aws

User Guide

AWS Ground Station

AWS Ground Station: User Guide

Table of Contents

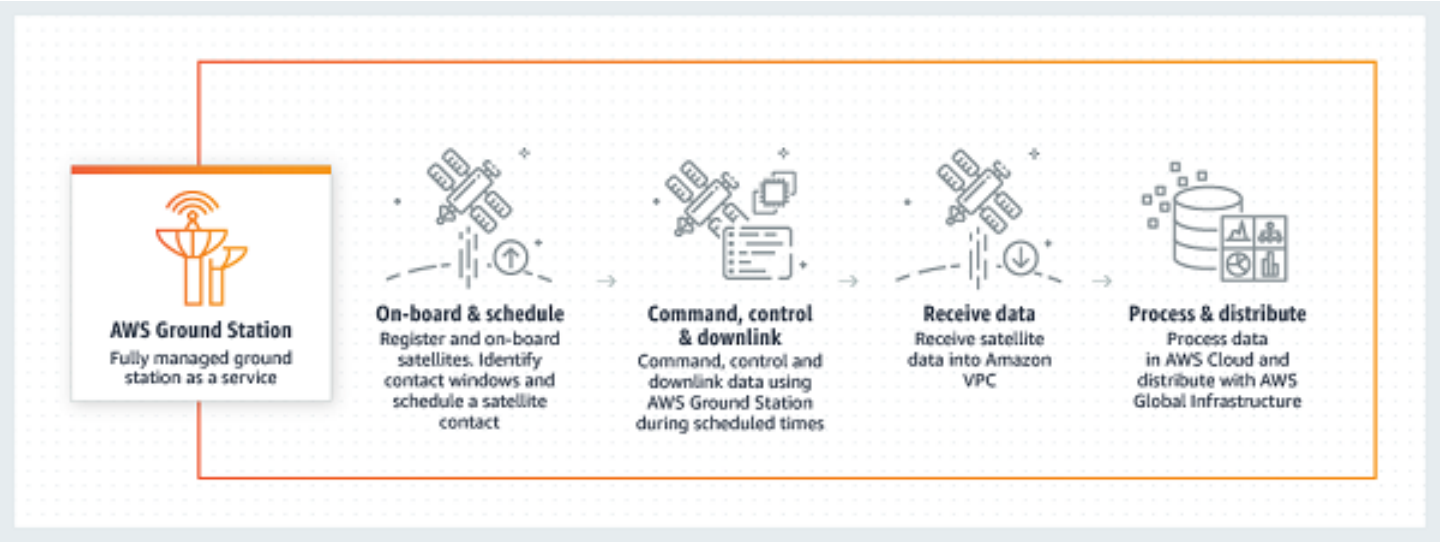
What is AWS Ground Station?

AWS Ground Station is a fully managed service that provides secure, fast, and predictable satellite communications across a global infrastructure. With AWS Ground Station, you no longer have to build, manage, or scale your own ground station infrastructure. AWS Ground Station enables you to focus on innovating and rapidly experimenting with new applications that ingest satellite data, rather than spend resources on building, operating, and scaling your own ground stations.

Using AWS's low-latency, high-bandwidth global fiber network, you can begin processing your satellite data within seconds of reception at the antenna system. This enables you to turn raw data into processed information or analyzed knowledge within a matter of seconds.

For organizations with specialized requirements, AWS Ground Station also offers AWS Ground Station Dedicated Antennas — custom-built antenna systems that AWS manages on your behalf, providing dedicated access to antennas built to your specifications.

Common use cases



AWS Ground Station allows you to communicate with your satellites bi-directionally and supports the following use cases:

- **Downlink data** – Receive data from your satellites, transmitting X-band and S-band frequencies, delivered to an Amazon EC2 instance in real-time (VITA-49 format), or directly to an Amazon S3 bucket in your account (PCAP format). Additionally, for satellites that use a supported

modulation and encoding scheme, you can choose between receiving data that is demodulated and decoded, or the raw digital intermediate frequency (DigIF) samples (VITA-49 format).

- **Uplink data** – Send data and commands to your satellites, that receive S-band frequencies, by sending DigIF data (VITA-49 format) to be transmitted by AWS Ground Station.
- **Uplink echo** – Validate commands sent to your spacecraft, and perform other advanced tasks, by receiving your transmitted signal on a physically co-located antenna.
- **Software Defined Radio (SDR) / Front End Processor (FEP)** – Use your existing SDR and/or FEP, that's capable of running on an Amazon EC2 instance, to process your data in real-time to send/receive your existing waveforms, and generate your data products.
- **Telemetry, Tracking, and Command (TT&C)** – Perform TT&C using a combination of the previously listed use cases to manage your satellite fleet.
- **Cross Region Data Delivery** – Operate multiple simultaneous contacts using AWS Ground Station's global antenna network from a single AWS Region.
- **Digital twin** – Test scheduling, verification of configurations, and proper error handling at a reduced cost without using production antenna capacity.

Next steps

We recommend that you begin by reading the following sections:

- To learn essential AWS Ground Station concepts, see How AWS Ground Station works.
- To learn how to set up your account and resources to use AWS Ground Station, see Get started.
- To programmatically use AWS Ground Station, please refer to the AWS Ground Station API Reference. The API Reference describes all the API operations for AWS Ground Station in detail. It also provides sample requests, responses, and errors for the supported web service protocols. You can use the AWS CLI, or an AWS SDK, in the language of your choice, to write code that interacts with AWS Ground Station.

How AWS Ground Station works

AWS Ground Station operates ground-based *antennas* to facilitate communication with your *satellite*. The physical characteristics of what the antennas can do are abstracted and are referred to as *capabilities*. The physical location of the antenna along with its current capabilities can be referenced in the AWS Ground Station Locations section. Please contact us through the AWS Support Center Console if your use case requires additional capabilities, additional location offerings, or more precise antenna locations.

To use one of the AWS Ground Station antennas you must reserve a time at a specific location. This reservation is referred to as a *contact*. To successfully schedule a contact, AWS Ground Station requires additional data to ensure its success.

- **Your satellite must be onboarded to one or more locations** – This ensures you have approval to operate the various capabilities at the requested location.
- **Your satellite must have a valid *ephemeris*** – This ensures the antennas have line of sight and can accurately point at your satellite during the contact.
- **You must have a valid *mission profile*** – This allows you to customize how this contact will behave including how you will receive and send data to your satellite. You may utilize multiple mission profiles for the same vehicle to create different contacts to fit different operating postures or scenarios you encounter.

Satellite onboarding

Onboarding a satellite into AWS Ground Station is a multistep process involving data collection, technical validation, spectrum licensing, with integration and testing. The Satellite onboarding section of the guide will walk you through this process.

Mission profile composition

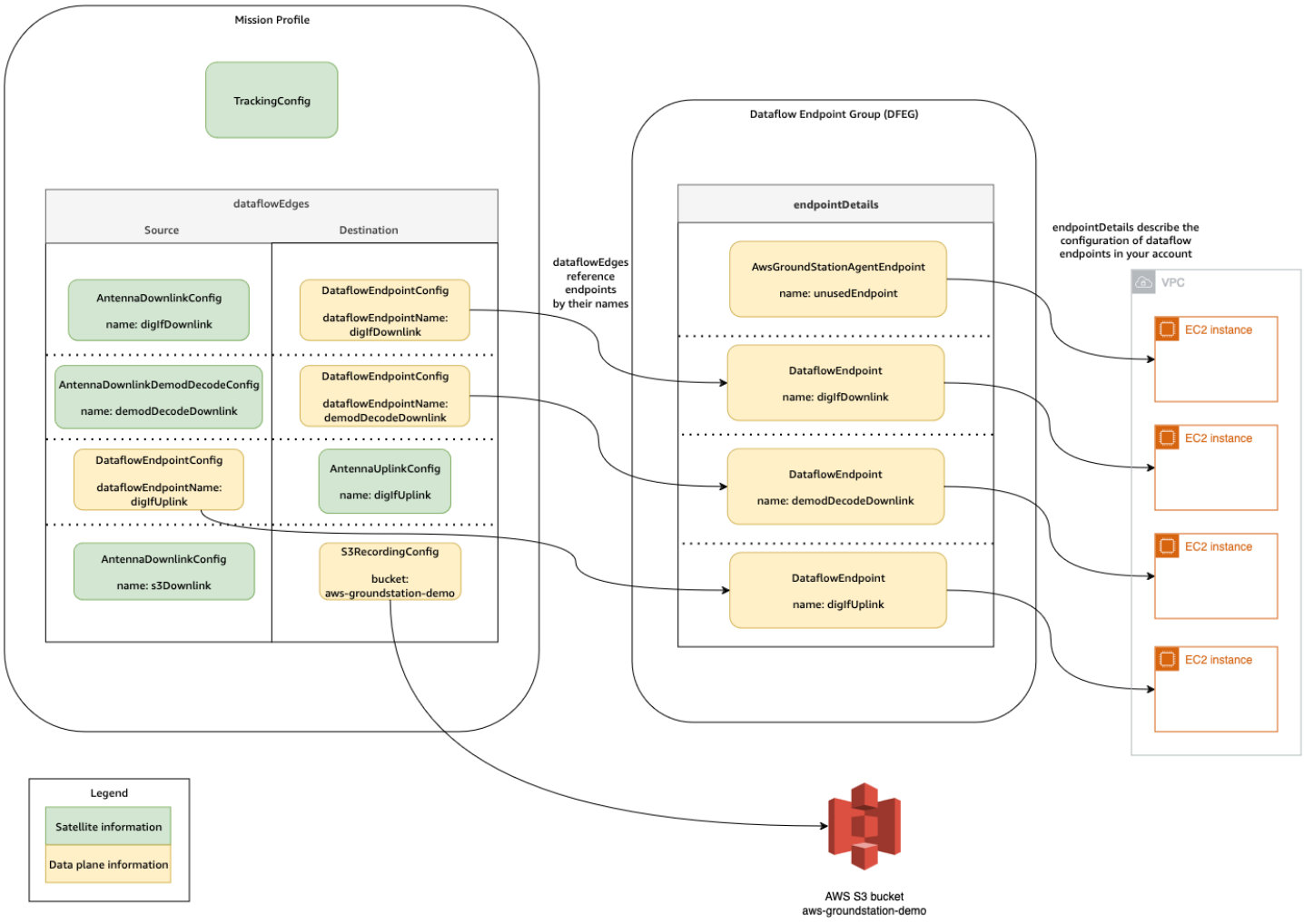
The satellite frequency information, data plane information, and other details are encapsulated into a mission profile. The mission profile is a collection of *config* components. This allows you to reuse config components across different mission profiles as suits your use case. Since mission profiles don't directly reference individual satellites, but instead only have information about their technical capabilities, mission profiles can also be reused by multiple satellites that have the same configuration.

A valid mission profile will have a *tracking config* and one or more *dataflows*. The tracking config will specify your preference for tracking during a contact. Each config pair within a dataflow establishes a source and destination. Depending on your satellite and its operational modes, the exact number of dataflows will vary in a mission profile to represent your uplink and downlink communication paths as well as any data processing aspects.

- For more information on configuring your Amazon VPC, Amazon S3, and Amazon EC2 resources that will be used during a contact, see Work with dataflows.
- For details on how each config behaves, see Use AWS Ground Station Configs.
- For specific details on all parameters expected, see Use AWS Ground Station Mission Profiles.
- For examples on how various mission profiles can be created to support your use case, see Example mission profile configurations.

