AWS Panorama

Developer Guide

AWS Panorama: Developer Guide

Table of Contents

What is AWS Panorama?

With AWS Panorama, you can build computer vision applications for your business or customers without purchasing special cameras. By using the AWS Panorama Appliance with your existing network cameras, you can run applications that use machine learning (ML) to collect data from video streams, output video with text and graphical overlays, and interact with other AWS services.

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 (p. 2).

Getting started with AWS Panorama

To get started with AWS Panorama, first learn about the service's concepts (p. 3) and the terminology used in this guide. Then you can use the AWS Panorama console to register your AWS Panorama Appliance (p. 5) and create an application (p. 9). 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 How to purchase.

The AWS Panorama sample application (p. 13) analyzes a video stream to tally the number of people detected and display the results on a connected display. 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 final two topics in this chapter detail requirements for models and cameras (p. 22), and the hardware specifications of the AWS Panorama Appliance (p. 23). 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 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*. 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 (p. 30).

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.

For more information, see Building AWS Panorama applications (p. 41).

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 nodes 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.

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 ??? (p. 51).

Setting up the AWS Panorama Appliance

To get started using your AWS Panorama Appliance, 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.

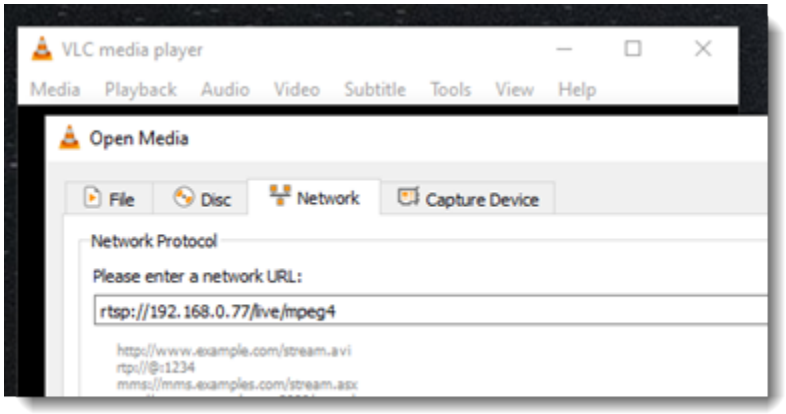
Sections

Prerequisites

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

- **Display** – A display with HDMI input for viewing the sample application output.
- **USB drive** (included) – A FAT32-formatted USB 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 (p. 26) 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

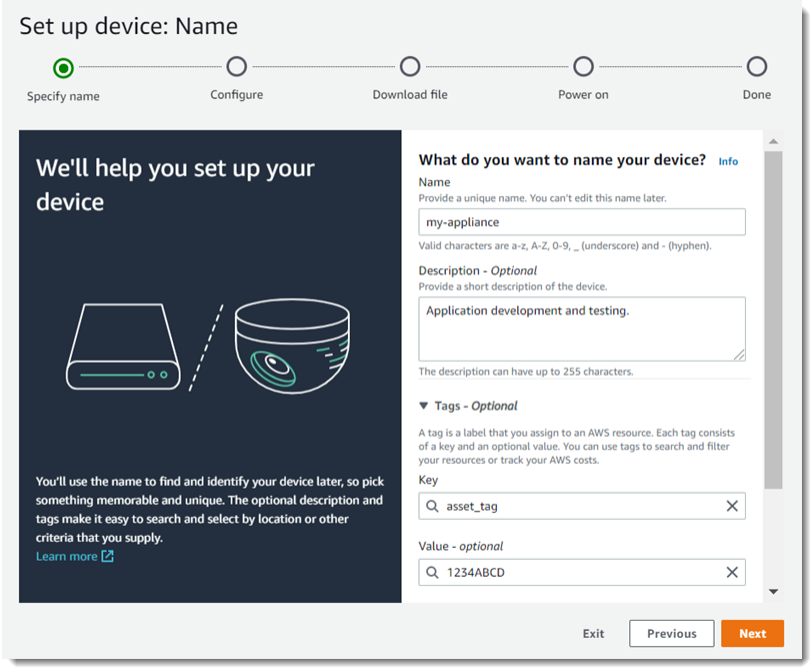
The AWS Panorama Appliance is a hardware device that connects to network-enabled cameras over a local network connection. It uses a Linux-based operating system that includes the AWS Panorama Application SDK and supporting software for running computer vision applications.

