aws

Developer Guide

AWS Panorama

AWS Panorama: Developer Guide

Table of Contents

What is AWS Panorama?

AWS Panorama is a service that brings computer vision to your on-premises camera network. You install the AWS Panorama Appliance or another compatible device in your datacenter, register it with AWS Panorama, and deploy computer vision applications from the cloud. AWS Panorama works with your existing real time streaming protocol (RTSP) network cameras. The appliance runs secure computer vision applications from AWS Partners, or applications that you build yourself with the AWS Panorama Application SDK.

The AWS Panorama Appliance is a compact edge appliance that uses a powerful system-on-module (SOM) that is optimized for machine learning workloads. The appliance can run multiple computer vision models against multiple video streams in parallel and output the results in real time. It is designed for use in commercial and industrial settings and is rated for dust and liquid protection (IP-62).

The AWS Panorama Appliance enables you to run self-contained computer vision applications at the edge, without sending images to the AWS Cloud. By using the AWS SDK, you can integrate with other AWS services and use them to track data from the application over time. By integrating with other AWS services, you can use AWS Panorama to do the following:

- **Analyze traffic patterns** – Use the AWS SDK to record data for retail analytics in Amazon DynamoDB. Use a serverless application to analyze the collected data over time, detect anomalies in the data, and predict future behavior.
- **Receive site safety alerts** – Monitor off-limits areas at an industrial site. When your application detects a potentially unsafe situation, upload an image to Amazon Simple Storage Service (Amazon S3) and send a notification to an Amazon Simple Notification Service (Amazon SNS) topic so recipients can take corrective action.
- **Improve quality control** – Monitor an assembly line's output to identify parts that don't conform to requirements. Highlight images of nonconformant parts with text and a bounding box and display them on a monitor for review by your quality control team.
- **Collect training and test data** – Upload images of objects that your computer vision model couldn't identify, or where the model's confidence in its guess was borderline. Use a serverless application to create a queue of images that need to be tagged. Tag the images and use them to retrain the model in Amazon SageMaker.

AWS Panorama uses other AWS services to manage the AWS Panorama Appliance, access models and code, and deploy applications. AWS Panorama does as much as possible without requiring you to interact with other services, but a knowledge of the following services can help you understand how AWS Panorama works.

- SageMaker – You can use SageMaker to collect training data from cameras or sensors, build a machine learning model, and train it for computer vision. AWS Panorama uses SageMaker Neo to optimize models to run on the AWS Panorama Appliance.
- Amazon S3 – You use Amazon S3 access points to stage application code, models, and configuration files for deployment to an AWS Panorama Appliance.
- AWS IoT – AWS Panorama uses AWS IoT services to monitor the state of the AWS Panorama Appliance, manage software updates, and deploy applications. You don't need to use AWS IoT directly.

To get started with the AWS Panorama Appliance and learn more about the service, continue to Getting started with AWS Panorama.

Getting started with AWS Panorama

To get started with AWS Panorama, first learn about the service's concepts and the terminology used in this guide. Then you can use the AWS Panorama console to register your AWS Panorama Appliance and create an application. In about an hour, you can configure the device, update its software, and deploy a sample application. To complete the tutorials in this section, you use the AWS Panorama Appliance and a camera that streams video over a local network.

> **Note**
>
> To purchase an AWS Panorama Appliance, visit the AWS Panorama console.

The AWS Panorama sample application demonstrates use of AWS Panorama features. It includes a model that has been trained with SageMaker and sample code that uses the AWS Panorama Application SDK to run inference and output video. The sample application include a AWS CloudFormation template and scripts that show how to automate development and deployment workflows from the command line.

The final two topics in this chapter detail requirements for models and cameras, and the hardware specifications of the AWS Panorama Appliance. If you haven't obtained an appliance and cameras yet, or plan on developing your own computer vision models, see these topics first for more information.

Topics

AWS Panorama concepts

In AWS Panorama, you create computer vision applications and deploy them to the AWS Panorama Appliance or a compatible device to analyze video streams from network cameras. You write application code in Python and build application containers with Docker. You use the AWS Panorama Application CLI to import machine learning models locally or from Amazon Simple Storage Service (Amazon S3). Applications use the AWS Panorama Application SDK to receive video input from a camera and interact with a model.

Concepts

The AWS Panorama Appliance

The AWS Panorama Appliance is the hardware that runs your applications. You use the AWS Panorama console to register an appliance, update its software, and deploy applications to it. The software on the AWS Panorama Appliance connects to camera streams, sends frames of video to your application, and displays video output on an attached display.

The AWS Panorama Appliance is an *edge device* powered by Nvidia Jetson AGX Xavier. Instead of sending images to the AWS Cloud for processing, it runs applications locally on optimized hardware. This enables you to analyze video in real time and process the results locally. The appliance requires an internet connection to report its status, to upload logs, and to perform software updates and deployments.

For more information, see Managing the AWS Panorama Appliance.

Compatible devices

In addition to the AWS Panorama Appliance, AWS Panorama supports compatible devices from AWS Partners. Compatible devices support the same features as the AWS Panorama Appliance. You register and manage compatible devices with the AWS Panorama console and API, and build and deploy applications in the same way.

- Lenovo ThinkEdge® SE70 – Powered by Nvidia Jetson Xavier NX

The content and sample applications in this guide are developed with the AWS Panorama Appliance. For more information about specific hardware and software features for your device, refer to the manufacturer's documentation.

Applications

Applications run on the AWS Panorama Appliance to perform computer vision tasks on video streams. You can build computer vision applications by combining Python code and machine learning models, and deploy them to the AWS Panorama Appliance over the internet. Applications can send video to a display, or use the AWS SDK to send results to AWS services.

To build and deploy applications, you use the AWS Panorama Application CLI. The AWS Panorama Application CLI is a command-line tool that generates default application folders and configuration files, builds containers with Docker, and uploads assets. You can run multiple applications on one device.

For more information, see Managing AWS Panorama applications.

Nodes

An application comprises multiple components called *nodes*, which represent inputs, outputs, models, and code. A node can be configuration only (inputs and outputs), or include artifacts (models and code). An application's code node are bundled in *node packages* that you upload to an Amazon S3 access point, where the AWS Panorama Appliance can access them. An *application manifest* is a configuration file that defines connections between the nodes.

For more information, see Application nodes.

Models

A computer vision model is a machine learning network that is trained to process images. Computer vision models can perform various tasks such as classification, detection, segmentation, and tracking. A computer vision model takes an image as input and outputs information about the image or objects in the image.

AWS Panorama supports models built with PyTorch, Apache MXNet, and TensorFlow. You can build models with Amazon SageMaker or in your development environment. For more information, see ???.

Setting up the AWS Panorama Appliance

To get started using your AWS Panorama Appliance or compatible device, register it in the AWS Panorama console and update its software. During the setup process, you create an appliance *resource* in AWS Panorama that represents the physical appliance, and copy files to the appliance with a USB drive. The appliance uses these certificates and configuration files to connect to the AWS Panorama service. Then you use the AWS Panorama console to update the appliance's software and register cameras.

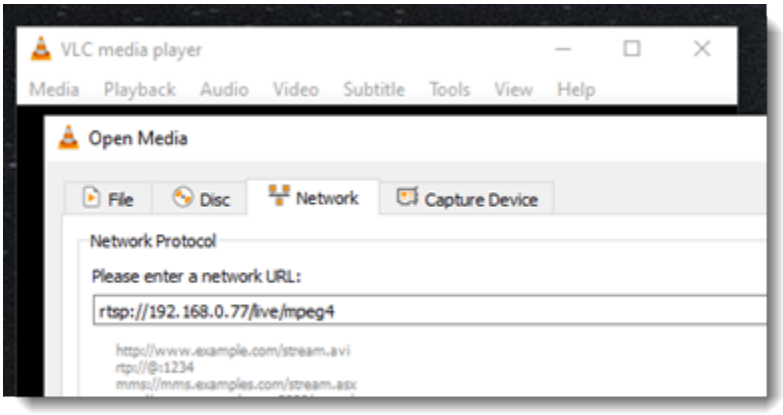
Sections

Prerequisites

To follow this tutorial, you need an AWS Panorama Appliance or compatible device and the following hardware:

- **Display** – A display with HDMI input for viewing the sample application output.
- **USB drive** (included with AWS Panorama Appliance) – A FAT32-formatted USB 3.0 flash memory drive with at least 1 GB of storage, for transferring an archive with configuration files and a certificate to the AWS Panorama Appliance.
- **Camera** – An IP camera that outputs an RTSP video stream.

Use the tools and instructions provided by your camera's manufacturer to identify the camera's IP address and stream path. You can use a video player such as VLC to verify the stream URL, by opening it as a network media source:



The AWS Panorama console uses other AWS services to assemble application components, manage permissions, and verify settings. To register an appliance and deploy the sample application, you need the following permissions:

- AWSPanoramaFullAccess – Provides full access to AWS Panorama, AWS Panorama access points in Amazon S3, appliance credentials in AWS Secrets Manager, and appliance logs in Amazon CloudWatch. Includes permission to create a service-linked role for AWS Panorama.
- **AWS Identity and Access Management (IAM)** – On first run, to create roles used by the AWS Panorama service and the AWS Panorama Appliance.

If you don't have permission to create roles in IAM, have an administrator open the AWS Panorama console and accept the prompt to create service roles.

Register and configure the AWS Panorama Appliance

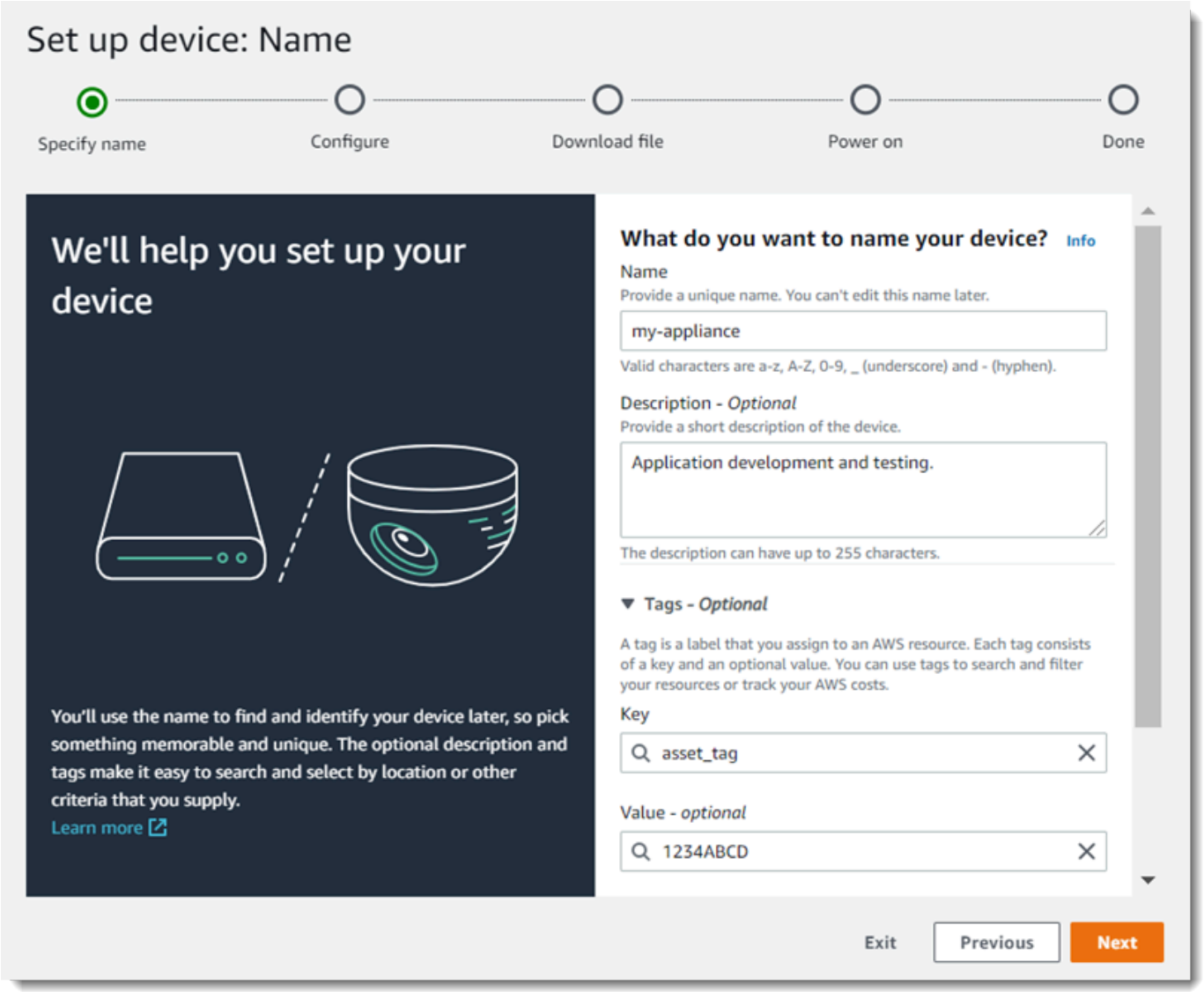
The AWS Panorama Appliance is a hardware device that connects to network-enabled cameras over a local network connection. It uses a Linux-based operating system that includes the AWS Panorama Application SDK and supporting software for running computer vision applications.

To connect to AWS for appliance management and application deployment, the appliance uses a device certificate. You use the AWS Panorama console to generate a provisioning certificate. The appliance uses this temporary certificate to complete initial setup and download a permanent device certificate.

> **Important**
>
> The provisioning certificate that you generate in this procedure is only valid for 5 minutes. If you do not complete the registration process within this time frame, you must start over.

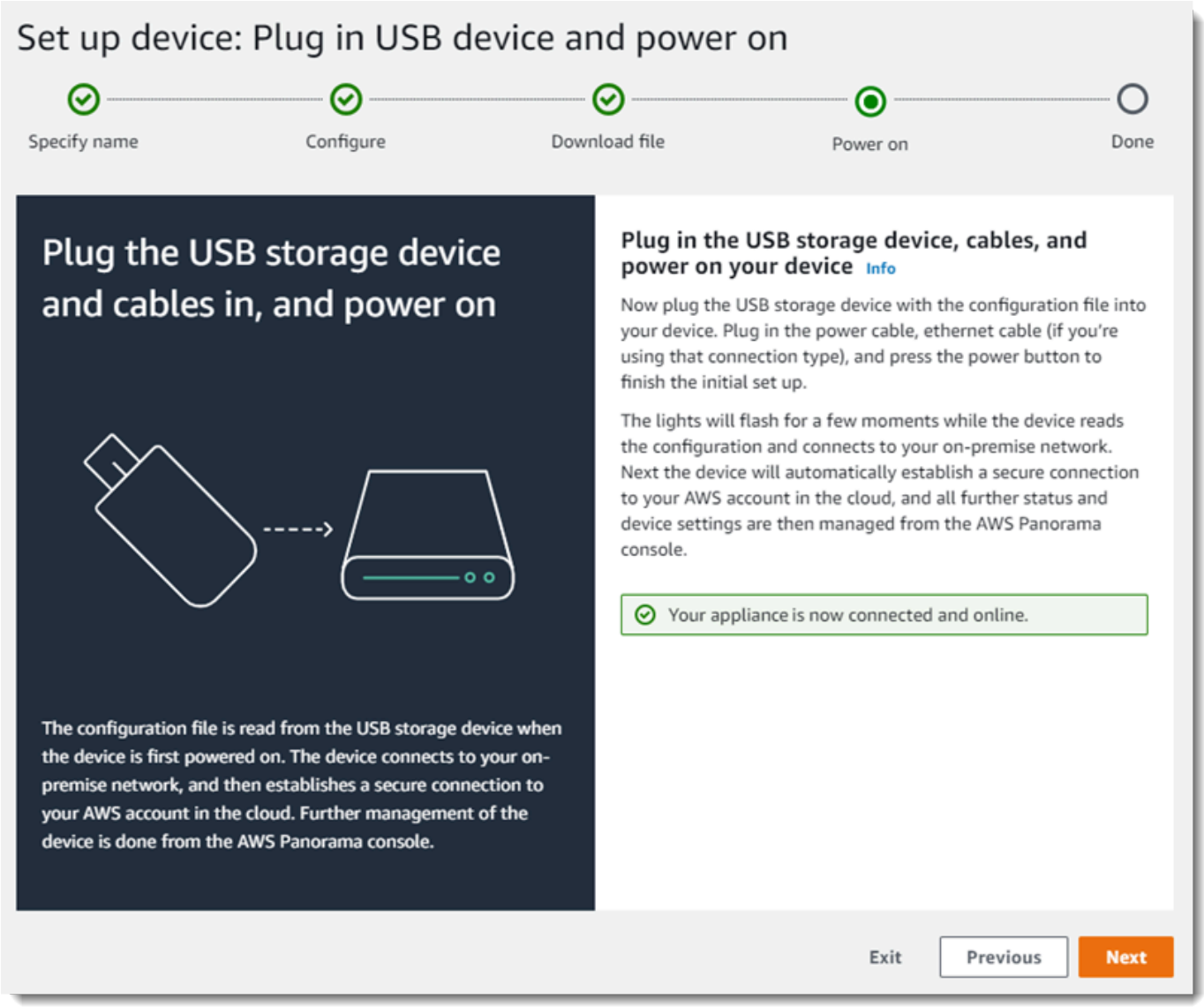
To register a appliance

1. Connect the USB drive to your computer. Prepare the appliance by connecting the network and power cables. The appliance powers on and waits for a USB drive to be connected.
2. Open the AWS Panorama console Getting started page.
3. Choose **Add device**.
4. Choose **Begin setup**.
5. Enter a name and description for the device resource that represents the appliance in AWS Panorama. Choose **Next**



6. If you need to manually assign an IP address, NTP server, or DNS settings, choose **Advanced network settings**. Otherwise, choose **Next**.
7. Choose **Download archive**. Choose **Next**.
8. Copy the configuration archive to the root directory of the USB drive.
9. Connect the USB drive to the USB 3.0 port on the front of the appliance, next to the HDMI port.

   When you connect the USB drive, the appliance copies the configuration archive and network configuration file to itself and connects to the AWS Cloud. The appliance's status light turns from green to blue while it completes the connection, and then back to green.
10. To continue, choose **Next**.