The following diagram shows an example mission profile and additional resources needed. Note that the example shows a dataflow endpoint which is not needed for this mission profile, named *unusedEndpoint*, to demonstrate the flexibility. The example supports the following dataflows:

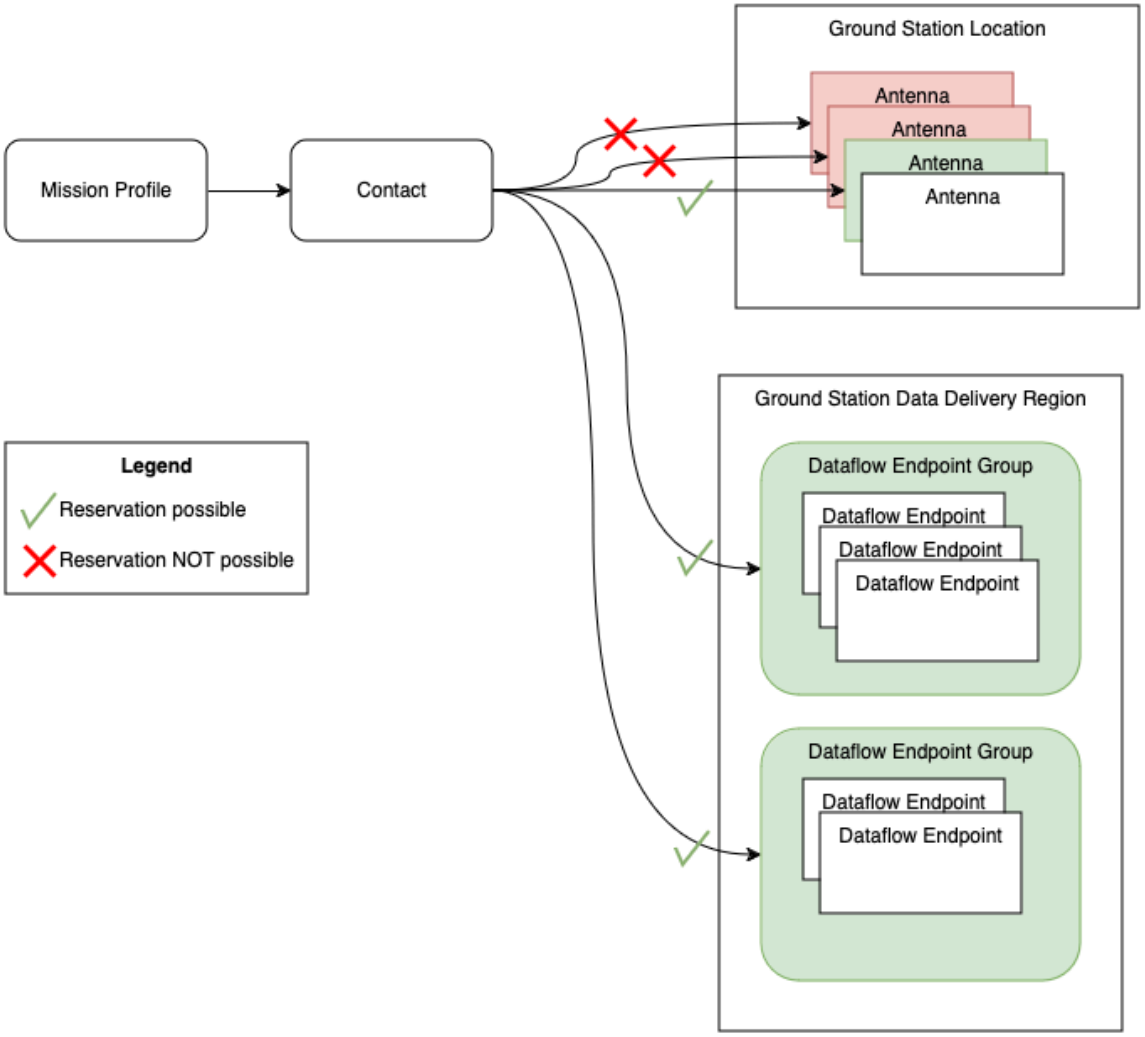
- Synchronous downlink of digital intermediate frequency data to an Amazon EC2 instance that you manage. Denoted by the name *digIfDownlink*.
- Asynchronous downlink of digital intermediate frequency data to an Amazon S3 bucket. Denoted by the bucket name *aws-groundstation-demo*.
- Synchronous downlink of demodulated and decoded data to an Amazon EC2 instance that you manage. Denoted by the name *demodDecodeDownlink*.
- Synchronous uplink of data from an Amazon EC2 instance that you manage to a AWS Ground Station managed antenna. Denoted by the name *digIfUplink*.



Contact scheduling

With a valid mission profile, you can request a contact with your onboarded satellites. The contact reservation request is asynchronous to allow time for the global antenna service to achieve a consistent schedule across all AWS Regions involved. During this process, various antennas at the requested ground station location are evaluated to determine if they are available and capable to process the contact. During this process, your configured *dataflow endpoints* are also evaluated to determine their availability. While this evaluation is occurring, the contact status will be in SCHEDULING.

This asynchronous scheduling process will finish within five minutes of the request, but typically finishes within one minute. Please review Automate AWS Ground Station with Events for event-based monitoring during scheduling time.



Contacts which can be performed and have availability result in *SCHEDULED* contacts. With a scheduled contact, the resources which are needed to perform your contact have been reserved across the needed AWS Regions as defined by your mission profile. Contacts which cannot be performed, or have unavailable parts will result in *FAILED_TO_SCHEDULE* contacts. See Troubleshoot FAILED_TO_SCHEDULE contacts for debugging details.

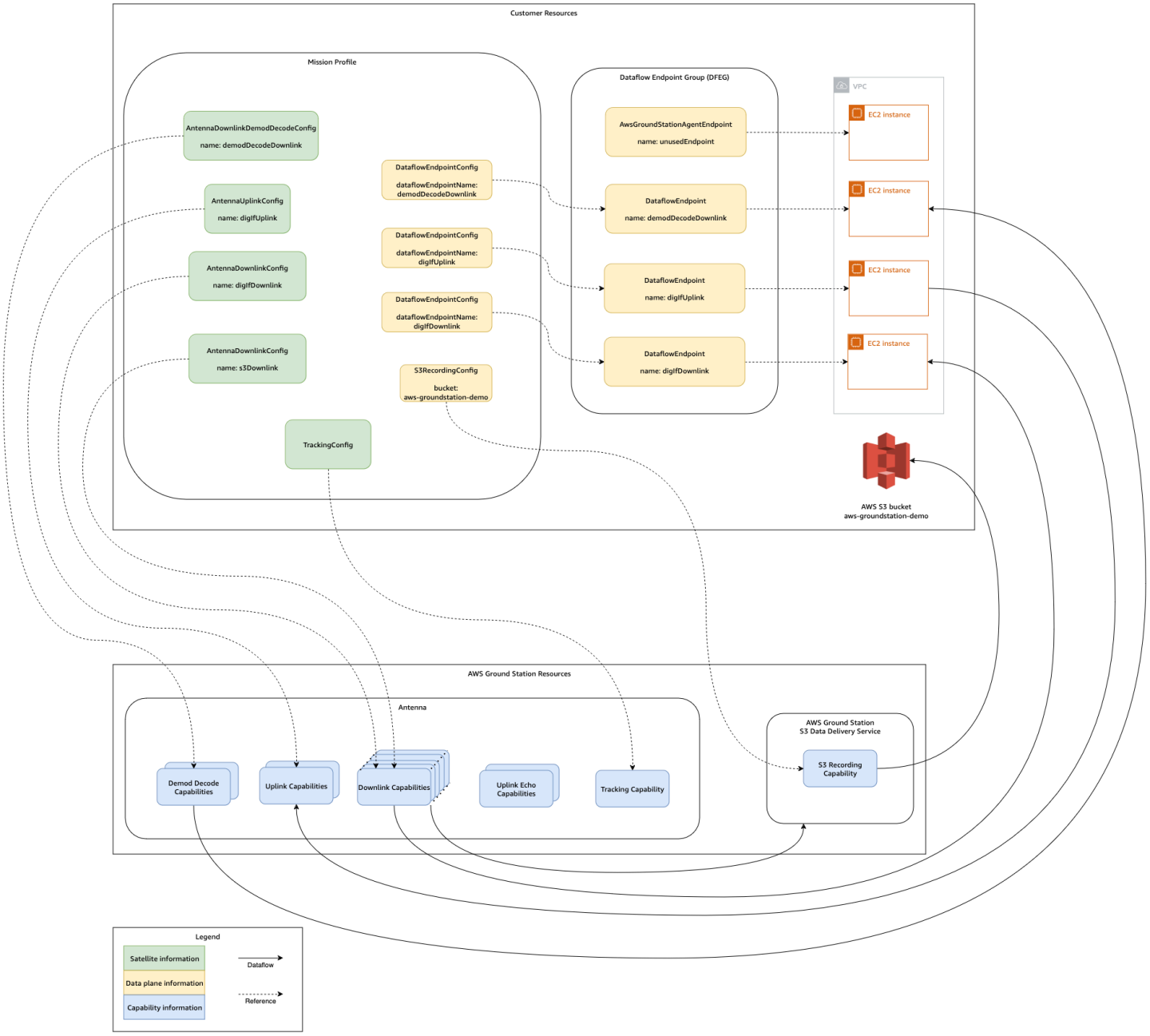
Contact execution

AWS Ground Station will automatically orchestrate your AWS managed resources during your contact reservation. If applicable, you are responsible for orchestrating EC2 resources defined by your mission profile as dataflow endpoints. AWS Ground Station provides AWS EventBridge Events for automating orchestration of your resources to reduce costs. See Automate AWS Ground Station with Events for more details.

During the contact, telemetry about your contact performance is delivered to AWS CloudWatch. For information about how to monitor your contact during execution, please see Understand monitoring with AWS Ground Station.

The following diagram continues the previous example by showing the same resources orchestrated during the contact.

> **Note**
>
> Not all the antenna capabilities were used in this example. For instance, there are more than a dozen antenna downlink capabilities available at each antenna that support multiple frequencies and polarizations. For more details about the number of each capability type available from AWS Ground Station antennas, and their supported frequencies and polarizations, see AWS Ground Station Site Capabilities.



At the end of your contact, AWS Ground Station will assess the performance of your contact and will determine a final contact status. Contacts where no errors are detected will result in a *COMPLETED* contact status. Contacts where service errors have caused data delivery issues during the contact will result in an *AWS_FAILED* status. Contacts where client or user errors have caused data delivery issues during the contact will result in a *FAILED* status. Errors outside a contact time, that is during pre-pass or post-pass, are not taken into account during the adjudication.

See Understand contact lifecycle for more information.