To connect to AWS for appliance management and application deployment, the AWS Panorama 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 an AWS Panorama Appliance

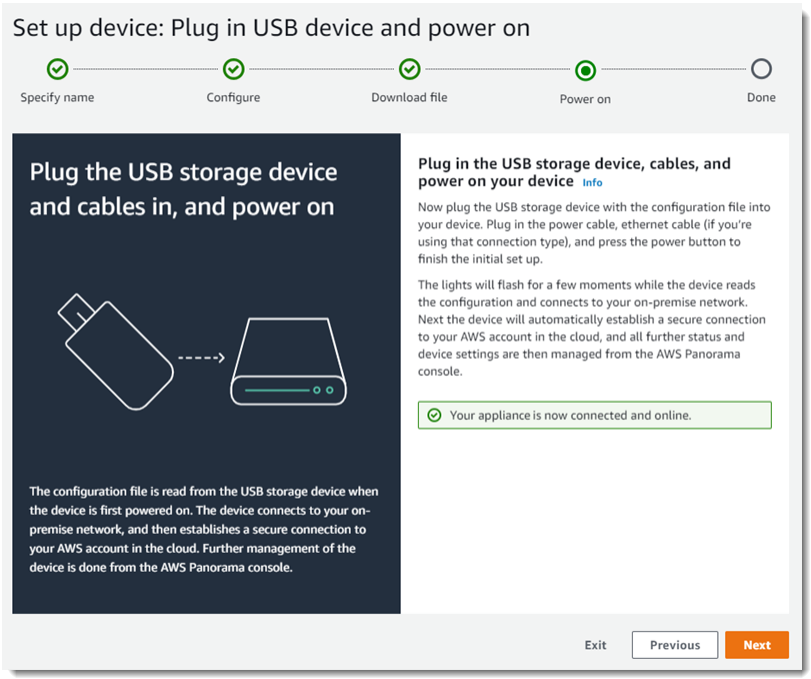
1. Connect the USB drive to your computer. Prepare the AWS Panorama Appliance by connecting the network and power cables. The appliance powers on and waits for a USB drive to be connected.
2. Open the AWS Panorama console Getting started page.
3. Choose **Add device**.
4. Choose **Begin setup**.
5. Enter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



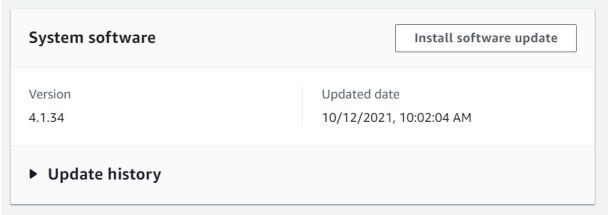
11. Choose **Done**.

Upgrade the appliance software

The AWS Panorama Appliance has several software components, including a Linux operating system, the AWS Panorama application SDK (p. 56), 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.

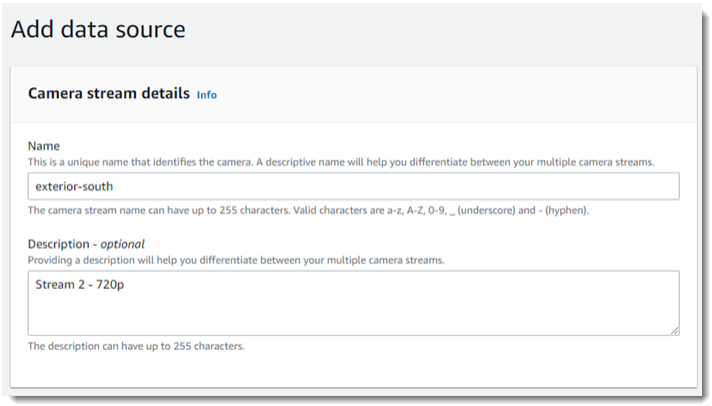
The upgrade process can take 30 minutes or more. You can monitor its progress in the AWS Panorama console or on a connected monitor. When the process completes, the appliance reboots.