11. Choose **Done**.

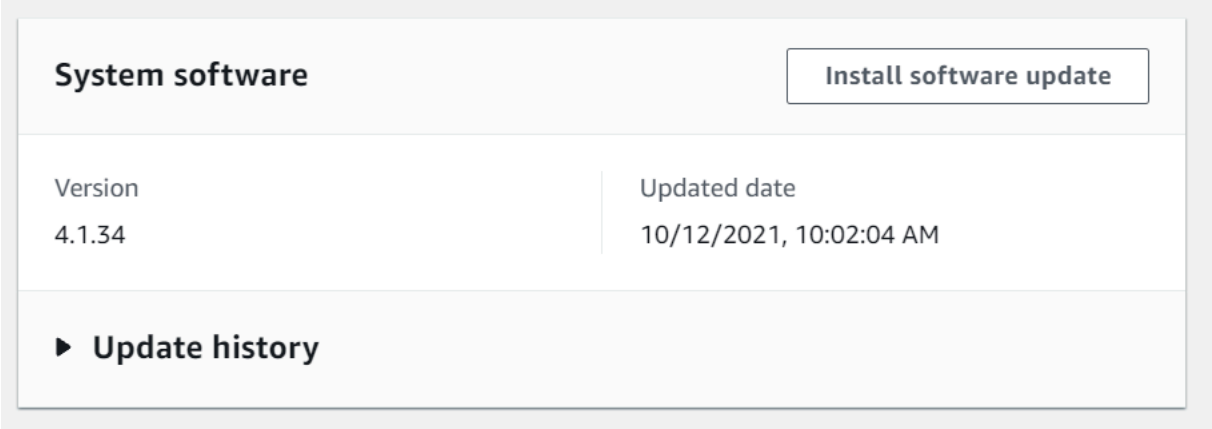
Upgrade the appliance software

The AWS Panorama Appliance has several software components, including a Linux operating system, the AWS Panorama application SDK, and supporting computer vision libraries and frameworks. To ensure that you can use the latest features and applications with your appliance, upgrade its software after setup and whenever an update is available.

To update the appliance software

1. Open the AWS Panorama console Devices page.

2. Choose an appliance.
3. Choose **Settings**
4. Under **System software**, choose **Install software update**.



5. Choose a new version and then choose **Install**.

> **Important**
>
> Before you continue, remove the USB drive from the appliance and format it to delete its contents. The configuration archive contains sensitive data and is not deleted automatically.

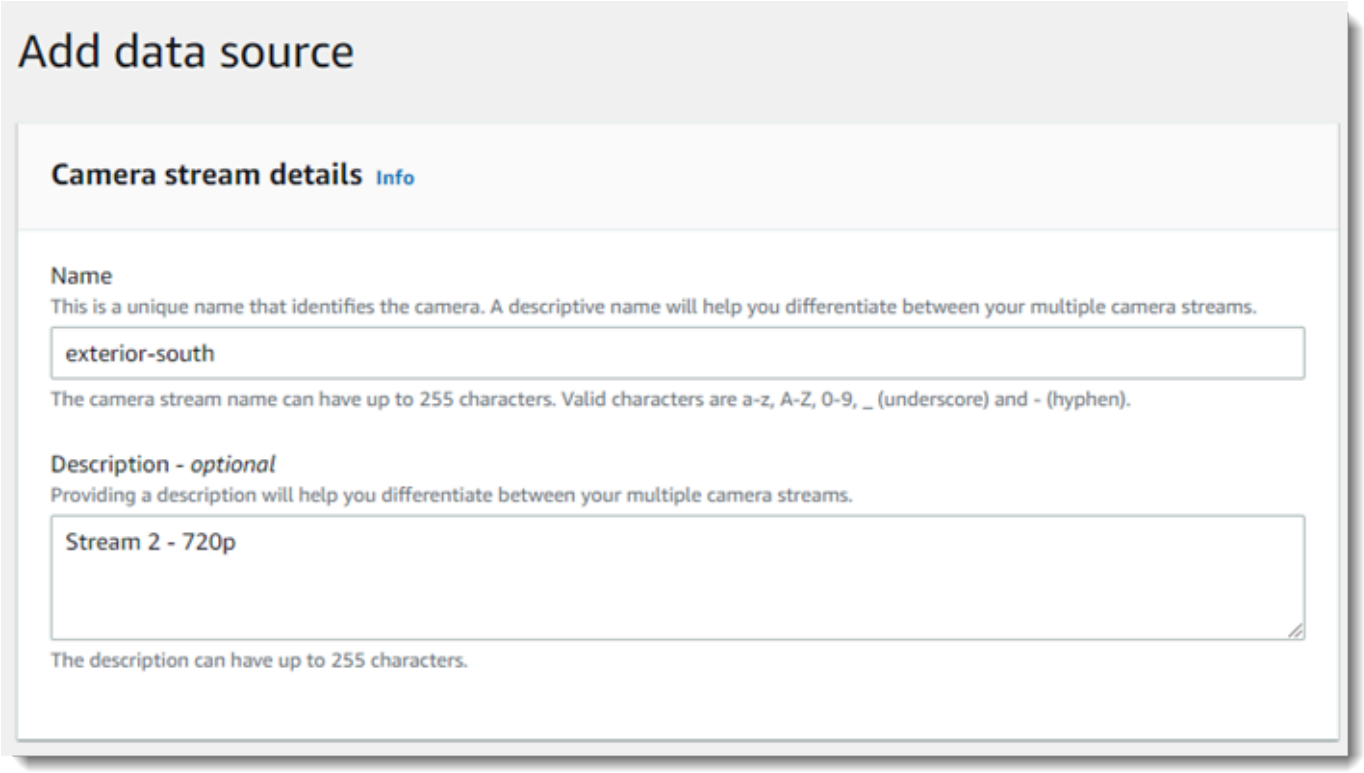
The upgrade process can take 30 minutes or more. You can monitor its progress in the AWS Panorama console or on a connected monitor. When the process completes, the appliance reboots.

Add a camera stream

Next, register a camera stream with the AWS Panorama console.

To register a camera stream

1. Open the AWS Panorama console Data sources page.
2. Choose **Add data source**.



3. Configure the following settings.

   - **Name** – A name for the camera stream.
   - **Description** – A short description of the camera, its location, or other details.
   - **RTSP URL** – A URL that specifies the camera's IP address and the path to the stream. For example, `rtsp://192.168.0.77/live/mpeg4/`
   - **Credentials** – If the camera stream is password protected, specify the username and password.
4. Choose **Save**.

AWS Panorama stores your camera's credentials securely in AWS Secrets Manager. Multiple applications can process the same camera stream simultaneously.

Next steps

If you encountered errors during setup, see Troubleshooting.

To deploy a sample application, continue to the next topic.

Deploying the AWS Panorama sample application

After you've set up your AWS Panorama Appliance or compatible device and upgraded its software, deploy a sample application. In the following sections, you import a sample application with the AWS Panorama Application CLI and deploy it with the AWS Panorama console.

The sample application uses a machine learning model to classify objects in frames of video from a network camera. It uses the AWS Panorama Application SDK to load a model, get images, and run the model. The application then overlays the results on top of the original video and outputs it to a connected display.

In a retail setting, analyzing foot traffic patterns enables you to predict traffic levels. By combining the analysis with other data, you can plan for increased staffing needs around holidays and other events, measure the effectiveness of advertisements and sales promotions, or optimize display placement and inventory management.

Sections

Prerequisites

To follow the procedures in this tutorial, you need a command line terminal or shell to run commands. In the code listings, commands are preceded by a prompt symbol ($) and the name of the current directory, when appropriate.

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

For long commands, we use an escape character (\) to split a command over multiple lines.

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can install the Windows Subsystem for Linux to get a Windows-integrated version of Ubuntu and Bash. For help setting up a development environment in Windows, see Setting up a development environment in Windows.

You use Python to develop AWS Panorama applications and install tools with pip, Python's package manager. If you don't already have Python, install the latest version. If you have Python 3 but not pip, install pip with your operating system's package manager, or install a new version of Python, which comes with pip.

In this tutorial, you use Docker to build the container that runs your application code. Install Docker from the Docker website: Get Docker

This tutorial uses the AWS Panorama Application CLI to import the sample application, build packages, and upload artifacts. The AWS Panorama Application CLI uses the AWS Command Line Interface (AWS CLI) to call service API operations. If you already have the AWS CLI, upgrade it to the latest version. To install the AWS Panorama Application CLI and AWS CLI, use `pip`.

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

Download the sample application, and extract it into your workspace.

- **Sample application** – aws-panorama-sample.zip

Import the sample application

To import the sample application for use in your account, use the AWS Panorama Application CLI. The application's folders and manifest contain references to a placeholder account number. To update these with your account number, run the `panorama-cli import-application` command.

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

The `SAMPLE_CODE` package, in the `packages` directory, contains the application's code and configuration, including a Dockerfile that uses the application base image, `panorama-application`. To build the application container that runs on the appliance, use the `panorama-cli build-container` command.

```
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
```

The final step with the AWS Panorama Application CLI is to register the application's code and model nodes, and upload assets to an Amazon S3 access point provided by the service. The assets include the code's container image, the model, and a descriptor file for each. To register the nodes and upload assets, run the `panorama-cli package-application` command.

```
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
```

Deploy the application

Use the AWS Panorama console to deploy the application to your appliance.

To deploy the application

1. Open the AWS Panorama console Deployed applications page.
2. Choose **Deploy application**.
3. Paste the contents of the application manifest, `graphs/aws-panorama-sample/graph.json`, into the text editor. Choose **Next**.
4. For **Application name**, enter `aws-panorama-sample`.
5. Choose **Proceed to deploy**.
6. Choose **Begin deployment**.
7. Choose **Next** without selecting a role.
8. Choose **Select device**, and then choose your appliance. Choose **Next**.
9. On the **Select data sources** step, choose **View input(s)**, and add your camera stream as a data source. Choose **Next**.
10. On the **Configure** step, choose **Next**.

11. Choose **Deploy**, and then choose **Done**.
12. In the list of deployed applications, choose **aws-panorama-sample**.

Refresh this page for updates, or use the following script to monitor the deployment from the command line.

Example monitor-deployment.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": {}
    }
```

```
]
...
```

If the application doesn't start running, check the application and device logs in Amazon CloudWatch Logs.

View the output

When the deployment is complete, the application starts processing the video stream and sends logs to CloudWatch.

To view logs in CloudWatch Logs

1. Open the Log groups page of the CloudWatch Logs console.
2. Find AWS Panorama application and appliance logs in the following groups:

   - **Device logs** – `/aws/panorama/devices/`*`device-id`*
   - **Application logs** – `/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
```

To view the application's video output, connect the appliance to a monitor with an HDMI cable. By default, the application shows any classification result that has more than 20% confidence.

Example [squeezenet_classes.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",
...
```

The sample model has 1000 classes including many animals, food, and common objects. Try pointing your camera at a keyboard or coffee mug.

For simplicity, the sample application uses a lightweight classification model. The model outputs a single array with a probability for each of its classes. Real-world applications more frequently use object detection models that have multidimensional output. For sample applications with more complex models, see Sample applications, scripts, and templates.

Enable the SDK for Python

The sample application uses the AWS SDK for Python (Boto) to send metrics to Amazon CloudWatch. To enable this functionality, create a role that grants the application permission to send metrics, and redeploy the application with the role attached.

The sample application includes a AWS CloudFormation template that creates a role with the permissions that it needs. To create the role, use the `aws cloudformation deploy` command.

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

To redeploy the application

1. Open the AWS Panorama console Deployed applications page.
2. Choose an application.
3. Choose **Replace**.
4. Complete the steps to deploy the application. In the **Specify IAM role**, choose the role that you created. Its name starts with `aws-panorama-sample-runtime`.
5. When the deployment completes, open the CloudWatch console and view the metrics in the `AWSPanoramaApplication` namespace. Every 150 frames, the application logs and uploads metrics for frame processing and inference time.

Clean up

If you are done working with the sample application, you can use the AWS Panorama console to remove it from the appliance.

To remove the application from the appliance

1. Open the AWS Panorama console Deployed applications page.
2. Choose an application.

3. Choose **Delete from device**.

Next steps

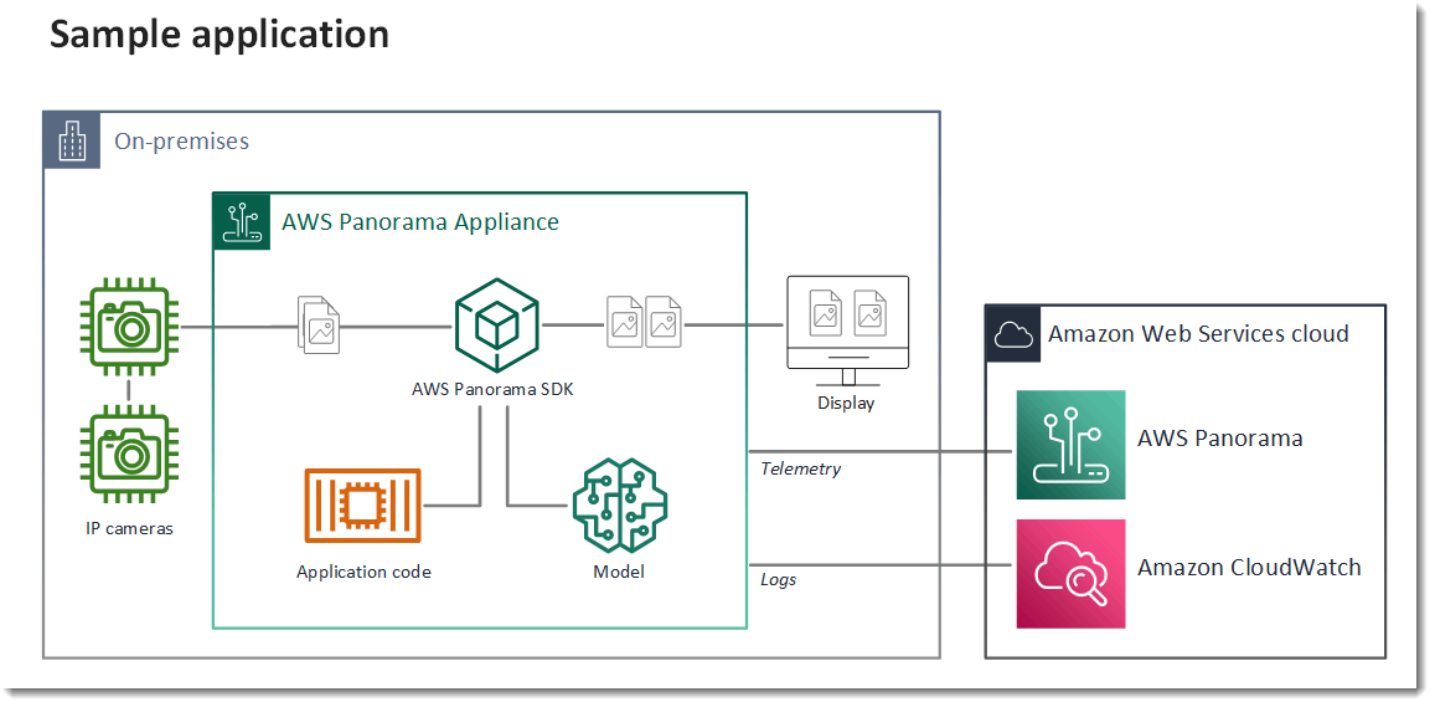
If you encountered errors while deploying or running the sample application, see Troubleshooting.

To learn more about the sample application's features and implementation, continue to the next topic.

Developing AWS Panorama applications

You can use the sample application to learn about AWS Panorama application structure, and as a starting point for your own application.

The following diagram shows the major components of the application running on an AWS Panorama Appliance. The application code uses the AWS Panorama Application SDK to get images and interact with the model, which it doesn't have direct access to. The application outputs video to a connected display but does not send image data outside of your local network.



In this example, the application uses the AWS Panorama Application SDK to get frames of video from a camera, preprocess the video data, and send the data to a computer vision model that detects objects. The application displays the result on an HDMI display connected to the appliance.

Sections

The application manifest

The application manifest is a file named `graph.json` in the `graphs` folder. The manifest defines the application's components, which are packages, nodes, and edges.

Packages are code, configuration, and binary files for application code, models, cameras, and displays. The sample application uses 4 packages:

Example `graphs/aws-panorama-sample/graph.json` – Packages

```
    "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"
        }
    ],
```

The first two packages are defined within the application, in the `packages` directory. They contain the code and model specific to this application. The second two packages are generic camera and display packages provided by the AWS Panorama service. The `abstract_rtsp_media_source` package is a placeholder for a camera that you override during deployment. The `hdmi_data_sink` package represents the HDMI output connector on the device.

Nodes are interfaces to packages, as well as non-package parameters that can have default values that you override at deploy time. The code and model packages define interfaces in `package.json` files that specify inputs and outputs, which can be video streams or a basic data type such as a float, boolean, or string.

For example, the `code_node` node refers to an interface from the `SAMPLE_CODE` package.

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

This interface is defined in the package configuration file, `package.json`. The interface specifies that the package is business logic and that it takes a video stream named `video_in` and a floating point number named `threshold` as inputs. The interface also specifies that the code requires a video stream buffer named `video_out` to output video to a display

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"
```

```
                    }
                ]
            }
        ]
    }
}
```

Back in the application manifest, the `camera_node` node represents a video stream from a camera. It includes a decorator that appears in the console when you deploy the application, prompting you to choose a camera stream.

Example `graphs/aws-panorama-sample/graph.json` – Camera node

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

A parameter node, `threshold_param`, defines the confidence threshold parameter used by the application code. It has a default value of 60, and can be overriden during deployment.

Example `graphs/aws-panorama-sample/graph.json` – Parameter node

```
            {
                "name": "threshold_param",
                "interface": "float32",
                "value": 60.0,
                "overridable": true,
                "decorator": {
                    "title": "Confidence threshold",
                    "description": "The minimum confidence for a classification to be
 recorded."
                }
            }
```

The final section of the application manifest, `edges`, makes connections between nodes. The camera's video stream and the threshold parameter connect to the input of the code node, and the video output from the code node connects to the display.