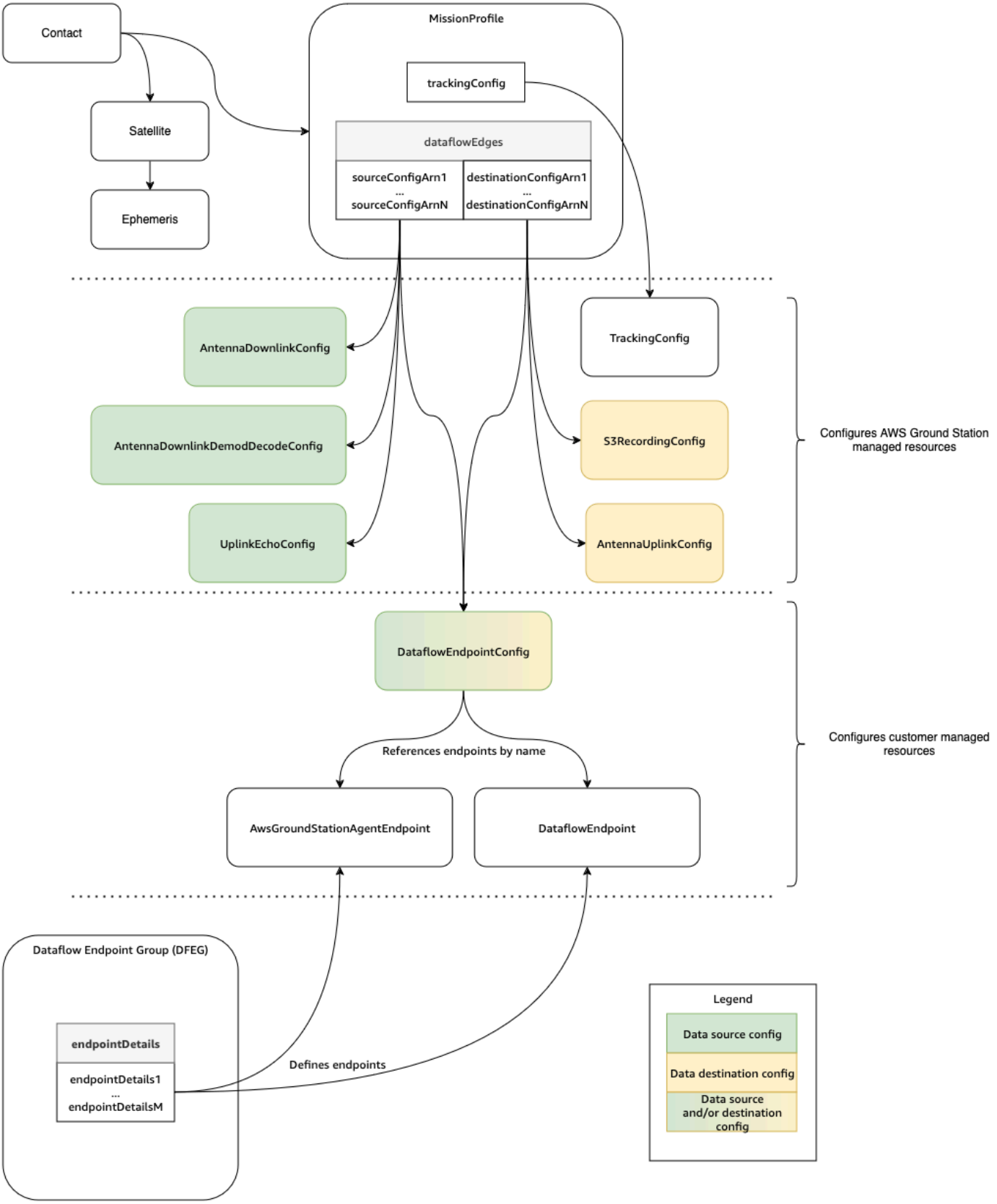
Digital twin

The digital twin feature for AWS Ground Station allows you to schedule contacts against virtual ground station locations. These virtual ground stations are exact replicas of production ground stations including antenna capabilities, site masks, and actual GPS coordinates. The digital twin feature enables you to test your contact orchestration workflow for a fraction of the cost compared to production ground stations. See Use the AWS Ground Station digital twin feature for more information.

Understand AWS Ground Station Core components

This section provides detailed definitions for the core components of AWS Ground Station.

The following diagram shows the core components of AWS Ground Station and how they relate to each other. The arrows indicate the direction of the dependencies between components, where each component points to its dependencies.


Contact
Satellite
Ephemeris
MissionProfile
trackingConfig
dataflowEdges
sourceConfigArn1
...
sourceConfigArnN
destinationConfigArn1
...
destinationConfigArnN
AntennaDownlinkConfig
AntennaDownlinkDemodDecodeConfig
UplinkEchoConfig
TrackingConfig
S3RecordingConfig
AntennaUplinkConfig
Configures AWS Ground Station managed resources
DataflowEndpointConfig
References endpoints by name
AwsGroundStationAgentEndpoint
DataflowEndpoint
Configures customer managed resources
Dataflow Endpoint Group (DFEG)
endpointDetails
endpointDetails1
...
endpointDetailsM
Defines endpoints
Legend
Data source config
Data destination config
Data source and/or destination config

The following topics describe the AWS Ground Station core components in detail.

Topics

Use AWS Ground Station Mission Profiles

Mission profiles contain configs and parameters for how contacts are executed. When you reserve a contact or search for available contacts, you supply the mission profile that you intend to use. Mission profiles bring all of your configs together and define how the antenna will be configured and where data will go during your contact.

Mission profiles can be shared across satellites which share the same radio characteristics. You can create additional dataflow endpoint groups to bound the maximum simultaneous contacts you want to perform for your constellation.

Tracking configs are specified as a unique field within the mission profile. Tracking configs are used to specify your preference for using program-tracking and auto-tracking during your contact. For more information, see Tracking Config.

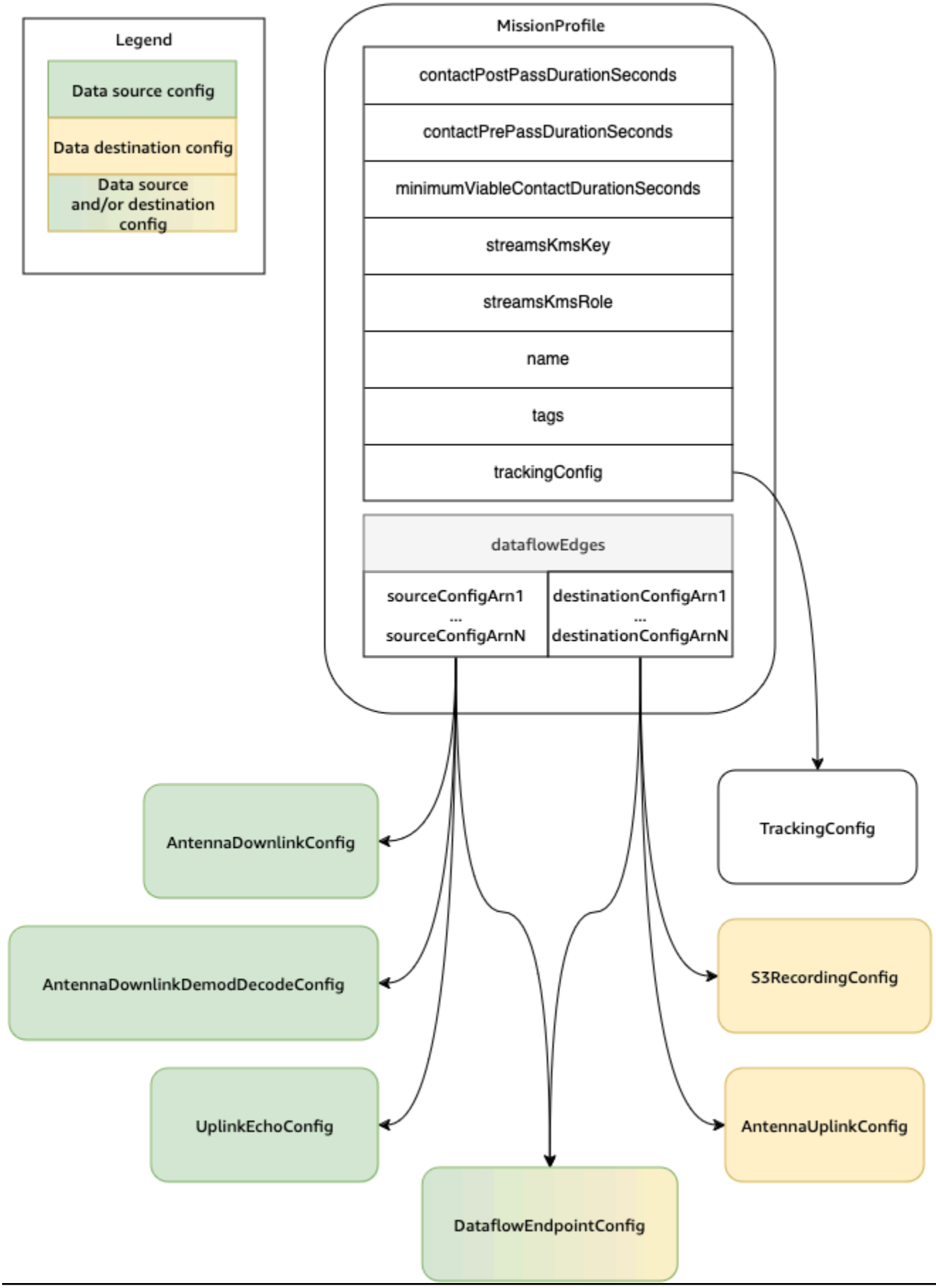
All other configs are contained in the `dataflowEdges` field of the mission profile. These configs can be thought of as dataflow nodes that each represent an AWS Ground Station managed resource that can send or receive data and its associated configuration. The `dataflowEdges` field defines which source and destination dataflow nodes (configs) are needed. A single dataflow edge is a list of two config Amazon Resource Names (ARNs)—the first is the *source* config and the second is the *destination* config. By specifying a dataflow edge between two configs, you are telling AWS Ground Station from where and to where data should flow during a contact. For more information, see Use AWS Ground Station Configs.

The `contactPrePassDurationSeconds` and `contactPostPassDurationSeconds` allow you to specify times relative to the contact where you will receive an CloudWatch Event notification. For a timeline of events related to your contact, please read Understand contact lifecycle.

The `name` field of the mission profile helps distinguish between the mission profiles that you create.

The `streamsKmsRole` and `streamsKmsKey` are used to define the encryption used by AWS Ground Station for your data delivery with AWS Ground Station Agent. Please see Data encryption during transit for AWS Ground Station.

The `telemetrySinkConfigArn` field is optional and allows you to enable AWS Ground Station telemetry during contacts. When specified, AWS Ground Station streams near real-time telemetry data to your account during the execution of your contacts. For more information about configuring and using telemetry, see Work with telemetry.


Legend
Data source config
Data destination config
Data source and/or destination config
MissionProfile
contactPostPassDurationSeconds
contactPrePassDurationSeconds
minimumViableContactDurationSeconds
streamsKmsKey
streamsKmsRole
name
tags
trackingConfig
dataflowEdges
sourceConfigArn1 ... sourceConfigArnN
destinationConfigArn1 ... destinationConfigArnN
AntennaDownlinkConfig
AntennaDownlinkDemodDecodeConfig
UplinkEchoConfig
TrackingConfig
S3RecordingConfig
AntennaUplinkConfig
DataflowEndpointConfig

A full list of parameters and examples is included at the following documentation.

- AWS::GroundStation::MissionProfile CloudFormation resource type

Use AWS Ground Station Configs

Configs are resources that AWS Ground Station uses to define the parameters for each aspect of your contact. Add the configs you want to a mission profile, and then that mission profile will be used when executing the contact. You can define several different types of configs. The configs can be grouped into three categories:

- Tracking configs
- Dataflow configs
- Telemetry configs

A *TrackingConfig* is the only type of tracking config. It's used to configure the autotrack setting of the antenna during a contact, and is required in a mission profile.

The configs that can be used in a mission profile dataflow can be thought of as dataflow nodes that each represent an AWS Ground Station managed resource that can send or receive data. A mission profile requires at least one pair of these configs, with one representing a source of data, and one representing a destination. These configs are summarized in the following table.

Config name	Dataflow source/destination
AntennaDownlinkConfig	Source
AntennaDownlinkDemodDecodeConfig	Source
UplinkEchoConfig	Source
S3RecordingConfig	Destination
AntennaUplinkConfig	Destination
DataflowEndpointConfig	Source and/or Destination

A *TelemetrySinkConfig* is the only type of telemetry config. It's used to configure where telemetry data will be delivered during a contact, and is optional in a mission profile. When included, AWS Ground Station streams near real-time telemetry to your account during the execution of your contacts.

See the following documentation for more information about how to perform operations on configs using CloudFormation, the AWS Command Line Interface, or the AWS Ground Station API. Links to documentation for specific config types are also provided below.

- AWS::GroundStation::Config CloudFormation resource type
- Config AWS CLI reference
- Config API reference

Tracking Config

You can use tracking configs in the mission profile to determine whether autotrack should be enabled during your contacts. This config has a single parameter: `autotrack`. The `autotrack` parameter can have the following values:

- `REQUIRED` - Autotrack is required for your contacts.
- `PREFERRED` - Autotrack is preferred for contacts, but contacts can still be executed without autotrack.
- `REMOVED` - No autotrack should be used for your contacts.

AWS Ground Station will utilize programmatic tracking which will point based on your ephemeris when autotrack is not used. Please reference Understand how AWS Ground Station uses ephemerides for details about how ephemeris is constructed.

Autotrack will use program tracking until the expected signal is found. Once that occurs, it will continue to track based on the strength of the signal.

See the following documentation for more information about how to perform operations on tracking configs using CloudFormation, the AWS Command Line Interface, or the AWS Ground Station API.

- AWS::GroundStation::Config TrackingConfig CloudFormation property
- Config AWS CLI reference (see the `trackingConfig -> (structure)` section)

- TrackingConfig API reference

Antenna Downlink Config

You can use antenna downlink configs to configure the antenna for downlink during your contact. They consist of a spectrum config that specifies the frequency, bandwidth, and polarization that should be used during your downlink contact.