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 (p. 65).

To deploy a sample application, continue to the next topic (p. 9).

Deploying the AWS Panorama sample application

After you've set up your AWS Panorama Appliance (p. 5) 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 detect people 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.

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 AWS Panorama 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": {}
    }
]
...
```

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:

   - **AWS Panorama Appliance system** – `/aws/panorama/devices/`*`device-id`*
   - **Application instance** – `/aws/panorama/devices/`*`device-id`*`/applications/`*`instance-id`*

If the application doesn't start running, check the application and device logs (p. 64) in Amazon CloudWatch Logs.

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

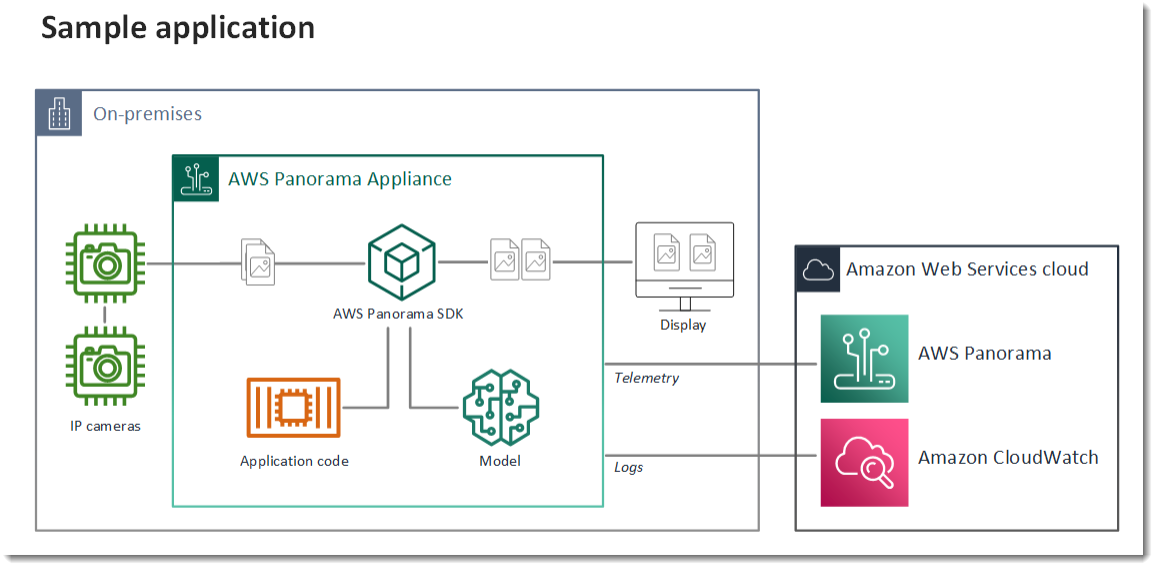
If you encountered errors while deploying or running the sample application, see Troubleshooting (p. 65).

To learn more about the sample application's features and implementation, continue to the next topic (p. 13).

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.

```
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 (p. 51), 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.

For more information about using models in AWS Panorama, see Computer vision models (p. 51).

Supported cameras

The AWS Panorama Appliance supports H.264 video streams from cameras that output RTSP over a local network. The following camera models have been tested for compatibility with the AWS Panorama Appliance:

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

For the appliance's hardware specifications, see AWS Panorama Appliance specifications (p. 23).

AWS Panorama Appliance specifications

The AWS Panorama Appliance has the following hardware specifications.