Example `graphs/aws-panorama-sample/graph.json` – Edges

```
    "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"
        }
    ]
```

Building with the sample application

You can use the sample application as a starting point for your own application.

The name of each package must be unique in your account. If you and another user in your account both use a generic package name such as `code` or `model`, you might get the wrong version of the package when you deploy. Change the name of the code package to one that represents your application.

To rename the code package

1. Rename the package folder: `packages/123456789012-`*`SAMPLE_CODE`*`-1.0/`.
2. Update the package name in the following locations.

   - **Application manifest** – `graphs/aws-panorama-sample/graph.json`
   - **Package configuration** – `packages/123456789012-SAMPLE_CODE-1.0/package.json`
   - **Build script** – `3-build-container.sh`

To update the application's code

1. Modify the application code in `packages/123456789012-SAMPLE_CODE-1.0/src/application.py`.
2. To build the container, run `3-build-container.sh`.

```
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
```

   The CLI automatically deletes the old container asset from the `assets` folder and updates the package configuration.
3. To upload the packages, run `4-package-application.py`.
4. Open the AWS Panorama console Deployed applications page.
5. Choose an application.
6. Choose **Replace**.

7. Complete the steps to deploy the application. If needed, you can make changes to the application manifest, camera streams, or parameters.

Changing the computer vision model

The sample application includes a computer vision model. To use your own model, modify the model node's configuration, and use the AWS Panorama Application CLI to import it as an asset.

The following example uses an MXNet SSD ResNet50 model that you can download from this guide's GitHub repo: ssd_512_resnet50_v1_voc.tar.gz

To change the sample application's model

1. Rename the package folder to match your model. For example, to `packages/123456789012-SSD_512_RESNET50_V1_VOC-1.0/`.
2. Update the package name in the following locations.

   - **Application manifest** – `graphs/aws-panorama-sample/graph.json`
   - **Package configuration** – `packages/123456789012-SSD_512_RESNET50_V1_VOC-1.0/package.json`
3. In the package configuration file (`package.json`). Change the `assets` value to a blank array.

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

4. Open the package descriptor file (`descriptor.json`). Update the `framework` and `shape` values to match your model.

```
{
    "mlModelDescriptor": {
        "envelopeVersion": "2021-01-01",
        "framework": "MXNET",
        "inputs": [
            {
```

```
                "name": "data",
                "shape": [ 1, 3, 512, 512 ]
            }
        ]
    }
}
```

The value for **shape**, `1,3,512,512`, indicates the number of images that the model takes as input (1), the number of channels in each image (3--red, green, and blue), and the dimensions of the image (512 x 512). The values and order of the array varies among models.

5. Import the model with the AWS Panorama Application CLI. The AWS Panorama Application CLI copies the model and descriptor files into the `assets` folder with unique names, and updates the package configuration.

```
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. To upload the model, run `panorama-cli package-application`.

```
$ 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. Update the application code. Most of the code can be reused. The code specific to the model's response is in the `process_results` method.

```
    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)
```

   Depending on your model, you might also need to update the `preprocess` method.

Preprocessing images

Before the application sends an image to the model, it prepares it for inference by resizing it and normalizing color data. The model that the application uses requires a 224 x 224 pixel image with

three color channels, to match the number of inputs in its first layer. The application adjusts each color value by converting it to a number between 0 and 1, subtracting the average value for that color, and dividing by the standard deviation. Finally, it combines the color channels and converts it to a NumPy array that the model can process.

Example application.py – Preprocessing

```
    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)
```

This process gives the model values in a predictable range centered around 0. It matches the preprocessing applied to images in the training dataset, which is a standard approach but can vary per model.

Uploading metrics with the SDK for Python

The sample application uses the SDK for Python to upload metrics to Amazon CloudWatch.

Example application.py – SDK for Python

```
    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.")
```

It gets permission from a runtime role that you assign during deployment. The role is defined in the `aws-panorama-sample.yml` AWS CloudFormation template.

Example [aws-panorama-sample.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/
```

The sample application installs the SDK for Python and other dependencies with pip. When you build the application container, the `Dockerfile` runs commands to install libraries on top of what comes with the base image.

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
```

To use the AWS SDK in your application code, first modify the template to add permissions for all API actions that the application uses. Update the AWS CloudFormation stack by running the `1-create-role.sh` each time you make a change. Then, deploy changes to your application code.

For actions that modify or use existing resources, it is a best practice to minimize the scope of this policy by specifying a name or pattern for the target `Resource` in a separate statement. For details on the actions and resources supported by each service, see Action, resources, and condition keys in the Service Authorization Reference

Next steps

For instructions on using the AWS Panorama Application CLI to build applications and create packages from scratch, see the CLI's README.

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

For more sample code and a test utility that you can use to validate your application code prior to deploying, visit the AWS Panorama samples repository.

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

Supported computer vision models and cameras

AWS Panorama supports models built with PyTorch, Apache MXNet, and TensorFlow. When you deploy an application, AWS Panorama compiles your model in SageMaker Neo. You can build models in Amazon SageMaker or in your development environment, as long as you use layers that are compatible with SageMaker Neo.

To process video and get images to send to a model, the AWS Panorama Appliance connects to an H.264 encoded video stream with the RTSP protocol. AWS Panorama tests a variety of common cameras for compatibility.

Sections

Supported models

When you build an application for AWS Panorama, you provide a machine learning model that the application uses for computer vision. You can use pre-built and pre-trained models provided by model frameworks, a sample model, or a model that you build and train yourself.

> **Note**
>
> For a list of pre-built models that have been tested with AWS Panorama, see Model compatibility.

When you deploy an application, AWS Panorama uses the SageMaker Neo compiler to compile your computer vision model. SageMaker Neo is a compiler that optimizes models to run efficiently on a target platform, which can be an instance in Amazon Elastic Compute Cloud (Amazon EC2), or an edge device such as the AWS Panorama Appliance.

AWS Panorama supports the versions of PyTorch, Apache MXNet, and TensorFlow that are supported for edge devices by SageMaker Neo. When you build your own model, you can use the framework versions listed in the SageMaker Neo release notes. In SageMaker, you can use the built-in image classification algorithm.

For more information about using models in AWS Panorama, see Computer vision models.

Supported cameras

The AWS Panorama Appliance supports H.264 video streams from cameras that output RTSP over a local network. For camera streams greater than 2 megapixels, the appliance scales down the image to 1920x1080 pixels or an equivalent size that preserves the stream's aspect ratio.

The following camera models have been tested for compatibility with the AWS Panorama Appliance:

- 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

For the appliance's hardware specifications, see AWS Panorama Appliance specifications.

AWS Panorama Appliance specifications

The AWS Panorama Appliance has the following hardware specifications. For other compatible devices, refer to the manufacturer's documentation.

Component	Specification
Processor and GPU	Nvidia Jetson AGX Xavier with 32GB RAM
Ethernet	2x 1000 Base-T (Gigabyte)
USB	1x USB 2.0 and 1x USB 3.0 type-A female
HDMI output	2.0a
Dimensions	7.75″ x 9.6″ x 1.6″ (197mm x 243mm x 40mm)
Weight	3.7lbs (1.7kg)
Power supply	100V-240V 50-60Hz AC 65W
Power input	IEC 60320 C6 (3-pin) receptacle
Dust and liquid protection	IP-62
EMI/EMC regulatory compliance	FCC Part-15 (US)
Thermal touch limits	IEC-62368
Operating temperature	-20°C to 60°C
Operating humidity	0% to 95% RH
Storage temperature	-20°C to 85°C
Storage humidity	Uncontrolled for low temperature. 90% RH at high temperature
Cooling	Forced-air heat extraction (fan)
Mounting options	Rackmount or free standing

Component	Specification
Power cord	6-foot (1.8 meter)
Power control	Push-button
Reset	Momentary switch
Status and network LEDs	Programmable 3-color RGB LED

Wi-Fi, Bluetooth and SD card storage are present on the appliance but are not usable.

The AWS Panorama Appliance includes two screws for mounting on a server rack. You can mount two appliances side-by-side on a 19-inch rack.

Service quotas

AWS Panorama applies quotas to the resources that you create in your account and the applications that you deploy. If you use AWS Panorama in multiple AWS Regions, quotas apply separately to each Region. AWS Panorama quotas are not adjustable.

Resources in AWS Panorama include devices, application node packages, and application instances.

- **Devices** – Up to 50 registered appliances per Region.
- **Node packages** – 50 packages per Region, with up to 20 versions per package.
- **Application instances** – Up to 10 applications per device. Each application can monitor up to 8 camera streams. Deployments are limited to 200 per day for each device.

When you use the AWS Panorama Application CLI, AWS Command Line Interface, or AWS SDK with the AWS Panorama service, quotas apply to the number of API calls that you make. You can make up to 5 requests total per second. A subset of API operations that create or modify resources apply an additional limit of 1 request per second.

For a complete list of quotas, visit the Service Quotas console, or see AWS Panorama endpoints and quotas in the Amazon Web Services General Reference.

AWS Panorama permissions

You can use AWS Identity and Access Management (IAM) to manage access to the AWS Panorama service and resources like appliances and applications. For users in your account that use AWS Panorama, you manage permissions in a permissions policy that you can apply to IAM roles. To manage permissions for an application, you create a role and assign it to the application.

To manage permissions for users in your account, use the managed policy that AWS Panorama provides, or write your own. You need permissions to other AWS services to get application and appliance logs, view metrics, and assign a role to an application.

An AWS Panorama Appliance also has a role that grants it permission to access AWS services and resources. The appliance's role is one of the service roles that the AWS Panorama service uses to access other services on your behalf.

An application role is a separate service role that you create for an application, to grant it permission to use AWS services with the AWS SDK for Python (Boto). To create an application role, you need administrative privileges or the help of an administrator.

You can restrict user permissions by the resource an action affects and, in some cases, by additional conditions. For example, you can specify a pattern for the Amazon Resource Name (ARN) of an application that requires a user to include their user name in the name of applications that they create. For the resources and conditions that are supported by each action, see Actions, resources, and condition keys for AWS Panorama in the Service Authorization Reference.

For more information, see What is IAM? in the IAM User Guide.

Topics

Identity-based IAM policies for AWS Panorama

To grant users in your account access to AWS Panorama, you use identity-based policies in AWS Identity and Access Management (IAM). Apply identity-based policies to IAM roles that are associated with a user. You can also grant users in another account permission to assume a role in your account and access your AWS Panorama resources.

AWS Panorama provides managed policies that grant access to AWS Panorama API actions and, in some cases, access to other services used to develop and manage AWS Panorama resources. AWS Panorama updates the managed policies as needed, to ensure that your users have access to new features when they're released.

- **AWSPanoramaFullAccess** – Provides full access to AWS Panorama, AWS Panorama access points in Amazon S3, appliance credentials in AWS Secrets Manager, and appliance logs in Amazon CloudWatch. Includes permission to create a service-linked role for AWS Panorama. View policy

The `AWSPanoramaFullAccess` policy allows you to tag AWS Panorama resources, but does not have all tag-related permissions used by the AWS Panorama console. To grant these permissions, add the following policy.

- **ResourceGroupsandTagEditorFullAccess** – View policy

The `AWSPanoramaFullAccess` policy does not include permission to purchase devices from the AWS Panorama console. To grant these permissions, add the following policy.

- **ElementalAppliancesSoftwareFullAccess** – View policy

Managed policies grant permission to API actions without restricting the resources that a user can modify. For finer-grained control, you can create your own policies that limit the scope of a user's permissions. Use the full-access policy as a starting point for your policies.

> **Creating service roles**
>
> The first time you use the AWS Panorama console, you need permission to create the service role used by the AWS Panorama Appliance. A service role gives a service permission to manage resources or interact with other services. Create this role before granting access to your users.

For details on the resources and conditions that you can use to limit the scope of a user's permissions in AWS Panorama, see Actions, resources, and condition keys for AWS Panorama in the Service Authorization Reference.

AWS Panorama service roles and cross-service resources

AWS Panorama uses other AWS services to manage the AWS Panorama Appliance, store data, and import application resources. A service role gives a service permission to manage resources or interact with other services. When you sign in to the AWS Panorama console for the first time, you create the following service roles:

- **AWSServiceRoleForAWSPanorama** – Allows AWS Panorama to manage resources in AWS IoT, AWS Secrets Manager, and AWS Panorama.

  Managed policy: AWSPanoramaServiceLinkedRolePolicy
- **AWSPanoramaApplianceServiceRole** – Allows an AWS Panorama Appliance to upload logs to CloudWatch, and to get objects from Amazon S3 access points created by AWS Panorama.

  Managed policy: AWSPanoramaApplianceServiceRolePolicy

To view the permissions attached to each role, use the IAM console. Wherever possible, the role's permissions are restricted to resources that match a naming pattern that AWS Panorama uses. For example, `AWSServiceRoleForAWSPanorama` grants only permission for the service to access AWS IoT resources that have `panorama` in their name.

Sections

Securing the appliance role

The AWS Panorama Appliance uses the `AWSPanoramaApplianceServiceRole` role to access resources in your account. The appliance has permission to upload logs to CloudWatch Logs, read camera stream credentials from AWS Secrets Manager, and to access application artifacts in Amazon Simple Storage Service (Amazon S3) access points that AWS Panorama creates.

> **Note**
>
> Applications don't use the appliance's permissions. To give your application permission to use AWS services, create an application role.

AWS Panorama uses the same service role with all appliances in your account, and does not use roles across accounts. For an added layer of security, you can modify the appliance role's trust policy to enforce this explicitly, which is a best practice when you use roles to grant a service permission to access resources in your account.

To update the appliance role trust policy

1. Open the appliance role in the IAM console: AWSPanoramaApplianceServiceRole
2. Choose **Edit trust relationship**.
3. Update the policy contents and then choose **Update trust policy**.

The following trust policy includes a condition that ensures that when AWS Panorama assumes the appliance role, it is doing so for an appliance in your account. The `aws:SourceAccount` condition compares the account ID specified by AWS Panorama to the one that you include in the policy.

Example trust policy – Specific account

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

If you want to restrict AWS Panorama further, and allow it to only assume the role with a specific device, you can specify the device by ARN. The `aws:SourceArn` condition compares the ARN of the appliance specified by AWS Panorama to the one that you include in the policy.

Example trust policy – Single appliance

```
{
  "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"
        }
      }
    }
  ]
}
```

If you reset and reprovision the appliance, you must remove the source ARN condition temporarily and then add it again with the new device ID.

For more information on these conditions, and security best practices when services use roles to access resources in your account, see The confused deputy problem in the IAM User Guide.

Use of other services

AWS Panorama creates or accesses resources in the following services:

- AWS IoT – Things, policies, certificates, and jobs for the AWS Panorama Appliance
- Amazon S3 – Access points for staging application models, code, and configurations.
- Secrets Manager – Short-term credentials for the AWS Panorama Appliance.

For information about Amazon Resource Name (ARN) format or permission scopes for each service, see the topics in the *IAM User Guide* that are linked to in this list.

Granting permissions to an application

You can create a role for your application to grant it permission to call AWS services. By default, applications do not have any permissions. You create an application role in IAM and assign it to an application during deployment. To grant your application only the permissions that it needs, create a role for it with permissions for specific API actions.

The sample application includes an AWS CloudFormation template and script that create an application role. It is a service role that AWS Panorama can assume. This role grants permission for the application to call CloudWatch to upload metrics.

Example aws-panorama-sample.yml – Application role

```
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/
```

You can extend this script to grant permissions to other services, by specifying a list of API actions or patterns for the value of `Action`.

For more information on permissions in AWS Panorama, see AWS Panorama permissions.

Managing the AWS Panorama Appliance

The AWS Panorama Appliance is the hardware that runs your applications. You use the AWS Panorama console to register an appliance, update its software, and deploy applications to it. The software on the AWS Panorama Appliance connects to camera streams, sends frames of video to your application, and displays video output on an attached display.

After setting up your appliance or another compatible device, you register cameras for use with applications. You manage camera streams in the AWS Panorama console. When you deploy an application, you choose which camera streams the appliance sends to it for processing.

For tutorials that introduce the AWS Panorama Appliance with a sample application, see Getting started with AWS Panorama.

Topics

Managing an AWS Panorama Appliance

You use the AWS Panorama console to configure, upgrade or deregister the AWS Panorama Appliance and other compatible devices.

To set up an appliance, follow the instructions in the getting started tutorial. The setup process creates the resources in AWS Panorama that track your appliance and coordinate updates and deployments.

To register an appliance with the AWS Panorama API, see Automate device registration.

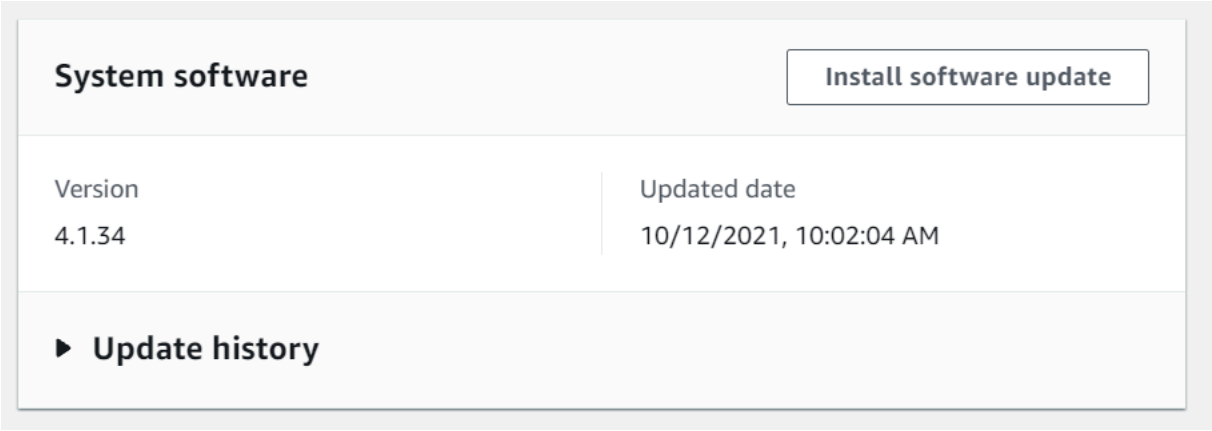
Sections

Update the appliance software

You view and deploy software updates for the appliance in the AWS Panorama console. Updates can be required or optional. When a required update is available, the console prompts you to apply it. You can apply optional updates on the appliance **Settings** page.