This config represents a source node in a dataflow. It is responsible for digitizing radio frequency data. Data streamed from this node will follow the Signal Data/IP Format. For more detailed information on how to construct dataflows with this config, see Work with dataflows

If your downlink use case requires demodulation or decoding, see the Antenna Downlink Demod Decode Config.

See the following documentation for more information about how to perform operations on antenna downlink configs using CloudFormation, the AWS Command Line Interface, or the AWS Ground Station API.

- AWS::GroundStation::Config AntennaDownlinkConfig CloudFormation property
- Config AWS CLI reference (see the `antennaDownlinkConfig -> (structure)` section)
- AntennaDownlinkConfig API reference

Antenna Downlink Demod Decode Config

Antenna downlink demod decode configs are a more complex and customizable config type that you can use to execute downlink contacts with demodulation and/or decoding. If you're interested in executing these types of contacts, please open an AWS Support ticket through the AWS Support Center Console. We'll help you define the right config and mission profile for your use case.

This config represents a source node in a dataflow. It is responsible for digitizing radio frequency data and performing the demodulation and decoding as specified. Data streamed from this node will follow the Demodulated/Decoded Data/IP Format. For more detailed information on how to construct dataflows with this config, see Work with dataflows

See the following documentation for more information about how to perform operations on antenna downlink demod decode configs using CloudFormation, the AWS Command Line Interface, or the AWS Ground Station API.

- AWS::GroundStation::Config AntennaDownlinkDemodDecodeConfig CloudFormation property
- Config AWS CLI reference (see the `antennaDownlinkDemodDecodeConfig -> (structure)` section)
- AntennaDownlinkDemodDecodeConfig API reference

Antenna Uplink Config

You can use antenna uplink configs to configure the antenna for uplink during your contact. They consist of a spectrum config with frequency, polarization, and target effective isotropic radiated power (EIRP). For information about how to configure a contact for uplink loopback, see Antenna Uplink Echo Config.

This config represents a destination node in a dataflow. It will convert the provided digitized radio frequency data signal into an analog signal and emit it for your satellite to receive. Data streamed to this node is expected to meet the Signal Data/IP Format. For more detailed information on how to construct dataflows with this config, see Work with dataflows

See the following documentation for more information about how to perform operations on antenna uplink configs using CloudFormation, the AWS Command Line Interface, or the AWS Ground Station API.

- AWS::GroundStation::Config AntennaUplinkConfig CloudFormation property
- Config AWS CLI reference (see the `antennaUplinkConfig -> (structure)` section)
- AntennaUplinkConfig API reference

Antenna Uplink Echo Config

Uplink echo configs tell the antenna how to execute an uplink echo. An uplink echo can be used to validate commands sent to your spacecraft, and perform other advanced tasks. This is achieved by recording the actual signal transmitted by the AWS Ground Station antenna (i.e. the uplink). This echoes the signal sent by the antenna back to your dataflow endpoint and should match the transmitted signal. An uplink echo config contains the ARN of an uplink config. The antenna uses the parameters from the uplink config pointed to by the ARN when executing an uplink echo.

This config represents a source node in a dataflow. Data streamed from this node will meet the Signal Data/IP Format. For more detailed information on how to construct dataflows with this config, see Work with dataflows

See the following documentation for more information about how to perform operations on uplink echo configs using CloudFormation, the AWS Command Line Interface, or the AWS Ground Station API.

- AWS::GroundStation::Config UplinkEchoConfig CloudFormation property
- Config AWS CLI reference (see the `uplinkEchoConfig -> (structure)` section)
- UplinkEchoConfig API reference

Dataflow Endpoint Config

> **Note**
>
> Dataflow endpoint configs are only used for data delivery to Amazon EC2 and are not used for data delivery to Amazon S3.

You can use dataflow endpoint configs to specify which dataflow endpoint in a dataflow endpoint group from which or to which you want data to flow during a contact. The two parameters of a dataflow endpoint config specify the name and region of the dataflow endpoint. When reserving a contact, AWS Ground Station analyzes the mission profile you specified and attempts to find a dataflow endpoint *group* within the AWS Region that contains all of the dataflow *endpoints* specified by the dataflow endpoint *configs* contained in your mission profile. If a suitable dataflow endpoint *group* is found, the contact status will become SCHEDULED, otherwise it will become FAILED_TO_SCHEDULE. For more information about the possible statuses of a contact, see AWS Ground Station contact statuses.

The `dataflowEndpointName` property of a dataflow endpoint config specifies which dataflow endpoint in a dataflow endpoint group to which or from which data will flow during a contact.

The `dataflowEndpointRegion` property specifies which region the dataflow endpoint resides in. If a region is specified in your dataflow endpoint config, AWS Ground Station looks for a dataflow endpoint in the region specified. If no region is specified, AWS Ground Station will default to the contact's ground station region. A contact is considered a cross region data delivery contact if your dataflow endpoint's region is not the same as the contact's ground station region. See Work with dataflows for more information on cross-region dataflows.

See Use AWS Ground Station Dataflow endpoint groups for tips on how different naming schemes for your dataflows can benefit your use case.

For more detailed information on how to construct dataflows with this config, see Work with dataflows

See the following documentation for more information about how to perform operations on dataflow endpoint configs using CloudFormation, the AWS Command Line Interface, or the AWS Ground Station API.

- AWS::GroundStation::Config DataflowEndpointConfig CloudFormation property
- Config AWS CLI reference (see the `dataflowEndpointConfig -> (structure)` section)
- DataflowEndpointConfig API reference

Amazon S3 Recording Config

> **Note**
>
> Amazon S3 recording configs are only used for data delivery to Amazon S3 and are not used for data delivery to Amazon EC2.

This config represents a destination node in a dataflow. This node will encapsulate incoming data from the dataflow's source node into pcap data. For more detailed information on how to construct dataflows with this config, see Work with dataflows

You can use S3 recording configs to specify an Amazon S3 bucket to which you want downlinked data delivered along with the naming convention used. The following specifies restrictions and details about these parameters:

- The Amazon S3 bucket's name must begin with `aws-groundstation` .
- The IAM role must have a trust policy that allows the `groundstation.amazonaws.com` service principal to assume the role. See the Example Trust Policy section below for an example. During config creation the config resource id does not exist, the trust policy must use an asterisk (*) in place of *`your-config-id`* and can be updated after creation with the config resource id.

Example Trust Policy

For more information on how to update a role's trust policy, see Managing IAM roles in the IAM User Guide.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "groundstation.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "999999999999"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:groundstation:us-
east-1:999999999999:config/s3-recording/your-config-id"
                }
            }
        }
    ]
}
```

- The IAM role must have an IAM policy that allows the role to perform the `s3:GetBucketLocation` action on the bucket and `s3:PutObject` action on the bucket's objects. If the Amazon S3 bucket has a bucket policy, then the bucket policy must also allow the IAM role to perform these actions. See the Example Role Policy section below for an example.

Example Role Policy

For more information on how to update or attach a role policy, see Managing IAM policies in the IAM User Guide.

JSON

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

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketLocation"
      ],
      "Resource": [
        "arn:aws:s3:::your-bucket-name"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::your-bucket-name/*"
      ]
    }
  ]
}
```

- The prefix will be used when naming the S3 data object. You can specify optional keys for substitution, these values will be replaced with the corresponding information from your contact details. For example, a prefix of `{satellite_id}/{year}/{month}/{day}` will be replaced and would result with an output like `fake_satellite_id/2021/01/10`

  Optional keys for substitution: `{satellite_id}` | `{config-name}` | `{config-id}` | `{year}` | `{month}` | `{day}` |

See the following documentation for more information about how to perform operations on S3 recording configs using CloudFormation, the AWS Command Line Interface, or the AWS Ground Station API.

- AWS::GroundStation::Config S3RecordingConfig CloudFormation property
- Config AWS CLI reference (see the `s3RecordingConfig -> (structure)` section)
- S3RecordingConfig API reference

Telemetry Sink Config

You can use telemetry sink configs to specify where you want telemetry data delivered during satellite contacts. A telemetry sink config is optional and is added to your mission profile to schedule telemetry-enabled contacts. The following specifies restrictions and details about these parameters:

- The IAM role must have a trust policy that allows the `groundstation.amazonaws.com` service principal to assume the role. See the Example Trust Policy section below for an example.

Example Trust Policy

For more information on how to update a role's trust policy, see Managing IAM roles in the IAM User Guide.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "groundstation.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- The IAM role must have an IAM policy that allows the role to perform the `kinesis:DescribeStream`, `kinesis:PutRecord` and `kinesis:PutRecords` actions on the stream. See the Example Role Policy section below for an example.

Example Role Policy

For more information on how to update or attach a role policy, see Managing IAM policies in the IAM User Guide.

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

```
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:DescribeStream",
        "kinesis:PutRecord",
        "kinesis:PutRecords"
      ],
      "Resource": "arn:aws:kinesis:us-east-2:999999999999:stream/your-stream-name"
    }
  ]
}
```

When you include a telemetry sink config in your mission profile, AWS Ground Station will stream telemetry data to your account during contacts. For more information about telemetry types, data format, and setting up the necessary AWS resources, see Work with telemetry.

See the following documentation for more information about how to perform operations on telemetry sink configs using CloudFormation, the AWS Command Line Interface, or the AWS Ground Station API.

- AWS::GroundStation::Config TelemetrySinkConfig CloudFormation property
- Config AWS CLI reference (see the `telemetrySinkConfig -> (structure)` section)
- TelemetrySinkConfig API reference

Use AWS Ground Station Dataflow endpoint groups

Dataflow endpoints define the location where you want the data to be synchronously streamed to or from during contacts. Dataflow endpoints are always created as part of a *dataflow endpoint group*. By including multiple dataflow endpoints in a group, you are asserting that the specified endpoints can all be used together during a single contact. For example, if a contact needs to send data to three separate dataflow endpoints, you must have three endpoints in a single dataflow endpoint group that match the dataflow endpoint configs in your mission profile.

Dataflow endpoint group versions

AWS Ground Station supports two versions of dataflow endpoint groups:

- **DataflowEndpointGroup** - The original implementation that supports uplink and downlink using a dataflow endpoint, and downlink-only for an AWS Ground Station Agent endpoint

- **DataflowEndpointGroupV2** - Updated version that supports both uplink and downlink dataflows for AWS Ground Station Agent endpoints with improved clarity and functionality

Dataflow endpoint group comparison

Feature	DataflowEndpointGroup	DataflowEndpointGroupV2
Supported endpoint types	DataflowEndpoint, AwsGroundStationAgentEndpoint	DownlinkAwsGroundStationAgentEndpoint, UplinkAwsGroundStationAgentEndpoint
Endpoints supporting uplink	DataflowEndpoint	UplinkAwsGroundStationAgentEndpoint
Endpoints supporting downlink	DataflowEndpoint, AwsGroundStationAgentEndpoint	DownlinkAwsGroundStationAgentEndpoint

DataflowEndpointGroupV2 was created to support uplink dataflows and to make the language surrounding dataflow endpoint groups clearer. We recommend using UplinkAwsGroundStationAgentEndpoint and DownlinkAwsGroundStationAgentEndpoint endpoints with a DataflowEndpointGroupV2 for all new use cases. DataflowEndpointGroup remains supported for backward compatibility, but DataflowEndpointGroupV2 provides enhanced functionality and clearer configuration options.

Tip

The dataflow endpoints are identified by a name of your choosing when executing contacts. These names do not need to be unique across the account. This allows multiple contacts across different satellites and antenna to be executed at the same time using the same mission profile. This can be useful if you have a constellation of satellites that have the same operating characteristics. You can scale the number of dataflow endpoint groups up to fit the maximum number of simultaneous contacts your constellation of satellite requires.

When one or more resources in a dataflow endpoint group is in use for a contact, the entire group is reserved for the duration of that contact. You may execute multiple contacts concurrently, but those contacts must be executed on different dataflow endpoint groups.

> **Important**
>
> Dataflow endpoint groups must be in a HEALTHY state to schedule contacts using them. For information on how to troubleshoot dataflow endpoint groups that are not in a HEALTHY state, see Troubleshoot DataflowEndpointGroups not in a HEALTHY state.

See the following documentation for more information about how to perform operations on dataflow endpoint groups using CloudFormation, the AWS Command Line Interface, or the AWS Ground Station API.

