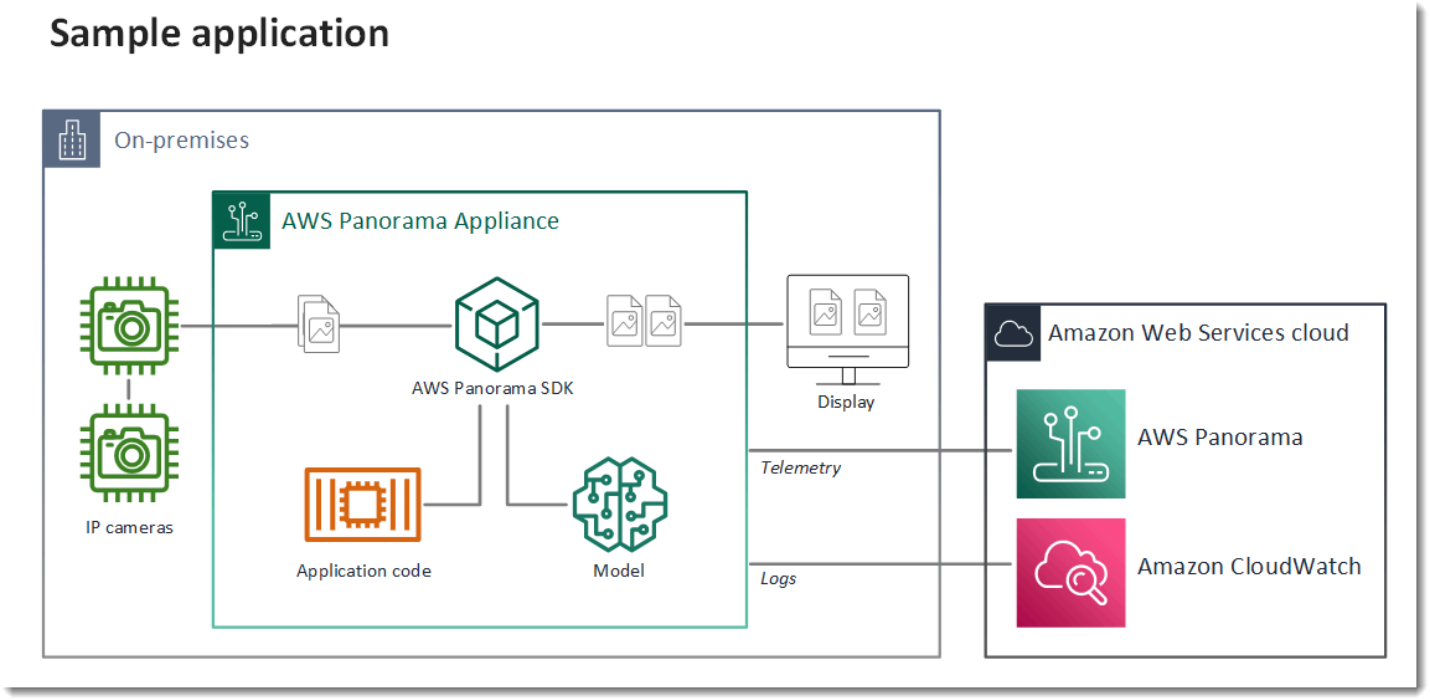
サンプルアプリケーションのデプロイまたは実行中にエラーが発生した場合は、トラブルシューティング を参照してください。

サンプルアプリケーションの機能と実装について詳しくは、次のトピックに進んでください。

AWS Panorama アプリケーションの開発

サンプルアプリケーションを使用して、AWS Panorama アプリケーションの構造について学んだり、独自のアプリケーションの開始点にしたりできます。

次の図表に、AWS Panorama アプライアンスで実行されるアプリケーションの主要なコンポーネントを示します。アプリケーションコードは AWS Panorama アプリケーションSDK を使用して画像を取得し、直接アクセスできないモデルとやり取りします。アプリケーションは接続されたディスプレイに動画を出力しますが、ローカルネットワークの外部には画像データを送信しません。



この例では、アプリケーションは AWS Panorama アプリケーション SDK を使用してカメラからビデオフレームを取得し、ビデオデータを前処理して、オブジェクトを検出するコンピュータビジョンモデルにデータを送信します。アプリケーションは、アプライアンスに接続された HDMI ディスプレイに結果を表示します。

セクション

- アプリケーションマニフェスト。
- サンプルアプリケーションでのビルド
- コンピュータービジョンモデルの変更
- 画像の前処理
- Python 用 SDK によるメトリクスのアップロード

- 次のステップ

アプリケーションマニフェスト。

アプリケーションマニフェストは、graphs フォルダー内の graph.json という名前のファイルです。マニフェストは、パッケージ、ノード、エッジなどのアプリケーションのコンポーネントを定義します。

パッケージは、アプリケーションコード、モデル、カメラ、ディスプレイのコード、構成、バイナリファイルです。サンプルアプリケーションは 4 つのパッケージを使用します。

Example **graphs/aws-panorama-sample/graph.json** - パッケージ

```
        "packages": [
            {
                "name": "123456789012::SAMPLE_CODE",
                "version": "1.0"
            },
            {
                "name": "123456789012::SQUEEZENET_PYTORCH_V1",
                "version": "1.0"
            },
            {
                "name": "panorama::abstract_rtsp_media_source",
                "version": "1.0"
            },
            {
                "name": "panorama::hdmi_data_sink",
                "version": "1.0"
            }
        ],
```

最初の 2 つのパッケージは、アプリケーション内の packages ディレクトリで定義されています。これらには、このアプリケーション固有のコードとモデルが含まれています。次の 2 つのパッケージは、AWS Panorama サービスが提供する汎用のカメラおよびディスプレイパッケージです。abstract_rtsp_media_source パッケージは、デプロイ時にオーバーライドするカメラのプレースホルダーです。hdmi_data_sink パッケージはデバイスの HDMI 出力コネクタを表します。

ノードはパッケージへのインターフェースであると同時に、デプロイ時にオーバーライドできるデフォルト値を持つ非パッケージパラメーターのインターフェースでもあります。コードパッケージと

モデルパッケージは、ビデオストリームでも、フロート、ブーリアン、文字列などの基本データ型でも、入力と出力を指定する package.json ファイル内のインターフェイスを定義します。

たとえば、code_node ノードは SAMPLE_CODE パッケージのインターフェースを参照します。

```
        "nodes": [
            {
                "name": "code_node",
                "interface": "123456789012::SAMPLE_CODE.interface",
                "overridable": false,
                "launch": "onAppStart"
            },
```

このインターフェースはパッケージ構成ファイル、package.json 内で定義されています。このインターフェースは、パッケージがビジネスロジックであり、video_in という名前のビデオストリームと thresholdという名前の浮動小数点数を入力として受け取ることを指定しています。また、このインターフェースでは、ビデオをディスプレイに出力するための video_out という名前のビデオストリームバッファがコードに必要であることも指定されています。

Example **packages/123456789012-SAMPLE_CODE-1.0/package.json**

```
{
    "nodePackage": {
        "envelopeVersion": "2021-01-01",
        "name": "SAMPLE_CODE",
        "version": "1.0",
        "description": "Computer vision application code.",
        "assets": [],
        "interfaces": [
            {
                "name": "interface",
                "category": "business_logic",
                "asset": "code_asset",
                "inputs": [
                    {
                        "name": "video_in",
                        "type": "media"
                    },
                    {
                        "name": "threshold",
                        "type": "float32"
                    }
```

```
                ],
                "outputs": [
                    {
                        "description": "Video stream output",
                        "name": "video_out",
                        "type": "media"
                    }
                ]
            }
        ]
    }
}
```

アプリケーションマニフェストに戻ると、camera_node ノードはカメラからのビデオストリームを表しています。これには、アプリケーションをデプロイするとコンソールに表示され、カメラストリームの選択を促すデコレータが含まれています。

Example **graphs/aws-panorama-sample/graph.json** – Camera ノード

```
            {
                "name": "camera_node",
                "interface": "panorama::abstract_rtsp_media_source.rtsp_v1_interface",
                "overridable": true,
                "launch": "onAppStart",
                "decorator": {
                    "title": "Camera",
                    "description": "Choose a camera stream."
                }
            },
```

パラメータノード threshold_param は、アプリケーションコードで使用される信頼度しきい値パラメータを定義します。デフォルト値は 60 で、デプロイ時に上書きできます。

Example **graphs/aws-panorama-sample/graph.json** — パラメータノード

```
            {
                "name": "threshold_param",
                "interface": "float32",
                "value": 60.0,
                "overridable": true,
                "decorator": {
                    "title": "Confidence threshold",
```

```
                    "description": "The minimum confidence for a classification to be
recorded."
                }
            }
```

アプリケーションマニフェストの最後のセクション edges では、ノード間の接続を行います。カメラのビデオストリームとしきい値パラメータはコードノードの入力に接続し、コードノードからのビデオ出力はディスプレイに接続します。

Example **graphs/aws-panorama-sample/graph.json** - エッジ

```
        "edges": [
            {
                "producer": "camera_node.video_out",
                "consumer": "code_node.video_in"
            },
            {
                "producer": "code_node.video_out",
                "consumer": "output_node.video_in"
            },
            {
                "producer": "threshold_param",
                "consumer": "code_node.threshold"
            }
        ]
```

サンプルアプリケーションでのビルド

サンプルコードを開始点として使用して、独自のアプリケーションを作成することができます。

各パッケージ名は一意にする必要があります。自分とアカウント内の別のユーザーの両方が code やmodel などの汎用パッケージ名を使用している場合、デプロイ時に間違ったバージョンのパッケージが表示される可能性があります。コードパッケージの名前を、自分のアプリケーションを表す名前に変更してください。

コードパッケージの名前を変更するには

1. パッケージフォルダーの名前を変更します:packages/123456789012-*SAMPLE_CODE*-1.0/。
2. 次の場所でパッケージ名を更新します。

- アプリケーションマニフェスト — graphs/aws-panorama-sample/graph.json
- パッケージ構成 – packages/123456789012-SAMPLE_CODE-1.0/package.json
- ビルドスクリプト — 3-build-container.sh

アプリケーションコードを更新するには

1. packages/123456789012-SAMPLE_CODE-1.0/src/application.py でアプリケーションコードを変更します。
2. コンテナを構築するには、3-build-container.sh を実行します。

```
aws-panorama-sample$ ./3-build-container.sh
TMPDIR=$(pwd) docker build -t code_asset packages/123456789012-SAMPLE_CODE-1.0
Sending build context to Docker daemon  61.44kB
Step 1/2 : FROM public.ecr.aws/panorama/panorama-application
 ---> 9b197f256b48
Step 2/2 : COPY src /panorama
 ---> 55c35755e9d2
Successfully built 55c35755e9d2
Successfully tagged code_asset:latest
docker export --output=code_asset.tar $(docker create code_asset:latest)
gzip -9 code_asset.tar
Updating an existing asset with the same name
{
    "name": "code_asset",
    "implementations": [
        {
            "type": "container",
            "assetUri":
 "98aaxmpl1c1ef64cde5ac13bd3be5394e5d17064beccee963b4095d83083c343.tar.gz",
            "descriptorUri":
 "1872xmpl129481ed053c52e66d6af8b030f9eb69b1168a29012f01c7034d7a8f.json"
        }
    ]
}
Container asset for the package has been succesfully built at  ~/aws-panorama-
sample-dev/
assets/98aaxmpl1c1ef64cde5ac13bd3be5394e5d17064beccee963b4095d83083c343.tar.gz
```

CLI は古いコンテナアセットを assets フォルダから自動的に削除し、パッケージ構成を更新します。

3. パッケージをアップロードするには、4-package-application.py を実行します。
4. AWS Panorama コンソールの [デプロイされたアプリケーション] ページを開きます。
5. アプリケーションを選択します。
6. [Replace (置換)] を選択します。
7. アプリケーションを実行するには、手順に従います。必要に応じて、アプリケーションマニフェスト、カメラストリーム、またはパラメータを変更できます。

コンピュータービジョンモデルの変更

サンプルアプリケーションにはコンピュータビジョンモデルが含まれています。独自のモデルを使用するには、モデルノードの構成を変更し、AWS Panorama アプリケーション CLI を使用してアセットとしてインポートします。

次の例では MXNet SSD ResNet50 モデルを使用しています。このモデルは、このガイドの GitHub リポジトリからダウンロードできます。ssd_512_resnet50_v1_voc.tar.gz

サンプルアプリケーションのモデルを変更するには

1. モデルに合うようにパッケージフォルダーの名前を変更します。たとえば、packages/*123456789012*-*SSD_512_RESNET50_V1_VOC*-1.0/ などにします。
2. 次の場所でパッケージ名を更新します。

   - アプリケーションマニフェスト — graphs/aws-panorama-sample/graph.json
   - パッケージ構成 – packages/*123456789012*-*SSD_512_RESNET50_V1_VOC*-1.0/package.json
3. パッケージ構成ファイル (package.json) 内。assets 値を空白の配列に変更します。

```
{
    "nodePackage": {
        "envelopeVersion": "2021-01-01",
        "name": "SSD_512_RESNET50_V1_VOC",
        "version": "1.0",
        "description": "Compact classification model",
```

```
        "assets": [],
```

4. パッケージ記述ファイル (descriptor.json) を開きます。モデルと一致するようにframework およびshape の値を更新します。

```
{
    "mlModelDescriptor": {
        "envelopeVersion": "2021-01-01",
        "framework": "MXNET",
        "inputs": [
            {
                "name": "data",
                "shape": [ 1, 3, 512, 512 ]
            }
        ]
    }
}
```

シェイプ の値は 1,3,512,512、モデルが入力として受け取る画像の数 (1)、各画像のチャネル数 (3— 赤、緑、青)、および画像のサイズ (512 x 512) を示します。配列の値と順序はモデルによって異なります。

5. AWS Panorama アプリケーション CLI を使用してモデルをインポートします。AWS Panorama アプリケーション CLI は、モデルファイルと記述子ファイルを一意の名前で assets フォルダにコピーし、パッケージ構成を更新します。

```
aws-panorama-sample$ panorama-cli add-raw-model --model-asset-name model-asset \
--model-local-path ssd_512_resnet50_v1_voc.tar.gz \
--descriptor-path packages/123456789012-SSD_512_RESNET50_V1_VOC-1.0/descriptor.json
 \
--packages-path packages/123456789012-SSD_512_RESNET50_V1_VOC-1.0
{
    "name": "model-asset",
    "implementations": [
        {
            "type": "model",
            "assetUri":
 "b1a1589afe449b346ff47375c284a1998c3e1522b418a7be8910414911784ce1.tar.gz",
            "descriptorUri":
 "a6a9508953f393f182f05f8beaa86b83325f4a535a5928580273e7fe26f79e78.json"
        }
    ]
```

```
}
```

6. モデルをアップロードするには、panorama-cli package-application を実行します。

```
$ panorama-cli package-application
Uploading package SAMPLE_CODE
Patch Version 1844d5a59150d33f6054b04bac527a1771fd2365e05f990ccd8444a5ab775809
 already registered, ignoring upload
Uploading package SSD_512_RESNET50_V1_VOC
Patch version for the package
 244a63c74d01e082ad012ebf21e67eef5d81ce0de4d6ad1ae2b69d0bc498c8fd
upload: assets/
b1a1589afe449b346ff47375c284a1998c3e1522b418a7be8910414911784ce1.tar.gz to
 s3://arn:aws:s3:us-west-2:454554846382:accesspoint/panorama-123456789012-
wc66m5eishf4si4sz5jefhx
63a/123456789012/nodePackages/SSD_512_RESNET50_V1_VOC/binaries/
b1a1589afe449b346ff47375c284a1998c3e1522b418a7be8910414911784ce1.tar.gz
upload: assets/
a6a9508953f393f182f05f8beaa86b83325f4a535a5928580273e7fe26f79e78.json to
 s3://arn:aws:s3:us-west-2:454554846382:accesspoint/panorama-123456789012-
wc66m5eishf4si4sz5jefhx63
a/123456789012/nodePackages/SSD_512_RESNET50_V1_VOC/binaries/
a6a9508953f393f182f05f8beaa86b83325f4a535a5928580273e7fe26f79e78.json
{
    "ETag": "\"2381dabba34f4bc0100c478e67e9ab5e\"",
    "ServerSideEncryption": "AES256",
    "VersionId": "KbY5fpESdpYamjWZ0YyGqHo3.LQQWUC2"
}
Registered SSD_512_RESNET50_V1_VOC with patch version
 244a63c74d01e082ad012ebf21e67eef5d81ce0de4d6ad1ae2b69d0bc498c8fd
Uploading package SQUEEZENET_PYTORCH_V1
Patch Version 568138c430e0345061bb36f05a04a1458ac834cd6f93bf18fdacdffb62685530
 already registered, ignoring upload
```

7. アプリケーションコードを更新します。ほとんどのコードは再利用できます。モデルのレスポンスに固有のコードは。process_results メソッド内にあります。

```
    def process_results(self, inference_results, stream):
        """Processes output tensors from a computer vision model and annotates a
 video frame."""
        for class_tuple in inference_results:
            indexes = self.topk(class_tuple[0])
        for j in range(2):
```

```
            label = 'Class [%s], with probability %.3f.'%
 (self.classes[indexes[j]], class_tuple[0][indexes[j]])
            stream.add_label(label, 0.1, 0.25 + 0.1*j)
```

モデルに応じて、preprocess メソッドを更新する必要があることもあります。

画像の前処理

アプリケーションがイメージをモデルに送信する前に、イメージのサイズを変更してカラーデータを正規化することで、イメージを推論できるように準備します。アプリケーションが使用するモデルには、最初のレイヤーの入力数に合わせて、3 つのカラーチャンネルを含む 224 x 224 ピクセルの画像が必要です。アプリケーションは、各色の値を 0 から 1 の間の数値に変換し、その色の平均値を引いて標準偏差で割ることによって調整します。最後に、カラーチャンネルを結合し、モデルが処理できるNumPy 配列に変換します。

Example アプリケーション.py — 前処理

```
    def preprocess(self, img, width):
        resized = cv2.resize(img, (width, width))
        mean = [0.485, 0.456, 0.406]
        std = [0.229, 0.224, 0.225]
        img = resized.astype(np.float32) / 255.
        img_a = img[:, :, 0]
        img_b = img[:, :, 1]
        img_c = img[:, :, 2]
        # Normalize data in each channel
        img_a = (img_a - mean[0]) / std[0]
        img_b = (img_b - mean[1]) / std[1]
        img_c = (img_c - mean[2]) / std[2]
        # Put the channels back together
        x1 = [[[], [], []]]
        x1[0][0] = img_a
        x1[0][1] = img_b
        x1[0][2] = img_c
        return np.asarray(x1)
```

このプロセスにより、0 を中心とした予測可能な範囲内のモデル値が得られます。これはトレーニングデータセット内の画像に適用される前処理と一致します。これは標準的な方法ですが、モデルごとに異なる場合があります。

Python 用 SDK によるメトリクスのアップロード

サンプルアプリケーションは Python 用 SDK を使用して Amazon CloudWatch にメトリクスをアップロードします。

Example アプリケーション.py — Python 用 SDK

```
    def process_streams(self):
        """Processes one frame of video from one or more video streams."""
        ...
            logger.info('epoch length: {:.3f} s ({:.3f} FPS)'.format(epoch_time,
epoch_fps))
            logger.info('avg inference time: {:.3f} ms'.format(avg_inference_time))
            logger.info('max inference time: {:.3f} ms'.format(max_inference_time))
            logger.info('avg frame processing time: {:.3f}
ms'.format(avg_frame_processing_time))
            logger.info('max frame processing time: {:.3f}
ms'.format(max_frame_processing_time))
            self.inference_time_ms = 0
            self.inference_time_max = 0
            self.frame_time_ms = 0
            self.frame_time_max = 0
            self.epoch_start = time.time()
            self.put_metric_data('AverageInferenceTime', avg_inference_time)
            self.put_metric_data('AverageFrameProcessingTime',
avg_frame_processing_time)

    def put_metric_data(self, metric_name, metric_value):
        """Sends a performance metric to CloudWatch."""
        namespace = 'AWSPanoramaApplication'
        dimension_name = 'Application Name'
        dimension_value = 'aws-panorama-sample'
        try:
            metric = self.cloudwatch.Metric(namespace, metric_name)
            metric.put_data(
                Namespace=namespace,
                MetricData=[{
                    'MetricName': metric_name,
                    'Value': metric_value,
                    'Unit': 'Milliseconds',
                    'Dimensions': [
                        {
                            'Name': dimension_name,
```

```
                            'Value': dimension_value
                        },
                        {
                            'Name': 'Device ID',
                            'Value': self.device_id
                        }
                    ]
                }]
            )
            logger.info("Put data for metric %s.%s", namespace, metric_name)
        except ClientError:
            logger.warning("Couldn't put data for metric %s.%s", namespace,
metric_name)
        except AttributeError:
            logger.warning("CloudWatch client is not available.")
```

デプロイ時に割り当てたランタイムロールから権限を取得します。ロールはaws-panorama-sample.yml AWS CloudFormation テンプレートで定義されます。

Example aws-panorama-サンプル.yml

```
Resources:
  runtimeRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - panorama.amazonaws.com
            Action:
              - sts:AssumeRole
      Policies:
        - PolicyName: cloudwatch-putmetrics
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              - Effect: Allow
                Action: 'cloudwatch:PutMetricData'
                Resource: '*'
```

```
      Path: /service-role/
```

サンプルアプリケーションは、Python 用の SDK とその他の依存関係をピップでインストールします。アプリケーションコンテナをビルドすると、Dockerfile はコマンドを実行して、ベースイメージに付属しているものの上にライブラリをインストールします。

Example Dockerfile

```
FROM public.ecr.aws/panorama/panorama-application
WORKDIR /panorama
COPY . .
RUN pip install --no-cache-dir --upgrade pip && \
    pip install --no-cache-dir -r requirements.txt
```

AWS SDK をアプリケーションコードで使用するには、まずテンプレートを変更して、アプリケーションが使用するすべての API アクションにアクセス権限を追加します。変更を加えるたびに 1-create-role.sh を実行してAWS CloudFormation スタックを更新します。次に、変更内容をアプリケーションコードにデプロイします。

既存のリソースを変更または使用するアクションでは、ターゲット Resource の名前またはパターンを別のステートメントで指定して、このポリシーの範囲を最小限に抑えるのがベストプラクティスです。各サービスでサポートされているアクションとリソースの詳細については、サービス承認リファレンスの「アクション、リソース、条件キー」を参照してください。

次のステップ

AWS Panorama アプリケーション CLI を使用してアプリケーションをビルドし、パッケージを最初から作成する手順については、CLI の README を参照してください。

- github.com/aws/aws-panorama-cli

デプロイ前にアプリケーションコードを検証するために使用できるその他のサンプルコードとテストユーティリティについては、AWS Panorama サンプルリポジトリをご覧ください。

- github.com/aws-samples/aws-panorama-samples

サポートされているコンピュータービジョンモデルとカメラ

AWS Panorama は PyTorch、Apache MXNet、TensorFlow で構築されたモデルをサポートしています。アプリケーションをデプロイすると、AWS Panorama はモデルを SageMaker Neo でコンパイルします。SageMaker Neo と互換性のあるレイヤーを使用している限り、Amazon SageMaker またはお使いの開発環境でモデルを構築できます。

動画を処理して画像を取得してモデルに送信するには、AWS Panorama アプライアンスは RTSP プロトコルを使用して H.264 でエンコードされたビデオストリームに接続します。AWS Panorama は、さまざまな一般的なカメラの互換性をテストします。

セクション

サポートされているモデル

AWS Panorama 用のアプリケーションを構築するときは、アプリケーションがコンピュータビジョンに使用する機械学習モデルを提供します。モデルフレームワークによって提供される、事前構築済みでトレーニング済みのモデル、サンプルモデル、または自分で構築してトレーニングしたモデルを使用できます。

> **Note**
>
> AWS Panorama でテストされたビルド済みモデルのリストについては、「モデルの互換性」を参照してください。

アプリケーションをデプロイすると、AWS Panorama は SageMaker Neo コンパイラーを使ってコンピュータービジョンモデルをコンパイルします。SageMaker Neo は、Amazon Elastic Compute Cloud (Amazon EC2) のインスタンスでも、AWS Panorama アプライアンスなどのエッジデバイスでも、ターゲットプラットフォームで効率的に実行されるようにモデルを最適化するコンパイラーです。

AWS Panorama は、SageMaker Neo がエッジデバイスでサポートしている PyTorch、Apache MXNet、TensorFlow のバージョンをサポートしています。独自のモデルを構築する場合、SageMaker Neo リリースノートに記載されているフレームワークバージョンを使用できます。SageMaker では、組み込み型の画像分類アルゴリズムを使用できます。

AWS Panorama でのモデルの使用の詳細については、「コンピュータービジョンモデル」を参照してください。

サポート対象カメラ

AWS Panorama アプライアンスは、ローカルネットワーク経由で RTSP を出力するカメラからの H.264 ビデオストリームをサポートします。2 メガピクセルを超えるカメラストリームの場合、アプライアンスは画像を 1920x1080 ピクセル、またはストリームの縦横比を維持する同等のサイズに縮小します。

以下のカメラモデルは、AWS Panorama アプライアンスとの互換性がテストされています。

- Axis — M3057-PLVE、M3058-PLVE、P1448-LE、P3225-LV Mk II
- LaVIEW — LV-PB3040W
- Vivotek – IB9360-H
- Amcrest – IP2M-841B
- Anpviz — IPC-B850W-S-3X、IPC-D250W-S
- WGCC – Dome PoE 4MP ONVIF

アプライアンスのハードウェア仕様については、「AWS Panorama アプライアンスの仕様」を参照してください。

AWS Panorama アプライアンスの仕様

AWS Panorama アプライアンスのハードウェア仕様は次のとおりです。その他の互換性のあるデバイスについては、製造元の説明書をご参照ください。

コンポーネント	仕様
プロセッサとGPU	Nvidia Jetson AGX Xavier (32 GB RAM 搭載)
イーサネット	1000 Base-T (ギガバイト) x 2
USB	USB 2.0 x 1 と USB 3.0 タイプ A メス x 1
HDMI 出力	2.0a
ディメンション	7.75インチ x 9.6インチ x 1.6インチ (197mm x 243mm x 40mm)
重量	3.7 ポンド (1.7 kg)
電源	100V-240V 50-60Hz AC 65W
電源入力	IEC 60320 C6 (3 ピン) レセプタクル
ほこりや液体からの保護	IP-62
EMI/EMC 規制コンプライアンス	FCC Part-15 (米国)
サーマル・タッチ・リミット	IEC-62368
動作温度	-20°C ~ 60°C
動作時湿度	相対湿度（RH）0% から 95%
保管温度	-20°C ~ 85°C
保管時の湿度	低温では制御されません。高温では 相対湿度（RH）90%
冷却	強制空気熱抽出 (ファン)

コンポーネント	仕様
取り付けオプション	ラックマウントまたはフリースタンディング
電源コード	6 フィート (1.8 メートル)
電源管理	プッシュボタン
リセット	モーメンタリスイッチ
ステータス LED とネットワーク LED	プログラム可能な 3 色 RGB LED

Wi-Fi、Bluetooth、SD カードストレージはアプライアンスに搭載されていますが、使用できません。

AWS Panorama アプライアンスには、サーバーラックに取り付けるためのネジが 2 本付属されています。2 つのアプライアンスを並べて 19 インチラックに取り付けることができます。

Service Quotas

AWS Panorama は、アカウントで作成したリソースとデプロイするアプリケーションにクォータを適用します。AWS Panorama を複数のAWS リージョンで使用する場合、クォータは各リージョンに個別に適用されます。AWS Panorama クォータは調整できません。

AWS Panorama のリソースには、デバイス、アプリケーションノードパッケージ、アプリケーションインスタンスが含まれます。

- デバイス — 1 リージョンあたり最大 50 件の登録済みアプライアンス。
- ノードパッケージ — 1 リージョンあたり 50 パッケージ、1 パッケージあたり最大 20 バージョン。
- アプリケーションインスタンス — デバイスあたり最大 10 個のアプリケーション。各アプリケーションは最大 8 つのカメラストリームを監視できます。各デバイスのデプロイは 1 日あたり 200 件に制限されています。

AWS Panorama アプリケーション CLI、AWS Command Line Interface または AWS SDK を AWS Panorama サービスで使用する場合、実行する API 呼び出しの数にクォータが適用されます。1 秒あたり合計 5 回までリクエストできます。リソースを作成または変更する API オペレーションの一部には、1 秒あたり 1 リクエストという追加の制限が適用されます。