To update the appliance software

1. Open the AWS Panorama console Devices page.
2. Choose an appliance.
3. Choose **Settings**
4. Under **System software**, choose **Install software update**.



5. Choose a new version and then choose **Install**.

Deregister an appliance

If you are done working with an appliance, you can use the AWS Panorama console to deregister it and delete the associated AWS IoT resources.

To delete an appliance

1. Open the AWS Panorama console Devices page.
2. Choose the appliance's name.
3. Choose **Delete**.
4. Enter the appliance's name and choose **Delete**.

When you delete an appliance from the AWS Panorama service, data on the appliance is not deleted automatically. A deregistered appliance can't connect to AWS services and can't be registered again until it is reset.

Reboot an appliance

You can reboot an appliance remotely.

To reboot an appliance

1. Open the AWS Panorama console Devices page.
2. Choose the appliance's name.
3. Choose **Reboot**.

The console sends a message to the appliance to reboot it. To receive the signal, the appliance must be able to connect to AWS IoT. To reboot an appliance with the AWS Panorama API, see Reboot appliances.

Reset an appliance

To use an appliance in a different Region or with a different account, you must reset it and reprovision it with a new certificate. Resetting the device applies the most recent required software version and deletes all account data.

To start a reset operation, the appliance must be plugged in and powered down. Press and hold both the power and reset buttons for five seconds. When you release the buttons, the status light blinks orange. Wait until the status light blinks green before provisioning or disconnecting the appliance.

You can also reset the appliance software without deleting certificates from the device. For more information, see Power and reset buttons.

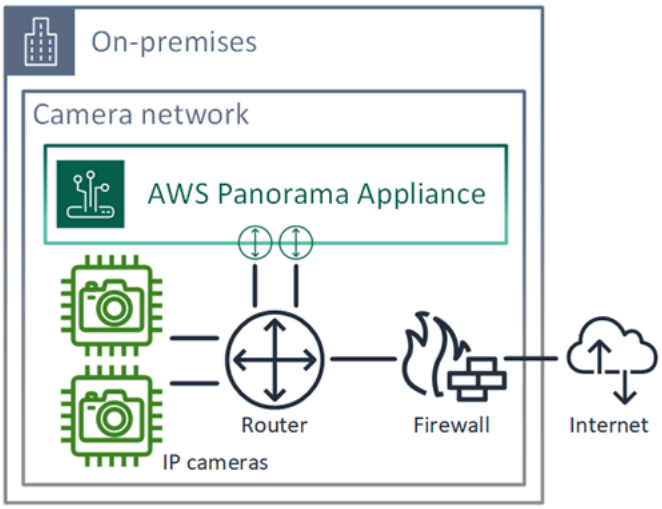
Connecting the AWS Panorama Appliance to your network

The AWS Panorama Appliance requires connectivity to both the AWS cloud and your on-premises network of IP cameras. You can connect the appliance to a single firewall that grants access to both, or connect each of the device's two network interfaces to a different subnet. In either case, you must secure the appliance's network connections to prevent unauthorized access to your camera streams.

Sections

Single network configuration

The appliance has two Ethernet ports. If you route all traffic to and from the device through a single router, you can use the second port for redundancy in case the physical connection to the first port is broken. Configure your router to allow the appliance to connect only to camera streams and the internet, and to block camera streams from otherwise leaving your internal network.

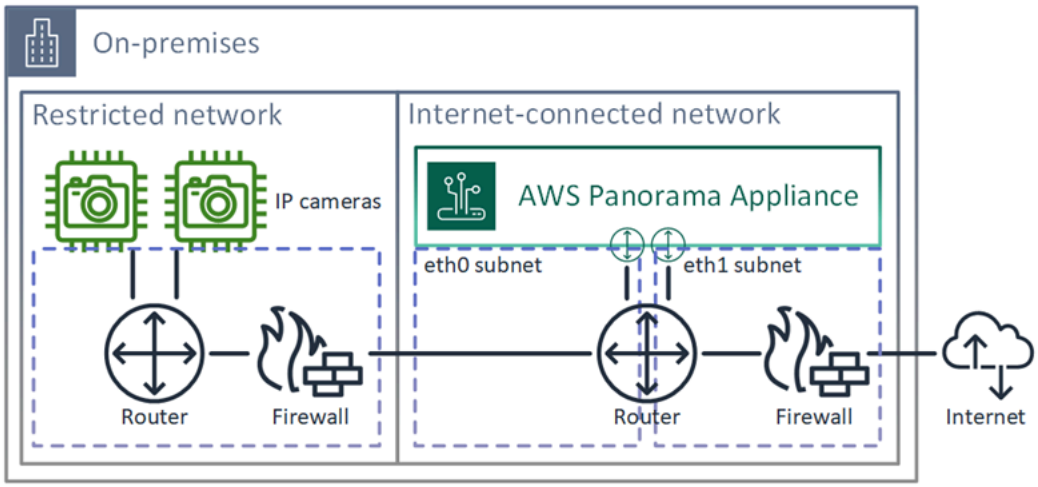


For details on the ports and endpoints that the appliance needs access to, see Configuring service access and Configuring local network access.

Dual network configuration

For an extra layer of security, you can place the appliance in an internet-connected network separate from your camera network. A firewall between your restricted camera network and the appliance's network only allows the appliance to access video streams. If your camera network was previously air-gapped for security purposes, you might prefer this method over connecting the camera network to a router that also grants access to the internet.

The following example shows the appliance connecting to a different subnet on each port. The router places the `eth0` interface on a subnet that routes to the camera network, and `eth1` on a subnet that routes to the internet.



You can confirm the IP address and MAC address of each port in the AWS Panorama console.

Configuring service access

During provisioning, you can configure the appliance to request a specific IP address. Choose an IP address ahead of time to simplify firewall configuration and ensure that the appliance's address doesn't change if it's offline for a long period of time.

The appliance uses AWS services to coordinate software updates and deployments. Configure your firewall to allow the appliance to connect to these endpoints.

Internet access

- **AWS IoT (HTTPS and MQTT, ports 443, 8443 and 8883)** – AWS IoT Core and device management endpoints. For details, see AWS IoT Device Management endpoints and quotas in the Amazon Web Services General Reference.

- **AWS IoT credentials (HTTPS, port 443)** – `credentials.iot.<region>.amazonaws.com` and subdomains.
- **Amazon Elastic Container Registry (HTTPS, port 443)** – `api.ecr.<region>.amazonaws.com`, `dkr.ecr.<region>.amazonaws.com` and subdomains.
- **Amazon CloudWatch (HTTPS, port 443)** – `monitoring.<region>.amazonaws.com`.
- **Amazon CloudWatch Logs (HTTPS, port 443)** – `logs.<region>.amazonaws.com`.
- **Amazon Simple Storage Service (HTTPS, port 443)** – `s3.<region>.amazonaws.com`, `s3-accesspoint.<region>.amazonaws.com` and subdomains.

If your application calls other AWS services, the appliance needs access to the endpoints for those services as well. For more information, see Service endpoints and quotas.

Configuring local network access

The appliance needs access to RTSP video streams locally, but not over the internet. Configure your firewall to allow the appliance to access RTSP streams on port 554 internally, and to not allow streams to go out to or come in from the internet.

Local access

- **Real-time streaming protocol (RTSP, port 554)** – To read camera streams.
- **Network time protocol (NTP, port 123)** – To keep the appliance's clock in sync. If you don't run an NTP server on your network, the appliance can also connect to public NTP servers over the internet.

Private connectivity

The AWS Panorama Appliance does not need internet access if you deploy it in a private VPC subnet with a VPN connection to AWS. You can use Site-to-Site VPN or AWS Direct Connect to create a VPN connection between an on-premises router and AWS. Within your private VPC subnet, you create endpoints that let the appliance connect to Amazon Simple Storage Service, AWS IoT, and other services. For more information, see Connecting an appliance to a private subnet.

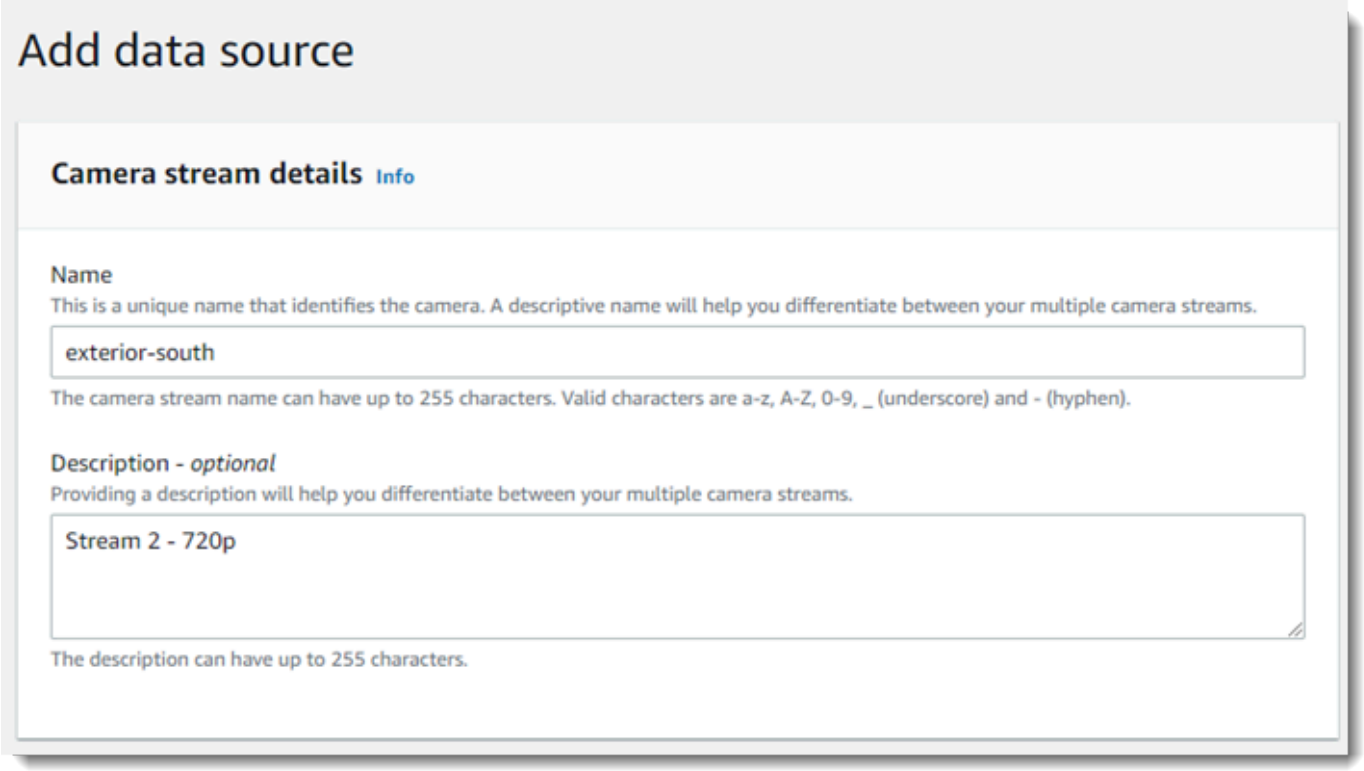
Managing camera streams in AWS Panorama

To register video streams as data sources for your application, use the AWS Panorama console. An application can process multiple streams simultaneously and multiple appliances can connect to the same stream.

> **Important**
>
> An application can connect to any camera stream that is routable from the local network it connects to. To secure your video streams, configure your network to allow only RTSP traffic locally. For more information, see Security in AWS Panorama.

To register a camera stream

1. Open the AWS Panorama console Data sources page.
2. Choose **Add data source**.



3. Configure the following settings.

   - **Name** – A name for the camera stream.

- **Description** – A short description of the camera, its location, or other details.
- **RTSP URL** – A URL that specifies the camera's IP address and the path to the stream. For example, `rtsp://192.168.0.77/live/mpeg4/`
- **Credentials** – If the camera stream is password protected, specify the username and password.

4. Choose **Save**.

To register a camera stream with the AWS Panorama API, see Automate device registration.

For a list of cameras that are compatible with the AWS Panorama Appliance, see Supported computer vision models and cameras.

Removing a stream

You can delete a camera stream in the AWS Panorama console.

To remove a camera stream

1. Open the AWS Panorama console Data sources page.
2. Choose a camera stream.
3. Choose **Delete data source**.

Removing a camera stream from the service does not stop running applications or delete camera credentials from Secrets Manager. To delete secrets, use the Secrets Manager console.

Manage applications on an AWS Panorama Appliance

An application is a combination of code, models, and configuration. From the **Devices** page in the AWS Panorama console, you can manage applications on the appliance.

To manage applications on an AWS Panorama Appliance

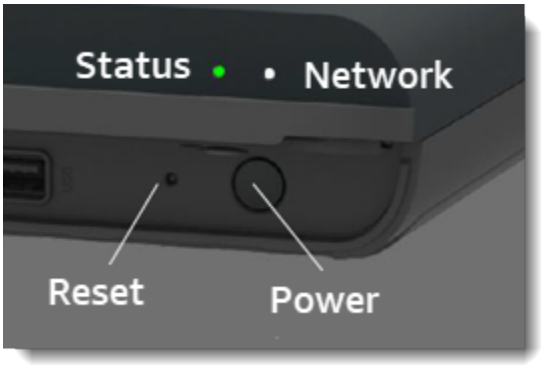
1. Open the AWS Panorama console Devices page.
2. Choose an appliance.

The **Deployed applications** page shows applications that have been deployed to the appliance.

Use the options on this page to remove deployed applications from the appliance, or replace a running application with a new version. You can also clone an application (running or deleted) to deploy a new copy of it.

AWS Panorama Appliance buttons and lights

The AWS Panorama Appliance has two LED lights above the power button that indicate the device status and network connectivity.



Status light

The LEDs change color and blink to indicate status. A slow blink is once every three seconds. A fast blink is once per second.

Status LED states

- **Fast blinking green** – The appliance is booting up.
- **Solid green** – The appliance is operating normally.
- **Slow blinking blue** – The appliance is copying configuration files and attempting to register with AWS IoT.
- **Fast blinking blue** – The appliance is copying a log image to a USB drive.
- **Fast blinking red** – The appliance encountered an error during startup or is overheated.
- **Slow blinking orange** – The appliance is restoring the latest software version.
- **Fast blinking orange** – The appliance is restoring the minimum software version.

Network light

The network LED has the following states:

Network LED states

- **Solid green** – An Ethernet cable is connected.

- **Blinking green** – The appliance is communicating over the network.
- **Solid red** – An Ethernet cable is not connected.

Power and reset buttons

The power and reset buttons are on the front of the device underneath a protective cover. The reset button is smaller and recessed. Use a small screwdriver or paperclip to press it.

To reset an appliance

1. The appliance must be plugged in and powered off. To power off the appliance, hold the power button for 1 second and wait for the shutdown sequence to complete. The shutdown sequence takes about 10 seconds.
2. To reset the appliance, use the following button combinations. A short press is 1 second. A long press is 5 seconds. For operations that require multiple buttons, press and hold both buttons simultaneously.

   - **Full reset** – Long press power and reset.

     Restores the minimum software version and deletes all configuration files and applications.
   - **Restore latest software version** – Short press reset.

     Reapplies the latest software update to the appliance.
   - **Restore minimum software version** – Long press reset.

     Reapplies the latest required software update to the appliance.
3. Release both buttons. The appliance powers on and the status light blinks orange for several minutes.
4. When the appliance is ready, the status light blinks green.

Resetting an appliance does not delete it from the AWS Panorama service. For more information, see Deregister an appliance.

Managing AWS Panorama applications

Applications run on the AWS Panorama Appliance to perform computer vision tasks on video streams. You can build computer vision applications by combining Python code and machine learning models, and deploy them to the AWS Panorama Appliance over the internet. Applications can send video to a display, or use the AWS SDK to send results to AWS services.

Topics

Deploy an application

To deploy an application, you use the AWS Panorama Application CLI import it to your account, build the container, upload and register assets, and create an application instance. This topic goes into each of these steps in detail and describes what goes on in the background.

If you have not deployed an application yet, see Getting started with AWS Panorama for a walkthrough.

For more information on customizing and extending the sample application, see Building AWS Panorama applications.

Sections

Install the AWS Panorama Application CLI

To install the AWS Panorama Application CLI and AWS CLI, use pip.

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

To build application images with the AWS Panorama Application CLI, you need Docker. On Linux, `qemu` and related system libraries are required as well. For more information on installing and configuring the AWS Panorama Application CLI, see the README file in the project's GitHub repository.

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

For instructions on setting up a build environment in Windows with WSL2, see Setting up a development environment in Windows.

Import an application

If you are working with a sample application or an application provided by a third party, use the AWS Panorama Application CLI to import the application.

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

This command renames application packages with your account ID. Package names start with the account ID of the account to which they are deployed. When you deploy an application to multiple accounts, you must import and package the application separately for each account.

For example, this guide's sample application a code package and a model package, each named with a placeholder account ID. The `import-application` command renames these to use the account ID that the CLI infers from your workspace's AWS credentials.

```
/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
```

`123456789012` is replaced with your account ID in the package directory names, and in the application manifest (`graph.json`), which refers to them. You can confirm your account ID by calling `aws sts get-caller-identity` with the AWS CLI.

```
$ aws sts get-caller-identity
{
    "UserId": "AIDAXMPL7W66UC3GFXMPL",
    "Account": "210987654321",
```

```
    "Arn": "arn:aws:iam::210987654321:user/devenv"
}
```

Build a container image

Your application code is packaged in a Docker container image, which includes the application code and libraries that you install in your Dockerfile. Use the AWS Panorama Application CLI `build-container` command to build a Docker image and export a filesystem image.

```
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
```

This command creates a Docker image named `code_asset` and exports a filesystem to a `.tar.gz` archive in the `assets` folder. The CLI pulls the application base image from Amazon Elastic Container Registry (Amazon ECR), as specified in the application's Dockerfile.