- AWS::GroundStation::DataflowEndpointGroup CloudFormation resource type
- Dataflow Endpoint Group AWS CLI reference
- Dataflow Endpoint Group API reference

Dataflow endpoints

The members of a dataflow endpoint group are dataflow endpoints. The supported endpoint types depend on which dataflow endpoint group version you use.

DataflowEndpointGroup endpoints

DataflowEndpointGroup supports uplink and downlink using a dataflow endpoint, and downlink-only for an AWS Ground Station Agent endpoint. For both types of endpoints, you will create the supporting constructs (e.g. IP addresses) prior to creating the dataflow endpoint group. Please see Work with dataflows for recommendations on which dataflow endpoint type to use and how to set up the supporting constructs.

The following sections describe both supported endpoint types.

> **Important**
>
> All dataflow endpoints within a single dataflow endpoint group must be of the same type. You cannot mix AWS Ground Station Agent endpoints with Dataflow endpoints in the same

group. If your use case requires both types of endpoints, you must create separate dataflow endpoint groups for each type.
For DataflowEndpointGroupV2, you can mix UplinkAwsGroundStationAgentEndpoint and DownlinkAwsGroundStationAgentEndpoint in the same group.

AWS Ground Station Agent endpoint

The AWS Ground Station Agent Endpoint utilizes the AWS Ground Station Agent as a software component to terminate connections. To construct an AWS Ground Station Agent Endpoint, you will only populate the `AwsGroundStationAgentEndpoint` field of the EndpointDetails. For more information about the AWS Ground Station Agent, see the full AWS Ground Station Agent User Guide.

The `AwsGroundStationAgentEndpoint` consists of the following:

- `Name` - The dataflow endpoint name. For the contact to use this dataflow endpoint, this name must match the name used in your dataflow endpoint config.
- `EgressAddress` - The IP and port address used to egress data from the Agent.
- `IngressAddress` - The IP and port address used to ingress data to the Agent.

Dataflow endpoint

The Dataflow Endpoint utilizes a networking application as a software component to terminate connections. Use Dataflow Endpoint when you want to uplink Digital Signal Data, downlink less-than 50MHz of Digital Signal Data, or downlink Demodulated/Decoded Signal Data. To construct a Dataflow Endpoint, you will populate the `Endpoint` and `Security Details` fields of the EndpointDetails.

The `Endpoint` consists of the following:

- `Name` - The dataflow endpoint name. For the contact to use this dataflow endpoint, this name must match the name used in your dataflow endpoint config.
- `Address` - The IP and port address used.

The `SecurityDetails` consists of the following:

- `roleArn` - The Amazon Resource Name (ARN) of a role that AWS Ground Station will assume to create Elastic Network Interfaces (ENIs) in your VPC. These ENIs serve as the ingress and egress points of data streamed during a contact.
- `securityGroupIds` - The security groups to attach to the elastic network interfaces.
- `subnetIds` - A list of subnets where AWS Ground Station may place elastic network interfaces to send streams to your instances. If multiple subnets are specified, they must be routable to one another. If the subnets are in different Availability Zones (AZs), you may incur cross-AZ data transfer charges.

The IAM role passed into `roleArn` must have a trust policy that allows the `groundstation.amazonaws.com` service principal to assume the role. See the Example Trust Policy section below for an example. During endpoint creation the endpoint resource id does not exist, so the trust policy must use an asterisk (*) in place of *`your-endpoint-id`*. This can be updated after creation to use the endpoint resource id in order to scope the trust policy to that specific dataflow endpoint group.

The IAM role must have an IAM policy that allows AWS Ground Station to set up the ENIs. See the Example Role Policy section below for an example.

Example Trust Policy

For more information on how to update a role's trust policy, see Managing IAM roles in the IAM User Guide.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "groundstation.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "999999999999"
                },
```

```
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:groundstation:us-
east-1:999999999999:dataflow-endpoint-group/your-endpoint-id"
                }
            }
        }
    ]
}
```

Example Role Policy

For more information on how to update or attach a role policy, see Managing IAM policies in the IAM User Guide.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:DeleteNetworkInterface",
        "ec2:CreateNetworkInterfacePermission",
        "ec2:DeleteNetworkInterfacePermission",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "ec2:DescribeSecurityGroups"
      ],
      "Resource": "*"
    }
  ]
}
```

DataflowEndpointGroupV2 endpoints

DataflowEndpointGroupV2 introduces specialized endpoint types that provide clearer configuration and enhanced functionality:

- **UplinkAwsGroundStationAgentEndpoint** - Optimized for uplink dataflows
- **DownlinkAwsGroundStationAgentEndpoint** - Optimized for downlink dataflows

These specialized endpoints replace the generic AwsGroundStationAgentEndpoint with direction-specific configurations that make it easier to set up and manage your dataflows.

Uplink AWS Ground Station Agent endpoint

The UplinkAwsGroundStationAgentEndpoint is specifically designed for uplink dataflows and provides clearer configuration options. Use this endpoint type when you need to provide data to AWS Ground Station to be uplinked to your satellite.

The `UplinkAwsGroundStationAgentEndpoint` consists of the following:

- `Name` - The dataflow endpoint name. For the contact to use this dataflow endpoint, this name must match the name used in your dataflow endpoint config.
- `IngressAddressAndPort` - Single IP and port address for data input to the agent
- `AgentIpAndPortAddress` - Port range for agent communication

Downlink AWS Ground Station Agent endpoint

The DownlinkAwsGroundStationAgentEndpoint is optimized for downlink dataflows, including narrowband downlink, wideband demodulation/decode, and uplink echo scenarios.

The `DownlinkAwsGroundStationAgentEndpoint` consists of the following:

- `Name` - The dataflow endpoint name. For the contact to use this dataflow endpoint, this name must match the name used in your dataflow endpoint config.
- `EgressAddressAndPort` - Single IP and port address for data output from the agent
- `AgentIpAndPortAddress` - Port range for agent communication

Creating dataflow endpoint groups

You can create dataflow endpoint groups using either version:

CreateDataflowEndpointGroup

Use CreateDataflowEndpointGroup for backward compatibility or when you need to use the generic AwsGroundStationAgentEndpoint or DataflowEndpoint types.

CreateDataflowEndpointGroupV2

Use CreateDataflowEndpointGroupV2 for new implementations to take advantage of specialized endpoint types that support both uplink and downlink dataflows. This API only supports UplinkAwsGroundStationAgentEndpoint and DownlinkAwsGroundStationAgentEndpoint.

Migration considerations

If you're currently using DataflowEndpointGroup, you can continue using your existing configuration without changes. AWS Ground Station maintains full backward compatibility.

If you'd like to migrate to use the new DataflowEndpointGroupV2, and are currently using a DataflowEndpoint with a Dataflow Endpoint Application to receive your data, you'll need to migrate to use the AWS Ground Station Agent instead. If you're already using an AWS Ground Station Agent for downlink, you can use the same agent instance for uplink as well - no additional agent instances are required.

To migrate to DataflowEndpointGroupV2:

1. If migrating from DataflowEndpoint, set up the AWS Ground Station Agent following the AWS Ground Station Agent User Guide
2. Identify your dataflow direction and create the appropriate endpoint type (UplinkAwsGroundStationAgentEndpoint or DownlinkAwsGroundStationAgentEndpoint)
3. Create the DataflowEndpointGroupV2 referencing those endpoints
4. Create a new dataflow endpoint config that references the new DataflowEndpointGroupV2 by name
5. Create a new mission profile that references the dataflow endpoint config as a dataflow edge
6. Use the new mission profile to schedule contacts
7. Test your configuration before deploying to production

For more information about the complete workflow, see Understand AWS Ground Station Core components and Create configs.

Use AWS Ground Station Agent

The AWS Ground Station Agent enables you to receive (downlink) synchronous Wideband Digital Intermediate Frequency (DigIF) dataflows during AWS Ground Station contacts.

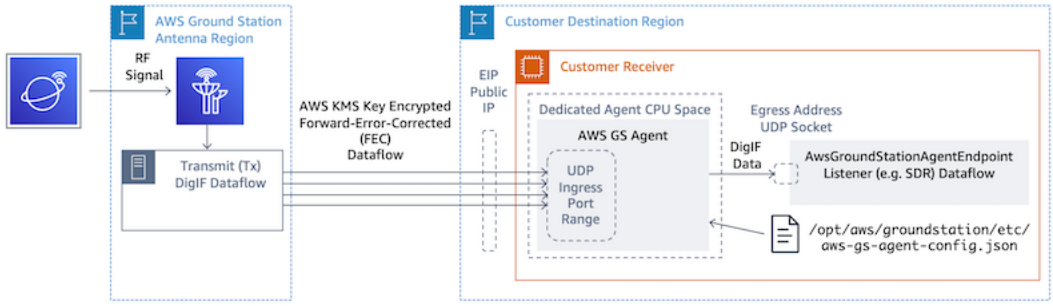
How it works

You can select two options for data delivery:

1. **Data delivery to an EC2 instance** - Data delivery to an EC2 instance that you own. You manage the AWS Ground Station Agent. This option may suit you best if you need near real-time data processing. See the Work with dataflows section for information about EC2 data delivery.
2. **Data delivery to an S3 bucket** - Data delivery to your AWS S3 bucket is fully managed by AWS Ground Station. See the Get started guide for information about S3 data delivery.

Both modes of data delivery require you to create a set of AWS resources. The use of CloudFormation to create your AWS resources is highly recommended to ensure reliability, accuracy, and supportability. Each contact can only deliver data to EC2 or S3 but not to both simultaneously.

The following diagram shows a DigIF dataflow from an AWS Ground Station Antenna Region to your EC2 instance with your Software-Defined Radio (SDR) or similar listener.



Additional information

For more detailed information, please see the full AWS Ground Station Agent User Guide.

Get started

Before you begin, you should familiarize yourself with the basic concepts in AWS Ground Station. For more information, see How AWS Ground Station works.

Below are the best practices for AWS Identity and Access Management (IAM) and what permissions you will need. After setting up the appropriate roles you can begin following the remainder of the steps.

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Add AWS Ground Station permissions to your AWS account

To use AWS Ground Station without requiring an administrative user, you need to create a new policy and attach it to your AWS account.

1. Sign in to the AWS Management Console and open the IAM console.
2. Create a new policy. Use the following steps:

   a. In the navigation pane, choose **Policies** and then choose **Create Policy**.

   b. In the **JSON** tab, edit the JSON with one of the following values. Use the JSON that works best for your application.

   - For Ground Station administrative privileges, set **Action** to **groundstation:*** as follows:

     JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "groundstation:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

   - For Read-only privileges, set **Action** to **groundstation:Get***, **groundstation:List***, and **groundstation:Describe*** as follows:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "groundstation:Get*",
        "groundstation:List*",
        "groundstation:Describe*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

- For additional security through multifactor authentication, set **Action** to **groundstation:***, and **Condition/Bool** to **aws:MultiFactorAuthPresent:true** as follows:

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "groundstation:*",
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "aws:MultiFactorAuthPresent": true
                }
            }
        }
    ]
}
```

3. In the IAM console, attach the policy you created to the desired user.

For more information about IAM users and attaching policies, see the IAM User Guide.

Onboard satellite

Onboarding a satellite into AWS Ground Station is a multistep process involving data collection, technical validation, spectrum licensing, with integration and testing. There are also non-disclosure agreements (NDAs) required.

Customer onboarding process overview

Satellite onboarding is a manual process that can be found on the Satellites and Resources section of the AWS Ground Station console page. The following describes the overall process.

1. Review the AWS Ground Station Locations section to determine if your satellite meets the geographical and radio frequency characteristics.
2. To start onboarding your satellite to AWS Ground Station, please submit a Satellite Onboarding Questionnaire on the Satellites and Resources section of the AWS Ground Station console page. Please include a brief summary of your mission and satellite needs, including your organization name, the frequencies required, when the satellites will be or were launched, the satellite's orbit type, and if you plan to use Use the AWS Ground Station digital twin feature.
3. Once your request is reviewed and approved, AWS Ground Station will apply for regulatory licensing at the specific locations you plan to use. The duration of this step will vary depending on the locations and any existing regulations.
4. After this approval is obtained, your satellite will be visible for you to use. AWS Ground Station will send you a notification of the successful update.

(Optional) Naming satellites

After onboarding, you may want to add a name to your satellite record to more easily recognize it. The AWS Ground Station console has the ability to display a user defined name for a satellite along with the Norad ID when using the Contacts page. Displaying the satellite name makes it much easier to select the correct satellite when scheduling. To do this, tags can be used.

Tagging AWS Ground Station Satellites can be done via the tag-resource API with the AWS CLI or one of the AWS SDKs. This guide will cover using the AWS Ground Station CLI to tag the public broadcast satellite Aqua (Norad ID 27424) in `us-west-2`.

AWS Ground Station CLI

The AWS CLI can be used to interact with AWS Ground Station. Before using AWS CLI to tag your satellites, the following AWS CLI prerequisites must be fulfilled:

- Ensure that AWS CLI is installed. For information about installing AWS CLI, see Installing the AWS CLI version 2.
- Ensure that AWS CLI is configured. For information about configuring AWS CLI, see Configuring the AWS CLI version 2.
- Save your frequently used configuration settings and credentials in files that are maintained by the AWS CLI. You need these settings and credentials to reserve and manage your AWS Ground Station contacts with AWS CLI. For more information about saving your configuration and credential settings, see Configuration and credential file settings.

Once AWS CLI is configured and ready to use, review the AWS Ground Station CLI Command Reference page to familiarize yourself with available commands. Follow the AWS CLI command structure when using this service and prefix your commands with `groundstation` to specify AWS Ground Station as the service you want to use. For more information on the AWS CLI command structure, see Command Structure in the AWS CLI page. An example command structure is provided below.