クォータの全リストについては、Service Quotas コンソールにアクセスするか、Amazon Web Services 全般のリファレンス のAWS Panorama エンドポイントとクォータを参照してください。

AWS Panorama アクセス権限

アプライアンスやアプリケーションなどの AWS Panorama サービスやリソース へのアクセスを管理するには、AWS Identity and Access Management (IAM) を使用します。AWS Panorama を使用するアカウントのユーザーについては、IAM ロールに適用できるアクセス権限ポリシーでアクセス権限を管理します。アプリケーションの権限を管理するには、ロールを作成してそのロールをアプリケーションに割り当てます。

アカウント内のユーザーの権限を管理するには、AWS Panorama 提供されているマネージドポリシーを使用するか、独自のポリシーを作成します。アプリケーションとアプライアンスのログを取得、メトリックの表示、アプリケーションへのロールの割り当てを行うには、他の AWS サービスに対する権限が必要です。

AWS Panorama アプライアンスには、AWS サービスやリソースにアクセスするための権限を付与するロールもあります。アプライアンスのロールは、AWS Panorama サービスがユーザーに代わって他のサービスにアクセスするために使用するサービスロールの1つです。

アプリケーションロールは、アプリケーション用に作成する独立したサービスロールであり、AWS SDK for Python (Boto) でAWS サービスを使用する権限をアプリケーションに付与します。アプリケーションロールを作成するには、管理者権限または管理者の助けが必要です。

ユーザーのアクセス許可は、アクションが影響を及ぼすリソース (場合によっては追加条件) を使用して制限することができます。例えば、ユーザーが作成するアプリケーションの名前にユーザー名を含めることを要求するアプリケーションの Amazon リソースネーム (ARN) のパターンを指定することができます。各アクションでサポートされているリソースと条件については、サービス承認リファレンスのアクション、リソース、および条件キー AWS Panoramaを参照してください。

詳細については、IAM ユーザーガイドのIAM とはを参照してください。

トピック

- AWS Panorama のアイデンティティベースの IAM ポリシー
- AWS Panorama サービスロールとクロスサービスリソース
- アプリケーションへの権限の付与

AWS Panorama のアイデンティティベースの IAM ポリシー

AWS Panorama へのアクセス権をアカウントのユーザーに付与するには、AWS Identity and Access Management (IAM) のアイデンティティベースのポリシーを使用します。ユーザーに関連付けられている IAM ロールにアイデンティティベースのポリシーを適用する。または、アカウントのロールを引き受け、AWS Panorama リソースにアクセスするためのアクセス権限を別のアカウントのユーザーに付与することもできます。

AWS Panorama は、AWS Panorama API アクションへのアクセスを許可し、場合によっては、AWS Panorama リソースの開発と管理に使用される他のサービスへのアクセスを許可する、マネージドポリシーを提供します。AWS Panorama は、新機能がリリースされたときにユーザーがそれらにアクセスできるよう、これらのマネージドポリシーを必要に応じて更新します。

- AWSPanoramaFullAccess — AWS Panorama、Amazon S3 の AWS Panorama アクセスポイント、AWS Secrets Manager のアプライアンス認証情報、Amazon CloudWatch のアプライアンスログへのフルアクセスを提供します。AWS Panorama のサービスにリンクされたロールを作成する権限が含まれます。ポリシーを表示

この `AWSPanoramaFullAccess` ポリシーでは、AWS Panorama リソースにタグを付けることはできますが、AWS Panorama コンソールで使用されるタグ関連の権限がすべて含まれているわけではありません。これらの権限を付与するには、次のポリシーを追加します。

- ResourceGroupsandTagEditorFullAccess – ポリシーを表示する

この `AWSPanoramaFullAccess` ポリシーには、AWS Panorama コンソールからデバイスを購入する権限は含まれていません。これらの権限を付与するには、次のポリシーを追加します。

- ElementalAppliancesSoftwareFullAccess - ポリシーを表示する

管理されたポリシーは、ユーザーが変更できるリソースを制限することなく、APIアクションに権限を付与します。きめ細かな制御では、ユーザーのアクセス許可の範囲を制限する独自のポリシーを作成することができます。フルアクセスポリシーをポリシーの出発点として使用してください。

> **サービスロールの作成**
>
> AWS Panorama コンソールを初めて使用するときは、AWS Panorama アプライアンスで使用されるサービスロールを作成する権限が必要です。サービスロールは、リソースを管理

したり、他のサービスとやり取りしたりする権限をサービスに付与します。このロールは、ユーザーにアクセス権を付与する前に作成してください。

AWS Panorama でユーザーの権限の範囲を制限するために使用できるリソースと条件の詳細については、「サービス認証リファレンス」の AWS Panorama 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。

AWS Panorama サービスロールとクロスサービスリソース

AWS Panorama は、他の AWS サービスを使用して AWS Panorama アプライアンスの管理、データの保存、アプリケーションリソースのインポートを行います。サービスロールは、リソースを管理したり、他のサービスとやり取りしたりする権限をサービスに付与します。AWS Panorama コンソールに初めてサインインするには、次のサービスロールを使用します。

- AWSServiceRoleForAWSPanorama — AWS Panorama が AWS IoT、AWS Secrets Manager、および AWS Panorama のリソースを管理できるようにします。

  マネージドポリシー: Panoramaサービスにリンクされたロールポリシー
- AWSPanoramaApplianceServiceRole — AWS Panorama アプライアンスが CloudWatch にログをアップロードし、AWS Panorama によって作成された Amazon S3 Access Points からオブジェクトを取得することを許可します。

  マネージドポリシー: AWSPanoramaApplianceServiceRolePolicy

各ロールにアタッチされている権限を表示するには、IAM コンソールを使用してください。ロールの権限は、可能な限り、AWS Panorama が使用する命名パターンに一致するリソースに制限されます。たとえば、`AWSServiceRoleForAWSPanorama` は名前に`panorama` が含まれているAWS IoT リソースにアクセスする権限のみをサービスに付与します。

セクション

- アプライアンスロールの保護
- 他のサービスの使用

アプライアンスロールの保護

AWS Panorama アプライアンスは`AWSPanoramaApplianceServiceRole` ロールを使用してアカウントのリソースにアクセスします。アプライアンスには、CloudWatch Logs にログをアップロードしたり、AWS Secrets Manager からカメラストリーム認証情報を読み取ったり、AWS Panorama が作成した Amazon Simple Storage Service (Amazon S3) アクセスポイントのアプリケーションアーティファクトにアクセスしたりする権限があります。

> **Note**
>
> アプリケーションはアプライアンスの権限を使用しません。アプリケーションにAWS サービスを使用する権限を与えるには、アプリケーションロールを作成します。

AWS Panorama は、アカウント内のすべてのアプライアンスで同じサービスロールを使用し、アカウント間でロールを使用することはありません。セキュリティを強化するために、アプライアンスロールの信頼ポリシーを変更してこれを明示的に適用できます。これは、ロールを使用してアカウント内のリソースにアクセスするためのサービス権限をサービスに付与する場合のベストプラクティスです。

アプライアンスロールの信頼ポリシーを更新するには

1. IAM コンソールでアプライアンスロール (AWSPanoramaApplianceServiceRole) を開きます。
2. [Edit trust relationship] (信頼関係の編集) をクリックします。
3. ポリシーの内容を更新し、[信頼ポリシーの更新]を選択します。

以下の信頼ポリシーには、AWS Panorama がアプライアンスの役割を引き受けるときに、アカウント内のアプライアンスに対してその役割を担うことを保証する条件が含まれています。`aws:SourceAccount` 条件は、AWS Panorama で指定されたアカウント ID を、ポリシーに含めたアカウント ID と比較します。

Example 信頼ポリシー — 特定のアカウント

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "panorama.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        }
      }
    }
```

```
  ]
}
```

AWS Panorama をさらに制限し、特定のデバイスでのみロールを引き受けられるようにしたい場合は、ARN でデバイスを指定できます。aws:SourceArn 条件は、AWS Panorama で指定されたアプライアンスの ARN を、ポリシーに含めたアカウント ID と比較します。

Example 信頼ポリシー — 単一アプライアンス

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "panorama.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:panorama:us-east-1:123456789012:device/
device-lk7exmplpvcr3heqwjmesw76ky"
        },
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

アプライアンスをリセットして再プロビジョニングする場合は、ソース ARN 条件を一時的に削除してから、新しいデバイス ID を使用して再度追加する必要があります。

これらの条件と、サービスがロールを使用してアカウント内のリソースにアクセスする場合のセキュリティのベストプラクティスについて詳しくは、「IAM ユーザーガイド」の「混乱した代理問題」を参照してください。

他のサービスの使用

AWS Panorama は、以下のサービスのリソースを作成またはアクセスします。

- AWS IoT — AWS Panorama アプライアンス用のモノ、ポリシー、証明書、ジョブ
- Amazon S3 — アプリケーションモデル、コード、構成をステージングするためのアクセスポイント。
- Secrets Manager — AWS Panorama アプライアンスの短期認証情報。

各サービスの Amazon リソースネーム (ARN) 形式または権限スコープについては、このリストにリンクされている IAM ユーザーガイドのトピックを参照してください。

アプリケーションへの権限の付与

アプリケーションのロールを作成して、AWS サービスを呼び出す権限を付与できます。デフォルトでは、アプリケーションには何も権限がありません。IAM でアプリケーションロールを作成し、デプロイ時にアプリケーションに割り当てます。アプリケーションに必要な権限のみを付与するには、特定の API アクションの権限を持つロールを作成します。

サンプルアプリケーションには、アプリケーションロールを作成する AWS CloudFormation テンプレートとスクリプトが含まれています。AWS Panorama が引き受けることができるサービスロールです。このロールは、アプリケーションに CloudWatch を呼び出してメトリックスをアップロードする権限を付与します。

Example aws-panorama-sample.yml — アプリケーションロール

```
Resources:
  runtimeRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - panorama.amazonaws.com
            Action:
              - sts:AssumeRole
      Policies:
        - PolicyName: cloudwatch-putmetrics
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              - Effect: Allow
                Action: 'cloudwatch:PutMetricData'
                Resource: '*'
      Path: /service-role/
```

Action の値に API アクションまたはパターンのリストを指定することで、このスクリプトを拡張して他のサービスにアクセス許可を付与できます。

AWS Panorama でのアクセス権限の詳細は、 AWS Panorama アクセス権限 をご参照ください。

AWS Panorama アプライアンスの管理

AWS Panorama アプライアンスはアプリケーションを実行するハードウェアです。アプライアンスの登録、ソフトウェアの更新、アプリケーションのデプロイを行うために AWS Panorama を使用します。AWS Panorama アプライアンス上のソフトウェアはカメラストリームに接続し、ビデオフレームをアプリケーションに送信し、接続されているディスプレイにビデオ出力を表示します。

アプライアンスまたは別の互換性のあるデバイスを設定したのち、アプリケーションで使用するカメラを登録します。カメラストリームの管理は AWS Panorama コンソールで行います。アプリケーションをデプロイする際は、アプライアンスがどのカメラストリームを送信して処理するかを選択します。

AWS Panorama アプライアンスとサンプルアプリケーションを紹介するチュートリアルについては、 AWS Panorama の開始方法 を参照してください。

トピック

- AWS Panorama アプライアンスの管理
- AWS Panorama アプライアンスをネットワークに接続します
- AWS Panorama でのカメラストリームの管理
- AWS Panorama アプライアンスでのアプリケーションの管理
- AWS Panorama アプライアンスのボタンとライト

AWS Panorama アプライアンスの管理

AWS Panorama コンソールを使用して、AWS Panorama アプライアンスやその他の互換性のあるデバイスを構成、アップグレード、または登録解除します。

アプライアンスをセットアップするには、入門チュートリアルの指示に従ってください。セットアッププロセスでは、アプライアンスを追跡し、更新とデプロイを調整するリソースが AWS Panorama に作成されます。

AWS Panorama API にアプライアンスを登録するには、「デバイス登録を自動化」を参照してください。

セクション

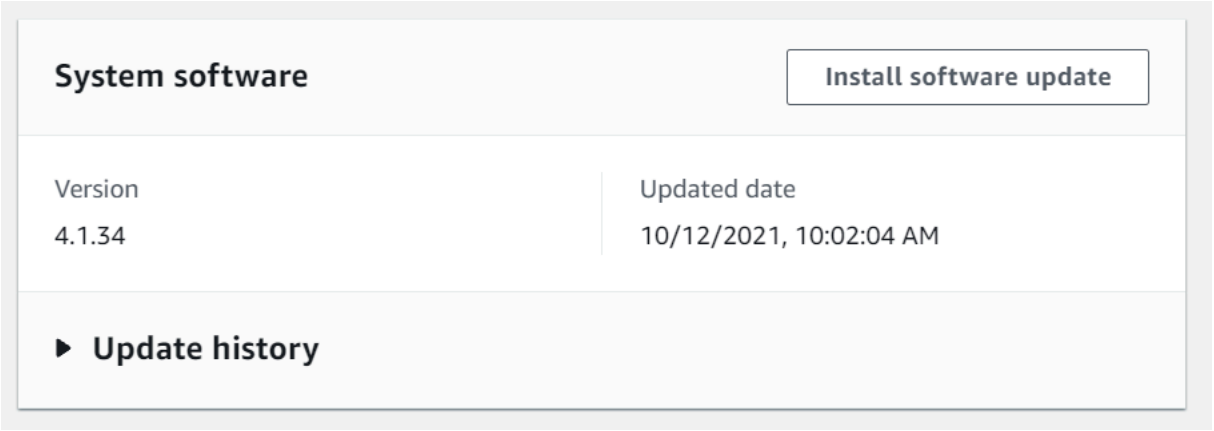
- アプライアンスソフトウェアを更新します。
- アプライアンスの登録解除
- アプライアンスの再起動
- アプライアンスのリセット

アプライアンスソフトウェアを更新します。

AWS Panorama コンソールでアプライアンスのソフトウェアアップデートを表示してデプロイします。更新は必須またはオプションです。必要なアップデートが入手可能になると、コンソールからそのアップデートを適用するよう求められます。アプライアンスの[設定] ページでオプションのアップデートを適用できます。

アプライアンスソフトウェアをアップグレードする

1. AWS Panorama コンソールの [デバイス] ページを開きます。
2. アプライアンスを選択します。
3. [設定] を選択します。
4. [システムのソフトウェア] で [ソフトウェアのインストール] を選択します。



5. 新しいバージョンを選択し、[インストール] を選択します。

アプライアンスの登録解除

アプライアンスの操作が終了したら、AWS Panorama コンソールを使用して登録を解除し、関連する AWS IoT リソースを削除できます。

アプライアンスを削除するには

1. AWS Panorama コンソールの [デバイス] ページを開きます。
2. アプライアンスの名前を選択します。
3. [Delete] (削除) をクリックします。
4. アプライアンスの名前を入力し、[削除] を選択します。

AWS Panorama サービスからアプライアンスを削除しても、アプライアンス上のデータは自動的に削除されません。登録解除されたアプライアンスは AWS サービスに接続できず、リセットされるまで再登録することもできません。

アプライアンスの再起動

アプライアンスはリモートで再起動できます。

アプライアンスを再起動するには

1. AWS Panorama コンソールの [デバイス] ページを開きます。
2. アプライアンスの名前を選択します。
3. [Reboot] を選択します。

コンソールはアプライアンスに再起動を求めるメッセージを送信します。信号を受信するには、アプライアンスがAWS IoT に接続できる必要があります。AWS Panorama API にアプライアンスを再起動するには、「アプライアンスの再起動」を参照してください。

アプライアンスのリセット

別のリージョンまたは別のアカウントのアプライアンスを使用するには、アプライアンスをリセットし、新しい証明書で再プロビジョニングする必要があります。デバイスをリセットすると、必要な最新のソフトウェアバージョンが適用され、すべてのアカウントデータが削除されます。

リセット操作を開始するには、アプライアンスを接続して電源を切る必要があります。電源ボタンとリセットボタンの両方を 5 秒間押し続けます。ボタンを離すと、ステータスライトがオレンジ色に点滅します。ステータスライトが緑色に点滅するまで待ってから、アプライアンスをプロビジョニングまたは切断してください。

デバイスから証明書を削除せずにアプライアンスソフトウェアをリセットすることもできます。詳細については、「電源ボタンとリセットボタン」を参照してください。

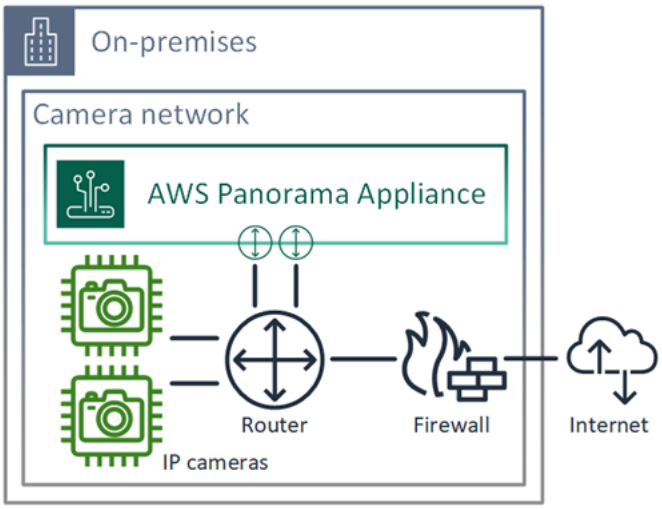
AWS Panorama アプライアンスをネットワークに接続します

AWS Panorama アプライアンスには、AWS クラウドと IP カメラのオンプレミスネットワークの両方への接続が必要です。アプライアンスを両方へのアクセスを許可する単一のファイアウォールに接続することも、デバイスの 2 つのネットワークインターフェイスをそれぞれ別のサブネットに接続することもできます。いずれの場合も、カメラストリームへの不正アクセスを防ぐために、アプライアンスのネットワーク接続を保護する必要があります。

セクション

単一ネットワークの構成

アプライアンスには 2 つのイーサネットポートがあります。デバイスとの間で送受信されるすべてのトラフィックを 1 台のルーター経由でルーティングすれば、1 つ目のポートとの物理的な接続が切断された場合に備えて、2 つ目のポートを冗長化できます。アプライアンスがカメラストリームとインターネットにのみ接続できるようにし、それ以外の場合はカメラストリームが内部ネットワークから送信されないようにルーターを構成します。

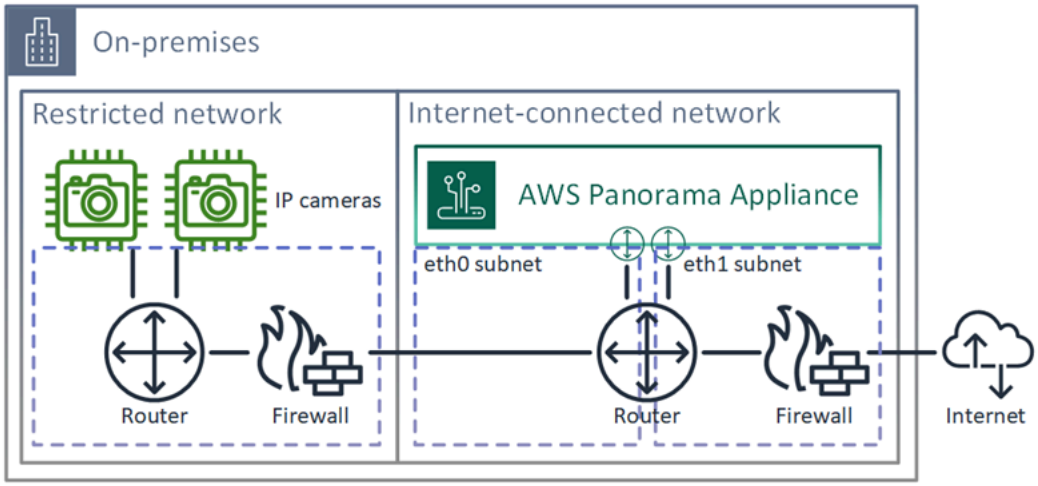


アプライアンスがアクセスする必要のあるポートとエンドポイントの詳細については、「サービスアクセスを構成する」と「ローカルネットワークアクセスの構成」を参照してください。

デュアルネットワーク構成

セキュリティをさらに強化するために、カメラネットワークとは別のインターネットに接続されたネットワークにアプライアンスを配置できます。制限されたカメラネットワークとアプライアンスのネットワーク間のファイアウォールにより、アプライアンスはビデオストリームにアクセスすることのみ許可されます。セキュリティ上の目的でカメラネットワークが工アギャップになっていた場合は、インターネットへのアクセスも許可するルーターにカメラネットワークを接続するよりも、この方法の方が適しているかもしれません。

以下の例は、アプライアンスを各ポートの異なるサブネットに接続しているところを示しています。ルーターは`eth0` インターフェイスをカメラネットワークにルーティングするサブネットと、インターネットにルーティングする `eth1` サブネットに配置します。



各ポートの IP アドレスと MAC アドレスは AWS Panorama コンソールで確認できます。

サービスアクセスを構成する

プロビジョニング中に、特定の IP アドレスをリクエストするようにアプライアンスを構成できます。IP アドレスを事前に選択しておくと、ファイアウォールの構成が簡単になり、長期間オフラインになってもアプライアンスのアドレスが変更されないようになります。

アプライアンスは、AWS サービスを使用してソフトウェアの更新と展開を調整します。アプライアンスがこれらのエンドポイントに接続できるようにファイアウォールを構成します。

インターネットアクセス

- AWS IoT(HTTPS と MQTT、ポート 443、8443、8883) — AWS IoT Core およびデバイス管理エンドポイント。詳細については、Amazon Web Services 全般のリファレンス の「AWS IoT Device Management エンドポイントとクォータ」を参照してください。
- AWS IoT 認証情報 (HTTPS、ポート 443) — `credentials.iot.<region>.amazonaws.com` およびサブドメイン。
- Amazon エラスティックコンテナレジストリ (HTTPS、ポート 443) — `api.ecr.<region>.amazonaws.com`、`dkr.ecr.<region>.amazonaws.com`およびサブドメイン。
- Amazon CloudWatch (HTTPS、ポート 443) — `monitoring.<region>.amazonaws.com`。
- Amazon CloudWatch Logs (HTTPS、ポート 443) — `logs.<region>.amazonaws.com`。
- Amazon シンプルストレージサービス (HTTPS、ポート 443) — `s3.<region>.amazonaws.com`、`s3-accesspoint.<region>.amazonaws.com`およびサブドメイン。

アプリケーションが他の AWS サービスを呼び出す場合、アプライアンスはそれらのサービスのエンドポイントにもアクセスする必要があります。詳細については、「サービスエンドポイントとクォータ」を参照してください。

ローカルネットワークアクセスの構成

アプライアンスはローカルで RTSP ビデオストリームにアクセスする必要がありますが、インターネット経由ではアクセスできません。アプライアンスがポート 554 の RTSP ストリームに内部的にアクセスできるようにし、ストリームがインターネットに出入りしないようにファイアウォールを構成します。

ローカルアクセス

- リアルタイムストリーミングプロトコル (RTSP、ポート 554) — カメラストリームを読み取ります。
- ネットワークタイムプロトコル (NTP、ポート 123) — アプライアンスの時計を同期させます。ネットワーク上で NTP サーバーを実行していない場合、アプライアンスはインターネット経由でパブリック NTP サーバーに接続することもできます。

プライベート接続

AWS へのVPN 接続を使用してプライベート VPC サブネットにデプロイする場合、AWS Panorama アプライアンスはインターネットアクセスを必要としません。Site-to-Site VPN または AWS Direct Connect を使用して、オンプレミスルーターとAWS との間の VPN 接続を作成できます。プライベート VPC サブネット内に、アプライアンスを Amazon Simple Storage Service、AWS IoT、およびその他のサービスに接続できるようにするエンドポイントを作成します。詳細については、「アプライアンスをプライベートサブネットに接続する」を参照してください。

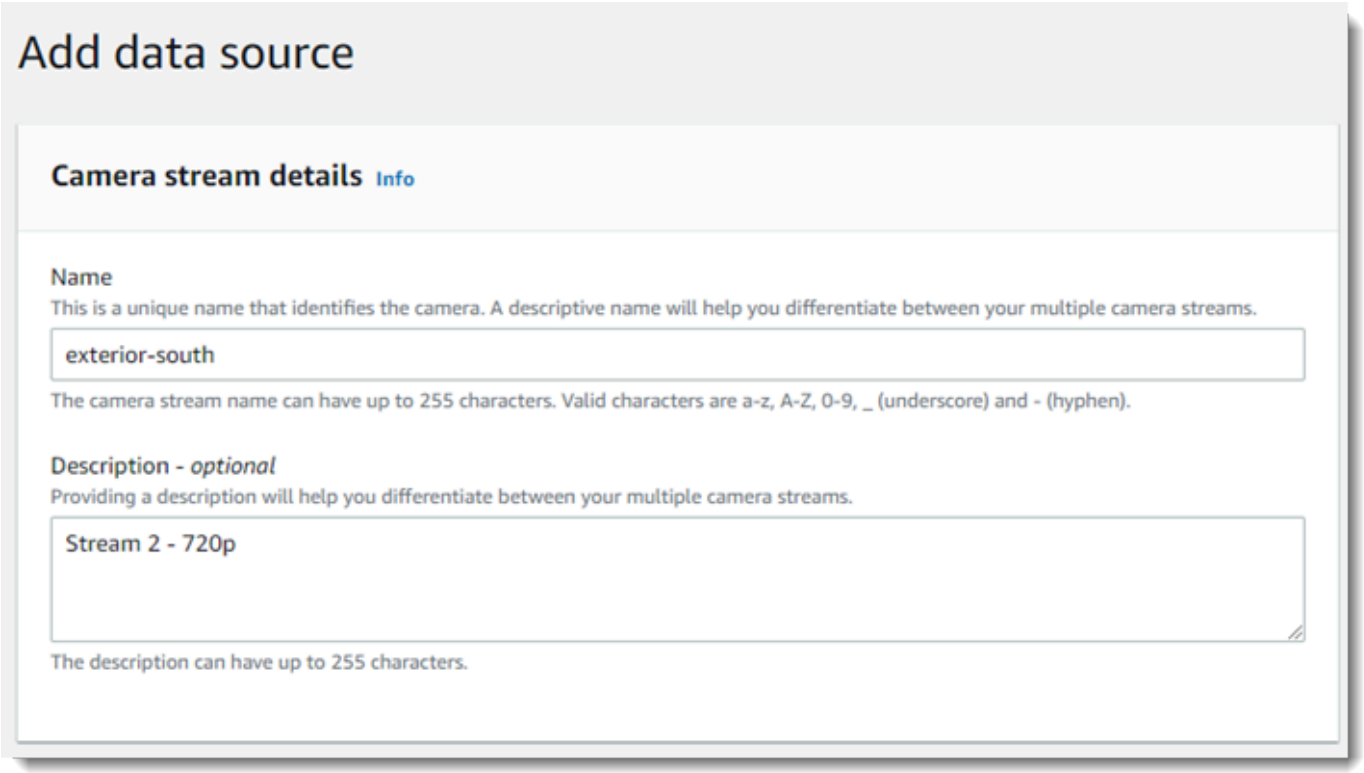
AWS Panorama でのカメラストリームの管理

ビデオストリームをアプリケーションのデータソースとして登録するには、AWS Panorama コンソールを使用します。アプリケーションは複数のストリームを同時に処理でき、複数のアプライアンスを同じストリームに接続できます。

> **Important**
>
> アプリケーションは、接続先のローカルネットワークからルーティング可能な任意のカメラストリームに接続できます。ビデオストリームを保護するには、ローカルで RTSP トラフィックのみを許可するようにネットワークを構成します。詳細については、「AWS Panorama セキュリティ」を参照してください。

カメラストリームを登録する

1. AWS Panorama コンソールの [データソース] ページを開きます。
2. [Add data source] (データソースを追加する) を選択する。



3. 以下を設定します。

- 名前 — カメラストリームの名前。
- 説明 — カメラ、その場所、その他の詳細に関する簡単な説明。
- RTSP URL — カメラの IP アドレスとストリームへのパスを指定する URL。例えば、`rtsp://192.168.0.77/live/mpeg4/` などです。
- 認証情報 — カメラストリームがパスワードで保護されている場合は、ユーザーネームとパスワードを指定します。

4. [Save (保存)] を選択します。

カメラストリームを AWS Panorama API に登録するには、「デバイス登録を自動化」を参照してください。

AWS Panorama アプライアンスと互換性のあるカメラのリストについては、サポートされているコンピュータービジョンモデルとカメラ を参照してください。

ストリームを削除します。

カメラストリームは AWS Panorama コンソールで削除できます。

カメラストリームを削除するには

1. AWS Panorama コンソールの [データソース] ページを開きます。
2. カメラストリームを選択します
3. [データソースを削除する] を選択します。

サービスからカメラストリームを削除しても、アプリケーションの実行が停止したり、Secrets Manager からカメラの認証情報が削除されたりすることはありません。シークレットを削除するには、Secrets Manager コンソールを使用してください。

AWS Panorama アプライアンスでのアプリケーションの管理

アプリケーションは、コード、モデル、構成を組み合わせたものです。AWS Panorama コンソールのデバイスページから、アプライアンス上のアプリケーションを管理できます。

AWS Panorama アプライアンスでアプリケーションを管理するには

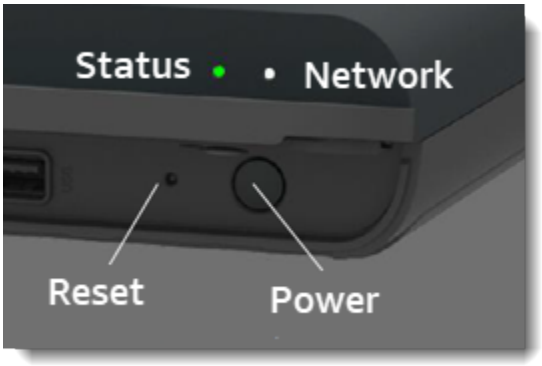
1. AWS Panorama コンソールの [デバイス] ページを開きます。
2. アプライアンスを選択します。

「デプロイされたアプリケーション」ページには、アプライアンスにデプロイされたアプリケーションが表示されます。

このページのオプションを使用して、デプロイされたアプリケーションをアプライアンスから削除したり、実行中のアプリケーションを新しいバージョンに置き換えたりします。また、(実行中または削除済みの)アプリケーションを複製して、その新しいコピーをデプロイすることもできます。

AWS Panorama アプライアンスのボタンとライト

AWS Panorama アプライアンスの電源ボタンの上には、デバイスのステータスとネットワーク接続を示す 2 つの LED ライトがあります。



ステータスライト

LED の色が変わり、点滅してステータスを示します。ゆっくり点滅するのは 3 秒に 1 回です。高速点滅は 1 秒に 1 回です。

ステータス LED の状態が表示されます。

- 緑色で高速点滅 — アプライアンスは起動中です。
- 緑色で点灯 — アプライアンスは正常に動作しています。
- 青色でゆっくり点滅 — アプライアンスは構成ファイルをコピーし、AWS IoT に登録しようとしています。
- 青色で高速点滅 — アプライアンスはログイメージを USB ドライブにコピーしています。
- 赤色で高速点滅 — アプライアンスのスタートアップ中にエラーが発生したか、過熱状態になっています。
- オレンジ色でゆっくり点滅 — アプライアンスは最新のソフトウェアバージョンを復元中です。
- オレンジ色で高速点滅 — アプライアンスは最低ソフトウェアバージョンを復元中です。

ネットワークライト

ネットワーク LED には以下の状態があります。

ネットワーク LED の状態

- 緑色で点灯 — イーサネットケーブルが接続されています。

- 緑色で点滅 — アプライアンスはネットワーク経由で通信中です。
- 赤色で点灯 — イーサネットケーブルが接続されています。

電源ボタンとリセットボタン

電源ボタンとリセットボタンは、デバイス前面の保護力バーの下にあります。リセットボタンは小さく、くぼんでいます。小さなドライバーやペーパークリップを使って押してください。

アプライアンスをリセットするには

1. アプライアンスを接続し、電源をオフにする必要があります。アプライアンスの電源を切るには、電源ボタンを 1 秒間押し続け、シャットダウンシーケンスが完了するまで待ちます。シャットダウンシーケンスには約 10 秒かかります。
2. アプライアンスを再起動するには、次のボタンの組合せを使用します。短く押すのは 1 秒です。長押しは 5 秒です。複数のボタンが必要な操作では、両方のボタンを同時に長押しします。

   - フルリセット — 電源を長押ししてリセットします。

     最低限のソフトウェアバージョンを復元し、すべての構成ファイルとアプリケーションを削除します。
   - 最新のソフトウェアバージョンを復元 — リセットを短く押します。

     最新のソフトウェアアップデートをアプライアンスに再適用します。
   - 最小ソフトウェアバージョンの復元 — リセットを長押しします。

     最新の必要なアップデートをアプライアンスに再適用します。
3. 両方のボタンを離します。アプライアンスの電源が入り、ステータスライトが数分間オレンジ色に点滅します。
4. アプライアンスの準備が整うと、ステータスライトが緑色に点滅します。

アプライアンスをリセットしても、AWS Panorama サービスからは削除されません。詳細については、「アプライアンスの登録解除」を参照してください。

AWS Panorama アプリケーションを管理する

アプリケーションは AWS Panorama アプライアンス上で動作し、ビデオストリーム上でコンピュータービジョンタスクを実行します。Pythonコードと機械学習モデルを組み合わせて、コンピュータビジョンアプリケーションをビルドし、インターネット経由で AWS Panorama アプライアンスにデプロイできます。アプリケーションはディスプレイに動画を送信したり、AWS SDK を使用して結果を AWS のサービスに送信したりできます。

トピック

アプリケーションをデプロイします

アプリケーションをデプロイするには、AWS Panorama Application CLI を使用してアカウントにインポートし、コンテナを構築し、アセットをアップロードして登録し、アプリケーションインスタンスを作成します。このトピックでは、これらの各ステップを詳細に説明し、バックグラウンドで何が行われているかを説明します。

まだアプリケーションをデプロイしていないのであれば、AWS Panorama の開始方法 を参照してウォークスルーを行ってください。

サンプルアプリケーションのカスタマイズと拡張の詳細については、AWS Panorama アプリケーションの構築 を参照してください。

セクション

- AWS Panorama Application CLI をインストールする
- アプリケーションをインポートします
- コンテナイメージの構築
- モデルのインポート
- アプリケーションアセットをアップロードします
- AWS Panorama コンソールでアプリケーションをデプロイ
- アプリケーションのデプロイを自動化する

AWS Panorama Application CLI をインストールする

AWS Panorama Application CLI と AWS CLI をインストールするには、pip を使用します。