Component	Specification
Processor and GPU	Nvidia Jetson Xavier AGX 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
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.

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 users, groups, or roles. To manage permissions for an application, you create a role and assign it to the application.

To manage permissions for users (p. 25) 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 (p. 26) that the AWS Panorama service uses to access other services on your behalf.

An application role (p. 29) 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). Identity-based policies can apply directly to IAM users, or to IAM groups and 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 (p. 26) for AWS Panorama. 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 (p. 26) 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 (p. 29).

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 (p. 13) includes an AWS CloudFormation template and script that create an application role. It is a service role (p. 26) 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 (p. 24).

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, you register cameras for use with applications. You manage camera streams (p. 36) 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 (p. 2).

Topics

Managing an AWS Panorama Appliance

You use the AWS Panorama console to configure, upgrade or deregister the AWS Panorama Appliance.

To set up an appliance, follow the instructions in the getting started tutorial (p. 5). The setup process creates the resources in AWS Panorama that track your appliance and coordinate updates and deployments.

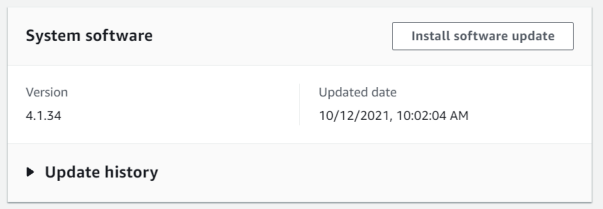
Sections

Update the appliance software

You view and deploy software updates for the AWS Panorama 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 the AWS Panorama Appliance, you can use the AWS Panorama console to deregister it and delete the associated AWS IoT resources.

When you delete an appliance from the AWS Panorama service, data on the appliance is not deleted automatically. This data includes applications, camera information, the appliance certificate, network configuration, and logs. You can remove applications (p. 38) from the device prior to deregistering it, or reset the device to its factory state.

To delete an appliance

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

To fully reset the device and delete all data, press both the power button and the reset button for over 5 seconds. For more information, see AWS Panorama Appliance buttons and lights (p. 39).

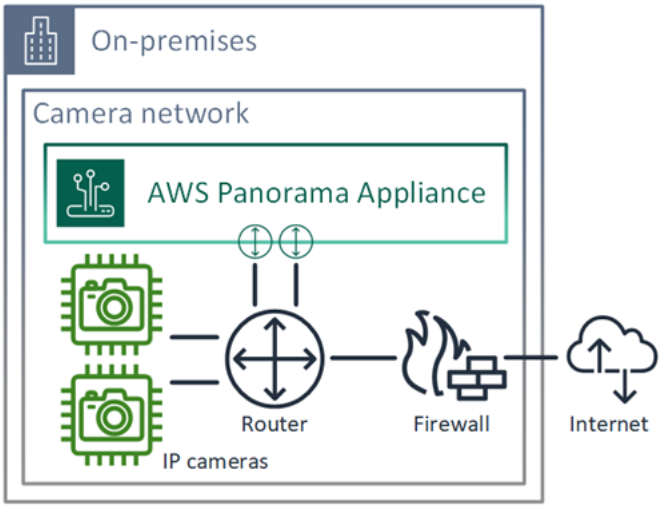
Connecting the AWS Panorama Appliance to your network

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.