```
aws groundstation <command> <subcommand> [options and parameters]
```

Name a Satellite

First you need to get the ARN for the satellite(s) you wish to tag. This can be done via the list-satellites API in the AWS CLI:

```
aws groundstation list-satellites --region us-west-2
```

Running the above CLI command will return an output similar to this:

```
{
    "satellites": [
        {
            "groundStations": [
                "Ohio 1",
                "Oregon 1"
```

```
            ],
            "noradSatelliteID": 27424,
            "satelliteArn":
 "arn:aws:groundstation::111111111111:satellite/11111111-2222-3333-4444-555555555555",
            "satelliteId": "11111111-2222-3333-4444-555555555555"
        }
    ]
}
```

Find the satellite you wish to tag and note down the `satelliteArn`. One important caveat for tagging is that the tag-resource API requires a regional ARN, and the ARN returned by list-satellites is global. For the next step, you should augment the ARN with the region you would like to see the tag in (likely the region you schedule in). For this example, we are using `us-west-2`. With this change, the ARN will go from:

```
arn:aws:groundstation::111111111111:satellite/11111111-2222-3333-4444-555555555555
```

to:

```
arn:aws:groundstation:us-
west-2:111111111111:satellite/11111111-2222-3333-4444-555555555555
```

In order to show the satellite name in the console, the satellite must have a tag with "Name" as the key. Additionally, because we are using the AWS CLI, the quotation marks must be escaped with a backslash. The tag will look something like:

```
{\"Name\":\"AQUA\"}
```

Next, you will call the tag-resource API to tag the satellite. This can be done with the AWS CLI like so:

```
aws groundstation tag-resource --region us-west-2 --resource-arn
 arn:aws:groundstation:us-
west-2:111111111111:satellite/11111111-2222-3333-4444-555555555555 --tags
 '{"Name":"AQUA"}'
```

After doing this, you'll be able to see the name you set for the satellite in the AWS Ground Station console.

Change the Name For a Satellite

If you want to change the name for a satellite, you can simply call tag-resource with the satellite ARN again with the same "Name" key, but with a different value in the tag. This will update the existing tag and show the new name in the console. An example call for this looks like:

```
aws groundstation tag-resource --region us-west-2 --resource-arn
 arn:aws:groundstation:us-
west-2:111111111111:satellite/11111111-2222-3333-4444-555555555555 --tags
 '{"Name":"NewName"}'
```

Remove the Name For a Satellite

The name set for a satellite can be removed with the untag-resource API. This API needs the satellite ARN with the region the tag is in, and a list of tag keys. For the name, the tag key is "Name". An example call to this API using the AWS CLI looks like:

```
aws groundstation untag-resource --region us-west-2 --resource-arn
 arn:aws:groundstation:us-
west-2:111111111111:satellite/11111111-2222-3333-4444-555555555555 --tag-keys Name
```

Public broadcast satellites

In addition to onboarding your own satellites, you may request to onboard with supported public broadcast satellites that provide a publicly accessible downlink communication path. This enables you to use AWS Ground Station to downlink data from these satellites.

> **Note**
>
> You will not be able to uplink to these satellites. You will only be able to use the publicly accessible downlink communication paths.

AWS Ground Station supports onboarding of the following satellites to downlink direct broadcast data:

- Aqua
- SNPP

- JPSS-1/NOAA-20
- Terra

Once onboarded, these satellites can be accessed for immediate use. AWS Ground Station maintains a number of preconfigured CloudFormation templates to make getting started with the service easier. See Example mission profile configurations for examples of how AWS Ground Station can be used.

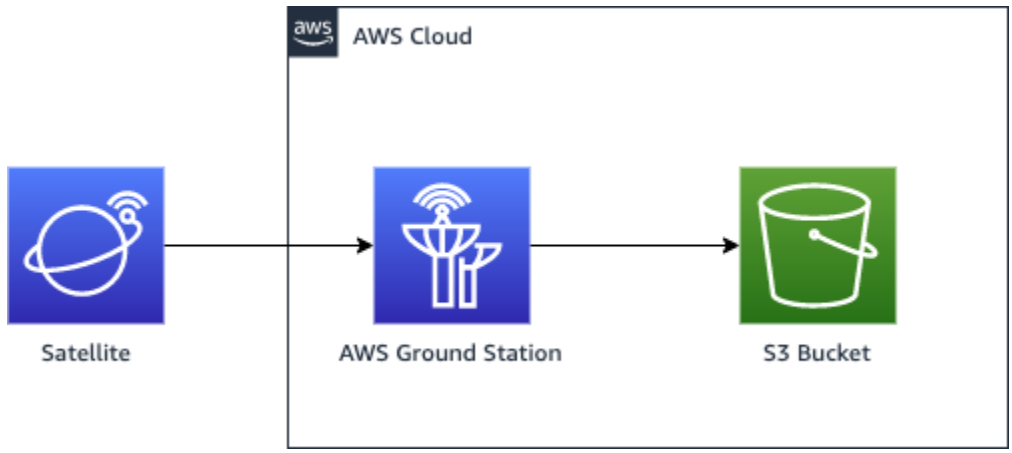
For more information about these satellites and the kind of data they transmit, see Aqua, JPSS-1/NOAA-20 and SNPP, and Terra.

Plan your dataflow communication paths

You have the choice between synchronous and asynchronous communication for each communication path on your satellite. Depending on your satellite and your use case, you may require one or both types. Synchronous communication paths allow for near real-time uplink as well as narrowband and wideband downlink operations. Asynchronous communication paths support narrowband and wideband downlink operations only.

Asynchronous data delivery

With data delivery to Amazon S3, your contact data is delivered asynchronously to an Amazon S3 bucket in your account. Your contact data is delivered as packet capture (pcap) files to allow replaying the contact data into a Software Defined Radio (SDR) or to extract the payload data from the pcap files for processing. The pcap files are delivered to your Amazon S3 bucket every 30 seconds as contact data is received by the antenna hardware to allow processing contact data during the contact if desired. Once received, you can process the data using your own post-processing software or use other AWS services like Amazon SageMaker AI or Amazon Rekognition. Data delivery to Amazon S3 is only available for downlinking data from your satellite; it is not possible to uplink data to your satellite from Amazon S3.



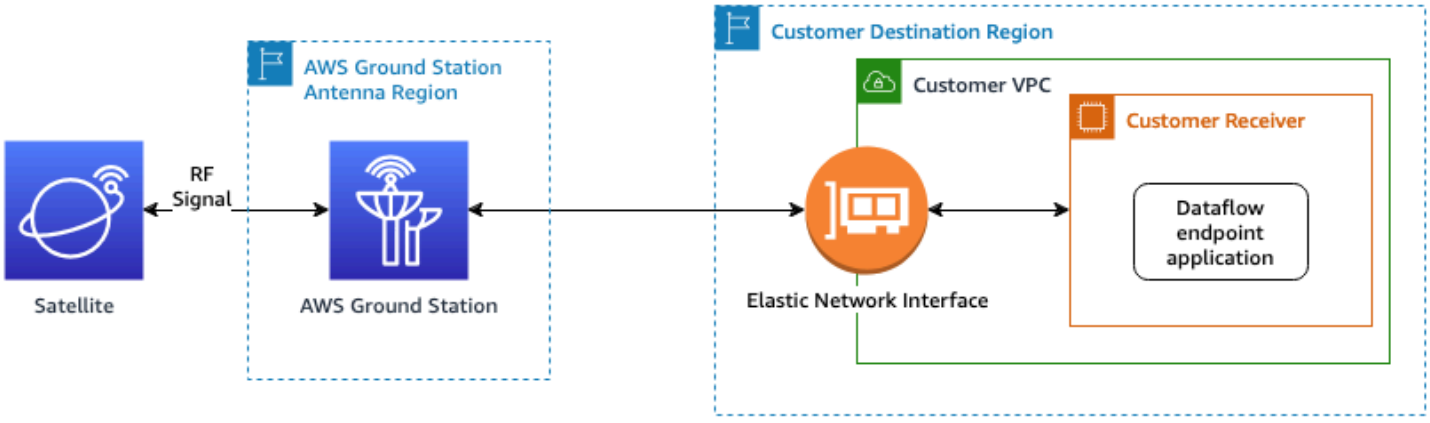
To utilize this path, you will use need to create an Amazon S3 bucket for AWS Ground Station to deliver the data into. In the next step, you'll also need to create a *S3 Recording Config* in the next step. Please reference the Amazon S3 Recording Config for restrictions on bucket naming and how to specify the naming convention used for your files.

Synchronous data delivery

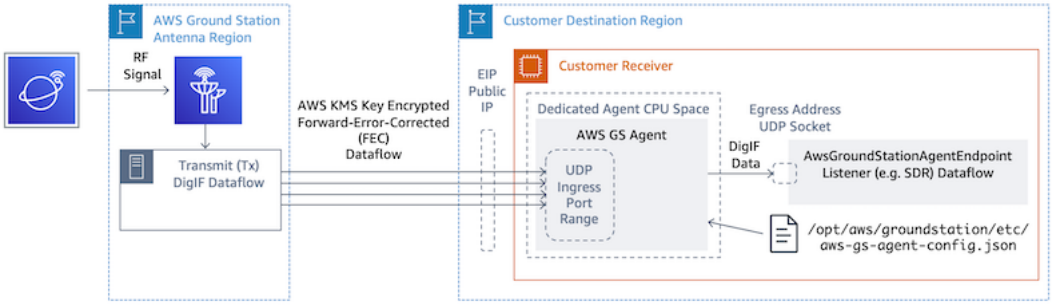
With data delivery to Amazon EC2, your contact data is streamed to and from your Amazon EC2 instance. You can process your data in real-time on your Amazon EC2 instance or forward the data for post-processing.

To utilize a synchronous path, you will use need to set up and configure your Amazon EC2 instances and create one or more *Dataflow Endpoint Groups*. To configure your Amazon EC2 instance reference the Set up and configure Amazon EC2. To create your Dataflow Endpoint Group, please reference the Use AWS Ground Station Dataflow endpoint groups.

The following shows the communication path if you are using the dataflow endpoint configuration.



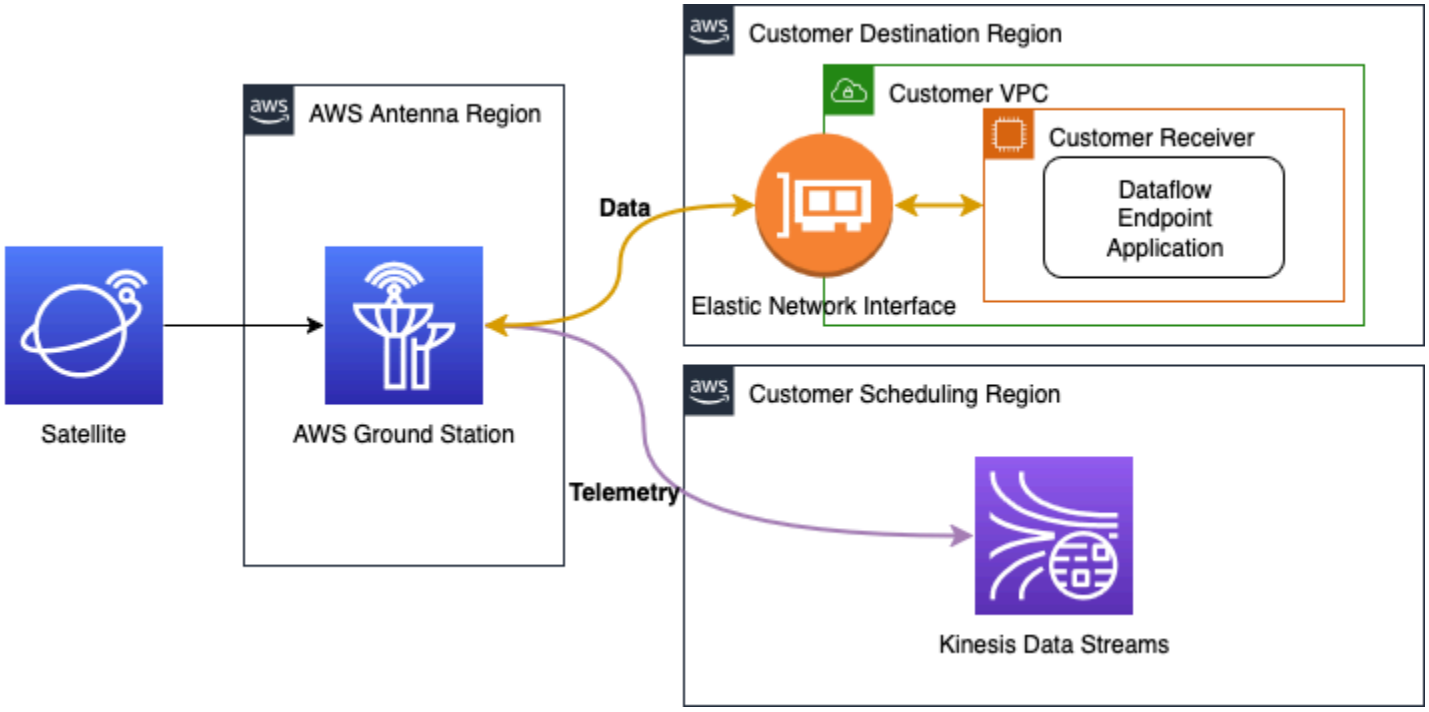
The following shows the communication path if you are using the AWS Ground Station Agent configuration.



Plan your telemetry

AWS Ground Station telemetry is an optional feature that streams metrics from AWS Ground Station antennas to your AWS account during satellite contacts. This allows you to monitor contact performance in near real-time and build custom monitoring solutions.

With AWS Ground Station telemetry, metrics from AWS Ground Station antennas are streamed directly to your account. Telemetry data begins streaming at contact start and continues throughout the contact duration. The telemetry data is delivered to your account in near real-time as it is sampled from the antenna hardware. Once received, you can process the data using your own post-processing software or use other AWS services like Amazon Data Firehose or AWS Lambda.



In the next step, you'll create the configs needed for your mission profile. If you want to enable telemetry, you'll create a *Telemetry Sink Config* in addition to your tracking config and dataflow configs. For detailed setup instructions, see Set up telemetry.

For more information about TelemetrySinkConfig, see Telemetry Sink Config.

Create configs

By this step you have identified the satellite, the communication paths, and the IAM, Amazon EC2, and Amazon S3 resources as needed. In this step you will create AWS Ground Station *configs* that store their respective parameters.

Data delivery configs

The first configs to create relate to where and how you want data delivered. Using the information from the previous step you will construct many of the following configuration types.

- **Amazon S3 Recording Config** - Deliver data to your Amazon S3 bucket.
- **Dataflow Endpoint Config** - Deliver data to your Amazon EC2 instance.

Telemetry config (optional)

If you want to receive near real-time telemetry during your contacts, you can create a TelemetrySinkConfig. This config is optional and specifies where AWS Ground Station will deliver telemetry data.

- **Telemetry Sink Config** - Deliver telemetry data to your account.

For detailed setup instructions, see Set up telemetry.

Satellite configs

The satellite configs relate how AWS Ground Station can communicate with your satellite. You will reference the information you gathered in Onboard satellite.

- **Tracking Config** - Sets preference for how your vehicle is physically tracked during a contact. This is required for mission profile construction.
- **Antenna Downlink Config** - Deliver digitized radio frequency data.

- **Antenna Downlink Demod Decode Config** - Deliver demodulated and decoded radio frequency data.
- **Antenna Uplink Config** - Uplink data to your satellite.
- **Antenna Uplink Echo Config** - Deliver an echo of your uplink signal data.

Create mission profile

With the *configs* constructed in the previous step, you have identified how to track your satellite, the possible ways to communicate with your satellite, and how to enable near real-time telemetry during contact execution. In this step you will construct one or more mission profiles. A mission profile represents the aggregation of the possible *configs* into an expected behavior that can be then scheduled and operated on.

For the latest parameters, please reference the AWS::GroundStation::MissionProfile CloudFormation resource type

1. Name your mission profile. This allows you to quickly understand its usage within your system. For example, you may have a *satellite-wideband-narrowband-nominal-operations* and a *satellite-narrowband-emergency-operations* if you have a separate narrowband carrier for emergency operations.
2. Set your tracking config.
3. Set your minimum viable contact durations. This allows you to filter potential contacts to meet your mission needs.
4. Set your *streamsKmsKey* and *streamsKmsRole* that are used to encrypt your data during transit. This is used for all AWS Ground Station Agent dataflows.
5. Set your dataflows. Create your dataflows to match your carrier signals using the configs you created in the previous step.
6. [Optional] Set your pre-pass and post-pass contact duration seconds. This is used to emit per-contact events prior-to and after the contact, respectively. See Automate AWS Ground Station with Events for more information.
7. [Optional] Set your *telemetrySinkConfigArn* to enable telemetry during contacts. This allows you to receive near real-time telemetry directly in your account for monitoring and analysis. See Work with telemetry for more information.
8. [Optional] You can associate Tags to your mission profile. These can be used to help programmatically differentiate your mission profiles.

You can reference the Example mission profile configurations, to see just some of the potential configurations.

Understand next steps

Now that you have an onboarded satellite and a valid mission profile, you are ready to schedule contacts and communicate with your satellite with AWS Ground Station.

You can schedule a contact in one of the following ways:

- The AWS Ground Station console.
- The AWS CLI reserve-contact command.
- The AWS SDK. ReserveContact API.

For information about how AWS Ground Station tracks the trajectory of your satellite and how that information is used, please reference Understand how AWS Ground Station uses ephemerides.

AWS Ground Station maintains a number of preconfigured CloudFormation templates to make getting started with the service easier. See Example mission profile configurations for examples of how AWS Ground Station can be used.

Processing the digital intermediate frequency data, or the demodulated and decoded data provided to you from AWS Ground Station will depend on your specific use case. The following blog posts can help you to understand some of the options available to you:

- Automated Earth observation using AWS Ground Station Amazon S3 data delivery (and it's associated GitHub repository awslabs/aws-groundstation-eos-pipeline)
- Virtualizing the satellite ground segment with AWS
- Earth observation using AWS Ground Station: A how to guide
- Building high-throughput satellite data downlink architectures with AWS Ground Station WideBand DigIF and Amphinicy Blink SDR (and it's associated GitHub repository aws-samples/aws-groundstation-wbdigif-snpp)

AWS Ground Station Locations

AWS Ground Station provides a global network of ground stations in close proximity to our global network of AWS infrastructure regions. You can configure your use of these locations from any supported AWS Region. This includes the AWS Region in which data is delivered.

Finding the AWS region for a ground station location

The AWS Ground Station global network includes ground station locations that are not physically located in the AWS Region to which they are connected. The list of ground stations that you have access to can be retrieved via the AWS SDK ListGroundStation response. The full list of ground station locations is presented below, with more coming soon. Please refer to the onboarding guide to add or modify site approvals for your satellites.

Ground Station Name	Ground Station Location	AWS Region Name	AWS Region Code	Notes
Alaska 1	Alaska, USA	US West (Oregon)	us-west-2	Not physicall y located in an AWS region

Ground Station Name	Ground Station Location	AWS Region Name	AWS Region Code	Notes
Bahrain 1	Bahrain	Middle East (Bahrain)	me-south-1	
Cape Town 1	Cape Town, South Africa	Africa (Cape Town)	af-south-1	
Dubbo 1	Dubbo, Australia	Asia Pacific (Sydney)	ap-southeast-2	Not physicall y located in an AWS region
Hawaii 1	Hawaii, USA	US West (Oregon)	us-west-2	Not physicall y located in an AWS region
Ireland 1	Ireland	Europe (Ireland)	eu-west-1	
Ohio 1	Ohio, USA	US East (Ohio)	us-east-2	
Oregon 1	Oregon, USA	US West (Oregon)	us-west-2	
Punta Arenas 1	Punta Arenas, Chile	South America (São Paulo)	sa-east-1	Not physicall y located in an AWS region
Seoul 1	Seoul, South Korea	Asia Pacific (Seoul)	ap-northeast-2	
Singapore 1	Singapore	Asia Pacific (Singapore)	ap-southeast-1	
Stockholm 1	Stockholm, Sweden	Europe (Stockholm)	eu-north-1	

AWS Ground Station supported AWS regions

You can deliver data and configure your **Contacts** via the AWS SDK or the AWS Ground Station console from supported AWS Regions. You can view the supported regions and their associated endpoints at the AWS Ground Station endpoints and quotas.

Digital twin availability

Use the AWS Ground Station digital twin feature is available in all AWS Regions where AWS Ground Station is available. Digital twin ground stations are exact copies of production ground stations with a modifying prefix to Ground Station Name of “Digital Twin ”. For example, "Digital Twin Ohio 1" is a digital twin ground station that is an exact copy of the "Ohio 1" production ground station.

Dedicated Antennas

In addition to the publicly available ground station locations listed above, AWS Ground Station offers Dedicated Antennas. A Dedicated Antenna is a custom-built antenna system that AWS manages on your behalf. A Dedicated Antenna is not restricted to existing AWS Ground Station ground station locations and can be built with capabilities beyond those of public ground stations as described in AWS Ground Station Site Capabilities. The locations and capabilities of Dedicated Antennas are not publicly disclosed.

For more information about Dedicated Antennas, see AWS Ground Station Dedicated Antennas. To learn more or to get started with Dedicated Antennas, contact AWS Support through the AWS Support Center Console.

Viewing antennas at a ground station

Each ground station location has one or more antennas. You can view the antennas at a ground station by using the ListAntennas API. This API returns the antennas at a specified ground station, including each antenna's name.

Antenna information is useful when combined with the ListGroundStationReservations API to understand capacity and availability at a ground station. For more information about viewing reservations, see View ground station reservations.

To call `ListAntennas`, you must have a satellite onboarded to the ground station or have azimuth elevation ephemeris permissions for the ground station. For more information, see Provide azimuth elevation ephemeris data.

Example: List antennas at a ground station

The following example lists all antennas at a ground station using the AWS SDK for Python (Boto3).