```
$ pip3 install --upgrade awscli panoramacli
```

AWS Panorama Application CLI を使用してアプリケーションイメージを構築するには、Docker が必要です。Linux では qemu および関連するシステムライブラリも必要です。AWS Panorama Application CLI のインストールと設定の詳細については、プロジェクトの GitHub リポジトリにある「README」 ファイルを参照してください。

- github.com/aws/aws-panorama-cli

Windows で WSL2 を使ってビルド環境を設定する手順については、Windows 開発環境でのセットアップ を参照してください。

アプリケーションをインポートします

サンプルアプリケーションまたはサードパーティが提供するアプリケーションを使用している場合は、AWS Panorama Application CLI を使用してアプリケーションをインポートします。

```
my-app$ panorama-cli import-application
```

このコマンドは、アカウント ID でアプリケーションパッケージの名前を変更します。Package 名は、デプロイ先のアカウントのアカウント ID で始まります。アプリケーションを複数のアカウントにデプロイする場合、アカウントごとにアプリケーションを個別にインポートしてパッケージ化する必要があります。

たとえば、このガイドのサンプルアプリケーションには、コードパッケージとモデルパッケージがあり、それぞれの名前にプレースホルダーアカウント ID が付けられています。`import-application` コマンドは、CLIがワークスペースの AWS 認証情報から推測したアカウント ID を使用するように名前を変更します。

```
/aws-panorama-sample
### assets
### graphs
#   ### my-app
#       ### graph.json
### packages
    ### 123456789012-SAMPLE_CODE-1.0
    #   ### Dockerfile
    #   ### application.py
    #   ### descriptor.json
    #   ### package.json
    #   ### requirements.txt
    #   ### squeezenet_classes.json
    ### 123456789012-SQUEEZENET_PYTORCH-1.0
        ### descriptor.json
        ### package.json
```

パッケージ・ディレクトリ名と、それらを参照するアプリケーション・マニフェスト (`graph.json`) では、`123456789012` はアカウント ID に置き換えられます。アカウント ID を確認するには、AWS CLI で `aws sts get-caller-identity` を呼び出します。

```
$ aws sts get-caller-identity
{
    "UserId": "AIDAXMPL7W66UC3GFXMPL",
    "Account": "210987654321",
    "Arn": "arn:aws:iam::210987654321:user/devenv"
}
```

コンテナイメージの構築

アプリケーションコードは Docker コンテナイメージにパッケージ化されており、これには Dockerfile にインストールするアプリケーションコードとライブラリが含まれます。AWS Panorama Application CLI `build-container` コマンドを使用して Docker イメージを構築し、ファイルシステムイメージをエクスポートします。

```
my-app$ panorama-cli build-container --container-asset-name code_asset --package-path
 packages/210987654321-SAMPLE_CODE-1.0
{
    "name": "code_asset",
    "implementations": [
        {
            "type": "container",
            "assetUri":
 "5fa5xmplbc8c16bf8182a5cb97d626767868d3f4d9958a4e49830e1551d227c5.tar.gz",
            "descriptorUri":
 "1872xmpl129481ed053c52e66d6af8b030f9eb69b1168a29012f01c7034d7a8f.json"
        }
    ]
}
Container asset for the package has been succesfully built at
 assets/5fa5xmplbc8c16bf8182a5cb97d626767868d3f4d9958a4e49830e1551d227c5.tar.gz
```

このコマンドは、という名前の Docker イメージを作成し`code_asset`、ファイルシステムをフォルダ内のアーカイブにエクスポートします。`.tar.gz` `assets`CLI は、アプリケーションの Dockerfile で指定されているように、Amazon Elastic Container Registry (Amazon ECR) からアプリケーションのベースイメージを取得します。

コンテナアーカイブに加えて、CLI はパッケージ記述子 (`descriptor.json`) のアセットを作成します。どちらのファイルも、元のファイルのハッシュを反映する一意の識別子で名前が変更されます。また、AWS Panorama Application CLI は、2 つのアセットの名前を記録するブロックをパッ

ケージ設定に追加します。これらの名前は、デプロイプロセス中にアプライアンスによって使用されます。

Example Packages/123456789012-sample_code-1.0/Package.json — アセットブロック付き

```
{
    "nodePackage": {
        "envelopeVersion": "2021-01-01",
        "name": "SAMPLE_CODE",
        "version": "1.0",
        "description": "Computer vision application code.",
        "assets": [
            {
                "name": "code_asset",
                "implementations": [
                    {
                        "type": "container",
                        "assetUri":
 "5fa5xmplbc8c16bf8182a5cb97d626767868d3f4d9958a4e49830e1551d227c5.tar.gz",
                        "descriptorUri":
 "1872xmpl129481ed053c52e66d6af8b030f9eb69b1168a29012f01c7034d7a8f.json"
                    }
                ]
            }
        ],
        "interfaces": [
            {
                "name": "interface",
                "category": "business_logic",
                "asset": "code_asset",
                "inputs": [
                    {
                        "name": "video_in",
                        "type": "media"
                    },
```

build-container コマンドで指定するコードアセットの名前は、パッケージ設定内の asset フィールドの値と一致する必要があります。前述の例では、両方の値が code_asset です。

モデルのインポート

アプリケーションの アセットフォルダーにモデルアーカイブがある場合や、個別にダウンロードするモデルアーカイブがある場合があります。新しいモデル、更新されたモデル、または更新されたモデル記述子ファイルがある場合は、add-raw-model コマンドを使用してインポートします。

```
my-app$ panorama-cli add-raw-model --model-asset-name model_asset \
      --model-local-path my-model.tar.gz \
      --descriptor-path packages/210987654321-SQUEEZENET_PYTORCH-1.0/descriptor.json \
      --packages-path packages/210987654321-SQUEEZENET_PYTORCH-1.0
```

記述子ファイルの更新だけが必要な場合は、アセットディレクトリにある既存のモデルを再利用できます。浮動小数点精度モードなどの機能を設定するには、記述子ファイルの更新が必要な場合があります。例えば、次のスクリプトはサンプルアプリでこれを行う方法を示しています。

Example util-scripts/update-model-config.sh

```
#!/bin/bash
set -eo pipefail
MODEL_ASSET=fd1axmplacc3350a5c2673adacffab06af54c3f14da6fe4a8be24cac687a386e
MODEL_PACKAGE=SQUEEZENET_PYTORCH
ACCOUNT_ID=$(ls packages | grep -Eo '[0-9]{12}' | head -1)
panorama-cli add-raw-model --model-asset-name model_asset --model-local-path assets/
${MODEL_ASSET}.tar.gz --descriptor-path packages/${ACCOUNT_ID}-${MODEL_PACKAGE}-1.0/
descriptor.json --packages-path packages/${ACCOUNT_ID}-${MODEL_PACKAGE}-1.0
cp packages/${ACCOUNT_ID}-${MODEL_PACKAGE}-1.0/package.json packages/${ACCOUNT_ID}-
${MODEL_PACKAGE}-1.0/package.json.bup
```

モデルパッケージディレクトリの記述子ファイルへの変更は、CLI で再インポートするまで適用されません。CLI は、コンテナを再構築するときにアプリケーションコードパッケージの設定を更新するのと同様に、新しいアセット名を使用してモデルパッケージ構成を更新します。

アプリケーションアセットをアップロードします

モデルアーカイブ、コンテナファイルシステムアーカイブ、およびそれらの記述子ファイルを含むアプリケーションのアセットをアップロードして登録するには、package-application コマンドを使用します。

```
my-app$ panorama-cli package-application
Uploading package SQUEEZENET_PYTORCH
```

```
Patch version for the package
 5d3cxmplb7113faa1d130f97f619655d8ca12787c751851a0e155e50eb5e3e96
Deregistering previous patch version
 e845xmpl8ea0361eb345c313a8dded30294b3a46b486dc8e7c174ee7aab29362
Asset fd1axmplacc3350a5c2673adacffab06af54c3f14da6fe4a8be24cac687a386e.tar.gz already
 exists, ignoring upload
upload: assets/87fbxmpl6f18aeae4d1e3ff8bbc6147390feaf47d85b5da34f8374974ecc4aaf.json
 to s3://arn:aws:s3:us-east-2:212345678901:accesspoint/
panorama-210987654321-6k75xmpl2jypelgzst7uux62ye/210987654321/nodePackages/
SQUEEZENET_PYTORCH/
binaries/87fbxmpl6f18aeae4d1e3ff8bbc6147390feaf47d85b5da34f8374974ecc4aaf.json
Called register package version for SQUEEZENET_PYTORCH with patch version
 5d3cxmplb7113faa1d130f97f619655d8ca12787c751851a0e155e50eb5e3e96
...
```

アセットファイルまたはパッケージ設定に変更がない場合、CLI はそれをスキップします。

```
Uploading package SAMPLE_CODE
Patch Version ca91xmplca526fe3f07821fb0c514f70ed0c444f34cb9bd3a20e153730b35d70 already
 registered, ignoring upload
Register patch version complete for SQUEEZENET_PYTORCH with patch version
 5d3cxmplb7113faa1d130f97f619655d8ca12787c751851a0e155e50eb5e3e96
Register patch version complete for SAMPLE_CODE with patch version
 ca91xmplca526fe3f07821fb0c514f70ed0c444f34cb9bd3a20e153730b35d70
All packages uploaded and registered successfully
```

CLI は、各パッケージのアセットを、アカウントに固有の Amazon S3 Access Points にアップロードします。AWS Panorama はアクセスポイントを管理し、DescribePackage API を通じてアクセスポイントに関する情報を提供します。CLI は各パッケージのアセットをそのパッケージに指定された場所にアップロードし、パッケージ設定で記述された設定で AWS Panorama サービスに登録します。

AWS Panorama コンソールでアプリケーションをデプロイ

AWS Panorama コンソールでアプリケーションをデプロイできます。デプロイプロセス中に、アプリケーションコードに渡すカメラストリームを選択し、アプリケーションの開発者が提供するオプションを設定します。

アプリケーションのデプロイ

1. AWS Panorama コンソールの [デプロイされたアプリケーション] ページを開きます。

2. [アプリケーションをデプロイ] を選択します。
3. アプリケーションマニフェスト、graph.json の内容をテキストエディタに貼り付けます。次へ をクリックします。
4. 名前と説明を入力します。
5. [デプロイに進む] を選択します。
6. [デプロイを開始] を選択します。
7. アプリケーションでロールを使用する場合は、ドロップダウンメニューからロールを選択します。次へ をクリックします。
8. [デバイスを選択]を選択し、次にアプライアンスを選択します。次へ をクリックします。
9. 「データソースを選択」 ステップで、「入力を表示」を選択し、カメラストリームをデータソースとして追加します。次へ をクリックします。
10. 設定 ステップでは、開発者が定義したアプリケーション固有の設定を行います。次へ をクリックします。
11. [デプロイ] を選択してから、[完了] を選択します。
12. デプロイされたアプリケーションのリストで、ステータスをモニタリングするアプリケーションを選択します。

デプロイプロセスには約 15～20 分かかります。アプリケーションの起動中は、アプライアンスの出力が長時間空白になることがあります。エラーが発生した場合は、トラブルシューティング を参照してください。

アプリケーションのデプロイを自動化する

CreateApplicationInstance API でアプリケーションのデプロイプロセスを自動化できます。この API は 2 つの設定ファイルを入力として受け取ります。アプリケーションマニフェストは、使用するパッケージとそれらの関係を指定します。2 つ目のファイルは、アプリケーションマニフェスト内の値をデプロイ時のオーバーライドを指定するオーバーライドファイルです。オーバーライドファイルを使用すると、同じアプリケーションマニフェストを使用してアプリケーションを異なるカメラストリームにデプロイしたり、その他のアプリケーション固有の設定を行ったりできます。

詳細と、このトピックの各ステップのスクリプト例については、アプリケーションのデプロイを自動化する を参照してください。

AWS Panorama コンソールでのアプリケーションの管理

AWS Panorama コンソールを使用してデプロイされたアプリケーションを管理します。

セクション

アプリケーションを更新またはコピーします。

アプリケーションを更新するには、置き換え オプションを使用します。アプリケーションを置き換えるときに、そのコードまたはモデルを更新できます。

アプリケーションを更新するには

1. AWS Panorama コンソールの [デプロイされたアプリケーション] ページを開きます。
2. アプリケーションを選択します。
3. [Replace (置換)] を選択します。
4. 手順に従って、新しいバージョンまたはアプリケーションを作成します。

「置き換え」と同様の動作をする「クローン」オプションもありますが、古いバージョンのアプリケーションは削除されません。このオプションを使用すると、実行中のバージョンを停止せずにアプリケーションの変更をテストしたり、すでに削除したバージョンを再デプロイしたりできます。

バージョンとアプリケーションを削除します。

未使用のアプリケーションバージョンをクリーンアップするには、アプライアンスから削除します。

アプリケーションを削除するには

1. AWS Panorama コンソールの [デプロイされたアプリケーション] ページを開きます。
2. アプリケーションを選択します。
3. [デバイスから削除] を選択します。

パッケージ設定

AWS Panorama アプリケーション CLI コマンド `panorama-cli package-application` を使用すると、CLI はアプリケーションのアセットを Amazon S3 にアップロードし、AWS Panorama に登録します。アセットには、AWS Panorama Appliance がデプロイ中にダウンロードするバイナリファイル (コンテナイメージとモデル) と記述子ファイルが含まれます。パッケージのアセットを登録するには、パッケージ、そのアセット、そのインターフェイスを定義する個別のパッケージ構成ファイルを提供します。

次の例は、1 つの入力と 1 つの出力を備えたコードノードのパッケージ構成を示しています。ビデオ入力では、カメラストリームの画像データにアクセスできます。出力ノードは処理された画像をディスプレイに送ります。

Example packages/1234567890-SAMPLE_CODE-1.0/package.json

```
{
    "nodePackage": {
        "envelopeVersion": "2021-01-01",
        "name": "SAMPLE_CODE",
        "version": "1.0",
        "description": "Computer vision application code.",
        "assets": [
            {
                "name": "code_asset",
                "implementations": [
                    {
                        "type": "container",
                        "assetUri":
 "3d9bxmplbdb67a3c9730abb19e48d78780b507f3340ec3871201903d8805328a.tar.gz",
                        "descriptorUri":
 "1872xmpl129481ed053c52e66d6af8b030f9eb69b1168a29012f01c7034d7a8f.json"
                    }
                ]
            }
        ],
        "interfaces": [
            {
                "name": "interface",
                "category": "business_logic",
                "asset": "code_asset",
                "inputs": [
                    {
```

```
                        "name": "video_in",
                        "type": "media"
                    }
                ],
                "outputs": [
                    {
                        "description": "Video stream output",
                        "name": "video_out",
                        "type": "media"
                    }
                ]
            }
        ]
    }
}
```

assets セクションでは、AWS Panorama アプリケーション CLI が Amazon S3 にアップロードしたアーティファクトの名前を指定します。サンプルアプリケーションまたは別のユーザーからアプリケーションをインポートする場合、このセクションは空だったり、アカウントにないアセットを参照していたりすることがあります。panorama-cli package-application を実行すると、AWS Panorama アプリケーション CLI によってこのセクションに正しい値が入力されます。

AWS Panorama アプリケーションマニフェスト

アプリケーションをデプロイするときは、アプリケーションマニフェストと呼ばれる設定ファイルを提供します。このファイルでは、アプリケーションをノードとエッジを含むグラフとして定義します。アプリケーションマニフェストはアプリケーションのソースコードの一部であり、graphs ディレクトリに保存されます。

Example graphs/aws-panorama-sample/graph.json

```
{
    "nodeGraph": {
        "envelopeVersion": "2021-01-01",
        "packages": [
            {
                "name": "123456789012::SAMPLE_CODE",
                "version": "1.0"
            },
            {
                "name": "123456789012::SQUEEZENET_PYTORCH_V1",
                "version": "1.0"
            },
            {
                "name": "panorama::abstract_rtsp_media_source",
                "version": "1.0"
            },
            {
                "name": "panorama::hdmi_data_sink",
                "version": "1.0"
            }
        ],
        "nodes": [
            {
                "name": "code_node",
                "interface": "123456789012::SAMPLE_CODE.interface"
            }
            {
                "name": "model_node",
                "interface": "123456789012::SQUEEZENET_PYTORCH_V1.interface"
            },
            {
                "name": "camera_node",
                "interface": "panorama::abstract_rtsp_media_source.rtsp_v1_interface",
                "overridable": true,
```

```
                "overrideMandatory": true,
                "decorator": {
                    "title": "IP camera",
                    "description": "Choose a camera stream."
                }
            },
            {
                "name": "output_node",
                "interface": "panorama::hdmi_data_sink.hdmi0"
            },
            {
                "name": "log_level",
                "interface": "string",
                "value": "INFO",
                "overridable": true,
                "decorator": {
                    "title": "Logging level",
                    "description": "DEBUG, INFO, WARNING, ERROR, or CRITICAL."
                }
            }
            ...
        ],
        "edges": [
            {
                "producer": "camera_node.video_out",
                "consumer": "code_node.video_in"
            },
            {
                "producer": "code_node.video_out",
                "consumer": "output_node.video_in"
            },
            {
                "producer": "log_level",
                "consumer": "code_node.log_level"
            }
        ]
    }
}
```

ノードは、ノードの入力と出力間のマッピングを指定するエッジによって接続されています。あるノードの出力は別のノードの入力に接続され、グラフを形成します。

JSON スキーマ

アプリケーションマニフェストとオーバーライドドキュメントの形式は JSON スキーマで定義されます。JSON スキーマを使用して、デプロイ前に設定ドキュメントを検証できます。JSON スキーマは、本ガイドの GitHub リポジトリから入手できます。

- JSON スキーマ – aws-panorama-developer-guide/resources

アプリケーションノード

ノードはモデル、コード、カメラストリーム、出力、パラメーターです。ノードには入力と出力を定義するインターフェースがあります。インターフェイスは、アカウントのパッケージ、AWS Panorama が提供するパッケージ、または組み込みタイプで定義できます。

次の例では、code_node と model_node Bはサンプルアプリケーションに含まれるサンプルコードとモデルパッケージを参照しています。 camera_node は AWS Panorama が提供するパッケージを使用して、デプロイ時に指定したカメラストリームのプレースホルダーを作成します。

Example graph.json — ノード

```
    "nodes": [
        {
            "name": "code_node",
            "interface": "123456789012::SAMPLE_CODE.interface"
        },
        {
            "name": "model_node",
            "interface": "123456789012::SQUEEZENET_PYTORCH_V1.interface"
        },
        {
            "name": "camera_node",
            "interface": "panorama::abstract_rtsp_media_source.rtsp_v1_interface",
            "overridable": true,
            "overrideMandatory": true,
            "decorator": {
                "title": "IP camera",
                "description": "Choose a camera stream."
            }
        }
]
```

エッジ

エッジは、あるノードからの出力を別のノードの入力にマップします。次の例では、最初のエッジがカメラストリームノードからの出力をアプリケーションコードノードの入力にマッピングします。video_in と video_out の名前はノードパッケージのインターフェースで定義されています。

Example graph.json — エッジ

```
        "edges": [
            {
                "producer": "camera_node.video_out",
                "consumer": "code_node.video_in"
            },
            {
                "producer": "code_node.video_out",
                "consumer": "output_node.video_in"
            },
```

アプリケーションコードでは、inputs および outputs の属性を使用して入力ストリームから画像を取得し、画像を出力ストリームに送信します。

Example application.py — ビデオの入力と出力

```
    def process_streams(self):
        """Processes one frame of video from one or more video streams."""
        frame_start = time.time()
        self.frame_num += 1
        logger.debug(self.frame_num)
        # Loop through attached video streams
        streams = self.inputs.video_in.get()
        for stream in streams:
            self.process_media(stream)
        ...
        self.outputs.video_out.put(streams)
```

抽象ノード

アプリケーションマニフェストでは、抽象ノードは AWS Panorama によって定義されたパッケージを指し、アプリケーションマニフェストのプレースホルダーとして使用できます。AWS Panorama には、2 つのタイプの抽象ノードがあります。

- カメラストリーム — デプロイ時にアプリケーションが使用するカメラストリームを選択します。

  パッケージ名 - `panorama::abstract_rtsp_media_source`

  インターフェイス名 - `rtsp_v1_interface`

- HDMI 出力 — アプリケーションがビデオを出力することを示します。

  パッケージ名 - `panorama::hdmi_data_sink`

  インターフェイス名 - `hdmi0`

次の例は、カメラストリームを処理してビデオをディスプレイに出力するアプリケーションのパッケージ、ノード、エッジの基本セットを示しています。AWS Panorama `abstract_rtsp_media_source` のパッケージのインターフェースを使用するカメラノードは、複数のカメラストリームを入力として受け入れることができます。hdmi_data_sink を参照する出力ノードは、アプライアンスのHDMIポートから出力されるビデオバッファへのアクセスをアプリケーションコードに提供します。

Example graph.json — 抽象ノード

```
{
    "nodeGraph": {
        "envelopeVersion": "2021-01-01",
        "packages": [
            {
                "name": "123456789012::SAMPLE_CODE",
                "version": "1.0"
            },
            {
                "name": "123456789012::SQUEEZENET_PYTORCH_V1",
                "version": "1.0"
            },
            {
                "name": "panorama::abstract_rtsp_media_source",
                "version": "1.0"
            },
            {
                "name": "panorama::hdmi_data_sink",
                "version": "1.0"
            }
        ],
        "nodes": [
            {
                "name": "camera_node",
                "interface": "panorama::abstract_rtsp_media_source.rtsp_v1_interface",
                "overridable": true,
                "decorator": {
```

```
                    "title": "IP camera",
                    "description": "Choose a camera stream."
                }
            },
            {
                "name": "output_node",
                "interface": "panorama::hdmi_data_sink.hdmi0"
            }
        ],
        "edges": [
            {
                "producer": "camera_node.video_out",
                "consumer": "code_node.video_in"
            },
            {
                "producer": "code_node.video_out",
                "consumer": "output_node.video_in"
            }
        ]
    }
}
```

アプリケーションパラメータ

パラメータは基本タイプのノードで、デプロイ時にオーバーライドできます。パラメータにはデフォルト値と、アプリケーションのユーザーにその構成方法を指示するデコレータを構成できます。

パラメータタイプ

- `string` - 文字列。例えば、`DEBUG` です。
- `int32` - 整数。例えば、`20` などです。
- `float32` - 浮動小数点数。例えば、`47.5` などです。
- `boolean` – `true` または `false`

次の例は、文字列と数値の 2 つのパラメータを示しており、これらは入力としてコードノードに送信されます。

Example グラフ.json — パラメータ