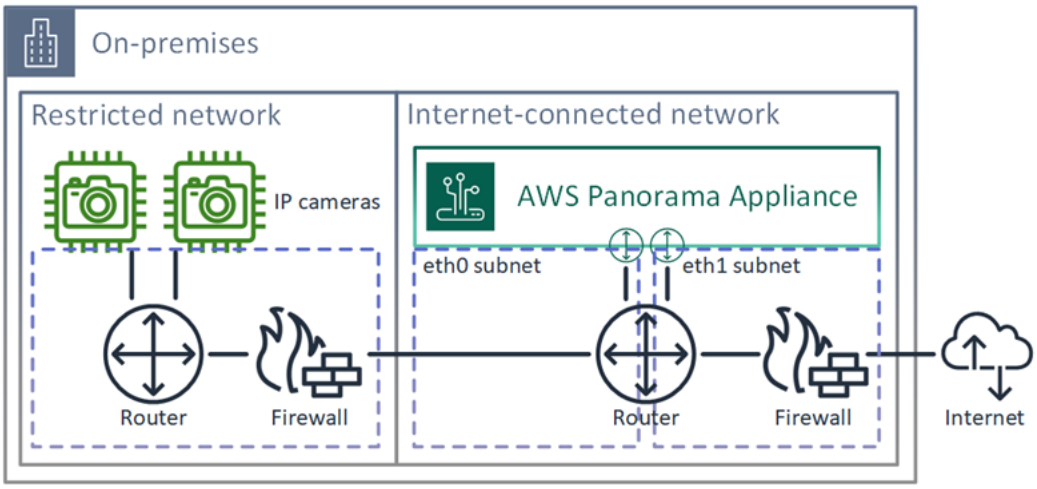


For details on the ports and endpoints that the appliance needs access to, see Configuring internet access (p. 34) and Configuring local network access (p. 34).

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 internet access

During provisioning (p. 5), 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 multiple AWS services in addition to AWS Panorama. Configure your firewall to allow the appliance to connect to these endpoints on port 443.

Internet access

- **AWS IoT (HTTPS and MQTT, port 443)** – AWS IoT Core and device management endpoints. For details, see AWS IoT Device Management endpoints and quotas in the Amazon Web Services General Reference.
- **Amazon CloudWatch (HTTPS, port 443)** – `monitoring.<region>.aws.amazon.com`.
- **Amazon CloudWatch Logs (HTTPS, port 443)** – `logs.<region>.aws.amazon.com`.
- **Amazon Simple Storage Service (HTTPS, port 443)** – `s3-accesspoint.<region>.aws.amazon.com`.

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.

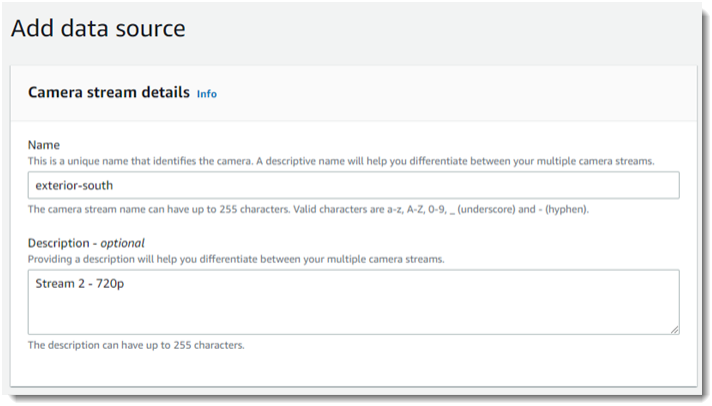
Managing camera streams in AWS Panorama

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 (p. 66).

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**.

For a list of cameras that are compatible with the AWS Panorama Appliance, see Supported computer vision models and cameras (p. 22).

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 Devices page.
2. Choose an appliance.
3. Choose **Camera streams**.
4. Choose a stream.
5. Choose **Remove stream**.

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

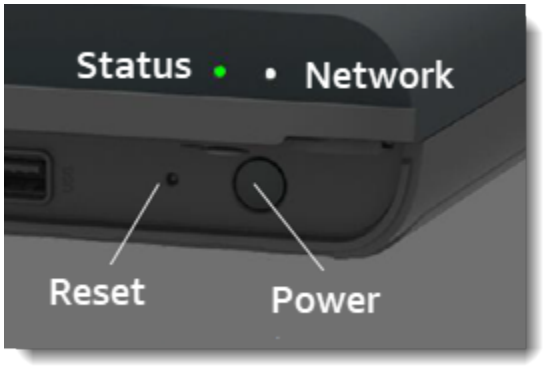
1. Open the AWS Panorama console Devices page.
2. Choose an appliance.