In addition to the container archive, the CLI creates an asset for the package descriptor (`descriptor.json`). Both files are renamed with a unique identifier that reflects a hash of the original file. The AWS Panorama Application CLI also adds a block to the package configuration that records the names of the two assets. These names are used by the appliance during the deployment process.

Example packages/123456789012-SAMPLE_CODE-1.0/package.json – with asset block

```
{
    "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"
                    },
```

The name of the code asset, specified in the `build-container` command, must match the value of the `asset` field in the package configuration. In the preceding example, both values are `code_asset`.

Import a model

Your application might have a model archive in its assets folder or that you download separately. If you have a new model, an updated model, or updated model descriptor file, use the `add-raw-model` command to import it.

```
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
```

If you just need to update the descriptor file, you can reuse the existing model in the assets directory. You might need to update the descriptor file to configure features such as floating point precision mode. For example, the following script shows how to do this with the sample app.

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
```

Changes to the descriptor file in the model package directory are not applied until you reimport it with the CLI. The CLI updates the model package configuration with the new asset names in-place, similar to how it updates the configuration for the application code package when you rebuild a container.

Upload application assets

To upload and register the application's assets, which include the model archive, container filesystem archive, and their descriptor files, use the `package-application` command.

```
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
```

```
...
```

If there are no changes to an asset file or the package configuration, the CLI skips it.

```
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
```

The CLI uploads the assets for each package to an Amazon S3 access point that is specific to your account. AWS Panorama manages the access point for you, and provides information about it through the DescribePackage API. The CLI uploads the assets for each package to the location provided for that package, and registers them with the AWS Panorama service with the settings described by the package configuration.

Deploy an application with the AWS Panorama console

You can deploy an application with the AWS Panorama console. During the deployment process, you choose which camera streams to pass to the application code, and configure options provided by the application's developer.

To deploy an application

1. Open the AWS Panorama console Deployed applications page.
2. Choose **Deploy application**.
3. Paste the contents of the application manifest, `graph.json`, into the text editor. Choose **Next**.
4. Enter a name and descroption.
5. Choose **Proceed to deploy**.
6. Choose **Begin deployment**.
7. If your application uses a role, choose it from the drop-down menu. Choose **Next**.
8. Choose **Select device**, and then choose your appliance. Choose **Next**.
9. On the **Select data sources** step, choose **View input(s)**, and add your camera stream as a data source. Choose **Next**.

10. On the **Configure** step, configure any application-specific settings defined by the developer. Choose **Next**.
11. Choose **Deploy**, and then choose **Done**.
12. In the list of deployed applications, choose the application to monitor its status.

The deployment process takes 15-20 minutes. The appliance's output can be blank for an extended period while the application starts. If you encounter an error, see Troubleshooting.

Automate application deployment

You can automate the application deployment process with the CreateApplicationInstance API. The API takes two configuration files as input. The application manifest specifies the packages used and their relationships. The second file is an overrides file that specifies deploy-time overrides of values in the application manifest. Using an overrides file lets you use the same application manifest to deploy the application with different camera streams, and configure other application-specific settings.

For more information, and example scripts for each of the steps in this topic, see Automate application deployment.

Managing applications in the AWS Panorama console

Use the AWS Panorama console to manage deployed applications.

Sections

Update or copy an application

To update an application, use the **Replace** option. When you replace an application, you can update its code or models.

To update an application

1. Open the AWS Panorama console Deployed applications page.
2. Choose an application.
3. Choose **Replace**.
4. Follow the instructions to create a new version or application.

There is also a **Clone** option that acts similar to **Replace**, but doesn't remove the old version of the application. You can use this option to test changes to an application without stopping the running version, or to redeploy a version that you've already deleted.

Delete versions and applications

To clean up unused application versions, delete them from your appliances.

To delete an application

1. Open the AWS Panorama console Deployed applications page.
2. Choose an application.
3. Choose **Delete from device**.

Package configuration

When you use the AWS Panorama Application CLI command `panorama-cli package-application`, the CLI uploads your application's assets to Amazon S3 and registers them with AWS Panorama. Assets include binary files (container images and models) and descriptor files, which the AWS Panorama Appliance downloads during deployment. To register a package's assets, you provide a separate package configuration file that defines the package, its assets, and its interface.

The following example shows a package configuration for a code node with one input and one output. The video input provides access to image data from a camera stream. The output node sends processed images out to a display.

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"
                    }
                ]
            }
        ]
    }
}
```

The `assets` section specifies the names of artifacts that the AWS Panorama Application CLI uploaded to Amazon S3. If you import a sample application or an application from another user, this section can be empty or refer to assets that aren't in your account. When you run `panorama-cli package-application`, the AWS Panorama Application CLI populates this section with the correct values.

The AWS Panorama application manifest

When you deploy an application, you provide a configuration file called an application manifest. This file defines the application as a graph with nodes and edges. The application manifest is part of the application's source code and is stored in the `graphs` directory.

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"
            }
        ]
    }
}
```

Nodes are connected by edges, which specify mappings between nodes' inputs and outputs. The output of one node connects to the input of another, forming a graph.

JSON schema

The format of application manifest and override documents is defined in a JSON schema. You can use the JSON schema to validate your configuration documents before deploying. The JSON schema is available in this guide's GitHub repository.

- **JSON schema** – aws-panorama-developer-guide/resources

Application nodes

Nodes are models, code, camera streams, output, and parameters. A node has an interface, which defines its inputs and outputs. The interface can be defined in a package in your account, a package provided by AWS Panorama, or a built-in type.

In the following example, `code_node` and `model_node` refer to the sample code and model packages included with the sample application. `camera_node` uses a package provided by AWS Panorama to create a placeholder for a camera stream that you specify during deployment.

Example graph.json – Nodes

```
    "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."
            }
        }
  ]
```

Edges

Edges map the output from one node to the input of another. In the following example, the first edge maps the output from a camera stream node to the input of an application code node. The names `video_in` and `video_out` are defined in the node packages' interfaces.

Example graph.json – edges

```
    "edges": [
```

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

In your application code, you use the `inputs` and `outputs` attributes to get images from the input stream, and send images to the output stream.

Example application.py – Video input and output

```
    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)
```

Abstract nodes

In an application manifest, an abstract node refers to a package defined by AWS Panorama, which you can use as a placeholder in your application manifest. AWS Panorama provides two types of abstract node.

- **Camera stream** – Choose the camera stream that the application uses during deployment.

  Package name – `panorama::abstract_rtsp_media_source`

  Interface name – `rtsp_v1_interface`
- **HDMI output** – Indicates that the application outputs video.

  Package name – `panorama::hdmi_data_sink`

Interface name – `hdmi0`

The following example shows a basic set of packages, nodes, and edges for an application that processes camera streams and outputs video to a display. The camera node, which uses the interface from the `abstract_rtsp_media_source` package in AWS Panorama, can accept multiple camera streams as input. The output node, which references `hdmi_data_sink`, gives application code access to a video buffer that is output from the appliance's HDMI port.

Example graph.json – Abstract nodes

```
{
    "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"
            }
        ]
    }
}
```

Application parameters

Parameters are nodes that have a basic type and can be overridden during deployment. A parameter can have a default value and a *decorator*, which instructs the application's user how to configure it.

Parameter types

- `string` – A string. For example, `DEBUG`.
- `int32` – An integer. For example, `20`
- `float32` – A floating point number. For example, `47.5`
- `boolean` – `true` or `false`.

The following example shows two parameters, a string and a number, which are sent to a code node as inputs.

Example graph.json – Parameters

```
       "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"
            }
            ...
        ]
    }
```

You can modify parameters directly in the application manifest, or provide new values at deploy-time with overrides. For more information, see Deploy-time configuration with overrides.

Deploy-time configuration with overrides

You configure parameters and abstract nodes during deployment. If you use the AWS Panorama console to deploy, you can specify a value for each parameter and choose a camera stream as input. If you use the AWS Panorama API to deploy applications, you specify these settings with an overrides document.

An overrides document is similar in structure to an application manifest. For parameters with basic types, you define a node. For camera streams, you define a node and a package that maps to a registered camera stream. Then you define an override for each node that specifies the node from the application manifest that it replaces.

Example overrides.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"
    }
}
```

In the preceding example, the document defines overrides for one string parameter and an abstract camera node. The `node0verrides` tells AWS Panorama which nodes in this document override which in the application manifest.

Building AWS Panorama applications

Applications run on the AWS Panorama Appliance to perform computer vision tasks on video streams. You can build computer vision applications by combining Python code and machine learning models, and deploy them to the AWS Panorama Appliance over the internet. Applications can send video to a display, or use the AWS SDK to send results to AWS services.

A model analyzes images to detect people, vehicles, and other objects. Based on images that it has seen during training, the model tells you what it thinks something is, and how confident it is in its guess. You can train models with your own image data or get started with a sample.

The application's code process still images from a camera stream, sends them to a model, and processes the result. A model might detect multiple objects and return their shapes and location. The code can use this information to add text or graphics to the video, or to send results to an AWS service for storage or further processing.

To get images from a stream, interact with a model, and output video, application code uses the AWS Panorama Application SDK. The application SDK is a Python library that supports models generated with PyTorch, Apache MXNet, and TensorFlow.

Topics

Computer vision models

A *computer vision model* is a software program that is trained to detect objects in images. A model learns to recognize a set of objects by first analyzing images of those objects through training. A computer vision model takes an image as input and outputs information about the objects that it detects, such as the type of object and its location. AWS Panorama supports computer vision models built with PyTorch, Apache MXNet, and TensorFlow.

> **Note**
>
> For a list of pre-built models that have been tested with AWS Panorama, see Model compatibility.

Sections

Using models in code

A model returns one or more results, which can include probabilities for detected classes, location information, and other data.The following example shows how to run inference on an image from a video stream and send the model's output to a processing function.

Example application.py – Inference

```
    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)
```

The following example shows a function that processes results from basic classification model. The sample model returns an array of probabilities, which is the first and only value in the results array.

Example application.py – Processing results

```
   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)
```

The application code finds the values with the highest probabilities and maps them to labels in a resource file that's loaded during initialization.

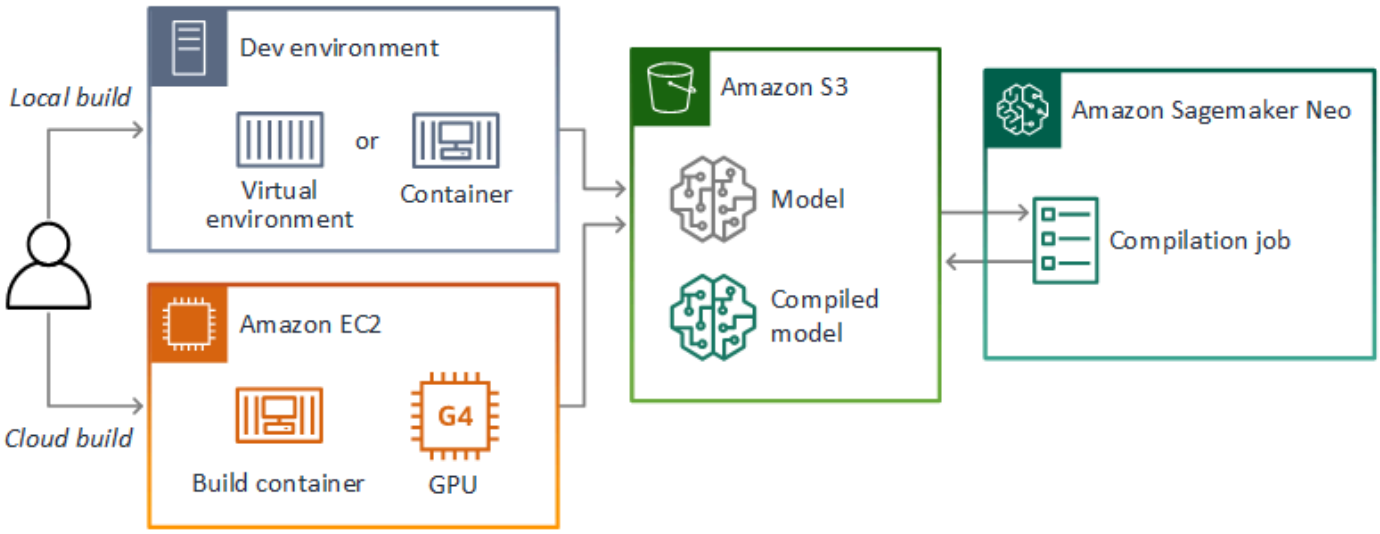
Building a custom model

You can use models that you build in PyTorch, Apache MXNet, and TensorFlow in AWS Panorama applications. As an alternative to building and training models in SageMaker, you can use a trained model or build and train your own model with a supported framework and export it in a local environment or in Amazon EC2.

> **Note**
>
> For details about the framework versions and file formats supported by SageMaker Neo, see Supported Frameworks in the Amazon SageMaker Developer Guide.

The repository for this guide provides a sample application that demonstrates this workflow for a Keras model in TensorFlow `SavedModel` format. It uses TensorFlow 2 and can run locally in a virtual environment or in a Docker container. The sample app also includes templates and scripts for building the model on an Amazon EC2 instance.

- Custom model sample application



AWS Panorama uses SageMaker Neo to compile models for use on the AWS Panorama Appliance. For each framework, use the format that's supported by SageMaker Neo, and package the model in a `.tar.gz` archive.

For more information, see Compile and deploy models with Neo in the Amazon SageMaker Developer Guide.

Packaging a model

A model package comprises a descriptor, package configuration, and model archive. Like in an application image package, the package configuration tells the AWS Panorama service where the model and descriptor are stored in Amazon S3.

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**
>
> Specify the framework version's major and minor version only. For a list of supported PyTorch, Apache MXNet, and TensorFlow versions versions, see Supported frameworks.

To import a model, use the AWS Panorama Application CLI `import-raw-model` command. If you make any changes to the model or its descriptor, you must rerun this command to update the application's assets. For more information, see Changing the computer vision model.

For the descriptor file's JSON schema, see assetDescriptor.schema.json.

Training models

When you train a model, use images from the target environment, or from a test environment that closely resembles the target environment. Consider the following factors that can affect model performance:

- **Lighting** – The amount of light that is reflected by a subject determines how much detail the model has to analyze. A model trained with images of well-lit subjects might not work well in a low-light or backlit environment.
- **Resolution** – The input size of a model is typically fixed at a resolution between 224 and 512 pixels wide in a square aspect ratio. Before you pass a frame of video to the model, you can downscale or crop it to fit the required size.
- **Image distortion** – A camera's focal length and lens shape can cause images to exhibit distortion away from the center of the frame. The position of a camera also determines which features of a subject are visible. For example, an overhead camera with a wide angle lens will show the top of a subject when it's in the center of the frame, and a skewed view of the subject's side as it moves farther away from center.

To address these issues, you can preprocess images before sending them to the model, and train the model on a wider variety of images that reflect variances in real-world environments. If a model needs to operate in a lighting situations and with a variety of cameras, you need more data for training. In addition to gathering more images, you can get more training data by creating variations of your existing images that are skewed or have different lighting.

Building an application image

The AWS Panorama Appliance runs applications as container filesystems exported from an image that you build. You specify your application's dependencies and resources in a Dockerfile that uses the AWS Panorama application base image as a starting point.

To build an application image, you use Docker and the AWS Panorama Application CLI. The following example from this guide's sample application demonstrates these use cases.

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
```

The following Dockerfile instructions are used.

- FROM – Loads the application base image (public.ecr.aws/panorama/panorama-application).
- WORKDIR – Set the working directory on the image. /panorama is used for application code and related files. This setting only persists during the build and does not affect the working directory for your application at runtime (/).
- COPY – Copies files from a local path to a path on the image. COPY . . copies the files in the current directory (the package directory) to the working directory on the image. For example, the application code is copied from packages/123456789012-SAMPLE_CODE-1.0/application.py to /panorama/application.py.
- RUN – Runs shell commands on the image during the build. A single RUN operation can run multiple commands in sequence by using && between commands. This example updates the pip package manager and then installs the libraries listed in requirements.txt.

You can use other instructions, such as ADD and ARG, that are useful at build time. Instructions that add runtime information to the container, such as ENV, do not work with AWS Panorama. AWS Panorama does not run a container from the image. It only uses the image to export a filesystem, which is transferred to the appliance.

Specifying dependencies

`requirements.txt` is a Python requirements file that specifies libraries used by the application. The sample application uses Open CV and the AWS SDK for Python (Boto3).

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

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

The `pip install` command in the Dockerfile installs these libraries to the Python `dist-packages` directory under `/usr/local/lib`, so that they can be imported by your application code.

Local storage

AWS Panorama reserves the `/opt/aws/panorama/storage` directory for application storage. Your application can create and modify files at this path. Files created in the storage directory persist across reboots. Other temporary file locations are cleared on boot.

Building image assets

When you build an image for your application package with the AWS Panorama Application CLI, the CLI runs `docker build` in the package directory. This builds an application image that contains your application code. The CLI then creates a container, exports its filesystem, compresses it, and stores it in the `assets` folder.