```
       "nodes": [
           {
               "name": "detection_threshold",
               "interface": "float32",
               "value": 20.0,
               "overridable": true,
               "decorator": {
                   "title": "Threshold",
                   "description": "The minimum confidence percentage for a positive
classification."
               }
           },
           {
               "name": "log_level",
               "interface": "string",
               "value": "INFO",
               "overridable": true,
               "decorator": {
                   "title": "Logging level",
                   "description": "DEBUG, INFO, WARNING, ERROR, or CRITICAL."
               }
           }
           ...
```

```
        ],
        "edges": [
            {
                "producer": "detection_threshold",
                "consumer": "code_node.threshold"
            },
            {
                "producer": "log_level",
                "consumer": "code_node.log_level"
            }
            ...
        ]
    }
```

アプリケーションマニフェストでパラメータを直接変更することも、デプロイ時にオーバーライドを使用して新しい値を指定することもできます。詳細については、「オーバーライド付きのデプロイ時構成」を参照してください。

オーバーライド付きのデプロイ時構成

デプロイ時にパラメーターと抽象ノードを構成します。AWS Panorama コンソールを使用してデプロイする場合、各パラメータの値を指定し、入力としてカメラストリームを選択できます。AWS Panorama API を使用してアプリケーションをデプロイする場合は、オーバーライドドキュメントでこれらの設定を指定します。

オーバーライドドキュメントは、アプリケーションマニフェストと構造が似ています。基本型のパラメータには、ノードを定義します。カメラストリームでは、登録したカメラストリームにマップするノードとパッケージを定義します。次に、置き換えるアプリケーションマニフェストからノードを指定するオーバーライドをノードごとに定義します。

Example オーバーライド.json

```
{
    "nodeGraphOverrides": {
        "nodes": [
            {
                "name": "my_camera",
                "interface": "123456789012::exterior-south.exterior-south"
            },
            {
                "name": "my_region",
                "interface": "string",
                "value": "us-east-1"
            }
        ],
        "packages": [
            {
                "name": "123456789012::exterior-south",
                "version": "1.0"
            }
        ],
        "nodeOverrides": [
            {
                "replace": "camera_node",
                "with": [
                    {
                        "name": "my_camera"
                    }
                ]
            },
```

```
            {
                "replace": "region",
                "with": [
                    {
                        "name": "my_region"
                    }
                ]
            }
        ],
        "envelopeVersion": "2021-01-01"
    }
}
```

前の例では、ドキュメントは 1 つの文字列パラメータと 1 つの抽象カメラノードにオーバーライドを定義しています。nodeOverrides は、このドキュメントのどのノードがアプリケーションマニフェストのどのノードをオーバーライドするかを AWS Panorama に伝えます。

AWS Panorama アプリケーションの構築

アプリケーションは AWS Panorama アプライアンス上で動作し、ビデオストリーム上でコンピュータービジョンタスクを実行します。Pythonコードと機械学習モデルを組み合わせて、コンピュータビジョンアプリケーションをビルドし、インターネット経由で AWS Panorama アプライアンスにデプロイできます。アプリケーションはディスプレイに動画を送信したり、AWS SDK を使用して結果を AWS のサービスに送信したりできます。

モデル は画像を分析して人物、車両、その他の物体を検出します。トレーニング中に見た画像に基づいて、モデルはそれが何だと思っているか、その推測にどの程度自信があるのかを教えてくれます。独自の画像データを使用してモデルをトレーニングすることも、サンプルから始めることもできます。

アプリケーションの コード は、カメラストリームからの静止画像を処理してモデルに送信し、結果を処理します。モデルは複数のオブジェクトを検出し、その形状と位置を返す場合があります。コードはこの情報を使用してビデオにテキストやグラフィックを追加したり、結果を AWS サービスに送信して保存したり、さらなる処理を行うことができます。

ストリームから画像を取得したり、モデルとやりとりしたり、ビデオを出力したりするために、アプリケーションコードは AWS Panorama Application SDK を使用します。アプリケーション SDK は PyTorch、Apache MXNet、TensorFlow で生成されたモデルをサポートする Python ライブラリです。

トピック

コンピュータービジョンモデル

コンピュータビジョンモデルは、画像内のオブジェクトを検出するようにトレーニングされたソフトウェアプログラムです。モデルは、最初にトレーニングを通じてそれらのオブジェクトの画像を分析することで、一連のオブジェクトを認識することを学習します。コンピュータビジョンモデルは、画像を入力として受け取り、検出したオブジェクトに関する情報 (オブジェクトのタイプや位置など) を出力します。AWS Panorama は PyTorch、Apache MXNet、TensorFlow で構築されたコンピュータビジョンモデルをサポートしています。

> **Note**
>
> AWS Panorama でテストされたビルド済みモデルのリストについては、「モデルの互換性」を参照してください。

セクション

コード内でのモデルの使用

モデルは 1 つ以上の結果を返します。これには、検出されたクラスの確率、位置情報、その他のデータが含まれる場合があります。次の例は、ビデオストリームの画像に対して推論を実行し、モデルの出力を処理関数に送信する方法を示しています。

Example アプリケーション.py — 推論

```
    def process_media(self, stream):
        """Runs inference on a frame of video."""
        image_data = preprocess(stream.image,self.MODEL_DIM)
        logger.debug('Image data: {}'.format(image_data))
        # Run inference
        inference_start = time.time()
        inference_results = self.call({"data":image_data}, self.MODEL_NODE)
         # Log metrics
```

```
        inference_time = (time.time() - inference_start) * 1000
        if inference_time > self.inference_time_max:
            self.inference_time_max = inference_time
        self.inference_time_ms += inference_time
        # Process results (classification)
        self.process_results(inference_results, stream)
```

次の例は、基本分類モデルから結果を処理する関数を示しています。サンプルモデルは確率の配列を返します。これは結果配列の最初で唯一の値です。

Example アプリケーション.py — 処理結果

```
    def process_results(self, inference_results, stream):
        """Processes output tensors from a computer vision model and annotates a video
frame."""
        if inference_results is None:
            logger.warning("Inference results are None.")
            return
        max_results = 5
        logger.debug('Inference results: {}'.format(inference_results))
        class_tuple = inference_results[0]
        enum_vals = [(i, val) for i, val in enumerate(class_tuple[0])]
        sorted_vals = sorted(enum_vals, key=lambda tup: tup[1])
        top_k = sorted_vals[::-1][:max_results]
        indexes =  [tup[0] for tup in top_k]

        for j in range(max_results):
            label = 'Class [%s], with probability %.3f.'% (self.classes[indexes[j]],
class_tuple[0][indexes[j]])
            stream.add_label(label, 0.1, 0.1 + 0.1*j)
```

アプリケーションコードは確率が最も高い値を見つけて、初期化時に読み込まれるリソースファイル内のラベルにマッピングします。

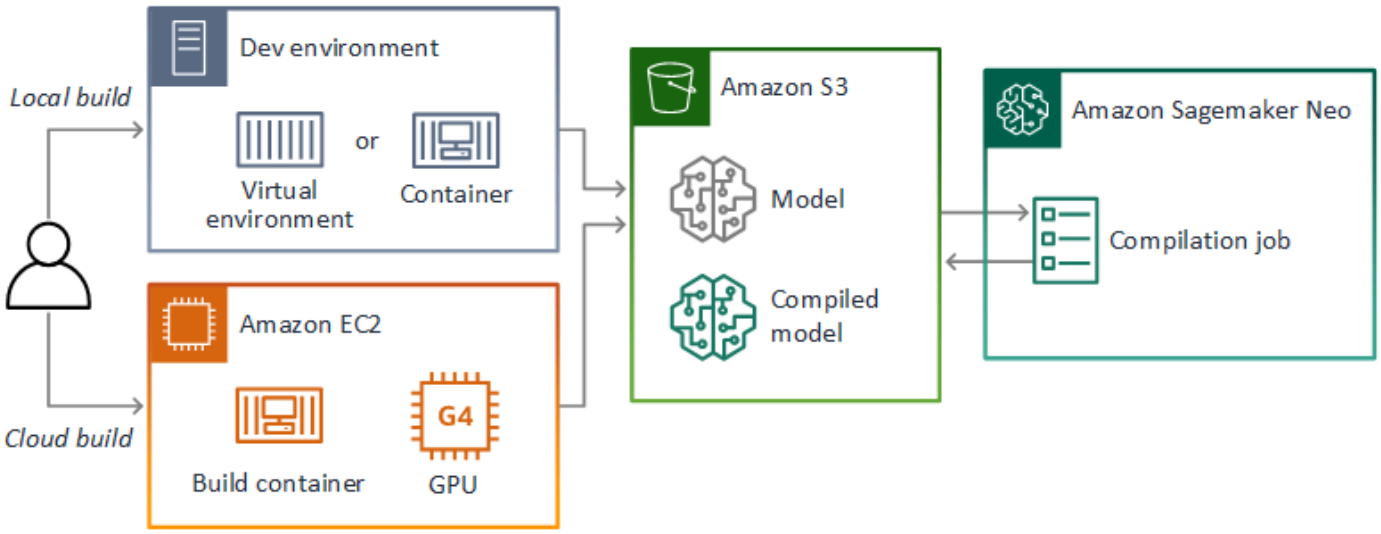
カスタムモデルの構築

PyTorch、Apache MXNet、TensorFlow で構築したモデルを AWS Panorama アプリケーションで使用できます。SageMaker でモデルを構築してトレーニングする代わりに、トレーニング済みのモデルを使用するか、サポートされているフレームワークを使用して独自のモデルを構築してトレーニングし、ローカル環境または Amazon EC2 にエクスポートできます。

> **Note**
>
> SageMaker Neo がサポートするフレームワークのバージョンとファイル形式の詳細については、Amazon SageMaker デベロッパーガイドの「サポートされているフレームワーク」を参照してください。

このガイドのリポジトリには、TensorFlow `SavedModel`形式の Keras モデルのこのワークフローを示すサンプルアプリケーションが用意されています。TensorFlow 2 を使用しており、仮想環境でローカルに実行することも、Docker コンテナで実行することもできます。サンプルアプリケーションには、Amazon EC2 インスタンスでモデルを構築するためのテンプレートとスクリプトも含まれています。

- カスタムモデルサンプルアプリケーション



AWS Panorama は SageMaker Neo を使用して AWS Panorama アプライアンスで使用するモデルをコンパイルします。フレームワークごとに、SageMaker Neo がサポートする形式を使用し、モデルを`.tar.gz`アーカイブにパッケージ化します。

詳細については、Amazon SageMaker デベロッパーガイドの「モデルをコンパイルしてデプロイする」を参照してください。

モデルのパッケージ化

モデルパッケージは、記述子、パッケージ構成、モデルアーカイブで構成されます。アプリケーションイメージパッケージと同様に、パッケージ構成は、モデルと記述子が Amazon S3 のどこに保存されているかを AWS Panorama サービスに伝えます。

Example packages/123456789012-SQUEEZENET_PYTORCH-1.0/descriptor.json

```
{
    "mlModelDescriptor": {
        "envelopeVersion": "2021-01-01",
        "framework": "PYTORCH",
        "frameworkVersion": "1.8",
        "precisionMode": "FP16",
        "inputs": [
            {
                "name": "data",
                "shape": [
                    1,
                    3,
                    224,
                    224
                ]
            }
        ]
    }
}
```

> **Note**
>
> フレームワークバージョンのメジャーバージョンとマイナーバージョンのみを指定してください。サポートされている PyTorch、Apache MXNet、および TensorFlow バージョンのリストについては、「サポートされているフレームワーク」を参照してください。

モデルをインポートするには、AWS Panorama アプリケーション CLI `import-raw-model` コマンドを使用します。モデルやその記述子に変更を加えた場合は、このコマンドを再実行してアプリケーションのアセットを更新する必要があります。詳細については、「コンピュータービジョンモデルの変更」を参照してください。

ディスクリプターファイルの JSON スキーマについては、「AssetDescriptor.schema.json」を参照してください。

モデルのトレーニング

モデルをトレーニングするときは、ターゲット環境、またはターゲット環境によく似たテスト環境のイメージを使用してください。モデルのパフォーマンスに影響を与える可能性がある以下の要因を考慮してください。

- 照明 — 被写体が反射する光の量によって、モデルがどの程度詳細に分析しなければならないかが決まります。明るい被写体の画像でトレーニングしたモデルは、暗い場所や逆光の環境ではうまく機能しない可能性があります。
- 解像度 — モデルの入力サイズは、通常、幅 224 ~ 512 ピクセル (縦横比) の解像度に固定されます。ビデオのフレームをモデルに渡す前に、必要なサイズに合うように縮小またはトリミングできます。
- 画像のゆがみ — カメラの焦点距離とレンズの形状により、画像がフレームの中心から離れるにつれて歪みが生じることがあります。また、カメラの位置によって、被写体のどの部分が見えるかが決まります。たとえば、広角レンズを搭載したオーバーヘッドカメラでは、被写体がフレームの中央にあるときは被写体の上部が表示され、中心から遠ざかるにつれて被写体の側面が歪んで表示されます。

このような問題に対処するには、画像をモデルに送る前に前処理を行い、実世界の環境の変化を反映するさまざまな画像でモデルをトレーニングできます。照明環境やさまざまなカメラでモデルを動作させる必要がある場合は、トレーニングのためにより多くのデータが必要になります。より多くの画像を収集するだけでなく、歪んだ画像や照明が異なる既存の画像のバリエーションを作成することで、より多くのトレーニングデータを取得できます。

アプリケーションイメージを構築する

AWS Panorama アプライアンス は、構築したイメージからエクスポートされたコンテナファイルシステムとしてアプリケーションを実行します。アプリケーションの依存関係とリソースは、AWS Panorama アプリケーションのベースイメージを開始点として使用する Dockerfile で指定します。

アプリケーションイメージを構築するには、Docker と AWS Panorama アプリケーション CLI を使用します。このガイドのサンプルアプリケーションの次の例は、これらのユースケースを示しています。

Example packages/123456789012-SAMPLE_CODE-1.0/Dockerfile

```
FROM public.ecr.aws/panorama/panorama-application
WORKDIR /panorama
COPY . .
RUN pip install --no-cache-dir --upgrade pip && \
    pip install --no-cache-dir -r requirements.txt
```

次の Dockerfile 指示を使用します。

- FROM — アプリケーションベースイメージ (public.ecr.aws/panorama/panorama-application) をロードします。
- WORKDIR — イメージに作業ディレクトリを設定します。 /panorama はアプリケーションコードおよび関連ファイルに使用されます。この設定は構築中のみ保持され、ランタイム (/) のアプリケーションの作業ディレクトリには影響しません。
- COPY — ファイルをローカルパスからイメージ上のパスにコピーします。COPY . . は現在のディレクトリ (パッケージディレクトリ) のファイルをイメージ上の作業ディレクトリにコピーします。たとえば、アプリケーションコードは packages/123456789012-SAMPLE_CODE-1.0/application.py から /panorama/application.py にコピーされます。
- RUN — 構築中にイメージ上でシェルコマンドを実行します。コマンド間に && を使用することで、1回の RUN 操作で複数のコマンドを順番に実行することができます。この例では、pip パッケージマネージャーを更新し、requirements.txt に記載されているライブラリをインストールします。

ADD や ARG など、構築時に役立つ他の指示も使用できます。ENV のような、ランタイム情報をコンテナに追加する指示は、AWS Panorama では機能しません。AWS Panorama はイメージからコンテ

ナを実行しません。イメージを使用してファイルシステムをエクスポートし、アプライアンスに転送するだけです。

依存関係の指定

`requirements.txt` は、アプリケーションが使用するライブラリを指定する Python 要件ファイルです。サンプルアプリケーションは Open CV と AWS SDK for Python (Boto3) を使用しています。

Example packages/123456789012-SAMPLE_CODE-1.0/requirements.txt

```
boto3==1.24.*
opencv-python==4.6.*
```

Dockerfile の `pip install` コマンドは、これらのライブラリを `/usr/local/lib` の Python `dist-packages` ディレクトリにインストールし、アプリケーションコードでインポートできるようにします。

ローカルストレージ

AWS Panorama はアプリケーションストレージ用の `/opt/aws/panorama/storage` ディレクトリを確保します。アプリケーションはこのパスでファイルを作成および変更できます。ストレージディレクトリに作成されたファイルは、再起動しても保持されます。他の一時ファイルの場所は起動時にクリアされます。

イメージアセットの構築

AWS Panorama Application CLI でアプリケーションパッケージのイメージを構築すると、CLI はパッケージディレクトリで `docker build` を実行します。これにより、アプリケーションコードを含むアプリケーションイメージが構築されます。次に、CLI はコンテナを作成し、そのファイルシステムをエクスポートして圧縮し、`assets` フォルダに保存します。

```
$ panorama-cli build-container --container-asset-name code_asset --package-path
 packages/123456789012-SAMPLE_CODE-1.0
docker build -t code_asset packages/123456789012-SAMPLE_CODE-1.0 --pull
docker export --output=code_asset.tar $(docker create code_asset:latest)
gzip -1 code_asset.tar
{
    "name": "code_asset",
    "implementations": [
        {
```

```
            "type": "container",
            "assetUri":
 "6f67xmpl32743ed0e60c151a02f2f0da1bf70a4ab9d83fe236fa32a6f9b9f808.tar.gz",
            "descriptorUri":
 "1872xmpl129481ed053c52e66d6af8b030f9eb69b1168a29012f01c7034d7a8f.json"
        }
    ]
}
Container asset for the package has been succesfully built at  /home/
user/aws-panorama-developer-guide/sample-apps/aws-panorama-sample/
assets/6f67xmpl32743ed0e60c151a02f2f0da1bf70a4ab9d83fe236fa32a6f9b9f808.tar.gz
```

出力の JSON ブロックは、CLI がパッケージ設定 (`package.json`) に追加して AWS Panorama サービスに登録するアセット定義です。CLI は、アプリケーションスクリプト (アプリケーションのエントリポイント) へのパスを指定する記述子ファイルもコピーします。

Example packages/123456789012-SAMPLE_CODE-1.0/descriptor.json

```
{
    "runtimeDescriptor":
    {
        "envelopeVersion": "2021-01-01",
        "entry":
        {
            "path": "python3",
            "name": "/panorama/application.py"
        }
    }
}
```

アセットフォルダーでは、記述子とアプリケーションイメージには SHA-256 チェックサムにちなんだ名前が付けられています。この名前は、アセットが Amazon S3 に保存されるときに、アセットの一意の識別子として使用されます。

アプリケーションコードから AWS サービスを呼び出す

AWS SDK for Python (Boto) を使用して、アプリケーションコードから AWS サービスを呼び出すことができます。たとえば、モデルが異常を検出した場合、Amazon CloudWatch にメトリクスを投稿したり、Amazon SNS で通知を送信したり、イメージを Amazon S3 に保存したり、Lambda 関数を呼び出してさらに処理したりすることができます。ほとんどの AWS サービスには、AWS SDK で使用できるパブリック API があります。

デフォルトでは、アプライアンスには AWS サービスへのアクセス権限がありません。アクセス権限を付与するには、アプリケーションのロールを作成し、デプロイ時にアプリケーションインスタンスに割り当てます。

セクション

- Amazon S3 の使用
- AWS IoT MQTT トピックを使用する

Amazon S3 の使用

Amazon S3 を使用して処理結果やその他のアプリケーションデータを保存できます。

```
import boto3
s3_client=boto3.client("s3")
s3_clients3.upload_file(data_file,
                    s3_bucket_name,
                    os.path.basename(data_file))
```

AWS IoT MQTT トピックを使用する

SDK for Python (Boto3) を使用して、AWS IoT の MQTT トピックにメッセージを送信できます。次の例では、アプリケーションは AWS IoT コンソールに表示されるアプライアンスの モノの名前にちなんだ名前のトピックに投稿します。

```
import boto3
iot_client=boto3.client('iot-data')
topic = "panorama/panorama_my-appliance_Thing_a01e373b"
iot_client.publish(topic=topic, payload="my message")
```

デバイス ID または任意のその他の識別子を示す名前を選択します。メッセージを公開するには、アプリケーションに `iot:Publish` を呼び出す権限が必要です。

MQTT キューをモニタリングするには

1. AWS IoT コンソールの Test (テスト) ページを開きます。
2. サブスクリプショントピックに、トピック名を入力します。例えば、`panorama/panorama_my-appliance_Thing_a01e373b` です。
3. トピックに登録する を選択します。

AWS Panorama アプリケーション SDK

AWS Panorama アプリケーション SDK は、AWS Panorama アプリケーションを開発するための Python ライブラリです。アプリケーションコードでは、AWS Panorama アプリケーション SDK を使用してコンピュータビジョンモデルをロードし、推論を実行し、ビデオをモニタに出力します。

> **Note**
>
> AWS Panorama アプリケーション SDK の最新機能にアクセスできるようにするには、アプライアンスソフトウェアをアップグレードしてください。

アプリケーション SDK が定義するクラスとそのメソッドの詳細については、「アプリケーション SDK リファレンス」を参照してください。

セクション

- 出力ビデオへのテキストとボックスの追加

出力ビデオへのテキストとボックスの追加

AWS Panorama SDK を使用すると、ビデオストリームをディスプレイに出力できます。動画には、モデルからの出力、アプリケーションの現在の状態、またはその他のデータを表示するテキストやボックスを含めることができます。

`video_in` 配列内の各オブジェクトは、アプライアンスに接続されたカメラストリームからの画像です。このオブジェクトのタイプは `panoramasdk.media` です。画像にテキストや長方形のボックスを追加して、`video_out` 配列に割り当てるメソッドがあります。

次の例で、サンプルアプリケーションは結果ごとに 1 つのラベルを追加します。各結果は同じ左位置に配置されますが、高さは異なります。

```
        for j in range(max_results):
            label = 'Class [%s], with probability %.3f.'% (self.classes[indexes[j]],
class_tuple[0][indexes[j]])
            stream.add_label(label, 0.1, 0.1 + 0.1*j)
```

出力イメージにボックスを追加するには、`add_rect` を使用します。このメソッドは、ボックスの左上隅と右下隅の位置を示す 0 から 1 までの値を 4 つ取ります。

```
        w,h,c = stream.image.shape
        stream.add_rect(x1/w, y1/h, x2/w, y2/h)
```

複数のスレッドの実行

アプリケーションロジックを処理スレッドで実行し、他のスレッドを他のバックグラウンドプロセスに使用することができます。たとえば、デバッグ用に HTTP トラフィックを処理するスレッドや、推論結果を監視してデータを AWS に送信するスレッドを作成できます。

複数のスレッドを実行するには、Python 標準ライブラリのスレッディングモジュールを使用して、プロセスごとにスレッドを作成します。次の例は、アプリケーションオブジェクトを作成し、それを使用して 3 つのスレッドを実行するデバッグサーバーのサンプルアプリケーションのメインループを示しています。

Example packages/123456789012-DEBUG_SERVER-1.0/application.py — メインループ

```
def main():
    panorama = panoramasdk.node()
    while True:
        try:
            # Instantiate application
            logger.info('INITIALIZING APPLICATION')
            app = Application(panorama)
            # Create threads for stream processing, debugger, and client
            app.run_thread = threading.Thread(target=app.run_cv)
            app.server_thread = threading.Thread(target=app.run_debugger)
            app.client_thread = threading.Thread(target=app.run_client)
            # Start threads
            logger.info('RUNNING APPLICATION')
            app.run_thread.start()
            logger.info('RUNNING SERVER')
            app.server_thread.start()
            logger.info('RUNNING CLIENT')
            app.client_thread.start()
            # Wait for threads to exit
            app.run_thread.join()
            app.server_thread.join()
            app.client_thread.join()
            logger.info('RESTARTING APPLICATION')
        except:
            logger.exception('Exception during processing loop.')
```

すべてのスレッドが終了すると、アプリケーションは自動的に再起動します。run_cv ループはカメラストリームからの画像を処理します。停止のシグナルを受け取ると、デバッガープロセスをシャッ

トダウンします。デバッガープロセスは HTTP サーバーを実行していますが、それ自体をシャットダウンすることはできません。各スレッドは独自のエラーを処理する必要があります。エラーが捕捉されログに記録されない場合、スレッドは静かに終了します。

Example packages/123456789012-DEBUG_SERVER-1.0/application.py — 処理ループ

```
    # Processing loop
    def run_cv(self):
        """Run computer vision workflow in a loop."""
        logger.info("PROCESSING STREAMS")
        while not self.terminate:
            try:
                self.process_streams()
                # turn off debug logging after 15 loops
                if logger.getEffectiveLevel() == logging.DEBUG and self.frame_num ==
15:
                    logger.setLevel(logging.INFO)
            except:
                logger.exception('Exception on processing thread.')
        # Stop signal received
        logger.info("SHUTTING DOWN SERVER")
        self.server.shutdown()
        self.server.server_close()
        logger.info("EXITING RUN THREAD")
```

スレッドはアプリケーションの `self` オブジェクトを介して通信します。アプリケーション処理ループを再開するには、デバッガースレッドが `stop` メソッドを呼び出します。このメソッドは、ほかのスレッドにシャットダウンを知らせる`terminate` 属性を設定します。

Example packages/123456789012-DEBUG_SERVER-1.0/application.py — 停止メソッド

```
    # Interrupt processing loop
    def stop(self):
        """Signal application to stop processing."""
        logger.info("STOPPING APPLICATION")
        # Signal processes to stop
        self.terminate = True
    # HTTP debug server
    def run_debugger(self):
        """Process debug commands from local network."""
        class ServerHandler(SimpleHTTPRequestHandler):
            # Store reference to application
```

```
            application = self
            # Get status
            def do_GET(self):
                """Process GET requests."""
                logger.info('Get request to {}'.format(self.path))
                if self.path == "/status":
                    self.send_200('OK')
                else:
                    self.send_error(400)
            # Restart application
            def do_POST(self):
                """Process POST requests."""
                logger.info('Post request to {}'.format(self.path))
                if self.path == '/restart':
                    self.send_200('OK')
                    ServerHandler.application.stop()
                else:
                    self.send_error(400)
```

インバウンドトラフィックへの対応

アプリケーションコードと一緒に HTTP サーバーを実行することで、アプリケーションをローカルでモニタリングまたはデバッグできます。外部トラフィックを処理するには、AWS Panorama アプライアンスのポートをアプリケーションコンテナのポートにマッピングします。

> **Important**
>
> デフォルトでは、AWS Panorama アプライアンスはどのポートでも着信トラフィックを受け付けません。アプライアンスのポートを開くことには、暗黙のセキュリティリスクがあります。この機能を使用するときは、外部トラフィックからアプライアンスを保護し、許可されたクライアントとアプライアンス間の通信を保護するための追加措置を講じる必要があります。
>
> このガイドに含まれるサンプルコードはデモンストレーションを目的としており、認証、承認、暗号化は実装されていません。

アプライアンスでは 8000 ~ 9000 の範囲のポートを開くことができます。これらのポートを開くと、ルーティング可能なすべてのクライアントからのトラフィックを受信できます。アプリケーションをデプロイするときは、開くポートを指定し、アプライアンス上のポートをアプリケーションコンテナのポートにマッピングします。アプライアンスソフトウェアはトラフィックをコンテナに転送し、要求者に応答を送り返します。リクエストは指定したアプライアンスポートで受信され、応答はランダムなエフェメラルポートに送信されます。

インバウンドポートの設定

ポートマッピングはアプリケーション設定の 3 つの場所で指定します。コードパッケージの `package.json` では、コードノードがリッスンするポートを `network` ブロック内で指定します。次の例では、ノードがポート 80 でリッスンすることを宣言しています。

Example packages/123456789012-debug_server-1.0/Package.json

```
            "outputs": [
                {
                    "description": "Video stream output",
                    "name": "video_out",
                    "type": "media"
                }
```

```
            ],
            "network": {
                "inboundPorts": [
                    {
                        "port": 80,
                        "description": "http"
                    }
                ]
            }
```

アプリケーションマニフェストでは、アプライアンスのポートをアプリケーションのコードコンテナのポートにマップするルーティングルールを宣言します。次の例では、デバイスのポート 8080 を code_node コンテナのポート 80 にマッピングするルールを追加します。

Example graphs/my-app/graph.json