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 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

To power down the appliance, press and hold the power button for 1 second. The shutdown sequence takes about 10 seconds.

To reset the appliance, use the following button combinations. A short press is 1 second. A long press is 5 seconds. To start a reset operation, the appliance must be powered down. For operations that require multiple buttons, press and hold both buttons simultaneously.

Reset operations

- **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.

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 (p. 51) 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 (p. 13) 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 (p. 56). The application SDK is a Python library that supports models generated with PyTorch, Apache MXNet, and TensorFlow.

Topics

Managing applications in AWS Panorama

Use the AWS Panorama console to manage applications and application versions. An *application* is a computer vision program that runs on the AWS Panorama Appliance. *Application versions* are immutable snapshots of an application's configuration. AWS Panorama saves previous versions of your applications so that you can roll back updates that aren't successful, or run different versions on different appliances.

To create an application, you need application and a computer vision model that is stored in Amazon SageMaker or Amazon Simple Storage Service (Amazon S3). The application code is a Python script that uses the AWS Panorama Application SDK to process inputs, run inference, and output video. If you have not created an application yet, see Getting started with AWS Panorama (p. 2) for a walkthrough.

Sections

Deploy an application

To deploy an application, use 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 (p. 29), 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 (p. 65).

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**.

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.

Sections

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."
                }
            }
    ]
```

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.

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)
```

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"
           }
           ...
       ]
   }
```

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"
            }
        ]
    }
}
```

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 `nodeOverrides` tells AWS Panorama which nodes in this document override which in the application manifest.

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

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.

You can use a sample model (p. 51) or build your own. A model can detect multiple objects in an image, and each result can have multiple outputs, such as the name of a class, a confidence rating, and a bounding box. You can train a model outside of AWS and store it in Amazon Simple Storage Service (Amazon S3), or train it with Amazon SageMaker. To build a model in SageMaker, you can use the built-in image classification algorithm. AWS Panorama can reference the training job to find the trained model that it created in Amazon S3.

Sections

Sample model

This guide uses a sample object detection model. The sample model uses the object detection algorithm to identify multiple objects in an image. For each object, the model outputs the type of object, a confidence score, and coordinates of a bounding box. It uses the Single Shot multibox detector (SSD) framework and the ResNet base network.