```
$ 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
```

The JSON block in the output is an asset definition that the CLI adds to the package configuration (`package.json`) and registers with the AWS Panorama service. The CLI also copies the descriptor file, which specifies the path to the application script (the application's entry point).

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

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

In the assets folder, the descriptor and application image are named for their SHA-256 checksum. This name is used as a unique identifier for the asset when it is stored is Amazon S3.

Calling AWS services from your application code

You can use the AWS SDK for Python (Boto) to call AWS services from your application code. For example, if your model detects something out of the ordinary, you could post metrics to Amazon CloudWatch, send an notification with Amazon SNS, save an image to Amazon S3, or invoke a Lambda function for further processing. Most AWS services have a public API that you can use with the AWS SDK.

The appliance does not have permission to access any AWS services by default. To grant it permission, create a role for the application, and assign it to the application instance during deployment.

Sections

Using Amazon S3

You can use Amazon S3 to store processing results and other application data.

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

Using the AWS IoT MQTT topic

You can use the SDK for Python (Boto3) to send messages to an MQTT topic in AWS IoT. In the following example, the application posts to a topic named after the appliance's *thing name*, which you can find in AWS IoT console.

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

Choose a name that indicates the device ID or other identifier of your choice. To publish messages, the application needs permission to call `iot:Publish`.

To monitor an MQTT queue

1. Open the AWS IoT console Test page.
2. For **Subscription topic**, enter the name of the topic. For example, `panorama/panorama_my-appliance_Thing_a01e373b`.
3. Choose **Subscribe to topic**.

The AWS Panorama Application SDK

The AWS Panorama Application SDK is a Python library for developing AWS Panorama applications. In your application code, you use the AWS Panorama Application SDK to load a computer vision model, run inference, and output video to a monitor.

> **Note**
>
> To ensure that you have access to the latest functionality of the AWS Panorama Application SDK, upgrade the appliance software.

For details about the classes that the application SDK defines and their methods, see Application SDK reference.

Sections

Adding text and boxes to output video

With the AWS Panorama SDK, you can output a video stream to a display. The video can include text and boxes that show output from the model, the current state of the application, or other data.

Each object in the `video_in` array is an image from a camera stream that is connected to the appliance. The type of this object is `panoramasdk.media`. It has methods to add text and rectangular boxes to the image, which you can then assign to the `video_out` array.

In the following example, the sample application adds a label for each of the results. Each result is positioned at the same left position, but at different heights.

```
        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)
```

To add a box to the output image, use `add_rect`. This method takes 4 values between 0 and 1, indicating the position of the top left and bottom right corners of the box.

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

Running multiple threads

You can run your application logic on a processing thread and use other threads for other background processes. For example, you can create a thread that serves HTTP traffic for debugging, or a thread that monitors inference results and sends data to AWS.

To run multiple threads, you use the threading module from the Python standard library to create a thread for each process. The following example shows the main loop of the debug server sample application, which creates an application object and uses it to run three threads.

Example packages/123456789012-DEBUG_SERVER-1.0/application.py – Main loop

```
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.')
```

When all of the threads exit, the application restarts itself. The `run_cv` loop processes images from camera streams. If it receives a signal to stop, it shuts down the debugger process, which runs an HTTP server and can't shut itself down. Each thread must handle its own errors. If an error is not caught and logged, the thread exits silently.

Example packages/123456789012-DEBUG_SERVER-1.0/application.py – Processing loop

```
    # 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")
```

Threads communicate via the application's `self` object. To restart the application processing loop, the debugger thread calls the `stop` method. This method sets a `terminate` attribute, which signals the other threads to shut down.

Example packages/123456789012-DEBUG_SERVER-1.0/application.py – Stop method

```
    # 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)
```

Serving inbound traffic

You can monitor or debug applications locally by running an HTTP server alongside your application code. To serve external traffic, you map ports on the AWS Panorama Appliance to ports on your application container.

> **Important**
>
> By default, the AWS Panorama Appliance does not accept incoming traffic on any ports. Opening ports on the appliance has implicit security risk. When you use this feature, you must take additional steps to secure your appliance from external traffic and secure communications between authorized clients and the appliance.
> The sample code included with this guide is for demonstration purposes and does not implement authentication, authorization, or encryption.

You can open up ports in the range 8000–9000 on the appliance. These ports, when opened, can receive traffic from any routable client. When you deploy your application, you specify which ports to open, and map ports on the appliance to ports on your application container. The appliance software forwards traffic to the container, and sends responses back to the requestor. Requests are received on the appliance port that you specify and responses go out on a random ephemeral port.

Configuring inbound ports

You specify port mappings in three places in your application configuration. The code package's `package.json`, you specify the port that the code node listens on in a `network` block. The following example declares that the node listens on port 80.

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"
                }
            ]
        }
```

In the application manifest, you declare a routing rule that maps a port on the appliance to a port on the application's code container. The following example adds a rule that maps port 8080 on the device to port 80 on the code_node container.

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."
            }
        }
    ]
```

When you deploy the application, you specify the same rules in the AWS Panorama console, or with an override document passed to the CreateApplicationInstance API. You must provide this configuration at deploy time to confirm that you want to open ports on the appliance.

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"
    }
}
```

If the device port specified in the application manifest is in use by another application, you can use the override document to choose a different port.

Serving traffic

With ports open on the container, you can open a socket or run a server to handle incoming requests. The `debug-server` sample shows a basic implementation of an HTTP server running alongside computer vision application code.

> **Important**
>
> The sample implementation is not secure for production use. To avoid making your appliance vulnerable to attacks, you must implement appropriate security controls in your code and network configuration.

Example packages/123456789012-DEBUG_SERVER-1.0/application.py – HTTP server

```
    # 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.')
```

The server accepts GET requests at the `/status` path to retrieve some information about the application. It also accepts a POST request to `/restart` to restart the application.

To demonstrate this functionality, the sample application runs an HTTP client on a separate thread. The client calls the `/status` path over the local network shortly after startup, and restarts the application a few minutes later.

Example packages/123456789012-DEBUG_SERVER-1.0/application.py – HTTP client

```
    # 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")
```

The main loop manages the threads and restarts the application when they exit.

Example packages/123456789012-DEBUG_SERVER-1.0/application.py – Main loop

```
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.')
```

To deploy the sample application, see the instructions in this guide's GitHub repository.

Using the GPU

You can access the graphics processor (GPU) on the AWS Panorama Appliance to use GPU-accelerated libraries, or run machine learning models in your application code. To turn on GPU access, you add GPU access as a requirement to the package configuration after building your application code container.

> **Important**
>
> If you enable GPU access, you can't run model nodes in any application on the appliance. For security purposes, GPU access is restricted when the appliance runs a model compiled with SageMaker Neo. With GPU access, you must run your models in application code nodes, and all applications on the device share access to the GPU.

To turn on GPU access for your application, update the package configuration after you build the package with the AWS Panorama Application CLI. The following example shows the `requirements` block that adds GPU access to the application code node.

Example package.json with requirements block

```
{
    "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": [
    ...
```

Update the package configuration between the build and packaging steps in your development workflow.

To deploy an application with GPU access

1. To build the application container, use the `build-container` command.

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

2. Add the `requirements` block to the package configuration.
3. To upload the container asset and package configuration, use the `package-application` command.

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

4. Deploy the application.

For sample applications that use GPU access, visit the aws-panorama-samples GitHub repository.

Setting up a development environment in Windows

To build a AWS Panorama application, you use Docker, command-line tools, and Python. In Windows, you can set up a development environment by using Docker Desktop with Windows Subsystem for Linux and Ubuntu. This tutorial walks you through the setup process for a development environment that has been tested with AWS Panorama tools and sample applications.

Sections

Prerequisites

To follow this tutorial, you need a version of Windows that supports Windows Subsystem for Linux 2 (WSL 2).

- Windows 10 version 1903 and higher (Build 18362 and higher) or Windows 11
- Windows features
  - Windows Subsystem for Linux
  - Hyper-V
  - Virtual machine platform

This tutorial was developed with the following software versions.

- Ubuntu 20.04
- Python 3.8.5
- Docker 20.10.8

Install WSL 2 and Ubuntu

If you have Windows 10 version 2004 and higher (Build 19041 and higher), you can install WSL 2 and Ubuntu 20.04 with the following PowerShell command.

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

For older Windows version, follow the instructions in the WSL 2 documentation: Manual installation steps for older versions

Install Docker

To install Docker Desktop, download and run the installer package from hub.docker.com. If you encounter issues, follow the instructions on the Docker website: Docker Desktop WSL 2 backend.

Run Docker Desktop and follow the first-run tutorial to build an example container.

> **Note**
>
> Docker Desktop only enables Docker in the default distribution. If you have other Linux distributions installed prior to running this tutorial, enable Docker in the newly installed Ubuntu distribution in the Docker Desktop settings menu under **Resources**, **WSL integration**.

Configure Ubuntu

You can now run Docker commands in your Ubuntu virtual machine. To open a command-line terminal, run the distribution from the start menu. The first time you run it, you configure a username and password that you can use to run administrator commands.

To complete configuration of your development environment, update the virtual machine's software and install tools.

To configure the virtual machine

1. Update the software that comes with Ubuntu.

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

2. Install development tools with apt.

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

3. Install Python libraries with pip.

```
$ pip3 install awscli panoramacli
```

4. Open a new terminal, and then run `aws configure` to configure the AWS CLI.

```
$ aws configure
```

   If you don't have access keys, you can generate them in the IAM console.

Finally, download and import the sample application.

To get the sample application

1. Download and extract the sample application.

```
$ 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. Run the included scripts to test compilation, build the application container, and upload packages to AWS Panorama.

```
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
```

The AWS Panorama Application CLI uploads packages and registers them with the AWS Panorama service. You can now deploy the sample app with the AWS Panorama console.

Next steps

To explore and edit the project files, you can use File Explorer or an integrated development environment (IDE) that supports WSL.

To access the virtual machine's file system, open File explorer and enter `\\wsl$` in the navigation bar. This directory contains a link to the virtual machine's file system (`Ubuntu-20.04`) and file systems for Docker's data. Under `Ubuntu-20.04`, your user directory is at home*`username`*.

> **Note**
>
> To access files in your Windows installation from within Ubuntu, navigate to the `/mnt/c` directory. For example, you can list files in your downloads directory by running `ls /mnt/c/Users/`*`windows-username`*`/Downloads`.

With Visual Studio Code, you can edit application code in your development environment and run commands with an integrated terminal. To install Visual Studio Code, visit code.visualstudio.com. After installation, add the Remote WSL extension.

Windows terminal is an alternative to the standard Ubuntu terminal that you've been running commands in. It supports multiple tabs and can run PowerShell, Command Prompt, and terminals for any other variety of Linux that you install. It supports copy and paste with **Ctrl+C** and **Ctrl+V**, clickable URLs, and other useful improvements. To install Windows Terminal, visit microsoft.com.

The AWS Panorama API

You can use the AWS Panorama service's public API to automate device and application management workflows. With the AWS Command Line Interface or the AWS SDK, you can develop scripts or applications that manage resources and deployments. This guide's GitHub repository includes scripts that you can use as a starting point for your own code.

- aws-panorama-developer-guide/util-scripts

Sections

Automate device registration

To provision an appliance, use the ProvisionDevice API. The response includes a ZIP file with the device's configuration and temporary credentials. Decode the file and save it in an archive with the prefix `certificates-omni_`.

Example provision-device.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}"
```

The credentials in the configuration archive expire after 5 minutes. Transfer the archive to your appliance with the included USB drive.

To register a camera, use the CreateNodeFromTemplateJob API. This API takes a map of template parameters for the camera's username, password, and URL. You can format this map as a JSON document by using Bash string manipulation.

Example register-camera.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)
```

Alternatively, you can load the JSON configuration from a file.

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

Manage appliances with the AWS Panorama API

You can automate appliance management tasks with the AWS Panorama API.

View devices

To get a list of appliances with device IDs, use the ListDevices API.

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

To get more details about an appliance, use the DescribeDevice API.

```
$ 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
}
```

Upgrade appliance software

If the `LatestSoftware` version is newer than the `CurrentSoftware`, you can upgrade the device. Use the CreateJobForDevices API to create an over-the-air (OTA) update job.

```
$ 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"
        }
    ]
}
```

In a script, you can populate the image version field in the job configuration file with Bash string manipulation.

Example check-updates.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
}
```

The appliance downloads the specified software version and updates itself. Watch the update's progress with the DescribeDeviceJob API.

```
$ 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
}
```

To get a list of all running jobs, use the ListDevicesJobs.

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

For a sample script that checks for and applies updates, see check-updates.sh in this guide's GitHub repository.

Reboot appliances

To reboot an appliance, use the CreateJobForDevices API.

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

```
    ]
}
```

In a script, you can get a list of devices and choose one to reboot interactively.

Example [reboot-device.sh] – usage

```
$ ./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"
        }
    ]
}
```

Automate application deployment

To deploy an application, you use both the AWS Panorama Application CLI and AWS Command Line Interface. After building the application container, you upload it and other assets to an Amazon S3 access point. You then deploy the application with the CreateApplicationInstance API.

For more context and instructions for using the scripts shown, follow the instructions in the sample application README.

Sections

Build the container

To build the application container, use the `build-container` command. This command builds a Docker container and saves it as a compressed file system in the `assets` folder.

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
```

You can also use command-line completion to fill in the path argument by typing part of the path, and then pressing **TAB**.

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

Upload the container and register nodes

To upload the application, use the `package-application` command. This command uploads assets from the `assets` folder to an Amazon S3 access point that AWS Panorama manages.

Example 4-package-app.sh

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

The AWS Panorama Application CLI uploads container and descriptor assets referenced by the package configuration (`package.json`) in each package, and registers the packages as nodes in AWS Panorama. You then refer to these nodes in your application manifest (`graph.json`) to deploy the application.

Deploy the application

To deploy the application, you use the CreateApplicationInstance API. This action takes the following parameters, among others.

- `ManifestPayload` – The application manifest (`graph.json`) that defines the application's nodes, packages, edges, and parameters.
- `ManifestOverridesPayload` – A second manifest that overrides parameters in the first. The application manifest can be considered as a static resource in the application source, where the override manifest provides deploy-time settings that customize the deployment.
- `DefaultRuntimeContextDevice` – The target device.
- `RuntimeRoleArn` – The ARN of an IAM role that the application uses to access AWS services and resources.
- `ApplicationInstanceIdToReplace` – The ID of an existing application instance to remove from the device.

The manifest and override payloads are JSON documents that must be provided as a string value nested inside of another document. To do this, the script loads the manifests from a file as a string and uses the jq tool to construct the nested document.

Example 5-deploy.sh – compose manifests

```
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)"' <<< {})"
```

The deploy script uses the ListDevices API to get a list of registered devices in the current Region, and saves the users choice to a local file for subsequent deployments.

Example 5-deploy.sh – find a device

```
    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)
```

The application role is created by another script (1-create-role.sh). The deploy script gets the ARN of this role from AWS CloudFormation. If the application is already deployed to the device, the script gets the ID of that application instance from a local file.

Example 5-deploy.sh – role ARN and replacement arguments

```
# 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
```

Finally, the script puts all of the pieces together to create an application instance and deploy the application to the device. The service responds with an instance ID which the script stores for later use.

Example 5-deploy.sh – deploy application

```
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
```

Monitor the deployment

To monitor a deployment, use the ListApplicationInstances API. The monitor script gets the device ID and application instance ID from files in the application directory and uses them to construct a CLI command. It then calls in a loop.

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
```

When the deployment completes, you can view logs by calling the Amazon CloudWatch Logs API. The view logs script uses the CloudWatch Logs `GetLogEvents` API.

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
```

Manage applications with the AWS Panorama API

You can monitor and manage applications with the AWS Panorama API.

View applications

To get a list of applications running on an appliance, use the ListApplicationInstances API.

```
$ 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"
            }
        },
    ]
}
```

To get more details about an application instance's nodes, use the ListApplicationInstanceNodeInstances API.

```
$ 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"
        }
    ]
}
```

Manage camera streams

You can pause and resume camera stream nodes with the SignalApplicationInstanceNodeInstances API.

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

In a script, you can get a list of nodes and choose one to pause or resume interactively.

Example pause-camera.sh – usage

```
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"
}
```

By pausing and resuming camera nodes, you can cycle through a larger number of camera streams than can be processed simultaneously. To do this, map multiple camera streams to the same input node in your override manifest.

In the following example, the override manifest maps two camera streams, `warehouse-floor` and `entrance-north` to the same input node (`camera_node`). The `warehouse-floor` stream is active when the application starts and the `entrance-north` node waits for a signal to turn on.

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"
                    }
                ]
            }
```

For details on deploying with the API, see Automate application deployment.

Using VPC endpoints

If you work in a VPC without internet access, you can create a VPC endpoint for use with AWS Panorama. A VPC endpoint lets clients running in a private subnet connect to an AWS service without an internet connection.

For details on ports and endpoints used by the AWS Panorama Appliance, see ???.

Sections

Creating a VPC endpoint

To establish a private connection between your VPC and AWS Panorama, create a *VPC endpoint*. A VPC endpoint is not required to use AWS Panorama. You only need to create a VPC endpoint if you work in a VPC without internet access. When the AWS CLI or SDK attempts to connect to AWS Panorama, the traffic is routed through the VPC endpoint.

Create a VPC endpoint for AWS Panorama using the following settings:

- **Service name** – **`com.amazonaws.us-west-2.panorama`**
- **Type** – **Interface**

A VPC endpoint uses the service's DNS name to get traffic from AWS SDK clients without any additional configuration. For more information about using VPC endpoints, see Interface VPC endpoints in the *Amazon VPC User Guide*.

Connecting an appliance to a private subnet

The AWS Panorama Appliance can connect to AWS over a private VPN connection with AWS Site-to-Site VPN or AWS Direct Connect. With these services, you can create a private subnet that extends to your data center. The appliance connects to the private subnet and accesses AWS services through VPC endpoints.

Site-to-Site VPN and AWS Direct Connect are services for connecting your data center to Amazon VPC securely. With Site-to-Site VPN, you can use commercially available network devices to connect. AWS Direct Connect uses an AWS device to connect.

- **Site-to-Site VPN** – What is AWS Site-to-Site VPN?
- **AWS Direct Connect** – What is AWS Direct Connect?

After you've connected your local network to a private subnet in a VPC, create VPC endpoints for the following services.

- **Amazon Simple Storage Service** – AWS PrivateLink for Amazon S3
- **AWS IoT Core** – Using AWS IoT Core with interface VPC endpoints (data plane and credential provider)
- **Amazon Elastic Container Registry** – Amazon Elastic Container Registry interface VPC endpoints
- **Amazon CloudWatch** – Using CloudWatch with interface VPC endpoints
- **Amazon CloudWatch Logs** – Using CloudWatch Logs with interface VPC endpoints

The appliance does not need connectivity to the AWS Panorama service. It communicates with AWS Panorama through a messaging channel in AWS IoT.

In addition to VPC endpoints, Amazon S3 and AWS IoT require the use of Amazon Route 53 private hosted zones. The private hosted zone routes traffic from subdomains, including subdomains for Amazon S3 access points and MQTT topics, to the correct VPC endpoint. For information on private hosted zones, see Working with private hosted zones in the Amazon Route 53 Developer Guide.