```
        {
            "producer": "model_input_width",
            "consumer": "code_node.model_input_width"
        },
        {
            "producer": "model_input_order",
            "consumer": "code_node.model_input_order"
        }
    ],
    "networkRoutingRules": [
        {
            "node": "code_node",
            "containerPort": 80,
            "hostPort": 8080,
            "decorator": {
                "title": "Listener port 8080",
                "description": "Container monitoring and debug."
            }
        }
    ]
```

アプリケーションをデプロイするときは、AWS Panorama コンソールで同じルールを指定するか、CreateApplicationInstance API に渡されるオーバーライドドキュメントを使用して指定します。アプライアンスでポートを開きたいことを確認するために、デプロイ時にこの設定を提供する必要があります。

Example graphs/my-app/override.json

```
            {
                "replace": "camera_node",
                "with": [
                    {
                        "name": "exterior-north"
                    }
                ]
            }
        ],
        "networkRoutingRules":[
            {
                "node": "code_node",
                "containerPort": 80,
                "hostPort": 8080
            }
        ],
        "envelopeVersion": "2021-01-01"
    }
}
```

アプリケーションマニフェストで指定されているデバイスポートが別のアプリケーションで使用されている場合は、オーバーライドドキュメントを使用して別のポートを選択できます。

トラフィックを処理する

コンテナのポートが開いていると、ソケットを開いたり、サーバーを実行して受信リクエストを処理したりできます。この debug-server サンプルは、コンピュータービジョンのアプリケーションコードと一緒に稼働する HTTP サーバーの基本的な実装を示しています。

> **Important**
>
> このサンプル実装は、実稼働環境では安全ではありません。アプライアンスが攻撃に対して脆弱にならないようにするには、コードとネットワーク構成に適切なセキュリティ制御を実装する必要があります。

Example packages/123456789012-DEBUG_SERVER-1.0/application.py — HTTP サーバー

```
    # HTTP debug server
```

```
    def run_debugger(self):
        """Process debug commands from local network."""
        class ServerHandler(SimpleHTTPRequestHandler):
            # Store reference to application
            application = self
            # Get status
            def do_GET(self):
                """Process GET requests."""
                logger.info('Get request to {}'.format(self.path))
                if self.path == '/status':
                    self.send_200('OK')
                else:
                    self.send_error(400)
            # Restart application
            def do_POST(self):
                """Process POST requests."""
                logger.info('Post request to {}'.format(self.path))
                if self.path == '/restart':
                    self.send_200('OK')
                    ServerHandler.application.stop()
                else:
                    self.send_error(400)
            # Send response
            def send_200(self, msg):
                """Send 200 (success) response with message."""
                self.send_response(200)
                self.send_header('Content-Type', 'text/plain')
                self.end_headers()
                self.wfile.write(msg.encode('utf-8'))
        try:
            # Run HTTP server
            self.server = HTTPServer(("", self.CONTAINER_PORT), ServerHandler)
            self.server.serve_forever(1)
            # Server shut down by run_cv loop
            logger.info("EXITING SERVER THREAD")
        except:
            logger.exception('Exception on server thread.')
```

サーバーは /status パスで GET リクエストを受け付けて、アプリケーションに関する情報を取得します。また、アプリケーションを再起動するための /restart への POST リクエストも受け付けます。

この機能を実証するために、サンプルアプリケーションは別のスレッドで HTTP クライアントを実行します。クライアントは起動後すぐにローカルネットワーク経由で /status パスを呼び出し、数分後にアプリケーションを再起動します。

Example packages/123456789012-DEBUG_SERVER-1.0/application.py — HTTP クライアント

```
    # HTTP test client
    def run_client(self):
        """Send HTTP requests to device port to demnostrate debug server functions."""
        def client_get():
            """Get container status"""
            r = requests.get('http://{}:{}/status'.format(self.device_ip,
 self.DEVICE_PORT))
            logger.info('Response: {}'.format(r.text))
            return
        def client_post():
            """Restart application"""
            r = requests.post('http://{}:{}/restart'.format(self.device_ip,
 self.DEVICE_PORT))
            logger.info('Response: {}'.format(r.text))
            return
        # Call debug server
        while not self.terminate:
            try:
                time.sleep(30)
                client_get()
                time.sleep(300)
                client_post()
            except:
                logger.exception('Exception on client thread.')
        # stop signal received
        logger.info("EXITING CLIENT THREAD")
```

メインループはスレッドを管理し、終了時にアプリケーションを再起動します。

Example packages/123456789012-DEBUG_SERVER-1.0/application.py — メインループ

```
def main():
    panorama = panoramasdk.node()
    while True:
        try:
            # Instantiate application
            logger.info('INITIALIZING APPLICATION')
```

```
            app = Application(panorama)
            # Create threads for stream processing, debugger, and client
            app.run_thread = threading.Thread(target=app.run_cv)
            app.server_thread = threading.Thread(target=app.run_debugger)
            app.client_thread = threading.Thread(target=app.run_client)
            # Start threads
            logger.info('RUNNING APPLICATION')
            app.run_thread.start()
            logger.info('RUNNING SERVER')
            app.server_thread.start()
            logger.info('RUNNING CLIENT')
            app.client_thread.start()
            # Wait for threads to exit
            app.run_thread.join()
            app.server_thread.join()
            app.client_thread.join()
            logger.info('RESTARTING APPLICATION')
        except:
            logger.exception('Exception during processing loop.')
```

サンプルアプリケーションをデプロイするには、このガイドの GitHub リポジトリにある「手順」を参照してください。

GPU を使用する

AWS Panorama アプライアンス のグラフィックプロセッサ (GPU) にアクセスして GPU アクセラレーションライブラリを使用したり、アプリケーションコードで機械学習モデルを実行したりできます。GPU アクセスを有効にするには、アプリケーションコードコンテナを構築した後に、パッケージ設定に GPU アクセスを要件として追加します。

> **Important**
>
> GPU アクセスを有効にすると、アプライアンス上のどのアプリケーションでもモデルノードを実行できなくなります。セキュリティ上の理由から、アプライアンスが SageMaker Neo でコンパイルされたモデルを実行する場合、GPU アクセスは制限されます。GPU アクセスでは、モデルをアプリケーションコードノードで実行し、デバイス上のすべてのアプリケーションが GPU へのアクセスを共有する必要があります。

アプリケーションの GPU アクセスを有効にするには、AWS Panorama Application CLI でパッケージを構築した後に パッケージ設定 を更新します。次の例は、GPU アクセスをアプリケーションコードノードに追加する `requirements` ブロックを示しています。

Example 要件ブロックを含む package.json

```
{
    "nodePackage": {
        "envelopeVersion": "2021-01-01",
        "name": "SAMPLE_CODE",
        "version": "1.0",
        "description": "Computer vision application code.",
        "assets": [
            {
                "name": "code_asset",
                "implementations": [
                    {
                        "type": "container",
                        "assetUri":
 "eba3xmpl71aa387e8f89be9a8c396416cdb80a717bb32103c957a8bf41440b12.tar.gz",
                        "descriptorUri":
 "4abdxmpl5a6f047d2b3047adde44704759d13f0126c00ed9b4309726f6bb43400ba9.json",
                        "requirements": [
                            {
                                "type": "hardware_access",
```

```
                            "inferenceAccelerators": [
                                {
                                    "deviceType": "nvhost_gpu",
                                    "sharedResourcePolicy": {
                                        "policy" : "allow_all"
                                    }
                                }
                            ]
                        }
                    ]
                }
            ]
        }
    ],
    "interfaces": [
    ...
```

開発ワークフローのビルドステップとパッケージングステップの間にパッケージ設定を更新します。

GPU アクセスでアプリケーションをデプロイするには

1. アプリケーションコンテナを構築するには、 `build-container` コマンドを使用します。

   ```
   $ panorama-cli build-container --container-asset-name code_asset --package-path
    packages/123456789012-SAMPLE_CODE-1.0
   ```

2. `requirements` ブロックをパッケージ設定に追加します。
3. コンテナアセットとパッケージ設定をアップロードするには、`package-application` コマンドを使用します。

   ```
   $ panorama-cli package-application
   ```

4. アプリケーションをデプロイします。

GPU アクセスを使用するサンプルアプリケーションについては、aws-panorama-samples GitHub リポジトリをご覧ください。

Windows 開発環境でのセットアップ

AWS Panorama アプリケーションを構築するには、Docker、コマンドラインツール、および Python を使用します。Windows では、Linux と Ubuntu 用の Docker デスクトップと Windows サブシステムを使用して開発環境をセットアップできます。このチュートリアルでは、AWS Panorama ツールとサンプルアプリケーションでテストされた開発環境の設定プロセスを順を追って説明します。

セクション

前提条件

このチュートリアルに沿って作業するためには、Linux 2 用 Windows サブシステム (WSL 2) をサポートするバージョンの Windows が必要です。

- Windows 10 バージョン 1903 以降 (ビルド 18362 以降) または Windows 11
- Windows の機能
  - Windows Subsystem for Linux
  - Hyper-V
  - 仮想マシンのプラットフォーム

このチュートリアルは、次のソフトウェアバージョンで開発されました。

- Ubuntu 20.04
- Python 3.8.5
- Docker 20.10.8

WSL 2、Ubuntu でインストールする

Windows 10 バージョン 2004 以降 (ビルド 19041 以降) をお使いの場合は、次の PowerShell コマンドを使用して WSL 2 と Ubuntu 20.04 をインストールできます。

```
> wsl --install -d Ubuntu-20.04
```

古いバージョンの Windows の場合は、WSL 2 ドキュメントの指示に従ってください：古いバージョンの手動インストール手順

Docker をインストールする

Docker Desktop をインストールするには、hub.docker.com からインストーラーパッケージをダウンロードして実行します。問題が発生した場合は、Docker ウェブサイトの「Docker デスクトップ WSL 2 バックエンド」に記載されている指示に従ってください。

Docker Desktop を実行し、初回実行のチュートリアルに従ってサンプルコンテナを構築します。

> **Note**
>
> Docker Desktop はデフォルトのディストリビューションでのみ Docker を有効にします。このチュートリアルを実行する前に他の Linux ディストリビューションをインストールしていた場合は、「リソース」、「WSL 統合」の「Docker Desktop 設定」メニューで、新しくインストールした Ubuntu ディストリビューションで Docker を有効にしてください。

Ubuntu の設定

これで Ubuntu 仮想マシンで Docker コマンドを実行できるようになりました。コマンドラインターミナルを開くには、スタートメニューからディストリビューションを実行します。初めて実行するときは、管理者コマンドの実行に使用できるユーザー名とパスワードを設定します。

開発環境の設定を完了するには、仮想マシンのソフトウェアを更新し、ツールをインストールします。

仮想マシンを設定するには

1. Ubuntu に付属するソフトウェアを更新します。

   ```
   $ sudo apt update && sudo apt upgrade -y && sudo apt autoremove
   ```

2. apt で開発ツールをインストールします。

```
$ sudo apt install unzip python3-pip
```

3. pip で Python ライブラリをインストールします。

```
$ pip3 install awscli panoramacli
```

4. 新しいターミナルを開き、`aws configure` を実行して AWS CLI を設定します。

```
$ aws configure
```

   アクセスキーがない場合は、IAM コンソールで生成することができます。

最後に、サンプルアプリケーションをダウンロードしてインポートします。

サンプルアプリケーションを入手する

1. サンプルアプリケーションをダウンロードして解凍する。

```
$ wget https://github.com/awsdocs/aws-panorama-developer-guide/releases/download/
v1.0-ga/aws-panorama-sample.zip
$ unzip aws-panorama-sample.zip
$ cd aws-panorama-sample
```

2. 付属のスクリプトを実行して、コンパイルをテストし、アプリケーションコンテナを構築し、パッケージを AWS Panorama にアップロードします。

```
aws-panorama-sample$ ./0-test-compile.sh
aws-panorama-sample$ ./1-create-role.sh
aws-panorama-sample$ ./2-import-app.sh
aws-panorama-sample$ ./3-build-container.sh
aws-panorama-sample$ ./4-package-app.sh
```

AWS Panorama アプリケーション CLI はパッケージをアップロードし、AWS Panorama サービスに登録します。これで、AWS Panorama コンソールでサンプルアプリケーションをデプロイできます。

次のステップ

プロジェクトファイルを確認して編集するには、ファイルエクスプローラーまたは WSL をサポートする統合開発環境 (IDE) を使用できます。

仮想マシンのファイルシステムにアクセスするには、ファイルエクスプローラーを開き、ナビゲーションバーに「 `\\wsl$` 」と入力します。このディレクトリには、仮想マシンのファイルシステム (`Ubuntu-20.04`) と Docker のデータ用のファイルシステムへのリンクが含まれています。`Ubuntu-20.04` の下、ユーザーディレクトリは `home\`*`username`* にあります。

> **Note**
>
> Ubuntu 内から Windows インストール内のファイルにアクセスするには、`/mnt/c` ディレクトリに移動します。たとえば、 を実行することで、ダウンロードディレクトリにあるファイルを一覧表示できます。`ls /mnt/c/Users/`*`windows-username`*`/Downloads`

Visual Studio Code では、開発環境でアプリケーションコードを編集し、統合ターミナルでコマンドを実行できます。Visual Studio Code をインストールするには、code.visualstudio.com にアクセスしてください。インストール後、リモート WSL 拡張機能を追加します。

Windows ターミナルは、これまでコマンドを実行してきた標準 Ubuntu ターミナルの代わりとなるものです。複数のタブをサポートし、PowerShell、コマンドプロンプト、およびインストールするその他のさまざまな Linux のターミナルを実行できます。Ctrl+C や Ctrl+V を使ったコピーアンドペースト、クリック可能な URL、その他の便利な機能強化をサポートしています。Windows ターミナルをインストールするには、microsoft.com にアクセスしてください。

AWS Panorama API

AWS Panorama サービスのパブリック API を使用して、デバイスとアプリケーションの管理ワークフローを自動化できます。AWS Command Line Interface または AWS SDK を使用すると、リソースとデプロイを管理するスクリプトやアプリケーションを開発できます。このガイドの GitHub リポジトリには、独自のコードの開始点として使用できるスクリプトが含まれています。

- aws-panorama-開発者ガイド/util-scripts

セクション

- デバイス登録を自動化
- AWS Panorama API によるアプライアンスの管理
- アプリケーションのデプロイを自動化する
- AWS Panorama API によるアプリケーションの管理
- VPC エンドポイントの使用

デバイス登録を自動化

アプライアンスをプロビジョニングするには、プロビジョンデバイス API を使用します。レスポンスには、デバイスの構成と一時的な認証情報が記載された ZIP ファイルが含まれます。ファイルをデコードし、プレフィックス `certificates-omni_` を付けてアーカイブに保存します。

Example プロビジョン-デバイス.sh

```
if [[ $# -eq 1 ]] ; then
    DEVICE_NAME=$1
else
    echo "Usage: ./provision-device.sh <device-name>"
    exit 1
fi
CERTIFICATE_BUNDLE=certificates-omni_${DEVICE_NAME}.zip
aws panorama provision-device --name ${DEVICE_NAME} --output text --query Certificates
 | base64 --decode > ${CERTIFICATE_BUNDLE}
echo "Created certificate bundle ${CERTIFICATE_BUNDLE}"
```

構成アーカイブ内の認証情報は 5 分後に期限切れになります。付属の USB ドライブを使用してアーカイブをアプライアンスに転送します。

カメラを登録するには、テンプレートジョブからノードを作成 API を使用します。この API は、カメラのユーザー名、パスワード、URL のテンプレートパラメータのマップを取得します。バッシュの文字列操作を使用して、このマップを JSON ドキュメントとしてフォーマットできます。

Example 登録カメラ.sh

```
if [[ $# -eq 3 ]] ; then
    NAME=$1
    USERNAME=$2
    URL=$3
else
    echo "Usage: ./register-camera.sh <stream-name> <username> <rtsp-url>"
    exit 1
fi
echo "Enter camera stream password: "
read PASSWORD
TEMPLATE='{"Username":"MY_USERNAME","Password":"MY_PASSWORD","StreamUrl": "MY_URL"}'
TEMPLATE=${TEMPLATE/MY_USERNAME/$USERNAME}
TEMPLATE=${TEMPLATE/MY_PASSWORD/$PASSWORD}
TEMPLATE=${TEMPLATE/MY_URL/$URL}
```

```
echo ${TEMPLATE}
JOB_ID=$(aws panorama create-node-from-template-job --template-type RTSP_CAMERA_STREAM
 --output-package-name ${NAME} --output-package-version "1.0" --node-name ${NAME} --
template-parameters "${TEMPLATE}" --output text)
```

または、JSON 構成をファイルからロードすることもできます。

```
--template-parameters file://camera-template.json
```

AWS Panorama API によるアプライアンスの管理

AWS Panorama API を使用してアプライアンス管理ワークタスクを自動化します。

デバイスを表示

デバイス ID を持つアプライアンスのリストを取得するには、デバイス一覧 API を使用します。

```
$ aws panorama list-devices
    "Devices": [
        {
            "DeviceId": "device-4tafxmplhtmzabv5lsacba4ere",
            "Name": "my-appliance",
            "CreatedTime": 1652409973.613,
            "ProvisioningStatus": "SUCCEEDED",
            "LastUpdatedTime": 1652410973.052,
            "LeaseExpirationTime": 1652842940.0
        }
    ]
}
```

アプライアンスに関する詳細情報を取得するには、デバイスを説明 API を使用してください。

```
$ aws panorama describe-device --device-id device-4tafxmplhtmzabv5lsacba4ere
{
    "DeviceId": "device-4tafxmplhtmzabv5lsacba4ere",
    "Name": "my-appliance",
    "Arn": "arn:aws:panorama:us-west-2:123456789012:device/
device-4tafxmplhtmzabv5lsacba4ere",
    "Type": "PANORAMA_APPLIANCE",
    "DeviceConnectionStatus": "ONLINE",
    "CreatedTime": 1648232043.421,
    "ProvisioningStatus": "SUCCEEDED",
    "LatestSoftware": "4.3.55",
    "CurrentSoftware": "4.3.45",
    "SerialNumber": "GFXMPL0013023708",
    "Tags": {},
    "CurrentNetworkingStatus": {
        "Ethernet0Status": {
            "IpAddress": "192.168.0.1/24",
            "ConnectionStatus": "CONNECTED",
            "HwAddress": "8C:XM:PL:60:C5:88"
        },
```

```
        "Ethernet1Status": {
            "IpAddress": "--",
            "ConnectionStatus": "NOT_CONNECTED",
            "HwAddress": "8C:XM:PL:60:C5:89"
        }
    },
    "LeaseExpirationTime": 1652746098.0
}
```

アプライアンスソフトウェアを アップグレードする

LatestSoftware バージョンが CurrentSoftware よりも新しい場合は、デバイスをアップグレードできます。デバイスのためにジョブを作成 API を使用して無線通信 (OTA) 更新ジョブを作成します。

```
$ aws panorama create-job-for-devices --device-ids device-4tafxmplhtmzabv5lsacba4ere \
  --device-job-config '{"OTAJobConfig": {"ImageVersion": "4.3.55"}}' --job-type OTA
{
    "Jobs": [
        {
            "JobId": "device-4tafxmplhtmzabv5lsacba4ere-0",
            "DeviceId": "device-4tafxmplhtmzabv5lsacba4ere"
        }
    ]
}
```

スクリプトでは、ジョブ構成ファイルのイメージバージョンフィールドにバッシュ文字列操作を入力できます。

Example 更新のチェック.sh

```
apply_update() {
    DEVICE_ID=$1
    NEW_VERSION=$2
    CONFIG='{"OTAJobConfig": {"ImageVersion": "NEW_VERSION"}}'
    CONFIG=${CONFIG/NEW_VERSION/$NEW_VERSION}
    aws panorama create-job-for-devices --device-ids ${DEVICE_ID} --device-job-config
 "${CONFIG}" --job-type OTA
}
```

アプライアンスは指定されたソフトウェアバージョンをダウンロードし、自動的にアップデートします。デバイスジョブを説明 API を使用してアップデートの進行状況を監視します。

```
$ aws panorama describe-device-job --job-id device-4tafxmplhtmzabv5lsacba4ere-0
{
    "JobId": "device-4tafxmplhtmzabv5lsacba4ere-0",
    "DeviceId": "device-4tafxmplhtmzabv5lsacba4ere",
    "DeviceArn": "arn:aws:panorama:us-west-2:559823168634:device/
device-4tafxmplhtmzabv5lsacba4ere",
    "DeviceName": "my-appliance",
    "DeviceType": "PANORAMA_APPLIANCE",
    "ImageVersion": "4.3.55",
    "Status": "REBOOTING",
    "CreatedTime": 1652410232.465
}
```

実行中のすべてのジョブのリストを取得するには、デバイスジョブ一覧を使用してください。

```
$ aws panorama list-devices-jobs
{
    "DeviceJobs": [
        {
            "DeviceName": "my-appliance",
            "DeviceId": "device-4tafxmplhtmzabv5lsacba4ere",
            "JobId": "device-4tafxmplhtmzabv5lsacba4ere-0",
            "CreatedTime": 1652410232.465
        }
    ]
}
```

更新を確認して適用するサンプルスクリプトについては、このガイドの GitHub リポジトリにある 更新のチェック.sh を参照してください。

アプライアンスの再起動

アプライアンスを再起動するには、デバイスのためにジョブを作成 API を使用します。

```
$ aws panorama create-job-for-devices --device-ids device-4tafxmplhtmzabv5lsacba4ere --
job-type REBOOT
{
    "Jobs": [
        {
            "JobId": "device-4tafxmplhtmzabv5lsacba4ere-0",
            "DeviceId": "device-4tafxmplhtmzabv5lsacba4ere"
        }
```

```
    ]
}
```

スクリプトでは、デバイスのリストを取得し、そのうちの 1 つを選択してインタラクティブに再起動できます。

Example デバイスの再起動.sh — 使用方法

```
$ ./reboot-device.sh
Getting devices...
0: device-53amxmplyn3gmj72epzanacniy     my-se70-1
1: device-6talxmpl5mmik6qh5moba6jium     my-manh-24
Choose a device
1
Reboot device device-6talxmpl5mmik6qh5moba6jium? (y/n)y
{
    "Jobs": [
        {
            "DeviceId": "device-6talxmpl5mmik6qh5moba6jium",
            "JobId": "device-6talxmpl5mmik6qh5moba6jium-8"
        }
    ]
}
```

アプリケーションのデプロイを自動化する

アプリケーションをデプロイするには、AWS Panorama アプリケーション CLI AWS Command Line Interface との両方を使用します。アプリケーションコンテナを構築したら、それと他のアセットを Amazon S3 Access Points にアップロードします。次に、CreateApplicationInstance API を使用してアプリケーションをデプロイします。

示されているスクリプトの詳細と使用方法については、サンプルアプリケーションの README の指示に従ってください。

セクション

- コンテナを構築します
- コンテナをアップロードしてノードを登録します
- アプリケーションのデプロイ
- デプロイをモニタリングします

コンテナを構築します

アプリケーションコンテナを構築するには、`build-container` コマンドを使用します。このコマンドは Docker コンテナを構築し、圧縮ファイルシステムとして `assets` フォルダに保存します。

Example 3-build-container.sh

```
CODE_PACKAGE=SAMPLE_CODE
ACCOUNT_ID=$(aws sts get-caller-identity --output text --query 'Account')
panorama-cli build-container --container-asset-name code_asset --package-path packages/
${ACCOUNT_ID}-${CODE_PACKAGE}-1.0
```

また、パスの一部を入力して TAB を押すことで、コマンドライン補完を使ってパスの引数を埋めることもできます。

```
$ panorama-cli build-container --package-path packages/TAB
```

コンテナをアップロードしてノードを登録します

アプリケーションをアップロードするには、`package-application` コマンドを使用します。このコマンドは、`assets` フォルダから AWS Panorama が管理する Amazon S3 Access Points にアセットをアップロードします。

Example 4-package-app.sh

```
panorama-cli package-application
```

AWS Panorama アプリケーション CLI は、各パッケージのパッケージ設定 (package.json) によって参照されるコンテナアセットと記述子アセットをアップロードし、パッケージを AWS Panorama のノードとして登録します。次に、アプリケーションマニフェスト (graph.json) でこれらのノードを参照してアプリケーションをデプロイします。

アプリケーションのデプロイ

アプリケーションをデプロイするには、CreateApplicationInstance API を使用します。このアクションは、特に以下のパラメーターを取ります。

- ManifestPayload — アプリケーションのノード、パッケージ、エッジ、パラメータを定義するアプリケーションマニフェスト (graph.json)。
- ManifestOverridesPayload — 最初のマニフェストのパラメータをオーバーライドする 2 つ目のマニフェスト。アプリケーション マニフェストはアプリケーション ソース内の静的リソースとみなすことができ、オーバーライド マニフェストはデプロイをカスタマイズするデプロイ時設定を提供します。
- DefaultRuntimeContextDevice - ターゲットデバイス。
- RuntimeRoleArn — AWS サービスおよびリソースにアクセスするためにアプリケーションが使用する IAM ロールの ARN。
- ApplicationInstanceIdToReplace — デバイスから削除する既存のアプリケーションインスタンスの ID。

マニフェストペイロードとオーバーライドペイロードは JSON ドキュメントであり、別のドキュメント内にネストされた文字列値として提供する必要があります。そのために、スクリプトはマニフェストをファイルから文字列として読み込み、jq ツールを使用してネストされたドキュメントをコンストラクトします。

Example 5-deploy.sh — マニフェストを作成

```
GRAPH_PATH="graphs/my-app/graph.json"
OVERRIDE_PATH="graphs/my-app/override.json"
```

```
# application manifest
GRAPH=$(cat ${GRAPH_PATH} | tr -d '\n' | tr -d '[:blank:]')
MANIFEST="$(jq --arg value "${GRAPH}" '.PayloadData="\($value)"' <<< {})"
# manifest override
OVERRIDE=$(cat ${OVERRIDE_PATH} | tr -d '\n' | tr -d '[:blank:]')
MANIFEST_OVERRIDE="$(jq --arg value "${OVERRIDE}" '.PayloadData="\($value)"' <<< {})"
```

デプロイスクリプトは ListDevices API を使用して現在のリージョンに登録されているデバイスのリストを取得し、ユーザーの選択を今後のデプロイ用にローカルファイルに保存します。

Example 5-deploy.sh — デバイスを検出

```
    echo "Getting devices..."
    DEVICES=$(aws panorama list-devices)
    DEVICE_NAMES=($((echo ${DEVICES} | jq -r '.Devices |=sort_by(.LastUpdatedTime) |
 [.Devices[].Name] | @sh') | tr -d \'\"))
    DEVICE_IDS=($((echo ${DEVICES} | jq -r '.Devices |=sort_by(.LastUpdatedTime) |
 [.Devices[].DeviceId] | @sh') | tr -d \'\"))
    for (( c=0; c<${#DEVICE_NAMES[@]}; c++ ))
    do
        echo "${c}: ${DEVICE_IDS[${c}]}     ${DEVICE_NAMES[${c}]}"
    done
    echo "Choose a device"
    read D_INDEX
    echo "Deploying to device ${DEVICE_IDS[${D_INDEX}]}"
    echo -n ${DEVICE_IDS[${D_INDEX}]} > device-id.txt
    DEVICE_ID=$(cat device-id.txt)
```

アプリケーションロールは別のスクリプト (1-create-role.sh) によって作成されます。デプロイスクリプトは、このロールの ARN を AWS CloudFormation から取得します。アプリケーションが既にデバイスにデプロイされている場合、スクリプトはローカルファイルからそのアプリケーションインスタンスの ID を取得します。

Example 5-deploy.sh — ロール ARN と置換引数

```
# application role
STACK_NAME=panorama-${NAME}
ROLE_ARN=$(aws cloudformation describe-stacks --stack-name panorama-${PWD##*/} --query
 'Stacks[0].Outputs[?OutputKey==`roleArn`].OutputValue' --output text)
ROLE_ARG="--runtime-role-arn=${ROLE_ARN}"
```

```
# existing application instance id
if [ -f "application-id.txt" ]; then
    EXISTING_APPLICATION=$(cat application-id.txt)
    REPLACE_ARG="--application-instance-id-to-replace=${EXISTING_APPLICATION}"
    echo "Replacing application instance ${EXISTING_APPLICATION}"
fi
```

最後に、このスクリプトはすべての要素をまとめてアプリケーションインスタンスを作成し、アプリケーションをデバイスにデプロイします。サービスは、後で使用するためにスクリプトが保存するインスタンス ID で応答します。

Example 5-deploy.sh — アプリケーションをデプロイ