- Download the sample model

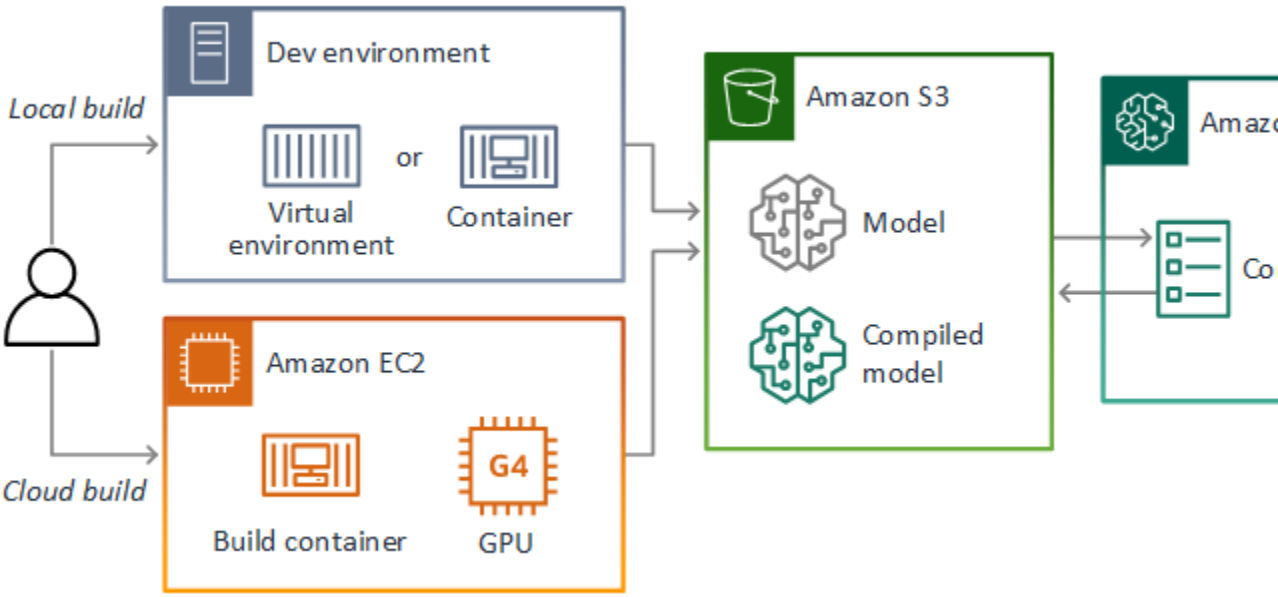
Building a custom model

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 1.15 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.

Using models in code

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)
```

A model returns one or more results, which can include probabilities for detected classes, location information, and other data. The following example shows a function that processes results from basic classification model. The model returns an array of probabilities, which is the first and only value in the results array. The application code finds the values with the highest probabilities and maps them to labels in a resource file that's loaded during initialization.

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)
```

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.

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 (p. 29), 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")
```

The topic name can be anything. To publish messages, the device 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**.

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)
```

For more information about the AWS Panorama application SDK, see The AWS Panorama Application SDK (p. 56).

The AWS Panorama Application SDK

The AWS Panorama Application SDK is a Python library for developing AWS Panorama applications. In your application code (p. 13), 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 (p. 31).

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

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 (p. 10) 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.

Migrate applications from preview

Applications from the AWS Panorama preview are not compatible with the hardware or software for general availability. If you've developed applications with the AWS Panorama Appliance Developer Kit, you must migrate them to the new application architecture. You can reuse portions of your application code, but dependency management, application SDK methods, and deployment tools are all new for general availability.

If you haven't already, deploy the sample application (p. 9) to familiarize yourself with the new development process and application structure.

Sections

Application code

In preview, you created an application class that inherits from `panoramasdk.base` and implements abstract methods for defining an interface, initialization, and processing images. For general availability, you inherit from `panoramasdk.node` and there are no abstract methods. You define the entrypoint for the application in a descriptor file and manage the lifecycle of the application class in a script, which can be the main method of your application class.

- descriptor.json–Descriptor file with entrypoint.
- application.py – Main method.

The sample application code contains many boilerplate methods that you can use without modification. The areas where you need to add your own code are the `preprocess` method, which prepares images for inference, and the `process_results` method, which processes the output from the model and modifies the output video stream.

AWS Panorama Application SDK

All of the AWS Panorama Application SDK methods are new or different. See the sample application code for examples of the new methods.

- application.py – Stream methods (`add_label` and `add_rect`).
- application.py – Video input.
- application.py – Video output.

Element	Preview	General availability
Base class	`panoramasdk.base`	`panoramasdk.node`
Interface	`interface(self)` instance method	Package configuration (p. 61)

Element	Preview	General availability
Initialization	`init(self, parameters, inputs, outputs)` abstract instance method	None
Process stream	`entry(self, inputs, outputs)` abstract instance method	None
Start application	`run()` instance method	None
Access parameters	`init()` `parameters` argument	`inputs` base class attribute `self.inputs.threshold.get()`
Read camera stream	`entry()` `inputs` argument	`inputs` base class attribute `self.inputs.video_in.get()`
Read camera stream	`entry()` `outputs` argument	`outputs` base class attribute `self.outputs.video_out.put(streams)`
Load a model	`model` class	None
Run inference	`model.batch(input_index, input_data)` instance method	`call()` base class method `self.call({"data":image_data}, 'model_node_name')`

For more information, see Application SDK reference.