For a sample VPC configuration with VPC endpoints and private hosted zones, see Sample AWS CloudFormation templates.

Sample AWS CloudFormation templates

The GitHub repository for this guide provides AWS CloudFormation templates that you can use to create resources for use with AWS Panorama. The templates create a VPC with two private subnets, a public subnet, and a VPC endpoint. You can use the private subnets in the VPC to host resources that are isolated from the internet. Resources in the public subnet can communicate with the private resources, but the private resources can't be accessed from the internet.

Example vpc-endpoint.yml – Private subnets

```
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
  ...
```

The `vpc-endpoint.yml` template shows how to create a VPC endpoint for AWS Panorama. You can use this endpoint to manage AWS Panorama resources with the AWS SDK or AWS CLI.

Example vpc-endpoint.yml – VPC endpoint

```
  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:
            - "*"
```

The `PolicyDocument` is a resource-based permissions policy that defines the API calls that can be made with the endpoint. You can modify the policy to restrict the actions and resources that can be accessed through the endpoint. For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

The `vpc-appliance.yml` template shows how to create VPC endpoints and private hosted zones for services used by the AWS Panorama Appliance.

Example vpc-appliance.yml – Amazon S3 access point endpoint with private hosted zone

```
  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 ] ] ]
```

The sample templates demonstrate the creation of Amazon VPC and Route 53 resources with a sample VPC. You can adapt these for your use case by removing the VPC resources and replacing the references to subnet, security group, and VPC IDs with the IDs of your resources.

Sample applications, scripts, and templates

The GitHub repository for this guide provides sample applications, scripts, and templates for AWS Panorama devices. Use these samples to learn best practices and automate development workflows.

Sections

Sample applications

Sample applications demonstrate use of AWS Panorama features and common computer vision tasks. These sample applications include scripts and templates that automate setup and deployment. With minimal configuration, you can deploy and update applications from the command line.

- aws-panorama-sample – Basic computer vision with a classification model. Use the AWS SDK for Python (Boto) to upload metrics to CloudWatch, instrument preprocessing and inference methods, and configure logging.
- debug-server – Open inbound ports on the device and forward traffic to an application code container. Use multithreading to run application code, an HTTP server, and an HTTP client simultaneously.
- custom-model – Export models from code and compile with SageMaker Neo to test compatibility with the AWS Panorama Appliance. Build locally in a Python development, in a Docker container, or on an Amazon EC2 instance. Export and compile all built-in application models in Keras for a specific TensorFlow or Python version.

For more sample applications, also visit the aws-panorama-samples repository.

Utility scripts

The scripts in the `util-scripts` directory manage AWS Panorama resources or automate development workflows.

- provision-device.sh – Provision a device.
- check-updates.sh – Check for and apply appliance software updates.
- reboot-device.sh – Reboot a device.
- register-camera.sh – Register a camera.
- deregister-camera.sh – Delete a camera node.
- view-logs.sh – View logs for an application instance.
- pause-camera.sh – Pause or resume a camera stream.
- push.sh – Build, upload, and deploy an application.
- rename-package.sh – Rename a node package. Updates directory names, configuration files, and the application manifest.
- samplify.sh – Replace your account ID with an example account ID, and restore backup configurations to remove local configuration.
- update-model-config.sh – Re-add the model to the application after updating the descriptor file.
- cleanup-patches.sh – Deregister old patch versions and delete their manifests from Amazon S3.

For usage details, see the README.

AWS CloudFormation templates

Use the AWS CloudFormation templates in the `cloudformation-templates` directory to create resources for AWS Panorama applications.

- alarm-application.yml – Create an alarm that monitors an application for errors. If the application instance raises errors or stops running for 5 minutes, the alarm sends a notification email.
- alarm-device.yml – Create an alarm that monitors a device's connectivity. If the device stops sending metrics for 5 minutes, the alarm sends a notification email.

- application-role.yml – Create an application role. The role includes permission to send metrics to CloudWatch. Add permissions to the policy statement for other API operations that your application uses.
- vpc-appliance.yml – Create a VPC with private subnet service access for the AWS Panorama Appliance. To connect the appliance to a VPC, use AWS Direct Connect or AWS Site-to-Site VPN.
- vpc-endpoint.yml – Create a VPC with private subnet service access to the AWS Panorama service. Resources inside of the VPC can connect to AWS Panorama to monitor and manage AWS Panorama resources without connecting to the internet.

The `create-stack.sh` script in this directory creates AWS CloudFormation stacks. It takes a variable number of arguments. The first argument is the name of the template, and the remaining arguments are overrides for parameters in the template.

For example, the following command creates an application role.

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

More samples and tools

The aws-panorama-samples repository has more sample applications and useful tools.

- Applications – Sample applications for various model architectures and use cases.
- Camera stream validation – Validate camera streams.
- PanoJupyter – Run JupyterLab on an AWS Panorama Appliance.
- Sideloading – Update application code without building or deploying an application container.

The AWS community has also developed tools and guidance for AWS Panorama. Check out the following open source projects on GitHub.

- cookiecutter-panorama – A Cookiecutter template for AWS Panorama applications.
- backpack – Python modules for accessing runtime environment details, profiling, and additional video output options.

Monitoring AWS Panorama resources and applications

You can monitor AWS Panorama resources in the AWS Panorama console and with Amazon CloudWatch. The AWS Panorama Appliance connects to the AWS Cloud over the internet to report its status and the status of connected cameras. While it is on, the appliance also sends logs to CloudWatch Logs in real time.

The appliance gets permission to use AWS IoT, CloudWatch Logs, and other AWS services from a service role that you create the first time that you use the AWS Panorama console. For more information, see AWS Panorama service roles and cross-service resources.

For help troubleshooting specific errors, see Troubleshooting.

Topics

Monitoring in the AWS Panorama console

You can use the AWS Panorama console to monitor your AWS Panorama Appliance and cameras. The console uses AWS IoT to monitor the state of the appliance.

To monitor your appliance in the AWS Panorama console

1. Open the AWS Panorama console.
2. Open the AWS Panorama console Devices page.
3. Choose an appliance.
4. To see the status of an application instance, choose it from the list.
5. To see the status of the appliance's network interfaces, choose **Settings**.

The overall status of the appliance appears at the top of the page. If the status is **Online**, then the appliance is connected to AWS and sending regular status updates.

Viewing AWS Panorama logs

AWS Panorama reports application and system events to Amazon CloudWatch Logs. When you encounter issues, you can use the event logs to help debug your AWS Panorama application or troubleshoot the application's configuration.

To view logs in CloudWatch Logs

1. Open the Log groups page of the CloudWatch Logs console.
2. Find AWS Panorama application and appliance logs in the following groups:

   - **Device logs** – `/aws/panorama/devices/`*`device-id`*
   - **Application logs** – `/aws/panorama/devices/`*`device-id`*`/applications/`*`instance-id`*

When you reprovision an appliance after updating the system software, you can also view logs on the provisioning USB drive.

Sections

Viewing device logs

The AWS Panorama Appliance creates a log group for the device, and a group for each application instance that you deploy. The device logs contain information about application status, software upgrades, and system configuration.

Device logs – `/aws/panorama/devices/`*`device-id`*

- `occ_log` – Output from the controller process. This process coordinates application deployments and reports on the status of each application instance's nodes.
- `ota_log` – Output from the process that coordinates over-the-air (OTA) software upgrades.

- `syslog` – Output from the device's syslog process, which captures messages sent between processes.
- `kern_log` – Events from the device's Linux kernel.
- `logging_setup_logs` – Output from the process that configures the CloudWatch Logs agent.
- `cloudwatch_agent_logs` – Output from the CloudWatch Logs agent.
- `shadow_log` – Output from the AWS IoT device shadow.

Viewing application logs

An application instance's log group contains a log stream for each node, named after the node.

Application logs – `/aws/panorama/devices/`*`device-id`*`/applications/`*`instance-id`*

- **Code** – Output from your application code and the AWS Panorama Application SDK. Aggregates application logs from `/opt/aws/panorama/logs`.
- **Model** – Output from the process that coordinates inference requests with a model.
- **Stream** – Output from the process that decodes video from a camera stream.
- **Display** – Output from the process that renders video output for the HDMI port.
- `mds` – Logs from the appliance metadata server.
- `console_output` – Captures standard output and error streams from code containers.

If you don't see logs in CloudWatch Logs, confirm that you are in the correct AWS Region. If you are, there might be an issue with the appliance's connection to AWS or with permissions on the appliance's AWS Identity and Access Management (IAM) role.

Configuring application logs

Configure a Python logger to write log files to `/opt/aws/panorama/logs`. The appliance streams logs from this location to CloudWatch Logs. To avoid using too much disk space, use a maximum file size of 10 MiB and a backup count of 1. The following example shows a method that creates a logger.

Example application.py – Logger configuration

```
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
```

Initialize the logger at the global scope and use it throughout your application code.

Example application.py – Initialize logger

```
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()
```

Viewing provisioning logs

During provisioning, the AWS Panorama Appliance copies logs to the USB drive that you use to transfer the configuration archive to the appliance. Use these logs to troubleshoot provisioning issues on appliances with the latest software version.

> **Important**
>
> Provisioning logs are available for appliances updated to software version 4.3.23 or newer.

Application logs

- `/panorama/occ.log` – AWS Panorama controller software logs.
- `/panorama/ota_agent.log` – AWS Panorama over-the-air update agent logs.
- `/panorama/syslog.log` – Linux system logs.
- `/panorama/kern.log` – Linux kernel logs.

Egressing logs from a device

If your device and application logs don't appear in CloudWatch Logs, you can use a USB drive to get an encrypted log image off of the device. The AWS Panorama service team can decrypt the logs on your behalf and assist in debugging.

Prerequisites

To follow the procedure you will need the following hardware:

- **USB drive** – A FAT32-formatted USB flash memory drive with at least 1 GB of storage, for transferring the log files off the AWS Panorama Appliance.

To egress logs from the device

1. Prepare a USB drive with a `managed_logs` folder inside of a `panorama` folder.

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

2. Connect the USB drive to the device.
3. Power off the AWS Panorama Appliance.
4. Power on the AWS Panorama Appliance.
5. The device copies logs to the device. The status LED blinks blue while this is in progress.
6. Log files can then be found inside `managed_logs` directory with the format `panorama_device_log_`*`v1_dd_hh_mm`*`.img`

You can't decrypt the log image yourself. Work with customer support, a technical account manager for AWS Panorama, or a solutions architect to coordinate with the service team.

Monitoring appliances and applications with Amazon CloudWatch

When an appliance is online, AWS Panorama sends metrics to Amazon CloudWatch. You can build graphs and dashboards with these metrics in the CloudWatch console to monitor appliance activity, and set alarms that notify you when devices go offline or applications encounter errors.

To view metrics in the CloudWatch console

1. Open the AWS Panorama console Metrics page (`PanoramaDeviceMetrics` namespace).
2. Choose a dimension schema.
3. Choose metrics to add them to the graph.
4. To choose a different statistic and customize the graph, use the options on the **Graphed metrics** tab. By default, graphs use the `Average` statistic for all metrics.

> **Pricing**
>
> CloudWatch has an Always Free tier. Beyond the free tier threshold, CloudWatch charges for metrics, dashboards, alarms, logs, and insights. For details, see CloudWatch pricing.

For more information about CloudWatch, see the *Amazon CloudWatch User Guide*.

Sections

Using device metrics

When an appliance is online, it sends metrics to Amazon CloudWatch. You can use these metrics to monitor device activity and trigger an alarm if devices go offline.

- `DeviceActive` – Sent periodically when the device is active.

  Dimensions – `DeviceId` and `DeviceName`.

View the `DeviceActive` metric with the `Average` statistic.

Using application metrics

When an application encounters an error, it sends metrics to Amazon CloudWatch. You can use these metrics to trigger an alarm if an application stops running.

- `ApplicationErrors` – The number of application errors recorded.

  Dimensions – `ApplicationInstanceName` and `ApplicationInstanceId`.

View the application metrics with the `Sum` statistic.

Configuring alarms

To get notifications when a metric exceeds a threshold, create an alarm. For example, you can create an alarm that sends a notification when the sum of the `ApplicationErrors` metric stays at 1 for 20 minutes.

To create an alarm

1. Open the Amazon CloudWatch console Alarms page.
2. Choose **Create alarm**.
3. Choose **Select metric** and locate a metric for your device, such as `ApplicationErrors` for `applicationInstance-gk75xmplqbqtenlnmz4ehiu7xa,my-application`.
4. Follow the instructions to configure a condition, action, and name for the alarm.

For detailed instructions, see Create a CloudWatch alarm in the *Amazon CloudWatch User Guide*.

Troubleshooting

The following topics provide troubleshooting advice for errors and issues that you might encounter when using the AWS Panorama console, appliance, or SDK. If you find an issue that is not listed here, use the **Provide feedback** button on this page to report it.

You can find logs for your appliance in the Amazon CloudWatch Logs console. The appliance uploads logs from your application code, the appliance software, and AWS IoT processes as they are generated. For more information, see Viewing AWS Panorama logs.

Provisioning

Issue: (macOS) *My computer doesn't recognize the included USB drive with a USB-C adapter.*

This can occur if you plug the USB drive into a USB-C adapter that is already connected to your computer. Try disconnecting the adapter and reconnecting it with the USB drive already attached.

Issue: *Provisioning fails when I use my own USB drive.*

Issue: *Provisioning fails when I use the appliance's USB 2.0 port.*

The AWS Panorama Appliance is compatible with USB flash memory devices between 1 and 32 GB, but not all are compatible. Some issues have been observed when using the USB 2.0 port for provisioning. For consistent results, use the included USB drive with the USB 3.0 port (next to the HDMI port).

For the Lenovo ThinkEdge® SE70, a USB drive is not included with the appliance. Use a USB 3.0 drive with at least 1 GB of storage.

Appliance configuration

Issue: *The appliance shows a blank screen during boot up.*

After completing the initial boot sequence, which takes about one minute, the appliance shows a blank screen for a minute or more while it loads your model and starts your application. Also, the appliance does not output video if you connect a display after it turns on.

Issue: *The appliance doesn't respond when I hold the power button down to turn it off.*

The appliance takes up to 10 seconds to shut down safely. You need to hold the power button down for only 1 second to start the shutdown sequence. For a complete list of button operations, see AWS Panorama Appliance buttons and lights.

Issue: *I need to generate a new configuration archive to change settings or replace a lost certificate.*

AWS Panorama does not store the device certificate or network configuration after you download it, and you can't reuse configuration archives. Delete the appliance using the AWS Panorama console and create a new one with a new configuration archive.

Application configuration

Issue: *When I run multiple applications, I can't control which uses the HDMI output.*

When you deploy multiple applications that have output nodes, the application that started most recently uses the HDMI output. If this application stops running, another application can use the output. To give only one application access to the output, remove the output node and corresponding edge from the other application's application manifest and redeploy.

Issue: *Application output doesn't appear in logs*

Configure a Python logger to write log files to `/opt/aws/panorama/logs`. These are captured in a log stream for the code container node. Standard output and error streams are captured in a separate log stream called `console-output`. If you use `print`, use the `flush=True` option to keep messages from getting stuck in the output buffer.

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

Source: AWS Panorama service

Each time you deploy a change to an application, you register a *patch version* that represents the package configuration and asset files for each package that it uses. Use the cleanup patches script to deregister unused patch versions.

Camera streams

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

Error: *liveMedia0: Failed to get SDP description: 404 Not Found; with the result code: 404*

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

Source: Camera node log

The appliance can't connect to the application's camera stream. When this happens, the video output is blank or freezes on the last processed frame while the application waits for a frame of video from the AWS Panorama Application SDK. The appliance software attempts to connect to the camera stream and logs timeout errors in the camera node log. Verify that your camera stream URL is correct and that RTSP traffic is routable between the camera and appliance within your network. For more information, see Connecting the AWS Panorama Appliance to your network.

Error: *ERROR finalizeInterface(35) Camera credential fetching for port [username] failed*

Source: OCC log

The AWS Secrets Manager secret with the camera stream's credentials can't be found. Delete the camera stream and recreate it.

Error: *Camera did not provide an H264 encoded stream*

Source: Camera node log

The camera stream has an encoding other than H.264, such as H.265. Redeploy the application with an H.264 camera stream. For details on supported cameras, see Supported cameras.

Security in AWS Panorama

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to AWS Panorama, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using AWS Panorama. The following topics show you how to configure AWS Panorama to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS Panorama resources.

Topics

AWS Panorama Appliance security features

To protect your applications, models, and hardware against malicious code and other exploits, the AWS Panorama Appliance implements an extensive set of security features. These include but are not limited to the following.

- **Full-disk encryption** – The appliance implements Linux unified key setup (LUKS2) full-disk encryption. All system software and application data are encrypted with a key that is specific to your device. Even with physical access to the device, an attacker cannot inspect the contents of its storage.
- **Memory layout randomization** – To protect against attacks that target executable code loaded into memory, the AWS Panorama Appliance uses address space layout randomization (ASLR). ASLR randomizes the location of operating system code as it is loaded into memory. This prevents the use of exploits that attempt to overwrite or run specific sections of code by predicting where it is stored at runtime.
- **Trusted execution environment** – The appliance uses a trusted execution environment (TEE) based on ARM TrustZone, with isolated storage, memory, and processing resources. Keys and other sensitive data stored in the trust zone can only be accessed by a trusted application, which runs in a separate operating system within the TEE. The AWS Panorama Appliance software runs in the untrusted Linux environment alongside application code. It can only access cryptographic operations by making a request to the secure application.
- **Secure provisioning** – When you provision an appliance, the credentials (keys, certificates, and other cryptographic material) that you transfer to the device are only valid for a short time. The appliance uses the short-lived credentials to connect to AWS IoT and requests a certificate for itself that's valid for a longer time. The AWS Panorama service generates credentials and encrypts them with a key that is hardcoded on the device. Only the device that requested the certificate can decrypt it and communicate with AWS Panorama.
- **Secure boot** – When the device starts up, each software component is authenticated before it runs. The boot ROM, software hardcoded in the processor that can't be modified, uses a hardcoded encryption key to decrypt the bootloader, which validates the trusted execution environment kernel, and so forth.
- **Signed kernel** – Kernel modules are signed with an asymmetric encryption key. The operating system kernel decrypts the signature with the public key and verifies that it matches the module's signature before loading the module into memory.