```
APPLICATION_ID=$(aws panorama create-application-instance ${REPLACE_ARG} --manifest-
payload="${MANIFEST}" --default-runtime-context-device=${DEVICE_ID} --name=${NAME}
 --description="command-line deploy" --tags client=sample --manifest-overrides-
payload="${MANIFEST_OVERRIDE}" ${ROLE_ARG} --output text)
echo "New application instance ${APPLICATION_ID}"
echo -n $APPLICATION_ID > application-id.txt
```

デプロイをモニタリングします

デプロイをモニタリングするには、ListApplicationInstances API を使用します。モニタースクリプトは、アプリケーションディレクトリ内のファイルからデバイス ID とアプリケーションインスタンス ID を取得し、それらを使用して CLI コマンドをコンストラクトします。その後、ループで呼び出されます。

Example 6-monitor-deployment.sh

```
APPLICATION_ID=$(cat application-id.txt)
DEVICE_ID=$(cat device-id.txt)
QUERY="ApplicationInstances[?ApplicationInstanceId==\`APPLICATION_ID\`]"
QUERY=${QUERY/APPLICATION_ID/$APPLICATION_ID}
MONITOR_CMD="aws panorama list-application-instances --device-id ${DEVICE_ID} --query
 ${QUERY}"
MONITOR_CMD=${MONITOR_CMD/QUERY/$QUERY}
while true; do
    $MONITOR_CMD
    sleep 60
done
```

デプロイが完了すると、Amazon CloudWatch Logs API を呼び出してログを表示できます。ログ表示スクリプトは CloudWatch Logs `GetLogEvents` API を使用します。

Example view-logs.sh

```
GROUP="/aws/panorama/devices/MY_DEVICE_ID/applications/MY_APPLICATION_ID"
GROUP=${GROUP/MY_DEVICE_ID/$DEVICE_ID}
GROUP=${GROUP/MY_APPLICATION_ID/$APPLICATION_ID}
echo "Getting logs for group ${GROUP}."
#set -x
while true
do
    LOGS=$(aws logs get-log-events --log-group-name ${GROUP} --log-stream-name
 code_node --limit 150)
    readarray -t ENTRIES < <(echo $LOGS | jq -c '.events[].message')
    for ENTRY in "${ENTRIES[@]}"; do
        echo "$ENTRY" | tr -d \"
    done
    sleep 20
done
```

AWS Panorama API によるアプリケーションの管理

AWS Panorama API を使用して、アプリケーションをモニタリングおよび管理できます。

アプリケーションの表示

アプライアンスで実行されているアプリケーションのリストを取得するには、ListApplicationInstances API を使用します。

```
$ aws panorama list-application-instances
    "ApplicationInstances": [
        {
            "Name": "aws-panorama-sample",
            "ApplicationInstanceId": "applicationInstance-ddaxxmpl2z7bg74ywutd7byxuq",
            "DefaultRuntimeContextDevice": "device-4tafxmplhtmzabv5lsacba4ere",
            "DefaultRuntimeContextDeviceName": "my-appliance",
            "Description": "command-line deploy",
            "Status": "DEPLOYMENT_SUCCEEDED",
            "HealthStatus": "RUNNING",
            "StatusDescription": "Application deployed successfully.",
            "CreatedTime": 1661902051.925,
            "Arn": "arn:aws:panorama:us-east-2:123456789012:applicationInstance/
applicationInstance-ddaxxmpl2z7bg74ywutd7byxuq",
            "Tags": {
                "client": "sample"
            }
        },
    ]
}
```

アプリケーションインスタンスのノードに関する詳細情報を取得するには、ListApplicationInstanceNodeInstances API を使用してください。

```
$ aws panorama list-application-instance-node-instances --application-instance-id
 applicationInstance-ddaxxmpl2z7bg74ywutd7byxuq
{
    "NodeInstances": [
        {
            "NodeInstanceId": "code_node",
            "NodeId": "SAMPLE_CODE-1.0-fd3dxmpl-interface",
            "PackageName": "SAMPLE_CODE",
```

```
            "PackageVersion": "1.0",
            "PackagePatchVersion":
 "fd3dxmpl2bdfa41e6fe1be290a79dd2c29cf014eadf7416d861ce7715ad5e8a8",
            "NodeName": "interface",
            "CurrentStatus": "RUNNING"
        },
        {
            "NodeInstanceId": "camera_node_override",
            "NodeId": "warehouse-floor-1.0-9eabxmpl-warehouse-floor",
            "PackageName": "warehouse-floor",
            "PackageVersion": "1.0",
            "PackagePatchVersion":
 "9eabxmple89f0f8b2f2852cca2a6e7971aa38f1629a210d069045e83697e42a7",
            "NodeName": "warehouse-floor",
            "CurrentStatus": "RUNNING"
        },
        {
            "NodeInstanceId": "output_node",
            "NodeId": "hdmi_data_sink-1.0-9c23xmpl-hdmi0",
            "PackageName": "hdmi_data_sink",
            "PackageVersion": "1.0",
            "PackagePatchVersion":
 "9c23xmplc4c98b92baea4af676c8b16063d17945a3f6bd8f83f4ff5aa0d0b394",
            "NodeName": "hdmi0",
            "CurrentStatus": "RUNNING"
        },
        {
            "NodeInstanceId": "model_node",
            "NodeId": "SQUEEZENET_PYTORCH-1.0-5d3cabda-interface",
            "PackageName": "SQUEEZENET_PYTORCH",
            "PackageVersion": "1.0",
            "PackagePatchVersion":
 "5d3cxmplb7113faa1d130f97f619655d8ca12787c751851a0e155e50eb5e3e96",
            "NodeName": "interface",
            "CurrentStatus": "RUNNING"
        }
    ]
}
```

カメラストリームの管理

SignalApplicationInstanceNodeInstances API を使用してカメラストリームノードを一時停止および再開できます。

```
$ aws panorama signal-application-instance-node-instances --application-instance-id
 applicationInstance-ddaxxmpl2z7bg74ywutd7byxuq \
       --node-signals '[{"NodeInstanceId": "camera_node_override", "Signal":
 "PAUSE"}]'
{
    "ApplicationInstanceId": "applicationInstance-ddaxxmpl2z7bg74ywutd7byxuq"
}
```

スクリプトでは、ノードのリストを取得し、インタラクティブに一時停止または再開するノードを 1 つ選択できます。

Example pause-camera.sh — 使用方法

```
my-app$ ./pause-camera.sh

Getting nodes...
0: SAMPLE_CODE              RUNNING
1: warehouse-floor          RUNNING
2: hdmi_data_sink           RUNNING
3: entrance-north           PAUSED
4: SQUEEZENET_PYTORCH       RUNNING
Choose a node
1
Signalling node warehouse-floor
+ aws panorama signal-application-instance-node-instances --application-instance-id
 applicationInstance-r3a7xmplcbmpjqeds7vj4b6pjy --node-signals '[{"NodeInstanceId":
 "warehouse-floor", "Signal": "PAUSE"}]'
{
    "ApplicationInstanceId": "applicationInstance-r3a7xmplcbmpjqeds7vj4b6pjy"
}
```

カメラノードを一時停止して再開することで、同時に処理できる数よりも多くのカメラストリームをサイクルさせることができます。そのためには、複数のカメラストリームをオーバーライドマニフェストの同じ入力ノードにマップします。

次の例では、オーバーライドマニフェストは warehouse-floor と entrance-north 、2 つのカメラストリームを同じ入力ノード (camera_node) にマップします。アプリケーションが起動し、entrance-north ノードが信号がオンになるのを待つと、warehouse-floor ストリームはアクティブになります。

Example override-multicam.json

```
    "nodeGraphOverrides": {
        "nodes": [
            {
                "name": "warehouse-floor",
                "interface": "123456789012::warehouse-floor.warehouse-floor",
                "launch": "onAppStart"
            },
            {
                "name": "entrance-north",
                "interface": "123456789012::entrance-north.entrance-north",
                "launch": "onSignal"
            },
        ...
        "packages": [
            {
                "name": "123456789012::warehouse-floor",
                "version": "1.0"
            },
            {
                "name": "123456789012::entrance-north",
                "version": "1.0"
            }
        ],
        "nodeOverrides": [
            {
                "replace": "camera_node",
                "with": [
                    {
                        "name": "warehouse-floor"
                    },
                    {
                        "name": "entrance-north"
                    }
                ]
            }
```

API でのデプロイの詳細については、アプリケーションのデプロイを自動化する を参照してください。

VPC エンドポイントの使用

インターネットにアクセスできない VPC で作業する場合は、AWS Panorama で使用する VPC エンドポイントを作成できます。VPC エンドポイントを使用すると、プライベートサブネットで実行されているクライアントは、インターネットに接続しなくても AWS サービスに接続できます。

AWS Panorama アプライアンスで使用されるポートとエンドポイントの詳細については、「 ??? 」を参照してください。

セクション

- VPC エンドポイントの作成
- アプライアンスをプライベートサブネットに接続する
- サンプル AWS CloudFormation テンプレート

VPC エンドポイントの作成

VPC とAWS Panorama とのプライベート接続を確立するには、VPC エンドポイントを作成します。AWS Panorama を使用するために VPC エンドポイントは必要ありません。VPC エンドポイントを作成する必要があるのは、インターネットにアクセスできない VPC で作業する場合だけです。AWS CLI または SDK が AWS Panorama に接続しようとすると、トラフィックは VPC エンドポイントを経由してルーティングされます。

次の設定を使用して AWS Panorama の VPC エンドポイントを作成します。

- サービス名 - **com.amazonaws.*us-west-2*.panorama**
- タイプ — インターフェイス

VPC エンドポイントはサービスの DNS 名を使用して、追加の設定なしで AWS SDK クライアントからのトラフィックを取得します。VPC エンドポイントの使い方の詳細については、「Amazon VPC ユーザーガイド」の「インターフェイス VPC エンドポイント 」を参照してください。

アプライアンスをプライベートサブネットに接続する

AWS Panorama アプライアンスは、AWS Site-to-Site VPN または AWS Direct Connect との AWS プライベート VPN 接続を介してに接続できます。これらのサービスを使用すると、データセンターにまで及ぶプライベートサブネットを作成できます。アプライアンスはプライベートサブネットに接続し、VPC エンドポイントを介して AWS サービスにアクセスします。

Site-to-Site VPN と AWS Direct Connect は、データセンターを Amazon VPC に安全に接続するためのサービスです。Site-to-Site VPN では、市販のネットワークデバイスを使用して接続することができます。 AWS Direct Connect は AWS デバイスを使用して接続します。

- Site-to-Site VPN - AWS Site-to-Site VPN とは何か？
- AWS Direct Connect - AWS Direct Connect とは何か？

ローカルネットワークを VPC 内のプライベートサブネットに接続したら、次のサービスの VPC エンドポイントを作成します。

- Amazon Simple Storage Service – AWS PrivateLink for Amazon S3
- AWS IoT Core — AWS IoT Core をインターフェイス VPC エンドポイント (データプレーンと認証情報プロバイダー) で使用する
- Amazon Elastic コンテナレジストリ — Amazon Elastic コンテナレジストリインターフェイス VPC エンドポイント
- Amazon CloudWatch - インターフェイス VPC エンドポイント で CloudWatch を使用する
- Amazon CloudWatch Logs - インターフェイス VPC エンドポイントで CloudWatch Logs を使用する

アプライアンスは AWS Panorama サービスに接続する必要はありません。AWS IoT のメッセージングチャネルを通じて AWS Panorama と通信します。

VPC エンドポイントに加えて、Amazon S3 および AWS IoT には Amazon Route 53 プライベートホストゾーンを使用する必要があります。プライベートホストゾーンは、Amazon S3 Access Points のサブドメインや MQTT トピックを含むサブドメインからのトラフィックを正しい VPC エンドポイントにルーティングします。プライベートホストゾーンの詳細については、「 Amazon Route 53 デベロッパーガイド」の「プライベートホストゾーンの使用」を参照してください。

VPC エンドポイントとプライベートホストゾーンを使用した VPC 設定のサンプルについては、サンプル AWS CloudFormation テンプレート を参照してください。

サンプル AWS CloudFormation テンプレート

このガイドの GitHub レポジトリには、AWS Panorama で使用するリソースを作成するために利用できる AWS CloudFormation テンプレートが用意されています。テンプレートでは、2 つのプライ

ベートサブネット、パブリックサブネット、および VPC エンドポイントを持つ VPC を作成します。VPC のプライベートサブネットを使用して、インターネットから隔離されたリソースをホストできます。パブリックサブネット内のリソースはプライベートリソースと通信できますが、プライベートリソースにはインターネットからアクセスできません。

Example vpc-endpoint.yml — プライベートサブネット

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  vpc:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: 172.31.0.0/16
      EnableDnsHostnames: true
      EnableDnsSupport: true
      Tags:
        - Key: Name
          Value: !Ref AWS::StackName
  privateSubnetA:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref vpc
      AvailabilityZone:
        Fn::Select:
         - 0
         - Fn::GetAZs: ""
      CidrBlock: 172.31.3.0/24
      MapPublicIpOnLaunch: false
      Tags:
        - Key: Name
          Value: !Sub  ${AWS::StackName}-subnet-a
  ...
```

vpc-endpoint.yml テンプレートでは、AWS Panorama 用の VPC エンドポイントを作成する方法を示します。このエンドポイントを使用して、AWS SDK または AWS CLI で AWS Panorama リソースを管理できます。

Example vpc-endpoint.yml — VPC エンドポイント

```
  panoramaEndpoint:
    Type: AWS::EC2::VPCEndpoint
    Properties:
```

```
      ServiceName: !Sub com.amazonaws.${AWS::Region}.panorama
      VpcId: !Ref vpc
      VpcEndpointType: Interface
      SecurityGroupIds:
      - !GetAtt vpc.DefaultSecurityGroup
      PrivateDnsEnabled: true
      SubnetIds:
      - !Ref privateSubnetA
      - !Ref privateSubnetB
      PolicyDocument:
        Version: 2012-10-17
        Statement:
        - Effect: Allow
          Principal: "*"
          Action:
            - "panorama:*"
          Resource:
            - "*"
```

PolicyDocument は、エンドポイントで実行できる API 呼び出しを定義するリソースベースのアクセス権限ポリシーです。ポリシーを変更して、エンドポイントからアクセスできるアクションとリソースを制限できます。詳細については、Amazon VPC ユーザーガイドのVPC エンドポイントによるサービスのアクセスコントロールを参照してください。

vpc-appliance.yml テンプレートは、AWS Panorama アプライアンス が使用するサービス用の VPC エンドポイントとプライベートホストゾーンを作成する方法を示しています。

Example vpc-appliance.yml — プライベートホストゾーンを持つ Amazon S3 Access Points エンドポイント

```
  s3Endpoint:
    Type: AWS::EC2::VPCEndpoint
    Properties:
      ServiceName: !Sub com.amazonaws.${AWS::Region}.s3
      VpcId: !Ref vpc
      VpcEndpointType: Interface
      SecurityGroupIds:
      - !GetAtt vpc.DefaultSecurityGroup
      PrivateDnsEnabled: false
      SubnetIds:
      - !Ref privateSubnetA
      - !Ref privateSubnetB
...
```

```
  s3apHostedZone:
    Type: AWS::Route53::HostedZone
    Properties:
      Name: !Sub s3-accesspoint.${AWS::Region}.amazonaws.com
      VPCs:
        - VPCId: !Ref vpc
          VPCRegion: !Ref AWS::Region
  s3apRecords:
    Type: AWS::Route53::RecordSet
    Properties:
      HostedZoneId: !Ref s3apHostedZone
      Name: !Sub "*.s3-accesspoint.${AWS::Region}.amazonaws.com"
      Type: CNAME
      TTL: 600
      # first DNS entry, split on :, second value
      ResourceRecords:
      - !Select [1, !Split [":", !Select [0, !GetAtt s3Endpoint.DnsEntries ] ] ]
```

サンプルテンプレートでは、サンプル VPC を使用して Amazon VPC と Route 53 リソースを作成する方法について説明しています。VPC リソースを削除し、サブネット、セキュリティグループ、VPC ID への参照をリソースの ID に置き換えることで、これらをユースケースに適合させることができます。

サンプルアプリケーション、スクリプト、テンプレート

このガイドの GitHub リポジトリには、AWS Panorama デバイス用のサンプルアプリケーション、スクリプト、およびテンプレートが用意されています。これらのサンプルを使ってベストプラクティスを学び、開発ワークフローを自動化します。

セクション

サンプルアプリケーション

サンプルアプリケーションでは、AWS Panorama 機能の使用方法や一般的なコンピュータービジョンタスクを示しています。これらのサンプルアプリケーションには、セットアップとデプロイを自動化するスクリプトとテンプレートが含まれています。最小限の設定で、コマンドラインからアプリケーションのデプロイと更新を実行できます。

- aws-panorama-sample — 分類モデルを使用した基本的なコンピュータービジョン。AWS SDK for Python (Boto) を使用して、CloudWatch へのメトリクスのアップロード、前処理メソッドと推論メソッドのインストゥルメント、ロギングの設定を行います。
- debug-server — デバイスのインバウンドポートを開き、トラフィックをアプリケーションコードコンテナに転送します。マルチスレッドを使用して、アプリケーションコード、HTTP サーバー、HTTP クライアントを同時に実行します。
- custom-model — コードからモデルをエクスポートし、SageMaker Neo でコンパイルして AWS Panorama アプライアンスとの互換性をテストします。Python 開発環境、Docker コンテナ、または Amazon EC2 インスタンスでローカルに構築します。特定の TensorFlow または Python バージョン用に、すべての組み込みアプリケーションモデルを Keras にエクスポートしてコンパイルします。

その他のサンプルアプリケーションについては、aws-panorama-samples リポジトリもご覧ください。

ユーティリティスクリプト

`util-scripts` ディレクトリ内のスクリプトは、AWS Panorama リソースを管理したり、開発ワークフローを自動化したりします。

- provision-device.sh — デバイスをプロビジョニングします。
- check-updates.sh — アプライアンスのソフトウェアアップデートを確認して適用します。
- reboot-device.sh — デバイスを再起動します。
- register-camera.sh — カメラを登録します。
- deregister-camera.sh — カメラノードを削除します。
- view-logs.sh — アプリケーションインスタンスのログを表示します。
- pause-camera.sh — カメラストリームを一時停止または再開します。
- push.sh — アプリケーションを構築、アップロード、デプロイします。
- rename-package.sh — ノードパッケージの名前を変更します。ディレクトリ名、設定ファイル、アプリケーションマニフェストを更新します。
- samplify.sh — アカウント ID をサンプルアカウント ID に置き換え、バックアップ設定を復元してローカル設定を削除します。
- update-model-config.sh — 記述子ファイルを更新した後に、モデルをアプリケーションに再度追加します。
- cleanup-patches.sh — 古いパッチバージョンの登録を解除し、そのマニフェストを Amazon S3 から削除します。

使用方法の詳細については、「README」を参照してください。

AWS CloudFormation テンプレート

`cloudformation-templates` ディレクトリ内の AWS CloudFormation テンプレートを使用して、AWS Panorama アプリケーション用のリソースを作成します。

- alarm-application.yml — アプリケーションのエラーをモニタリングするアラームを作成します。アプリケーションインスタンスでエラーが発生したり、5 分間実行が停止したりすると、アラームは通知メールを送信します。

- alarm-device.yml — デバイスの接続をモニタリングするアラームを作成します。デバイスがメトリクスの送信を 5 分間停止すると、アラームは通知メールを送信します。
- application-role.yml — アプリケーションロールを作成します。ロールには CloudWatch にメトリクスを送信する権限が含まれます。アプリケーションが使用する他の API オペレーションの権限をポリシーステートメントに追加します。
- vpc-appliance.yml — AWS Panorama アプライアンスのプライベートサブネットサービスアクセスを含む VPC を作成します。アプライアンスを VPC に接続するには、AWS Direct Connect または AWS Site-to-Site VPN を使用します。
- vpc-endpoint.yml — プライベートサブネットサービスが AWS Panorama サービスにアクセスできる VPC を作成します。VPC 内のリソースはインターネットに接続することなく、AWS Panorama に接続して AWS Panorama リソースのモニタリングと管理ができます。

このディレクトリの `create-stack.sh` スクリプトは AWS CloudFormation スタックを作成します。引数の数は可変です。最初の引数はテンプレートの名前で、残りの引数はテンプレートのパラメーターのオーバーライドです。

例えば、以下のコマンドはアプリケーションロールを作成します。

```
$ ./create-stack.sh application-role
```

その他のサンプルとツール

aws-panorama-samples リポジトリには、さらに多くのサンプルアプリケーションと便利なツールがあります。

- アプリケーション — さまざまなモデルアーキテクチャとユースケースに対応するサンプルアプリケーション。
- カメラストリームの検証 — カメラストリームを検証します。
- PanoJupyter — AWS Panorama アプライアンスで JupyterLab を実行します。
- サイドローディング — アプリケーションコンテナを構築またはデプロイせずにアプリケーションコードを更新します。

AWS コミュニティでは、AWS Panorama のためのツールやガイダンスも開発しています。GitHub で次のオープンソースプロジェクトをチェックしてください。

- cookiecutter-panorama — AWS Panorama アプリケーション用のクッキーカッターテンプレート。
- バックパック — ランタイム環境の詳細、プロファイリング、その他のビデオ出力オプションにアクセスするための Python モジュール。

モニタリング AWS Panorama リソースとアプリケーション

AWS Panorama リソースは AWS Panorama コンソール内と Amazon CloudWatch でモニタリングできます。AWS Panorama アプライアンスはインターネット経由で AWS クラウドに接続し、そのステータスと、接続されているカメラのステータスのリポートを行います。また、オンになっている間、アプライアンスは CloudWatch Logs にリアルタイムでログを送信します。

アプライアンスは、AWS Panorama コンソールを初めて使用するときに作成するサービスロールから、AWS IoT CloudWatch Logs、およびその他の AWS サービスの使用許可を取得します。詳細については、「AWS Panorama サービスロールとクロスサービスリソース」を参照してください。

特定のエラーのトラブルシューティングのヘルプについては、「トラブルシューティング」を参照してください。

トピック

- AWS Panorama コンソールでのモニタリング
- AWS Panorama ログを表示する
- Amazon CloudWatch によるアプライアンスとアプリケーションのモニタリング

AWS Panorama コンソールでのモニタリング

AWS Panorama コンソールを使用して AWS Panorama アプライアンスとカメラをモニタリングできます。コンソールは AWS IoT を使用してアプライアンスの状態をモニタリングします。

AWS Panorama コンソールでアプライアンスをモニタリングするには

1. AWS Panorama コンソールを開きます。
2. AWS Panorama コンソールの [デバイス] ページを開きます。
3. アプライアンスを選択します。
4. アプリケーションインスタンスのステータスを確認するには、リストから選択します。
5. アプライアンスのネットワークインターフェースのステータスを確認するには、Settings を選択します。

ページの上部にアプライアンスの全体的なステータスが表示されます。ステータスがオンラインの場合、アプライアンスは AWS に接続されており、定期的にステータス更新を送信しています。

AWS Panorama ログを表示する

AWS Panorama は、アプリケーションイベントとシステムイベントを Amazon CloudWatch Logs にレポートします。問題が発生した場合は、イベントログを使用して AWS Panorama アプリケーションのデバッグやアプリケーション設定のトラブルシューティングを行うことができます。

CloudWatch Logs でログを表示する

1. CloudWatch Logs コンソールの [[ロググループ] ページ] を開きます。
2. 次のグループで AWS Panorama アプリケーションログとアプライアンスログを検索します:

   - デバイスログ — `/aws/panorama/devices/device-id`
   - アプリケーションログ – `/aws/panorama/devices/device-id/applications/instance-id`

システムソフトウェアの更新後にアプライアンスを再プロビジョニングすると、プロビジョニング USB ドライブのログも表示できます。

セクション

- デバイスログの表示
- アプリケーションログの表示
- アプリケーションログの設定をする
- プロビジョニングログの表示
- デバイスからのログ出力

デバイスログの表示

AWS Panorama アプライアンス は、デバイスのロググループと、デプロイする各アプリケーションインスタンスのグループを作成します。デバイスログには、アプリケーションのステータス、ソフトウェアのアップグレード、システム設定に関する情報が含まれます。

デバイスログ — **`/aws/panorama/devices/device-id`**

- `occ_log` — コントローラープロセスからの出力。このプロセスは、アプリケーションのデプロイを調整し、各アプリケーションインスタンスのノードの状態をレポートします。

- ota_log — 無線通信 (OTA) ソフトウェアアップグレードを調整するプロセスから出力されます。
- syslog — プロセス間で送信されたメッセージをキャプチャする、デバイスの syslog プロセスからの出力。
- kern_log — デバイスの Linux カーネルからのイベント。
- logging_setup_logs — CloudWatch Logs エージェントを設定するプロセスからの出力。
- cloudwatch_agent_logs — CloudWatch Logs エージェントからの出力。
- shadow_log — AWS IoT デバイスシャドウからの出力。

アプリケーションログの表示

アプリケーションインスタンスのロググループには、ノードにちなんで名付けられた各ノードのログストリームが含まれます。

アプリケーションログ – **/aws/panorama/devices/*device-id*/applications/*instance-id***

- コード — アプリケーションコードと AWS Panorama アプリケーション SDK からの出力。/opt/aws/panorama/logs からのアプリケーションログを集約します。
- モデル — 推論リクエストをモデルと連携させるプロセスからの出力。
- ストリーム — カメラストリームからのビデオをデコードするプロセスからの出力。
- ディスプレイ — HDMI ポートのビデオ出力をレンダリングするプロセスからの出力。
- mds — アプライアンスメタデータサーバーからのログ。
- console_output — 標準出力ストリームとエラーストリームをコードコンテナからキャプチャします。

CloudWatch Logs にログが表示されない場合は、正しい AWS リージョンにあることを確認します。その場合は、アプライアンスの AWS への接続またはアプライアンスの AWS Identity and Access Management (IAM) ロールの権限に問題がある可能性があります。

アプリケーションログの設定をする

/opt/aws/panorama/logs にログファイルを書き込むための Python ロガーを設定します。アプライアンスはこの場所から CloudWatch Logs にログをストリーミングします。ディスク容量を過剰に使用しないようにするには、最大ファイルサイズを 10 MiB、バックアップ数を 1 にします。次の例は、ロガーを作成する方法を示しています。

Example application.py — ロガー設定

```
def get_logger(name=__name__,level=logging.INFO):
    logger = logging.getLogger(name)
    logger.setLevel(level)
    LOG_PATH = '/opt/aws/panorama/logs'
    handler = RotatingFileHandler("{}/app.log".format(LOG_PATH), maxBytes=10000000,
 backupCount=1)
    formatter = logging.Formatter(fmt='%(asctime)s %(levelname)-8s %(message)s',
                                  datefmt='%Y-%m-%d %H:%M:%S')
    handler.setFormatter(formatter)
    logger.addHandler(handler)
    return logger
```

ロガーをグローバルスコープで初期化し、アプリケーションコード全体で使用します。

Example application.py — ロガーを初期化します

```
def main():
    try:
        logger.info("INITIALIZING APPLICATION")
        app = Application()
        logger.info("PROCESSING STREAMS")
        while True:
            app.process_streams()
            # turn off debug logging after 150 loops
            if logger.getEffectiveLevel() == logging.DEBUG and app.frame_num == 150:
                logger.setLevel(logging.INFO)
    except:
        logger.exception('Exception during processing loop.')

logger = get_logger(level=logging.INFO)
main()
```

プロビジョニングログの表示

プロビジョニング中、AWS Panorama アプライアンスは設定アーカイブをアプライアンスに転送するために使用する USB ドライブにログをコピーします。これらのログを使用して、最新のソフトウェアバージョンのアプライアンスのプロビジョニングに関する問題のトラブルシューティングを行います。

> **Important**
>
> プロビジョニングログは、ソフトウェアバージョン 4.3.23 以降に更新されたアプライアンスで使用できます。

アプリケーションログ

- `/panorama/occ.log` — AWS Panorama コントローラーのソフトウェアログ。
- `/panorama/ota_agent.log` — AWS Panorama 無線通信経由 (OTA) 更新エージェントのログ。
- `/panorama/syslog.log` — Linux システムログ。
- `/panorama/kern.log` – Linux カーネルログ。

デバイスからのログ出力

デバイスとアプリケーションのログが CloudWatch Logs に表示されない場合は、USB ドライブを使用して暗号化されたログイメージをデバイスから取得できます。AWS Panorama サービスチームが代わりにログを復号し、デバッグを支援します。