Interface

In preview, you defined an interface in the application class to declare parameters, inputs, and outputs. For general availability, you use the package manifest to declare inputs and outputs. You create nodes for camera streams and models, and map inputs and outputs in the application manifest.

- package.json – Package configuration.
- graph.json – Application manifest.

Dependencies

For preview, you installed Python modules locally and packaged them into a ZIP file with your application code. For general availability, you can use the application's `Dockerfile` to install libraries with `pip`, or copy modules into the application container alongside your code.

- Dockerfile – Install dependencies with pip.

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 (p. 26).

For help troubleshooting specific errors, see Troubleshooting (p. 65).

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 event logs in CloudWatch 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:

   - **AWS Panorama Appliance system** – `/aws/panorama/devices/device-id`
   - **Application instance** – `/aws/panorama/devices/device-id/applications/instance-id`

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

- `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.
- `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.

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

Application logs

- **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.

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 (p. 26).

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 event logs in CloudWatch Logs (p. 64).

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).

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 (p. 39).

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.

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

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. This content includes 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 user accounts with 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 recommend TLS 1.2 or later.
- 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 personal data that is stored in Amazon S3.
- If you require FIPS 140-2 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) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form 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 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

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 (p. 39) 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 (p. 78).

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 employees 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 (p. 73).

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 (p. 73).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see Signing in to the AWS Management Console as an IAM user or root user in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 signing process in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also 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 Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you first create an AWS account, you begin with a single 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 do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing access keys for IAM users in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

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:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated users and roles in the *IAM User Guide*.

- **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 How IAM roles differ from resource-based policies 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.
  - **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, Resources, and Condition Keys for AWS Panorama in the *Service Authorization Reference*.
  - **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 IAM 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 IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. 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**.

Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

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 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 How SCPs work 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 (p. 24).

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

Identity-based policies are very powerful. They 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 using AWS managed policies** – To start using AWS Panorama quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get started using permissions with AWS managed policies in the *IAM User Guide*.
- **Grant least privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant least privilege in the *IAM User Guide*.
- **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.
- **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to

specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON policy elements: Condition 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 (p. 25)

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

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ViewOnlyAccess** AWS managed policy provides read-only access to many AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions 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 (p. 26) 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 role to allow users to manage the AWS Panorama service-linked role (p. 75), 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 (p. 43)
- Deregister an appliance (p. 31)

> **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 (p. 26).

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

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, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies 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 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 asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

Important
Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the *IAM User Guide*.

I'm an administrator and want to allow others to access AWS Panorama

To allow others to access AWS Panorama, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in AWS Panorama.

To get started right away, see Creating your first IAM delegated user and group in the *IAM User Guide*.

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 (p. 73).
- 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 How IAM roles differ from resource-based policies in the *IAM User Guide*.

Compliance validation for AWS Panorama

AWS Panorama is not in scope of any AWS compliance programs. For a list of AWS services in scope of specific compliance programs, see AWS services in scope by compliance program. 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 Panorama 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 security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA security and compliance whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS compliance resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Config – This AWS 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 that helps you check your compliance with security industry standards and best practices.

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 the AWS global network security procedures that are described in the Amazon Web Services: Overview of security processes whitepaper.

You use AWS published API calls to access AWS Panorama through the network. Clients must support Transport Layer Security (TLS) 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). 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 AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

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 (p. 33). For details on encryption, see Data protection in AWS Panorama (p. 67).

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 (p. 31).

You can use libraries in your application code by installing them in the application's`Dockerfile` To ensure application stability across deployments, 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 (p. 31) to the latest software version. For more information on a release, see the linked topic.

update-history-change	update-history-description	update-history-date
Appliance software update (p. 83)	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 (p. 83)	AWS Identity and Access Management managed policies for AWS Panorama have been updated. For details, see AWS managed policies.	October 20, 2021
General availability (p. 83)	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 (p. 83)	AWS Panorama is available by invitation in the US East (N. Virginia) and US West (Oregon) Regions.	December 1, 2020