- **dm-verity** – Similar to how kernel modules are validated, the appliance uses the Linux Device Mapper's `dm-verity` feature to verify the integrity of the appliance software image before mounting it. If the appliance software is modified, it won't run.
- **Rollback prevention** – When you update the appliance software, the appliance blows an electronic fuse on the SoC (system on a chip). Each software version expects an increasing number of fuses to be blown, and can't run if more are blown.

AWS Panorama Appliance security best practices

Keep in mind the following best practices when using the AWS Panorama appliance.

- **Physically secure the appliance** – Install the appliance in an enclosed server rack or secure room. Limit physical access to the device to authorized personnel.
- **Secure the appliance's network connection** – Connect the appliance to a router that limits access to internal and external resources. The appliance needs to connect to cameras, which can be on a secure internal network. It also needs to connect to AWS. Use the second Ethernet port only for physical redundancy, and configure the router to allow only required traffic.

  Use one of the recommended network configurations to plan your network layout. For more information, see Connecting the AWS Panorama Appliance to your network.
- **Format the USB drive** – After provisioning an appliance, remove the USB drive and format it. The appliance does not use the USB drive after it registers with the AWS Panorama service. Format the drive to remove temporary credentials, configuration files, and provisioning logs.
- **Keep the appliance up to date** – Apply appliance software updates in a timely manner. When you view an appliance in the AWS Panorama console, the console notifies you if a software update is available. For more information, see Managing an AWS Panorama Appliance.

  With the DescribeDevice API operation, you can automate checking for updates by comparing the `LatestSoftware` and `CurrentSoftware` fields. When the latest software version differs from the current version, apply the update with the console or by using the CreateJobForDevices operation.
- **If you stop using an appliance, reset it** – Before you move the appliance out of your secure data center, fully reset it. With the appliance powered down and plugged in, press both the power and reset button simultaneously for 5 seconds. This deletes account credentials, applications, and logs from the appliance.

  For more information, see AWS Panorama Appliance buttons and lights.
- **Limit access to AWS Panorama and other AWS services** – The AWSPanoramaFullAccess provides access to all AWS Panorama API operations and, as necessary, access to other services. Where possible, the policy limits access to resources based on naming conventions. For example, it provides access to AWS Secrets Manager secrets that have names starting with `panorama`. For users that need read-only access, or access to a more specific set of resources, use the managed policy as a starting point for your least-privilege policies.

For more information, see Identity-based IAM policies for AWS Panorama.

Data protection in AWS Panorama

The AWS shared responsibility model applies to data protection in AWS Panorama. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with AWS Panorama or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Sections

- Other services

Encryption in transit

AWS Panorama API endpoints support secure connections only over HTTPS. When you manage AWS Panorama resources with the AWS Management Console, AWS SDK, or the AWS Panorama API, all communication is encrypted with Transport Layer Security (TLS). Communication between the AWS Panorama Appliance and AWS is also encrypted with TLS. Communication between the AWS Panorama Appliance and cameras over RTSP is not encrypted.

For a complete list of API endpoints, see AWS Regions and endpoints in the *AWS General Reference*.

AWS Panorama Appliance

The AWS Panorama Appliance has physical ports for Ethernet, HDMI video, and USB storage. The SD card slot, Wi-Fi, and Bluetooth are not usable. The USB port is only used during provisioning to transfer a configuration archive to the appliance.

The contents of the configuration archive, which includes the appliance's provisioning certificate and network configuration, are not encrypted. AWS Panorama does not store these files; they can only be retrieved when you register an appliance. After you transfer the configuration archive to an appliance, delete it from your computer and USB storage device.

The entire file system of the appliance is encrypted. Additionally, the appliance applies several system-level protections, including rollback protection for required software updates, signed kernel and bootloader, and software integrity verification.

When you stop using the appliance, perform a full reset to delete your application data and reset the appliance software.

Applications

You control the code that you deploy to your appliance. Validate all application code for security issues before deploying it, regardless of its source. If you use 3rd party libraries in your application, carefully consider the licensing and support policies for those libraries.

Application CPU, memory, and disk usage are not constrained by the appliance software. An application using too many resources can negatively impact other applications and the device's operation. Test applications separately before combining or deploying to production environments.

Application assets (codes and models) are not isolated from access within your account, appliance, or build environment. The container images and model archives generated by the AWS Panorama Application CLI are not encrypted. Use separate accounts for production workloads and only allow access on an as-needed basis.

Other services

To store your models and application containers securely in Amazon S3, AWS Panorama uses server-side encryption with a key that Amazon S3 manages. For more information, see Protecting data using encryption in the Amazon Simple Storage Service User Guide.

Camera stream credentials are encrypted at rest in AWS Secrets Manager. The appliance's IAM role grants it permission to retrieve the secret in order to access the stream's username and password.

The AWS Panorama Appliance sends log data to Amazon CloudWatch Logs. CloudWatch Logs encrypts this data by default, and can be configured to use a customer managed key. For more information, see Encrypt log data in CloudWatch Logs using AWS KMS in the Amazon CloudWatch Logs User Guide.

Identity and access management for AWS Panorama

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use AWS Panorama resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in AWS Panorama.

Service user – If you use the AWS Panorama service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS Panorama features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in AWS Panorama, see Troubleshooting AWS Panorama identity and access.

Service administrator – If you're in charge of AWS Panorama resources at your company, you probably have full access to AWS Panorama. It's your job to determine which AWS Panorama features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AWS Panorama, see How AWS Panorama works with IAM.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS Panorama. To view example AWS Panorama identity-based policies that you can use in IAM, see AWS Panorama identity-based policy examples.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see Signing AWS API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For

the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.

- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If

you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see Service control policies in the *AWS Organizations User Guide*.

- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How AWS Panorama works with IAM

Before you use IAM to manage access to AWS Panorama, you should understand what IAM features are available to use with AWS Panorama. To get a high-level view of how AWS Panorama and other AWS services work with IAM, see AWS services that work with IAM in the *IAM User Guide*.

For an overview of permissions, policies, and roles as they are used by AWS Panorama, see AWS Panorama permissions.

AWS Panorama identity-based policy examples

By default, IAM users and roles don't have permission to create or modify AWS Panorama resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating policies on the JSON tab in the *IAM User Guide*.

Topics

- Policy best practices

- Using the AWS Panorama console
- Allow users to view their own permissions

Policy best practices

Identity-based policies determine whether someone can create, access, or delete AWS Panorama resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see IAM Access Analyzer policy validation in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Configuring MFA-protected API access in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the AWS Panorama console

To access the AWS Panorama console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the AWS Panorama resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

For more information, see Identity-based IAM policies for AWS Panorama

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "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 managed policies for AWS Panorama

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the *IAM User Guide*.

AWS Panorama provides the following managed policies. For the full contents and change history of each policy, see the linked pages in the IAM console.

- AWSPanoramaFullAccess – Provides full access to AWS Panorama, AWS Panorama access points in Amazon S3, appliance credentials in AWS Secrets Manager, and appliance logs in Amazon CloudWatch. Includes permission to create a service-linked role for AWS Panorama.
- AWSPanoramaServiceLinkedRolePolicy – Allows AWS Panorama to manage resources in AWS IoT, AWS Secrets Manager, and AWS Panorama.
- AWSPanoramaApplianceServiceRolePolicy – Allows an AWS Panorama Appliance to upload logs to CloudWatch, and to get objects from Amazon S3 access points created by AWS Panorama.

AWS Panorama updates to AWS managed policies

The following table describes updates to managed policies for AWS Panorama.

Change	Description	Date
AWSPanoramaFullAccess – Update to an existing policy	Added permissions to the user policy to allow users to view log groups in the CloudWatch Logs console.	2022-01-13
AWSPanoramaFullAccess – Update to an existing policy	Added permissions to the user policy to allow users to manage the AWS Panorama service-linked role, and to access AWS Panorama resources in other services including IAM, Amazon S3, CloudWatch, and Secrets Manager.	2021-10-20
AWSPanoramaApplianceServiceRolePolicy – New policy	New policy for the AWS Panorama Appliance service role	2021-10-20
AWSPanoramaServiceLinkedRolePolicy – New policy	New policy for the AWS Panorama service-linked role.	2021-10-20
AWS Panorama started tracking changes	AWS Panorama started tracking changes for its AWS managed policies.	2021-10-20

Using service-linked roles for AWS Panorama

AWS Panorama uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to AWS Panorama. Service-linked roles are predefined by AWS Panorama and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up AWS Panorama easier because you don't have to manually add the necessary permissions. AWS Panorama defines the permissions of its service-linked roles, and unless defined otherwise, only AWS Panorama can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your AWS Panorama resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-linked role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Sections

Service-linked role permissions for AWS Panorama

AWS Panorama uses the service-linked role named **AWSServiceRoleForAWSPanorama** – Allows AWS Panorama to manage resources in AWS IoT, AWS Secrets Manager, and AWS Panorama..

The AWSServiceRoleForAWSPanorama service-linked role trusts the following services to assume the role:

- `panorama.amazonaws.com`

The role permissions policy allows AWS Panorama to complete the following actions:

- Monitor AWS Panorama resources
- Manage AWS IoT resources for the AWS Panorama Appliance
- Access AWS Secrets Manager secrets to get camera credentials

For a full list of permissions, view the AWSPanoramaServiceLinkedRolePolicy policy in the IAM console.

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating a service-linked role for AWS Panorama

You don't need to manually create a service-linked role. When you register an appliance in the AWS Management Console, the AWS CLI, or the AWS API, AWS Panorama creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you register an appliance, AWS Panorama creates the service-linked role for you again.

Editing a service-linked role for AWS Panorama

AWS Panorama does not allow you to edit the AWSServiceRoleForAWSPanorama service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for AWS Panorama

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don’t have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

To delete the AWS Panorama resources used by the AWSServiceRoleForAWSPanorama, use the procedures in the following sections of this guide.

- Delete versions and applications
- Deregister an appliance

> **Note**
>
> If the AWS Panorama service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete the AWSServiceRoleForAWSPanorama service-linked role, use the IAM console, the AWS CLI, or the AWS API. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Supported Regions for AWS Panorama service-linked roles

AWS Panorama supports using service-linked roles in all of the regions where the service is available. For more information, see AWS Regions and endpoints.

Cross-service confused deputy prevention

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource policies to limit the permissions that AWS Panorama gives another service to the resource. If you use both global condition context keys, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement.

The value of `aws:SourceArn` must be the ARN of an AWS Panorama device.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcards (*) for the unknown portions of the ARN. For example, `arn:aws:servicename::123456789012:*`.

For instructions on securing the service role that AWS Panorama uses to give permission to the AWS Panorama Appliance, see Securing the appliance role.

Troubleshooting AWS Panorama identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS Panorama and IAM.

Topics

- I am not authorized to perform an action in AWS Panorama
- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my AWS Panorama resources

I am not authorized to perform an action in AWS Panorama

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about an appliance but does not have `panorama:DescribeAppliance` permissions.

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

In this case, Mateo asks his administrator to update his policies to allow him to access the `my-appliance` resource using the `panorama:DescribeAppliance` action.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to AWS Panorama.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in AWS Panorama. However, the action requires the service to have

permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my AWS Panorama resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether AWS Panorama supports these features, see How AWS Panorama works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Compliance validation for AWS Panorama

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- Architecting for HIPAA Security and Compliance on Amazon Web Services – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

  > **Note**
  >
  > Not all AWS services are HIPAA eligible. For more information, see the HIPAA Eligible Services Reference.

- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your

compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.

- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Additional considerations for when people are present

Below are some best practices to consider when using AWS Panorama for scenarios where people might be present:

- Ensure that you are aware of and compliant with all applicable laws and regulations for your use case. This may include laws related to the positioning and field of view of your cameras, notice and signage requirements when placing and using cameras, and the rights of people that may be present in your videos, including their privacy rights.
- Take into account the effect of your cameras on people and their privacy. In addition to legal requirements, consider whether it would be appropriate to place notice in areas where your cameras are located, and whether cameras should be placed in plain sight and free of any occlusions, so people are not surprised that they may be on camera.
- Have appropriate policies and procedures in place for the operation of your cameras and review of data obtained from the cameras.
- Consider appropriate access controls, usage limitations, and retention periods for the data obtained from your cameras.

Infrastructure security in AWS Panorama

As a managed service, AWS Panorama is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access AWS Panorama through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Deploying the AWS Panorama Appliance in your datacenter

The AWS Panorama Appliance needs internet access to communicate with AWS services. It also needs access to your internal network of cameras. It is important to consider your network configuration carefully and only provide each device the access that it needs. Be careful if your configuration allows the AWS Panorama Appliance to act as a bridge to a sensitive IP camera network.

You are responsible for the following:

- The physical and logical network security of the AWS Panorama Appliance.
- Securely operating the network-attached cameras when you use the AWS Panorama Appliance.
- Keeping the AWS Panorama Appliance and camera software updated.
- Complying with any applicable laws or regulations associated with the content of the videos and images you gather from your production environments, including those related to privacy.

The AWS Panorama Appliance uses unencrypted RTSP camera streams. For more information on connecting the AWS Panorama Appliance to your network, see Connecting the AWS Panorama Appliance to your network. For details on encryption, see Data protection in AWS Panorama.

Runtime environment software in AWS Panorama

AWS Panorama provides software that runs your application code in an Ubuntu Linux–based environment on the AWS Panorama Appliance. AWS Panorama is responsible for keeping software in the appliance image up to date. AWS Panorama regularly releases software updates, which you can apply by using the AWS Panorama console.

You can use libraries in your application code by installing them in the application's `Dockerfile`. To ensure application stability across builds, choose a specific version of each library. Update your dependencies regularly to address security issues.

Releases

The following table shows when features and software updates were released for the AWS Panorama service, software, and documentation. To ensure that you have access to all features, update your AWS Panorama Appliance to the latest software version. For more information on a release, see the linked topic.

Change	Description	Date
Appliance software update	Version 7.0.13 is a major version update that changes how the appliance manages software updates. If you restrict network communication outbound from the appliance, or connect it to a private VPC subnet, you must allow access to additional endpoints and ports before applying the update. For more information, see the change log.	December 28, 2023
Appliance software update	Version 6.2.1 includes bug fixes. For more information, see the change log.	September 6, 2023
Appliance software update	Version 6.0.8 includes bug fixes and security improvements. For more information, see the change log.	July 6, 2023
Appliance software update	Version 5.1.7 includes bug fixes and error handling improvements. For more information, see the change log.	March 31, 2023

Console update	You can now purchase the AWS Panorama Appliance from the management console. To grant a user permission to purchase devices, see Identity-based IAM policies for AWS Panorama.	February 2, 2023
Appliance software update	Version 5.0.74 includes bug fixes and error handling improvements. For more information, see the change log.	January 23, 2023
API update	Added `AllowMajorVersionUpdate` option to `OTAJobConfig` to make appliance software major version updates opt-in. For more information, see CreateJobForDevices.	January 19, 2023
New tool for developers	A new tool, "sideloading", is available in the AWS Panorama samples GitHub repository. You can use this tool to update application code without building and deploying a container. For more information, see the README.	November 16, 2022

Application base image update	Version 1.2.0 adds a timeout option to `video_in.get()` , sets the `AWS_REGION` environment variable, and improves error handling. For more information, see the change log.	November 16, 2022
Appliance software update	Version 5.0.42 includes bug fixes and security updates. For more information, see the change log.	November 16, 2022
Appliance software update	Version 5.0.7 adds support for rebooting appliances remotely and pausing camera streams remotely. For more information, see the change log.	October 13, 2022
Appliance software update	Version 4.3.93 adds support for retrieving logs from an offline device. For more information, see the change log.	August 24, 2022
Appliance software update	Version 4.3.72 includes bug fixes and security updates. For more information, see the change log.	June 23, 2022
AWS PrivateLink support	AWS Panorama supports VPC endpoints for managing AWS Panorama resources from a private subnet. For more information, see Using VPC endpoints.	June 2, 2022

Appliance software update	Version 4.3.55 improves storage utilization for the `console_output.log`. For more information, see the change log.	May 5, 2022
Lenovo ThinkEdge® SE70	A new appliance for AWS Panorama is available from Lenovo. The Lenovo ThinkEdge® SE70, powered by Nvidia Jetson Xavier NX, supports the same features as the AWS Panorama Appliance . For more information, see Compatible devices.	April 6, 2022
Application base image update	Version 1.1.0 improves performance when running background threads and adds a flag (is_cached) to media objects that indicates if the image is fresh. For more information, see gallery.ecr.aws.	March 29, 2022
Appliance software update	Version 4.3.45 adds support for GPU access and inbound ports. For more information, see the change log.	March 24, 2022
Appliance software update	Version 4.3.35 improves security and performance. For more information, see the change log.	February 22, 2022

Updated managed policies	AWS Identity and Access Management managed policies for AWS Panorama have been updated. For details, see AWS managed policies.	January 13, 2022
Provisioning logs	With appliance software 4.3.23, the appliance writes logs to a USB drive during provisioning. For more information, see Logs.	January 13, 2022
NTP server configuration	You can now configure the AWS Panorama Appliance to use a specific NTP server for clock syncronization. Configure NTP settings during appliance setup with other networking settings. For more information, see Setting up.	January 13, 2022
Additional regions	AWS Panorama is now available in the Asia Pacific (Singapore) and Asia Pacific (Sydney) Regions.	January 13, 2022
Appliance software update	Version 4.3.4 adds support for the `precisionMode` setting for models and updates logging behavior. For more information, see the change log.	November 8, 2021

Updated managed policies	AWS Identity and Access Management managed policies for AWS Panorama have been updated. For details, see AWS managed policies.	October 20, 2021
General availability	AWS Panorama is now available to all customers in the US East (N. Virginia) , US West (Oregon), Europe (Ireland), and Canada (Central) Regions. To purchase an AWS Panorama Appliance, visit AWS Panorama.	October 20, 2021
Preview	AWS Panorama is available by invitation in the US East (N. Virginia) and US West (Oregon) Regions.	December 1, 2020