前提条件

手順を実行するには、次のハードウェアが必要になります：

- USB ドライブ — AWS Panorama アプライアンスからログファイルを転送するための、少なくとも 1 GB のストレージを備えた FAT32 形式の USB フラッシュメモリドライブ。

デバイスからログを出力する

1. `panorama` フォルダー内にある `managed_logs` フォルダーが入った USB ドライブを用意します。

   ```
   /
    ### panorama
        ### managed_logs
   ```

2. USB ドライブをデバイスに接続します。
3. AWS Panorama アプライアンスの電源を切ります。

4. AWS Panorama アプライアンスの電源を入れます。
5. デバイスがログをデバイスにコピーします。この処理中は、ステータス LED が青色に点滅します。
6. その後、ログファイルは managed_logs ディレクトリ内に panorama_device_log_*v1_dd_hh_mm*.img フォーマットで見つけることができます

ログイメージを自分で復号化することはできません。カスタマーサポート、AWS Panorama のテクニカルアカウントマネージャー、またはソリューションアーキテクトと協力してサービスチームとの調整を行ってください。

Amazon CloudWatch によるアプライアンスとアプリケーションのモニタリング

アプライアンスをオンラインにすると、AWS Panorama はメトリクスを Amazon CloudWatch に送信します。CloudWatch コンソールでこれらのメトリクスを使ってグラフやダッシュボードを構築してアプライアンスのアクティビティをモニタリングし、デバイスがオフラインになったり、アプリケーションにエラーが発生したときに通知するアラームを設定することができます。

CloudWatch コンソールでメトリクスを表示する

1. AWS Panorama コンソールのメトリクスページ (PanoramaDeviceMetrics 名前空間) を開きます。
2. ディメンションスキーマを選択します。
3. メトリクスを選択してグラフに追加します。
4. 別の統計を選択してグラフをカスタマイズするには、グラフ化されたメトリックス タブのオプションを使用します。デフォルトでは、グラフはすべてのメトリクスで Average 統計を使用します。

> **料金**
>
> CloudWatch では、無期限の無料利用枠をご利用いただけます。無料利用枠の限度を超えた場合、メトリクス、ダッシュボード、アラーム、ログ、インサイトに対する CloudWatch 料金が発生します。詳細については、CloudWatch 料金表を参照してください。

CloudWatch 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。

セクション

- デバイスメトリクスを使用する
- アプリケーションメトリクスを使用する
- アラームの設定

デバイスメトリクスを使用する

アプライアンスがオンラインになると、メトリクスを Amazon CloudWatch に送信します。これらのメトリックスを使用してデバイスのアクティビティをモニタリングし、デバイスがオフラインになった場合にアラームをトリガーすることができます。

- `DeviceActive`— デバイスがアクティブになると定期的に送信されます。

  寸法 — `DeviceId` と`DeviceName`。

`DeviceActive` メトリクスと共に `Average` 統計を表示します。

アプリケーションメトリクスを使用する

アプリケーションでエラーが発生すると、メトリクスを Amazon CloudWatch に送信します。これらのメトリクスを使用して、アプリケーションが実行を停止した場合にアラームをトリガーできます。

- `ApplicationErrors` — 記録されたアプリケーションエラーの数。

  寸法 — `ApplicationInstanceName` と`ApplicationInstanceId`。

`Sum` 統計と共に、アプリケーションメトリクスを表示します。

アラームの設定

メトリックスがしきい値を超えたときに通知を受け取るには、アラームを作成します。例えば、`ApplicationErrors` メトリックスの合計が1の状態が20分間継続した場合に通知を送信するアラームを作成できます。

アラームを作成する

1. Amazon CloudWatch コンソールのアラームページを開きます。
2. アラームの作成 を選択します。
3. メトリクスの選択を選択し、`applicationInstance-gk75xmplqbqtenlnmz4ehiu7xa`や`my-application`の `ApplicationErrors` のようなデバイスのメトリックスを探します。
4. 手順に従って、アラームの条件、アクション、名前を設定します。

詳細な手順については、Amazon CloudWatch ユーザーガイドの、Amazon CloudWatch アラームを作成するを参照してください。

トラブルシューティング

以下のトピックでは、API、AWS Panorama コンソール、アプライアンス、または SDK の使用時に発生する可能性のあるエラーや問題のトラブルシューティングに関するアドバイスを提供します。ここに記載されていない問題が見つかった場合は、このページの [フィードバックを提供する] ボタンを使用して報告することができます。

アプライアンスのログは Amazon CloudWatch Logs コンソールで確認できます。アプライアンスは、アプリケーションコード、アプライアンスソフトウェア、AWS IoT プロセスのログの生成時にアップロードします。詳細については、「AWS Panorama ログを表示する」を参照してください。

プロビジョニング

問題: (macOS) 付属の USB ドライブ (USB-C アダプター付き) をコンピューターが認識しません。

これは、既にコンピューターに接続されている USB-C アダプターに USB ドライブを接続した場合に発生することがあります。アダプターを取り外し、既に接続されている USB ドライブに再接続してみてください。

問題: 自分の USB ドライブを使用するとプロビジョニングに失敗します。

問題: アプライアンスの USB 2.0 ポートを使用するとプロビジョニングが失敗します。

AWS Panorama アプライアンスは 1 ~ 32 GB の USB フラッシュメモリデバイスと互換性がありますが、すべてに互換性があるわけではありません。USB 2.0 ポートをプロビジョニングに使用すると問題が発生しています。一貫した結果を得るには、付属の USB 3.0 ポート (HDMI ポートの隣) の USB ドライブを使用してください。

レノボ ThinkEdge® SE70 の場合、USB ドライブはアプライアンスに含まれていません。1 GB 以上のストレージを備えた USB 3.0 ドライブを使用してください。

アプライアンスの構成

問題: アプライアンスの起動中に空白の画面が表示されます。

約 1 分かかる初期起動シーケンスを完了すると、アプライアンスはモデルをロードしてアプリケーションを起動する間、1 分以上にわたって空白の画面を表示します。また、ディスプレイがオンになった後にディスプレイを接続しても、アプライアンスはビデオを出力しません。

問題: 電源ボタンを長押しして電源を切っても、アプライアンスが反応しません。

アプライアンスを安全にシャットダウンするには最大 10 秒かかります。電源ボタンを 1 秒間押し続けるだけで、シャットダウンシーケンスを開始できます。ボタン操作の詳細なリストについては、AWS Panorama アプライアンスのボタンとライト を参照してください。

問題: 構成を変更したり、紛失した証明書を交換したりするために、新しい構成アーカイブを生成する必要があります。

AWS Panorama はダウンロード後のデバイス証明書やネットワーク構成を保存せず、構成アーカイブは再利用できません。AWS Panorama コンソールを使用してアプライアンスを削除し、新しい構成アーカイブで新しいアプライアンスを作成します。

アプリケーションの構成

問題: 複数のアプリケーションを実行すると、HDMI 出力を使用するアプリケーションを制御できません。

出力ノードを持つ複数のアプリケーションをデプロイすると、最後に起動したアプリケーションが HDMI 出力を使用します。このアプリケーションが実行を停止すると、別のアプリケーションが出力を使用することができます。1 つのアプリケーションだけが出力にアクセスできるようにするには、出力ノードと対応するエッジを他のアプリケーションのアプリケーションマニフェストから削除し、再デプロイします。

問題: アプリケーションの出力がログに表示されません

`/opt/aws/panorama/logs` にログファイルを書き込むための Python ロガーを設定します。。これらはコードコンテナノードのログストリームにキャプチャされます。標準出力ストリームとエラーストリームは、`console-output` という別のログストリームにキャプチャされます。`print` を使用する場合は、`flush=True` オプションを使用してメッセージが出力バッファに滞留しないようにします。

エラー: You've reached the maximum number of versions for package SAMPLE_CODE. Deregister unused package versions and try again.

ソース: AWS Panoramaサービス

アプリケーションに変更をデプロイするたびに、そのアプリケーションが使用する各パッケージのパッケージ構成とアセットファイルを表すパッチバージョンを登録します。クリーンアップパッチスクリプトを使用して、未使用のパッチバージョンを登録解除します。

カメラストリーム

エラー: liveMedia0: Failed to get SDP description: Connection to server failed: Connection timed out (-115)

エラー: liveMedia0: Failed to get SDP description: 404 Not Found; with the result code: 404

エラー: liveMedia0: Failed to get SDP description: DESCRIBE send() failed: Broken pipe; with the result code: -32

ソース: カメラノードログ

アプライアンスはアプリケーションのカメラストリームに接続できません。この場合、アプリケーションが AWS Panorama アプリケーション SDK からのビデオフレームを待っている間、ビデオ出力は空白になるか、最後に処理されたフレームでフリーズします。アプライアンスソフトウェアはカメラストリームへの接続を試み、カメラノードログにタイムアウトエラーを記録します。カメラストリーム URL が正しく、RTSP トラフィックがネットワーク内のカメラとアプライアンス間でルーティング可能であることを確認します。詳細については、「AWS Panorama アプライアンスをネットワークに接続します」を参照してください。

エラー: ERROR finalizeInterface(35) Camera credential fetching for port [username] failed

ソース: OCC ログ

カメラストリームの認証情報を含む AWS Secrets Manager シークレットが見つかりません。カメラストリームを削除して再作成してください。

エラー: Camera did not provide an H264 encoded stream

ソース: カメラノードログ

カメラストリームのエンコーディングは H.264 以外 (H.265 など) です。H.264 カメラストリームを使用してアプリケーションを再デプロイします。サポートされているカメラの詳細については、サポート対象カメラ を参照してください。

AWS Panorama セキュリティ

AWS では、クラウドのセキュリティが最優先事項です。AWS の顧客は、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャから利点を得られます。

セキュリティは、AWS と顧客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して説明しています。

- クラウドのセキュリティ - AWS は、AWS クラウドで AWS のサービスを実行するインフラストラクチャを保護する責任を負います。また、AWS は、使用するサービスを安全に提供します。AWS コンプライアンスプログラムの一環として、サードパーティーの監査が定期的にセキュリティの有効性をテストおよび検証しています。AWS Panorama に適用されるコンプライアンスプログラムの詳細については、「コンプライアンスプログラムによる対象範囲内の AWS サービス」を参照してください。
- クラウド内のセキュリティ - ユーザーの責任は、使用する AWS のサービスに応じて異なります。またお客様は、データの機密性、企業要件、適用法令と規制などのその他の要因に対しても責任を担います。

このドキュメントは、AWS Panorama を使用する際に責任共有モデルを適用する方法を理解するのに役立ちます。次のトピックでは、セキュリティおよびコンプライアンスの目的達成のための AWS Panorama の設定方法を示します。また、AWS Panorama リソースのモニタリングや保護に役立つ他の AWS のサービスの使用方法についても説明します。

トピック

AWS Panorama アプライアンスのセキュリティ機能

アプリケーション、モデル、ハードウェアを悪意のあるコードやその他の悪用から保護するために、AWS Panorama アプライアンスには広範なセキュリティ機能が実装されています。以下の理由がありますが、これらに限定されるものではありません。

- フルディスク暗号化 — アプライアンスにはLinux 統合キーセットアップ (LUKS2) のフルディスク暗号化が実装されています。すべてのシステムソフトウェアとアプリケーションデータは、デバイス固有のキーで暗号化されます。デバイスに物理的にアクセスしても、攻撃者はストレージの内容を検査することはできません。
- メモリレイアウトのランダム化 — メモリにロードされた実行コードを標的とする攻撃から保護するために、AWS Panorama アプライアンスはアドレス空間レイアウトのランダム化 (ASLR) を使用しています。ASLR は、オペレーティングシステムコードがメモリに読み込まれるときに、その位置をランダム化します。これにより、ランタイム中にコードの保存場所を予測することで、コードの特定のセクションを上書きしたり実行したりする悪用を防ぐことができます。
- 信頼できる実行環境 — アプライアンスは、独立したストレージ、メモリ、および処理リソースを備えた ARM TrustZone に基づく信頼できる実行環境 (TEE) を使用します。トラストゾーンに保存されている鍵やその他の機密データには、TEE 内の別のオペレーティングシステムで動作する信頼できるアプリケーションのみがアクセスできます。AWS Panorama アプライアンスソフトウェアは、アプリケーションコードとともに信頼できない Linux 環境で実行されます。暗号化操作にアクセスできるのは、安全なアプリケーションにリクエストを行う場合のみです。
- 安全なプロビジョニング — アプライアンスをプロビジョニングする場合、デバイスに転送する認証情報 (鍵、証明書、その他の暗号資料) は短期間しか有効になりません。アプライアンスは有効期間の短い認証情報を使用して AWS IoT に接続し、より長期間有効な証明書を自身に要求します。AWS Panorama サービスは認証情報を生成し、デバイスにハードコーディングされたキーで暗号化します。証明書をリクエストしたデバイスのみが、証明書を復号化して AWS Panorama と通信できます。
- セキュアブート — デバイスの起動時に、各ソフトウェアコンポーネントは実行前に認証されます。ブート ROM は、プロセッサにハードコーディングされた変更不可能なソフトウェアで、ハードコードされた暗号化キーを使用してブートローダーを復号化し、信頼できる実行環境のカーネルなどを検証します。
- 署名付きカーネル — カーネルモジュールは非対称暗号鍵で署名されます。オペレーティングシステムカーネルは、モジュールをメモリにロードする前に、公開鍵を使用して署名を復号化し、モジュールの署名と一致することを確認します。

- dm-verity — カーネルモジュールの検証と同様に、アプライアンスは Linux デバイスマッパー dm-verity の機能を使用して、アプライアンスのソフトウェアイメージをマウントする前にその整合性を検証します。アプライアンスソフトウェアが変更されると、動作しなくなります。
- ロールバック防止 — アプライアンスソフトウェアを更新すると、アプライアンスはSoC (システムオンチップ) の電子ヒューズを切断します。各ソフトウェアバージョンでは、切れるヒューズの数が増えることが予想され、ヒューズがさらに切れると動作しなくなります。

AWS Panorama アプライアンスセキュリティのベストプラクティス

AWS Panorama アプライアンスを使用するときは、次の点に注意してください。

- アプライアンスを物理的に保護する — アプライアンスは密閉されたサーバーラックまたは安全な部屋に設置してください。デバイスへの物理的なアクセスは、権限のある担当者に限定してください。
- アプライアンスのネットワーク接続を保護する — 内部リソースと外部リソースへのアクセスを制限するルーターにアプライアンスを接続します。アプライアンスは、安全な内部ネットワーク上にあるカメラに接続する必要があります。また、AWSに接続する必要があります。2 つ目のイーサネットポートは物理的な冗長性のためだけに使用し、必要なトラフィックのみを許可するようにルーターを構成します。

  推奨ネットワーク構成のいずれかを使用して、ネットワークレイアウトを計画してください。詳細については、「AWS Panorama アプライアンスをネットワークに接続します」を参照してください。
- USB ドライブのフォーマット — アプライアンスをプロビジョニングしたら、USB ドライブを取り外してフォーマットします。アプライアンスは AWS Panorama サービスに登録した後は USB ドライブを使用しません。一時的な認証情報、構成ファイル、プロビジョニングログを削除するようにドライブをフォーマットします。
- アプライアンスを最新の状態に保つ — アプライアンスのソフトウェアアップデートを適時に適用してください。AWS Panorama コンソールでアプライアンスを表示すると、ソフトウェアアップデートが利用可能な場合コンソールから通知されます。詳細については、「AWS Panorama アプライアンスの管理」を参照してください。

  デバイスを説明 API オペレーションでは、`LatestSoftware`および`CurrentSoftware` フィールドを比較することで更新の確認を自動化できます。最新のソフトウェアバージョンが現在のバージョンと異なる場合は、コンソールまたは デバイスのためにジョブを作成 オペレーションを使用してアップデートを適用します。
- アプライアンスの使用をやめた場合は、リセットしてください – アプライアンスを安全なデータセンターから移動する前に、アプライアンスを完全にリセットしてください。アプライアンスの電源を切り、電源を接続した状態で、電源ボタンとリセットボタンを同時に5秒間押します。これにより、アカウント認証情報、アプリケーション、およびログがアプライアンスから削除されます。

  詳細については、「AWS Panorama アプライアンスのボタンとライト」を参照してください。

- AWS Panorama やその他のサービスへのアクセスを制限する — AWSPanoramaFullAccess を使用すると、すべての AWS Panorama API オペレーションにアクセスできるほか、必要に応じて他のサービスにもアクセスできます。このポリシーは、可能な限り、命名規則に基づいてリソースへのアクセスを制限します。たとえば、名前が panorama で始まるAWS Secrets Manager シークレットへのアクセスを許可します。読み取り専用アクセス権が必要なユーザーや、より具体的なリソースセットへのアクセスを必要とするユーザーには、最小特権ポリシーを起点として管理ポリシーを使用してください。

  詳細については、「AWS Panorama のアイデンティティベースの IAM ポリシー」を参照してください。

AWS Panorama でのデータ保護

AWS 責任共有モデルは、AWS Panorama のデータ保護に適用されます。このモデルで説明されているように、AWS は、AWS クラウド のすべて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があります。お客様は、このインフラストラクチャでホストされてい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のよくある質問」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。

データを保護するため、AWS アカウント の認証情報を保護し、AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーをセットアップ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を各ユーザーに付与できます。また、次の方法でデータを保護することをおすすめします。

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 が必須です。TLS 1.3 が推奨されます。
- AWS CloudTrail で API とユーザーアクティビティロギングをセットアップします。
- AWS のサービス内でデフォルトである、すべてのセキュリティ管理に加え、AWS の暗号化ソリューションを使用します。
- Amazon Macie などの高度なマネージド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 により AWS にアクセスするときに FIPS 140-2 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) 140-2」を参照してください。

お客様の E メールアドレスなどの機密情報やセンシティブ情報は、タグや [名前] フィールドなどの自由形式のフィールドに配置しないことを強くお勧めします。これには、コンソール、API、AWS CLI、または AWS SDK を使用して、AWS Panorama または他の AWS のサービス で作業する場合も含まれます。名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。外部サーバーへの URL を提供する場合は、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 に含めないように強くお勧めします。

セクション

- 転送中の暗号化

- AWS Panorama アプライアンス
- アプリケーション
- その他のサービス

転送中の暗号化

AWS Panorama API エンドポイントでは、HTTPS 経由の安全な接続のみがサポートされます。AWS Panorama リソースを AWS Management Console、AWS SDK、または AWS Panorama API を使用して管理する場合、すべての通信は Transport Layer Security (TLS) で暗号化されます。AWS Panorama アプライアンスと AWS 間の通信も TLS で暗号化されます。RTSP 上の AWS Panorama アプライアンスとカメラ間の通信は暗号化されません。

API エンドポイントの詳細なリストについては、AWS 全般のリファレンス の「 のリージョンとエンドポイント」を参照してください。

AWS Panorama アプライアンス

AWS Panorama アプライアンスには、イーサネット、HDMI ビデオ、USB ストレージ用の物理ポートがあります。SD カードスロット、Wi-Fi、ブルートゥースは使用できません。USB ポートは、プロビジョニング中に構成アーカイブをアプライアンスに転送するためにのみ使用されます。

アプライアンスのプロビジョニング証明書とネットワーク構成を含む構成アーカイブの内容は暗号化されません。AWS Panorama はこれらのファイルを保存しません。アプライアンスを登録したときにのみ取得できます。構成アーカイブをアプライアンスに転送したら、コンピュータと USB ストレージデバイスから削除します。

アプライアンスのファイルシステム全体が暗号化されます。さらに、アプライアンスは必要なソフトウェアアップデートのロールバック保護、署名付きカーネルとブートローダー、ソフトウェア整合性検証など、システムレベルの保護をいくつか適用します。

アプライアンスの使用を停止したら、フルリセットを実行してアプリケーションデータを削除し、アプライアンスソフトウェアをリセットします。

アプリケーション

アプライアンスにデプロイするコードを制御できます。ソースに関係なく、デプロイする前にすべてのアプリケーションコードにセキュリティ上の問題がないか検証してください。アプリケーションでサードパーティのライブラリを使用する場合は、そのライブラリのライセンスポリシーとサポートポリシーを慎重に検討してください。

アプリケーションの CPU、メモリ、およびディスク使用量は、アプライアンスソフトウェアによる制約を受けません。アプリケーションがリソースを大量に使用すると、他のアプリケーションやデバイスの動作に悪影響を及ぼす可能性があります。アプリケーションは、組み合わせたり、実稼働環境にデプロイしたりする前に個別にテストしてください。

アプリケーション資産 (コードとモデル) は、アカウント、アプライアンス、またはビルド環境内のアクセスから切り離されているわけではありません。AWS Panorama アプリケーション CLI によって生成されたコンテナイメージとモデルアーカイブは暗号化されません。本番環境のワークロードには別のアカウントを使用し、アクセスは必要な場合にのみ許可してください。

その他のサービス

モデルとアプリケーションコンテナを Amazon S3 に安全に保存するために、AWS Panorama は Amazon S3 が管理するキーによるサーバー側の暗号化を使用しています。詳細については、「Amazon Simple Storage Service ユーザーガイド」の「暗号化を使用したデータの保護」を参照してください。

カメラストリームの認証情報は保存時に AWS Secrets Manager で暗号化されます。アプライアンスの IAM ロールは、ストリームのユーザー名とパスワードにアクセスするためのシークレットを取得する権限を付与します。

AWS Panorama アプライアンスは、ログデータを Amazon CloudWatch Logs. CloudWatch Logs に送信し、デフォルトでこのデータを暗号化し、カスタマーマネージドキーを使用するように設定できます。詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「 を使用してログのログデータを暗号化AWS KMSする」を参照してください。 CloudWatch

AWS Panorama のアイデンティティとアクセス管理

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するために役立つ AWS のサービス です。IAM 管理者は、AWS Panorama リソースの使用を 認証 (サインイン) し、 認可 (アクセス権限を持つ) できるユーザーを制御します。IAM は、追加費用なしで使用できる AWS のサービスです。

トピック

対象者

AWS Identity and Access Management (IAM) の用途は、AWS Panorama で行う作業によって異なります。

サービスユーザー - ジョブを実行するために AWS Panorama サービスを使用する場合は、管理者が必要なアクセス権限と認証情報を用意します。作業をするためにさらに多くの AWS Panorama 機能を使用するとき、追加のアクセス許可が必要になる場合があります。アクセスの管理方法を理解しておくと、管理者に適切な許可をリクエストするうえで役立ちます。AWS Panorama の機能にアクセスできない場合、「AWS Panorama 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 」を参照してください。

サービス管理者 – 社内の AWS Panorama リソースを担当している場合は、おそらく AWS Panorama へのフルアクセスがあります。サービスのユーザーがどの AWS Panorama 機能やリソースにアクセスするかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。AWS Panorama で IAM を利用する方法の詳細については、「AWS Panorama が IAM と連携する仕組み」 を参照してください。

IAM 管理者 - IAM 管理者は、AWS Panorama へのアクセスを管理するポリシーの作成方法の詳細について確認する場合があります。IAM で使用できる AWS Panorama 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、「AWS Panorama のアイデンティティベースのポリシーの例」 を参照してください。

アイデンティティによる認証

認証とは、アイデンティティ認証情報を使用して AWSにサインインする方法です。ユーザーは、AWS アカウントのルートユーザーもしくは IAM ユーザーとして、または IAM ロールを引き受けることによって、認証を受ける (AWS にサインインする) 必要があります。

ID ソースから提供された認証情報を使用して、フェデレーティッドアイデンティティとして AWS にサインインできます。 AWS IAM Identity Centerフェデレーティッドアイデンティティの例としては、(IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook の認証情報などがあります。フェデレーティッドアイデンティティとしてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーションを使用して AWSにアクセスする場合、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、AWS Management Console または AWS アクセスポータルにサインインできます。AWS への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。

プログラムで AWS にアクセスする場合、AWS は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) を提供し、認証情報でリクエストに暗号で署名します。AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。リクエストに署名する推奨方法の使用については、『IAM ユーザーガイド』の「AWS API リクエストの署名」を参照してください。

使用する認証方法を問わず、追加のセキュリティ情報の提供が求められる場合もあります。例えば、AWS では、アカウントのセキュリティ強化のために多要素認証 (MFA) の使用をお勧めしています。詳細については、「AWS IAM Identity Center ユーザーガイド」の「Multi-factor authentication (多要素認証)」 および「IAM ユーザーガイド」の「AWS での多要素認証 (MFA) の使用」を参照してください。

AWS アカウントのルートユーザー

AWS アカウントを作成する場合は、そのアカウントのすべての AWS のサービスとリソースに対して完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。このアイデンティティは AWS アカウントのルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることによってアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報は保護し、ルートユーザーで

しか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、『IAM ユーザーガイド』の「ルートユーザー認証情報が必要なタスク」を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、1 人のユーザーまたは 1 つのアプリケーションに対して特定の権限を持つ AWS アカウント 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、「IAM ユーザーガイド」の「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する権限を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳細については、「IAM ユーザーガイド」の「IAM ユーザー (ロールではなく) の作成が適している場合」を参照してください。

IAM ロール

IAM ロールは、特定の権限を持つ、AWS アカウント 内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。ロールを切り替えることによって、AWS Management Console で IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、AWS CLI または AWS API オペレーションを呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「IAM ロールの使用」を参照してください。

一時的な認証情報を持った IAM ロールは、以下の状況で役立ちます。

- フェデレーションユーザーユーザーアクセス – フェデレーションアイデンティティに権限を割り当てるには、ロールを作成してそのロールの権限を定義します。フェデレーティッドアイデンティティが認証されると、そのアイデンティティはロールに関連付けられ、ロールで定義されている

権限が付与されます。フェデレーションの詳細については、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー向けロールの作成」を参照してください。IAM アイデンティティセンターを使用する場合、権限セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。権限セットの詳細については、「AWS IAM Identity Center ユーザーガイド」の「権限セット」を参照してください。

- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の AWS のサービスでは、(ロールをプロキシとして使用する代わりに) リソースにポリシーを直接アタッチできます。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、「IAM ユーザーガイド」の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。
- クロスサービスアクセス - 一部の AWS のサービスでは、他の AWS のサービスの機能を使用します。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの権限、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用して AWS でアクションを実行するユーザーは、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、AWS のサービスを呼び出すプリンシパルの権限を、AWS のサービスのリクエストと合わせて使用し、ダウンストリームのサービスに対してリクエストを行います。FAS リクエストは、サービスが、完了するために他の AWS のサービス またはリソースとのやりとりを必要とするリクエストを受け取ったときにのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。
  - サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。
  - サービスリンクロール - サービスリンクロールは、AWS のサービス にリンクされたサービスロールの一種です。サービスがロールを引き受け、ユーザーに代わってアクションを実行できる

ようになります。サービスリンクロールは、AWS アカウント に表示され、サービスによって所有されます。IAM 管理者は、サービスリンクロールの権限を表示できますが、編集することはできません。

- Amazon EC2 で実行されているアプリケーション - EC2 インスタンスで実行され、AWS CLI または AWS API 要求を行っているアプリケーションの一時的な認証情報を管理するには、IAM ロールを使用できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスに添付されたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、「IAM ユーザーガイド」の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用してアクセス許可を付与する」を参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、「IAM ユーザーガイド」の「(IAM ユーザーではなく) IAM ロールをいつ作成したら良いのか?」を参照してください。

ポリシーを使用したアクセス権の管理

AWS でアクセス権を管理するには、ポリシーを作成して AWS アイデンティティまたはリソースにアタッチします。ポリシーは AWS のオブジェクトであり、アイデンティティやリソースに関連付けて、これらの権限を定義します。AWS は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うと、これらのポリシーを評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。大半のポリシーは JSON ドキュメントとして AWSに保存されます。JSON ポリシードキュメントの構造と内容の詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は AWSJSON ポリシーを使用して、だれが何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの権限を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。このポリシーがあるユーザーは、AWS Management Console、AWS CLI、または AWS API からロール情報を取得できます。

アイデンティティベースポリシー

アイデンティティベースポリシーは、IAM ユーザー、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 権限ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件を制御します。アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

アイデンティティベースポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれます。管理ポリシーは、AWS アカウント 内の複数のユーザー、グループ、およびロールにアタッチできるスタンドアロンポリシーです。マネージドポリシーには、AWS マネージドポリシーとカスタマー管理ポリシーがあります。マネージドポリシーまたはインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「マネージドポリシーとインラインポリシーの比較」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには、例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあります。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。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーションユーザー、または AWS のサービスを含めることができます。

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は使用できません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための権限を持つかをコントロールします。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Simple Storage Service (Amazon S3)、AWS WAF、および Amazon VPC は、ACL をサポートするサービスの例です。ACL の詳細については、「Amazon Simple Storage Service デベロッパーガイド」の「アクセスコントロールリスト (ACL) の概要」を参照してください。

その他のポリシータイプ

AWS では、他の一般的ではないポリシータイプをサポートしています。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- 権限の境界 - 権限の境界は、アイデンティティベースのポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる許可の上限を設定する高度な機能です。エンティティに権限の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとその権限の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、権限の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。権限の境界の詳細については、「IAM ユーザーガイド」の「IAM エンティティの権限の境界」を参照してください。
- サービスコントロールポリシー (SCP) - SCP は、AWS Organizations で組織や組織単位 (OU) の最大権限を指定する JSON ポリシーです。AWS Organizations は、顧客のビジネスが所有する複数の AWS アカウント をグループ化し、一元的に管理するサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP はメンバーアカウントのエンティティに対する権限を制限します (各 AWS アカウントのルートユーザー など)。Organizations と SCP の詳細については、『AWS Organizations ユーザーガイド』の「SCP の仕組み」を参照してください。
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限の範囲は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、「IAM ユーザーガイド」の「セッションポリシー」をご参照ください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。複数のポリシータイプが関連するとき、リクエストを許可するかどうかを AWS が決定する方法の詳細については、『IAM ユーザーガイド』の「Policy evaluation logic (ポリシーの評価ロジック)を参照してください。

AWS Panorama が IAM と連携する仕組み

IAM を使用して AWS Panorama へのアクセスを管理する前に、AWS Panorama で使用できる IAM 機能について理解しておく必要があります。AWS Panorama およびその他 AWS のサービスが IAM

と連携する方法の概要を把握するには、「IAM ユーザーガイド」の「IAM と連携する AWS のサービス」を参照してください。

AWS Panoramaで使用する、アクセス権限、ポリシー、およびロールの概要については、「AWS Panorama アクセス権限」 を参照してください。

AWS Panorama 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、IAM ユーザーとロールには AWS Panorama リソースを作成または変更するためのアクセス許可はありません。AWS Management Console、AWS CLI、または AWS API を使用してタスクを実行することもできません。IAM 管理者は、ユーザーとロールに必要な、指定されたリソースで特定の API オペレーションを実行する権限をユーザーとロールに付与する IAM ポリシーを作成する必要があります。続いて、管理者はそれらの権限が必要な IAM ユーザーまたはグループにそのポリシーをアタッチする必要があります。

これらの JSON ポリシードキュメント例を使用して IAM の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「JSON タブでのポリシーの作成」を参照してください。

トピック

- ポリシーのベストプラクティス
- AWS Panorama コンソールを使う
- ユーザーが自分のアクセス許可を表示できるようにする方法

ポリシーのベストプラクティス

ID ベースのポリシーは、ユーザーのアカウントで誰が AWS Panorama リソースを作成、アクセス、または削除できるどうかを決定します。これらのアクションを実行すると、AWS アカウント に料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS マネージドポリシーを使用して開始し、最小特権の権限に移行する – ユーザーとワークロードへの権限の付与を開始するには、多くの一般的なユースケースのために権限を付与する AWS マネージドポリシーを使用します。これらは AWS アカウントで使用できます。ユースケースに応じた AWS カスタマーマネージドポリシーを定義することで、権限をさらに減らすことをお勧めします。詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」または「AWS ジョブ機能の管理ポリシー」を参照してください。

- 最小特権を適用する – IAM ポリシーで権限を設定するときは、タスクの実行に必要な権限のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権権限とも呼ばれています。IAM を使用して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 でのポリシーと権限」を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。また、AWS CloudFormation などの特定の AWS のサービス を介して使用する場合、条件を使用して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。詳細については、「IAM ユーザーガイド」の「IAM JSON ポリシー要素: 条件」を参照してください。
- IAM Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、『IAM ユーザーガイド』の「IAM Access Analyzer ポリシーの検証」を参照してください。
- 多要素認証 (MFA) を要求する - AWS アカウント内の IAM ユーザーまたはルートユーザーを要求するシナリオがある場合は、セキュリティを強化するために MFA をオンにします。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、「IAM ユーザーガイド」の「MFA 保護 API アクセスの設定」を参照してください。

IAM でのベストプラクティスの詳細については、『IAM ユーザーガイド』の「IAM でのセキュリティのベストプラクティス」を参照してください。

AWS Panorama コンソールを使う

AWS Panorama コンソールにアクセスするには、一連の最小限のアクセス権限が必要です。これらのアクセス権限により、AWS アカウントの AWS Panorama リソースの一覧と詳細を表示できます。最小限必要な許可よりも厳しく制限されたアイデンティティベースポリシーを作成すると、そのポリシーを添付したエンティティ (IAM 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。

詳細については、「AWS Panorama のアイデンティティベースの IAM ポリシー」を参照してください。

ユーザーが自分のアクセス許可を表示できるようにする方法

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWS CLI か AWS API を使用してプログラム的に、このアクションを完了する権限が含まれています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS Panorama のAWS マネージドポリシー

AWS マネージドポリシーは、AWS が作成および管理するスタンドアロンポリシーです。AWS マネージドポリシーは、多くの一般的なユースケースでアクセス許可を提供できるように設計されているため、ユーザー、グループ、ロールへのアクセス許可の割り当てを開始できます。

AWS マネージドポリシーは、ご利用の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ご注意ください。AWS のすべてのお客様が使用できるようになるのを避けるためです。ユースケース別にカスタマーマネージドポリシーを定義することで、アクセス許可を絞り込むことをお勧めします。

AWS マネージドポリシーで定義したアクセス権限は変更できません。AWS が AWS マネージドポリシーに定義されているアクセス許可を更新すると、更新はポリシーがアタッチされているすべてのプリンシパルアイデンティティ (ユーザー、グループ、ロール) に影響します。新しい AWS のサービスを起動するか、既存のサービスで新しい API オペレーションが使用可能になると、AWS が AWS マネージドポリシーを更新する可能性が最も高くなります。

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 Panorama は、次のマネージドポリシーを提供されます。各ポリシーの全内容と変更履歴については、IAM コンソールのリンク先ページを参照してください。

- AWSPanoramaFullAccess — AWS Panorama、Amazon S3 の AWS Panorama アクセスポイント、AWS Secrets Manager のアプライアンス認証情報、Amazon CloudWatch のアプライアンスログへのフルアクセスを提供します。AWS Panorama のサービスにリンクされたロールを作成する権限が含まれます。
- AWSPanoramaServiceLinkedRolePolicy — AWS Panorama が AWS IoT、AWS Secrets Manager、および AWS Panorama のリソースを管理できるようにします。
- AWSPanoramaServiceLinkedRolePolicy — AWS Panorama アプライアンスが CloudWatch にログをアップロードし、AWS Panorama によって作成された Amazon S3 Access Points からオブジェクトを取得することを許可します。

AWS マネージドポリシーに対する AWS Panorama 更新

次の表は、AWS Panorama のマネージドポリシーを更新したものです。

変更	説明	日付
AWSPanoramaFullAccess — 既存のポリシーへの更新	ユーザーが CloudWatch Logs コンソールでロググループを表示できるようにするアクセス許可をユーザーポリシーに追加しました。	2022-01-13
AWSPanoramaFullAccess — 既存のポリシーへの更新	ユーザーポリシーにアクセス許可を追加して、ユーザーが AWS Panorama サービスにリンクされたロールを管理し、IAM、Amazon S3、CloudWatch、Secrets Manager などの他のサービスの AWS Panorama リソースにアクセスできるようにしました。	2021-10-20
AWSPanoramaApplianceServiceRolePolicy — 新しいポリシー	AWS Panorama アプライアンスサービスロールの新しいポリシー	2021-10-20
AWSPanoramaServiceLinkedRolePolicy — 新しいポリシー	AWS Panorama サービスロールの新しいポリシー	2021-10-20
AWS Panorama が変更の追跡を開始	AWS Panorama でAWS マネージドポリシーの変更の追跡が開始されました。	2021-10-20

AWS Panorama のサービスにリンクされたロールの使用

AWS Panorama は AWS Identity and Access Management (IAM) サービスにリンクされたロールを使用します。サービスにリンクされたロールは、AWS Panorama に直接リンクされた一意のタイプの IAM ロールです。サービスにリンクされたロールは、AWS Panorama による事前定義済みのロー

ルであり、ユーザーに代わってサービスから他の AWS のサービスを呼び出すために必要なすべての許可を備えています。

サービスリンクロールを使用すると、必要なアクセス許可を手動で追加する必要がなくなるため、AWS Panorama の設定が簡単になります。AWS Panorama は、このサービスリンクロールのアクセス許可を定義します。特に定義されている場合を除き、AWS Panorama のみがそのロールを引き受けることができます。定義される許可は、信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンクロールを削除するには、まずその関連リソースを削除します。これにより、リソースへの意図しないアクセスによるアクセス許可の削除が防止され、AWS Panorama リソースは保護されます。

サービスにリンクされたロールをサポートする他のサービスについては、「IAM と連動する AWS サービス」を参照し、[Service-linked roles] (サービスにリンクされたロール) の列内で [Yes] (はい) と表記されたサービスを確認してください。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[Yes] (はい) リンクを選択します。

セクション

- AWS Panorama のサービスにリンクされたロールの許可
- AWS Panorama のサービスにリンクされたロールの作成
- AWS Panorama のサービスにリンクされたロールの編集
- AWS Panorama のサービスにリンクされたロールの削除
- AWS Panorama のサービスにリンクされたロールをサポートするリージョン

AWS Panorama のサービスにリンクされたロールの許可

AWS Panorama は、ServiceRoleForAWSPanorama という名前のサービスにリンクされたロールを使用します — AWS Panorama が AWS IoT、AWS Secrets Manager、および AWS Panorama のリソースを管理できるようにします。

サービスにリンクされたロール AWSServiceRoleForAWSPanorama は、次のサービスを信頼してロールを引き受けます。

- `panorama.amazonaws.com`

ロールのアクセス許可ポリシーは、AWS Panorama が次のアクションを完了することを許可します。

- AWS Panorama リソースのモニタリング
- AWS Panorama アプライアンスのAWS IoT リソースを管理する
- AWS Secrets Manager シークレットにアクセスしカメラの認証情報を取得する

権限の完全なリストについては、IAM コンソールで AWSPanoramaServiceLinkedRolePolicyをご覧ください。

サービスにリンクされたロールの作成、編集、削除をIAM エンティティ (ユーザー、グループ、ロールなど) に許可するには、許可を設定する必要があります。詳細については、IAM ユーザーガイド の「サービスリンクロールのアクセス許可」を参照してください。

AWS Panorama のサービスにリンクされたロールの作成

サービスにリンクされたロールを手動で作成する必要はありません。AWS Management Console 、AWS CLI、または AWS APIでアプライアンスを登録すると、AWS Panorama は、サービスにリンクされたロールを作成します。

このサービスにリンクされたロールを削除した後で再度作成する必要が生じた場合は、同じ方法でアカウントにロールを再作成できます。アプライアンスを登録すると、AWS Panorama によって、サービスにリンクされたロールが再度作成されます。

AWS Panorama のサービスにリンクされたロールの編集

AWS Panorama では、AWSServiceRoleForAWSPanorama のサービスにリンクされたロールを編集することはできません。サービスにリンクされたロールを作成すると、多くのエンティティによってロールが参照される可能性があるため、ロール名を変更することはできません。ただし、IAM を使用したロールの説明の編集はできます。詳細については、IAM ユーザーガイドの「サービスリンクロールの編集」を参照してください。

AWS Panorama のサービスにリンクされたロールの削除

サービスにリンクされたロールが必要な機能またはサービスが不要になった場合には、そのロールを削除することをお勧めします。そうすることで、積極的にモニタリングまたは保守されていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスにリンクされたロールのリソースをクリーンアップする必要があります。

AWSServiceRoleForAWSPanorama が使用している AWS Panorama リソースを削除するには、このガイドの以下のセクションにある手順を使用してください。

- バージョンとアプリケーションを削除します。
- アプライアンスの登録解除

> **Note**
>
> リソースを削除する際に、AWS Panorama のサービスでロールが使用されている場合、削除は失敗することがあります。失敗した場合は、数分待ってから操作を再試行してください。

AWSServiceRoleForAWSPanorama サービスにリンクされたロールを削除するには、IAM コンソール、AWS CLI、または AWS API を使用します。詳細については、IAM ユーザーガイド の「サービスにリンクされたロールの削除」を参照してください。

AWS Panorama のサービスにリンクされたロールをサポートするリージョン

AWS Panorama は、サービスを利用できるすべてのリージョンで、サービスにリンクされたロールの使用をサポートします。詳細については、「AWS リージョンとエンドポイント」を参照してください。

サービス間の混乱した代理の防止

混乱した代理問題とは、アクションを実行する許可を持たないエンティティが、より高い特権を持つエンティティにそのアクションの実行を強制できるというセキュリティ問題です。AWS では、サービス間でのなりすましが、混乱した代理問題を生じさせることがあります。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、別の顧客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、アカウント内のリソースへのアクセス権が付与されたサービスプリンシパルですべてのサービスのデータを保護するために役立つツールを提供しています。

リソースポリシーで `aws:SourceArn` および `aws:SourceAccount` のグローバル条件コンテキストキーを使用して、AWS Panorama が別のサービスに付与する許可をそのリソースに制限することをお勧めします。両方のグローバル条件コンテキストキーを使用しており、それらが同じポリシーステートメントで使用されるときは、`aws:SourceAccount` 値と、`aws:SourceArn` 値のアカウントが同じアカウント ID を使用する必要があります。

`aws:SourceArn` の値は AWS Panorama デバイスの ARN でなければなりません。

混乱した代理問題から保護するための最も効果的な方法は、リソースの完全な ARN を指定しながら、aws:SourceArn グローバル条件コンテキストキーを使用することです。リソースの完全な ARN が不明な場合や、複数のリソースを指定する場合には、グローバルコンテキスト条件キー aws:SourceArn で、ARN の未知部分を示すためにワイルドカード (*) を使用します。例えば、arn:aws:*servicename*::*123456789012*:* です。

AWS Panorama アプライアンスに権限を与えるために AWS Panorama が使用するサービスロールを保護する手順については、アプライアンスロールの保護 を参照してください。

AWS Panorama 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、AWS Panoramaと IAM の使用に伴って発生する可能性がある一般的な問題の診断や修復に役立ちます。

トピック

- AWS Panorama でアクションを実行する権限がない
- iam を実行する権限がありません。PassRole
- 自分の AWS アカウント以外のユーザーに AWS Panorama リソースへのアクセスを許可したい

AWS Panorama でアクションを実行する権限がない

AWS Management Console から、アクションを実行することが認可されていないと通知された場合、管理者に問い合わせ、サポートを依頼する必要があります。管理者とは、ユーザーにユーザー名とパスワードを提供した人です。

次のエラー例は、mateojackson IAM ユーザーがコンソールを使用してアプライアンスの詳細を表示する際に、panorama:DescribeAppliance アクセス権限を持っていない場合に発生します。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 panorama:DescribeAppliance on resource: my-appliance
```

この場合、Mateo は、panorama:DescribeAppliance アクションを使用して my-appliance リソースへのアクセスが許可されるように、管理者にポリシーの更新を依頼します。

iam を実行する権限がありません。PassRole

iam:PassRole アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して AWS Panorama 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やサービスリンクロールを作成せずに、既存のロールをサービスに渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

次のエラー例は、marymajor という IAM ユーザーがコンソールを使用して AWS Panorama でアクションを実行しようする場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。Mary には、ロールをサービスに渡す権限がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新して、Mary 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 管理者に問い合わせてください。管理者とは、サインイン認証情報を提供した担当者です。

自分の AWS アカウント以外のユーザーに AWS Panorama リソースへのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセス制御リスト (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:

- AWS Panorama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、「AWS Panorama が IAM と連携する仕組み」 を参照してください。
- 所有している AWS アカウント 全体のリソースへのアクセス権を提供する方法については、『IAM ユーザーガイド』の「所有している別の AWS アカウント アカウントへのアクセス権を IAM ユーザーに提供」を参照してください。
- サードパーティーの AWS アカウント にリソースへのアクセス権を提供する方法については、『IAM ユーザーガイド』の「第三者が所有する AWS アカウント へのアクセス権を付与する」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でのロールとリソースベースのポリシーの使用の違いの詳細については、『IAM ユーザーガイド』の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

AWS Panorama のコンプライアンス検証

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、「コンプライアンスプログラム別の範囲」の「AWS のサービス」と「」の「AWS のサービス」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てください。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。

AWS Artifact 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。詳細については、「Downloading Reports in AWS Artifact」を参照してください。

AWS のサービスを使用する際のユーザーのコンプライアンス責任は、ユーザーのデータの機密性や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。AWS では、コンプライアンスに役立つ次のリソースを提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド - これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境を AWS にデプロイするための手順を示します。
- 「Amazon Web Services での HIPAA のセキュリティとコンプライアンスのためのアーキテクチャ」 - このホワイトペーパーは、企業が AWS を使用して HIPAA 対象アプリケーションを作成する方法を説明しています。

  > **Note**
  >
  > すべての AWS のサービスが HIPAA 適格であるわけではありません。詳細については、「HIPAA 対応サービスのリファレンス」を参照してください。

- AWS コンプライアンスのリソース - このワークブックおよびガイドのコレクションは、顧客の業界と拠点に適用されるものである場合があります。
- AWS Customer Compliance Guide — コンプライアンスの観点から見た責任共有モデルを理解できます。このガイドは、AWS のサービスを保護するためのベストプラクティスを要約したものであり、複数のフレームワーク (米国標準技術研究所 (NIST)、ペイメントカード業界セキュリティ標準評議会 (PCI)、国際標準化機構 (ISO) など) にわたるセキュリティ統制へのガイダンスがまとめられています。
- 「AWS Config デベロッパーガイド」の「ルールでのリソースの評価」 - AWS Config サービスは、自社のプラクティス、業界ガイドライン、および規制に対するリソースの設定の準拠状態を評価します。

- AWS Security Hub - この AWS のサービスは、AWS 内のセキュリティ状態の包括的なビューを提供します。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールのリストについては、「Security Hub のコントロールリファレンス」を参照してください。
- AWS Audit Manager - この AWS のサービスは、AWS の使用状況を継続的に監査して、リスクの管理方法や、規制および業界標準へのコンプライアンスの管理方法を簡素化するために役立ちます。

人がいる場合に関するその他の考慮事項

人がいる可能性があるシナリオで AWS Panorama を使用する際に考慮すべきいくつかのベストプラクティスを以下に示します：

- ユースケースに適用されるすべての法律と規制を把握し、遵守していることを確認してください。これには、カメラの位置や視野に関する法律、カメラを設置・使用する際の通知や標識の要件、プライバシー権を含む映像に映り込む可能性のある人々の権利などが含まれる場合があります。
- カメラが人々とプライバシーに及ぼす影響を考慮に入れてください。法的な要件に加え、人々がカメラで撮影されていることに驚かないように、カメラが設置されている場所に告知を行うことが適切かどうか、また、カメラは見通しの良い場所に、遮蔽物のないように設置すべきかどうかを検討してください。
- カメラの操作とカメラから取得したデータの確認については、適切なポリシーと手順を整備してください。
- カメラから取得したデータの適切なアクセス制御、使用制限、保存期間を検討してください。

AWS Panorama 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 Panorama 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 AWS がインフラストラクチャを保護する方法については、「AWS クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには、「セキュリティの柱 - AWS Well-Architected Framework」の「インフラストラクチャ保護」を参照してください。

AWS が公開した API コールを使用して、ネットワーク経由で AWS Panorama にアクセスします。クライアントは以下をサポートする必要があります。

- Transport Layer Security (TLS) TLS 1.2 および TLS 1.3 をお勧めします。
- DHE (Ephemeral Diffie-Hellman) や ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートです。これらのモードは、Java 7 以降など、最近のほとんどのシステムでサポートされています。

また、リクエストは、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時セキュリティ認証情報を生成し、リクエストに署名することもできます。

データセンターへの AWS Panorama アプライアンスのデプロイ

AWS Panorama アプライアンスは、AWS サービスと通信するためにインターネットアクセスが必要です。また、カメラの内部ネットワークへのアクセスも必要です。ネットワーク構成を慎重に検討し、必要なアクセスのみを各デバイスに提供することが重要です。AWS Panorama アプライアンスが機密性の高い IP カメラネットワークへのブリッジとして機能するように構成されている場合は注意してください。

お客様は次を行う必要があります。

- AWS Panorama アプライアンスの物理的および論理的なネットワークセキュリティ。
- AWS Panorama アプライアンスを使用する際に、ネットワークに接続されたカメラを安全に操作する。
- AWS Panorama アプライアンスとカメラソフトウェアを最新の状態に保つ。
- プライバシーに関するものも含め、本番環境から収集する動画や画像のコンテンツに関連する適用法や規制を遵守する。

AWS Panorama アプライアンスは暗号化されていない RTSP カメラストリームを使用します。AWS Panorama アプライアンスをネットワークに接続する方法の詳細については、AWS Panorama アプライアンスをネットワークに接続します を参照してください。暗号化の詳細については、AWS Panorama でのデータ保護 を参照してください。

AWS Panorama ランタイム環境ソフトウェア

AWS Panorama は、AWS パノラマアプライアンス上の Ubuntu Linux ベースの環境でアプリケーションコードを実行するソフトウェアを提供します。AWS Panorama は、アプライアンスイメージ内のソフトウェアを最新の状態に保つ責任があります。AWS Panorama は定期的にソフトウェアアップデートをリリースしており、これは Panorama コンソールを使用して適用できます。

ライブラリをアプリケーションコードで使用するには、アプリケーションの `Dockerfile` にライブラリをインストールします。ビルド全体でアプリケーションの安定性を確保するには、各ライブラリの特定のバージョンを選択してください。セキュリティ上の問題に対処するため、依存関係を定期的に更新してください。

リリース

次の表は、AWS Panorama サービス、ソフトウェア、およびドキュメントの機能とソフトウェアアップデートがいつリリースされたかを示しています。すべての機能にアクセスできるように、AWS Panorama アプライアンスを最新のソフトウェアバージョンに更新します。詳細については、「リンク先のトピック」を参照してください。

変更	説明	日付
アプライアンスソフトウェア更新	バージョン 7.0.13 は、アプライアンスがソフトウェア更新を管理する方法を変更するメジャーバージョン更新です。アプライアンスからのネットワーク通信の送信を制限するか、プライベート VPC サブネットに接続する場合は、更新を適用する前に追加のエンドポイントとポートへのアクセスを許可する必要があります。詳細については、「変更ログ」を参照してください。	2023 年 12 月 28 日
アプライアンスソフトウェア更新	バージョン 6.2.1 にはバグ修正が含まれています。詳細については、「変更ログ」を参照してください。	2023 年 9 月 6 日
アプライアンスソフトウェア更新	バージョン 6.0.8 では、バグ修正とセキュリティの向上が含まれています。詳細については、「変更ログ」を参照してください。	2023 年 7 月 6 日
アプライアンスソフトウェア更新	バージョン 5.1.7 には、バグ修正とエラー処理の改善が含まれています。詳細について	2023 年 3 月 31 日

	は、「変更ログ」を参照してください。	
コンソールの更新	管理コンソールから AWS Panorama アプライアンスを購入できるようになりました。デバイスを購入する権限をユーザーに付与するには、「AWS Panorama の ID ベース IAM ポリシー」を参照してください。	2023 年 2 月 2 日
アプライアンスソフトウェア更新	バージョン 5.0.74 には、バグ修正とエラー処理の改善が含まれています。詳細については、「変更ログ」を参照してください。	2023 年 1 月 23 日
API 更新	アプライアンスソフトウェアのメジャーバージョン更新をオプトインする `AllowMajorVersionUpdate` オプションが `OTAJobConfig` に追加されました。詳細については、「」を参照してくださいCreateJobForDevices。	2023 年 1 月 19 日
開発者向けの新しいツール	新しいツール「サイドロード」がAWS Panoramaサンプル GitHub リポジトリで利用できます。このツールを使えば、コンテナを構築してデプロイしなくてもアプリケーションコードを更新できます。詳細については、「README」を参照してください。	2022 年 11 月 16 日

アプリケーションベースのイメージの更新	バージョン 1.2.0 では、video_in.get() にタイムアウトオプションを追加し、AWS_REGION 環境変数を設定し、エラー処理を改善しています。詳細については、「変更ログ」を参照してください。	2022 年 11 月 16 日
アプライアンスソフトウェア更新	バージョン 5.0.42 には、バグ修正とセキュリティアップデートが含まれています。詳細については、「変更ログ」を参照してください。	2022 年 11 月 16 日
アプライアンスソフトウェア更新	バージョン 5.0.7 では、アプライアンスのリモート再起動とカメラストリームのリモートでの一時停止のサポートが追加されています。詳細については、「変更ログ」を参照してください。	2022 年 10 月 13 日
アプライアンスソフトウェア更新	バージョン 4.3.93 では、オフラインデバイスからのログ取得のサポートが追加されています。詳細については、「変更ログ」を参照してください。	2022 年 8 月 24 日
アプライアンスソフトウェア更新	バージョン4.3.72 には、バグ修正とセキュリティアップデートが含まれています。詳細については、「変更ログ」を参照してください。	2022 年 6 月 23 日

AWS PrivateLink のサポート	AWS Panorama は、プライベートサブネットから AWS Panorama リソースを管理するためのVPCエンドポイントをサポートしています。詳細については、「VPC エンドポイントの使用」を参照してください。	2022 年 6 月 2 日
アプライアンスソフトウェア更新	バージョン 4.3.55 では、console_output ログのストレージ使用率が向上しています。詳細については、「変更ログ」を参照してください。	2022 年 5 月 5 日
TAK ThinkEdgeTAK SE70	AWS Panorama 用の新しいアプライアンスがレノボから入手可能になりました。Nvidia Jetson Xavier NX を搭載した ThinkEdgeTAK SE70 は、AWS Panoramaアプライアンスと同じ機能をサポートしています。詳細については、「互換性のあるデバイス」を参照してください。	2022 年 4 月 6 日
アプリケーションベースのイメージの更新	バージョン 1.1.0 では、バックグラウンドスレッド実行時のパフォーマンスが向上し、イメージが最新かどうかを示すフラグ (is_cached) がメディアオブジェクトに追加されました。詳細については、「gallery.ecr.aws」を参照してください。	2022 年 3 月 29 日

アプライアンスソフトウェア更新	バージョン 4.3.45 では GPU アクセスとインバウンドポートのサポートが追加されています。詳細については、「変更ログ」を参照してください。	2022 年 3 月 24 日
アプライアンスソフトウェア更新	バージョン 4.3.35 はセキュリティとパフォーマンスを向上させます。詳細については、「変更ログ」を参照してください。	2022 年 2 月 22 日
更新されたマネージドポリシー	AWS Panorama の AWS Identity and Access Management マネージドポリシーが更新されました。詳細については、「AWS マネージドポリシー」を参照してください。	2022 年 1 月 13 日
プロビジョニングログ	アプライアンスソフトウェア 4.3.23 では、アプライアンスはプロビジョニング中に USB ドライブにログを書き込みます。詳細については、「ログ」を参照してください。	2022 年 1 月 13 日

NTP サーバーの設定	AWS Panorama アプライアンスを設定して、特定の NTP サーバをクロック同期に使用できるようになりました。アプライアンスのセットアップ中に、他のネットワーク設定を使用して NTP 設定を行います。詳細については、「セットアップ」を参照してください。	2022 年 1 月 13 日
追加のリージョン	AWS Panorama は、アジアパシフィック (シンガポール) と アジアパシフィック (シドニー) の各リージョンで利用可能になりました。	2022 年 1 月 13 日
アプライアンスソフトウェア更新	バージョン4.3.4 では、`precisionMode` モデル設定のサポートが追加され、ロギング動作が更新されます。詳細については、「変更ログ」を参照してください。	2021 年 11 月 8 日
更新されたマネージドポリシー	AWS Panorama の AWS Identity and Access Management マネージドポリシーが更新されました。詳細については、「AWS マネージドポリシー」を参照してください。	2021 年 10 月 20 日

一般提供	AWS Panorama は、米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド)、カナダ (中部) の各リージョンでご利用できるようになりました。AWS Panorama アプライアンスを購入するには、AWS Panorama にアクセスしてください。	2021 年 10 月 20 日
プレビュー	AWS Panorama は、米国東部 (バージニア北部)および米国西部 (オレゴン州) リージョンで招待制で利用可能です。	2020 年 12 月 1 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。