```
import boto3

# Create AWS Ground Station client
ground_station_client = boto3.client("groundstation")

# The ground station ID to list antennas for.
# Use the ListGroundStations API to find available ground station IDs.
ground_station_id = "Ohio 1"

# List all antennas at a ground station.
# This is useful for understanding the capacity of a ground station
# and for planning multi-antenna operations.
print(f"Listing antennas for ground station '{ground_station_id}'...")

paginator = ground_station_client.get_paginator("list_antennas")
page_iterator = paginator.paginate(
    groundStationId=ground_station_id,
    PaginationConfig={
        "MaxItems": 100,
        "PageSize": 20,
    },
)

for page in page_iterator:
    for antenna in page["antennaList"]:
        print(f"  Antenna: {antenna['antennaName']}")
        print(f"    Ground Station: {antenna['groundStationName']}")
        print(f"    Region: {antenna['region']}")
        print()
```

View ground station reservations

You can view reservations across antennas at a ground station by using the ListGroundStationReservations API. Reservations represent time blocks on antennas, including your scheduled contacts. AWS Ground Station Dedicated Antennas customers also see maintenance windows.

This information helps you understand antenna availability when planning contact schedules and provides visibility into what is happening on the antennas at a ground station.

Listing reservations

To list reservations, call ListGroundStationReservations with a ground station identifier and a time range. The API returns reservations across all antennas at the ground station within the specified time window.

The reservations you see depend on your access level:

- **Public AWS Ground Station customers** — You can see only your own contact reservations. Maintenance windows and contacts owned by other accounts are not included.
- **AWS Ground Station Dedicated Antennas customers** — You can see all reservations on your Dedicated Antennas, including maintenance windows and contacts scheduled by other accounts. Contact identifiers are only included for contacts that you own. For more information, see AWS Ground Station Dedicated Antennas.

Reservation types

Each reservation has a type that indicates what the antenna time is being used for:

- **Contact** — A contact reservation represents antenna time reserved for satellite communication. The reservation start and end times reflect the full antenna reservation, including pre-pass and post-pass time, not just the satellite pass window.
- **Maintenance** — A maintenance reservation represents a time period when the antenna is unavailable due to maintenance. Maintenance reservations include a `maintenanceType` that indicates whether the maintenance was planned or unplanned.

Code example

The following example lists reservations at a ground station for the next 7 days using the AWS SDK for Python (Boto3), including filtering by reservation type.

```
import boto3
from datetime import datetime, timezone, timedelta

# Create AWS Ground Station client
ground_station_client = boto3.client("groundstation")

# The ground station ID to list reservations for
ground_station_id = "Ohio 1"

# Define the time range to query. Reservations include both your
# scheduled contacts and maintenance windows at the ground station.
start_time = datetime.now(timezone.utc)
end_time = start_time + timedelta(days=7)

# List all reservations at a ground station for the next 7 days.
# You can filter by reservation type to see only contacts or
# only maintenance windows.
print(f"Listing reservations for ground station '{ground_station_id}'...")
print(f"Time range: {start_time} to {end_time}")

paginator = ground_station_client.get_paginator("list_ground_station_reservations")
page_iterator = paginator.paginate(
    groundStationId=ground_station_id,
    startTime=start_time,
    endTime=end_time,
    PaginationConfig={
        "MaxItems": 100,
        "PageSize": 20,
    },
)

for page in page_iterator:
    for reservation in page["reservationList"]:
        reservation_type = reservation["reservationType"]
        antenna_name = reservation["antennaName"]
        res_start = reservation["startTime"]
        res_end = reservation["endTime"]
```

```
        print(f"  Type: {reservation_type}")
        print(f"    Antenna: {antenna_name}")
        print(f"    Start: {res_start}")
        print(f"    End: {res_end}")

        details = reservation["reservationDetails"]
        if "contact" in details:
            contact_id = details["contact"].get("contactId", "N/A")
            print(f"    Contact ID: {contact_id}")
        elif "maintenance" in details:
            maintenance_type = details["maintenance"]["maintenanceType"]
            print(f"    Maintenance Type: {maintenance_type}")

        print()

# For Dedicated Antenna customers, you can also filter to show only maintenance windows
print("Listing only maintenance reservations...")

page_iterator = paginator.paginate(
    groundStationId=ground_station_id,
    startTime=start_time,
    endTime=end_time,
    reservationTypes=["MAINTENANCE"],
    PaginationConfig={
        "MaxItems": 100,
        "PageSize": 20,
    },
)

for page in page_iterator:
    for reservation in page["reservationList"]:
        maintenance_type = reservation["reservationDetails"]["maintenance"][
            "maintenanceType"
        ]
        print(
            f"  {maintenance_type} maintenance on {reservation['antennaName']}: "
            f"{reservation['startTime']} to {reservation['endTime']}"
        )
```

AWS Ground Station site masks

Each AWS Ground Station antenna location has associated site masks. These masks block antennas at that location from transmitting or receiving when pointing in some directions, typically close to the horizon. The masks may take into account:

- **Features of the geographic terrain surrounding the antenna** – For example, this includes things like mountains or buildings, that would block a radio frequency (RF) signal or prevent transmitting.
- **Radio Frequency Interference (RFI)** – This affects both the ability to receive (external RFI sources impacting a downlink signal into AWS Ground Station antennas) and transmit (the RF signal transmitted by AWS Ground Station antennas adversely impacting external receivers).
- **Legal authorizations** – Local site authorizations to operate AWS Ground Station in each region may include specific restrictions, such as a minimum elevation angle for transmitting.

These site masks may change over time. For example, new buildings could be constructed near an antenna location, RFI sources may change, or legal authorization may be renewed with different restrictions. The AWS Ground Station site masks are available to you under a non-disclosure agreement (NDA).

Customer-specific masks

In addition to the AWS Ground Station site masks at each site, you may have additional masks due to restrictions on your own legal authorization to communicate with your satellites in a given region. Such masks can be configured in AWS Ground Station on a case-by-case basis to ensure compliance when using AWS Ground Station to communicate with these satellites. Contact the AWS Ground Station team for details.

Impact of site masks on available contact times

There are two kinds of site masks: uplink (transmit) site masks, and downlink (receive) site masks.

When listing available contact times using the ListContacts operation, AWS Ground Station will return visibility times based on when your satellite will rise above and set below the downlink mask. Available contact times are based on this downlink mask visibility window. This ensures that you do not reserve time when your satellite is below the downlink mask.

Uplink site masks are *not* applied to the available contact times, even if the Mission Profile includes an Antenna Uplink Config in a dataflow edge. This allows you to use all available contact time for downlink, even if uplink may not be available for portions of that time due to the uplink site mask. However, the uplink signal may not be transmitted for some or all of the time reserved for a satellite contact. You are responsible for accounting for the provided uplink mask when scheduling uplink transmissions.

The portion of a contact that is unavailable for uplink varies depending on the satellite trajectory during the contact, relative to the uplink site mask at the antenna location. In regions where the uplink and downlink site masks are similar, this duration will typically be short. In other regions, where the uplink mask may be considerably higher than the downlink site mask, this could result in significant portions, or even all, of the contact duration being unavailable for uplink. The full contact time is billed to you, even if portions of the reserved time are unavailable for uplink.

AWS Ground Station Site Capabilities

To simplify your experience, AWS Ground Station determines a common set of capabilities for an antenna type and then deploys multiple antenna to a ground station location. Part of the onboarding steps ensures your satellite is compatible with the antenna types at a specific location. When you reserve a contact, you indirectly determine the antenna type used. This ensures your experience at a particular ground station location remains the same over time regardless of which antennas are being used. The specific performance of your contact will vary due to a wide variety of environmental concerns such as weather at the site.

Currently, all sites support the following capabilities:

> **Note**
>
> Each row in the following table indicates an independent communication path, unless otherwise indicated. Duplicate rows exist to reflect our multi-channel capabilities that allow multiple communication paths to be used concurrently.

Capability Type	Frequency Range	Bandwidth Range	Polarization	Common Name	Notes
antenna-downlink	7750 - 8500 MHz	50 - 400 MHz	RHCP	X-band wideband downlink	This capability requires the use of the AWS Ground Station Agent. This capability is not supported in Alaska 1 or Punta Arenas 1. The aggregate bandwidth must not exceed 400MHz per polarization at each location. All utilized frequency ranges must be non-overlapping.
antenna-downlink	7750 - 8500 MHz	50 - 400 MHz	RHCP		
antenna-downlink	7750 - 8500 MHz	50 - 400 MHz	RHCP		
antenna-downlink	7750 - 8500 MHz	50 - 400 MHz	RHCP		
antenna-downlink	7750 - 8500 MHz	50 - 400 MHz	RHCP		
antenna-downlink	7750 - 8500 MHz	50 - 400 MHz	LHCP		
antenna-downlink	7750 - 8500 MHz	50 - 400 MHz	LHCP		
antenna-downlink	7750 - 8500 MHz	50 - 400 MHz	LHCP		
antenna-downlink	7750 - 8500 MHz	50 - 400 MHz	LHCP		
antenna-downlink	7750 - 8500 MHz	50 - 400 MHz	LHCP		
antenna-downlink	2200 - 2290 MHz	Up to 40 MHz	RHCP	S-band downlink	Only one polarization can be used at a time
antenna-downlink	2200 - 2290 MHz	Up to 40 MHz	LHCP		

Capability Type	Frequency Range	Bandwidth Range	Polarization	Common Name	Notes
antenna-downlink	7750 - 8500 MHz	Up to 40 MHz	RHCP	X-band narrowband downlink	Only one polarization can be used at a time
antenna-downlink	7750 - 8500 MHz	Up to 40 MHz	LHCP		
antenna-uplink	2025 - 2110 MHz	Up to 40 MHz	RHCP	S-band uplink	Only one polarization can be used at a time EIRP 20-50 dBW
antenna-uplink	2025 - 2110 MHz	Up to 40 MHz	LHCP		
antenna-uplink-echo	2025 - 2110 MHz	2 MHz	RHCP	Uplink echo	Matches antenna-uplink restrictions
antenna-uplink-echo	2025 - 2110 MHz	2 MHz	LHCP		
antenna-downlink-demod-decode	7750 - 8500 MHz	Up to 500 MHz	RHCP	X-band demodulated and decoded downlink	
antenna-downlink-demod-decode	7750 - 8500 MHz	Up to 500 MHz	LHCP		
tracking	N/A	N/A	N/A	N/A	Support for auto-tracking and program tracking

* RHCP = right-handed circular polarization, and LHCP = left-handed circular polarization. For more information on polarization, see Circular polarization.

Understand how AWS Ground Station uses ephemerides

An ephemeris, plural ephemerides, is a file or data structure providing the trajectory of astronomical objects. Historically, this file only referred to tabular data but, gradually, it has come to direct to a wide variety of data files indicating a spacecraft trajectory.

The Ephemeris API allows custom ephemerides to be uploaded to AWS Ground Station for use with a satellite. These ephemerides override the default ephemerides from Space-Track (see: Default ephemeris data). We support receiving ephemeris data in Orbit Ephemeris Message (OEM), two-line element (TLE), and azimuth elevation formats.

AWS Ground Station uses ephemeris data to determine when contacts become available based on the provided ephemeris and correctly command antennas in the AWS Ground Station network. By default, no action is required to provide AWS Ground Station with ephemerides if your satellite has an assigned NORAD ID.

Uploading custom ephemerides can improve the quality of tracking, handle early operations where no Space-Track ephemerides are available to AWS Ground Station, and account for maneuvers.

Alternatively, AWS Ground Station supports an azimuth elevation format, which allow you to directly specify antenna pointing directions without providing satellite orbital information. This is useful for scenarios where precise antenna pointing is required because satellite trajectory information is imprecise or unknown.

Topics

Default ephemeris data

By default, AWS Ground Station uses publicly available data from Space-Track, and no action is required to supply AWS Ground Station with these default ephemerides. These ephemerides are

two-line element sets (TLEs) associated with your satellite's NORAD ID. All default ephemerides have a priority of 0. As a result, they will be overridden, always, by any non-expired, custom ephemerides uploaded via the ephemeris API, which must always have a priority of 1, or greater.

Satellites without a NORAD ID must upload custom ephemeris data to AWS Ground Station. For example, satellites that have just launched or that are intentionally omitted from the Space-Track catalog would have no NORAD ID and would need to have custom ephemerides uploaded. For more information on providing custom ephemeris data, see: Providing Custom Ephemeris Data.

Provide custom ephemeris data

> **Important**
>
> The ephemeris API is currently in a Preview state

Access to the Ephemeris API is provided only on an as-needed basis. If you require the ability to upload custom ephemeris data, please open an AWS Support ticket through the AWS Support Center Console. Our team will work with you to enable this capability for your specific requirements.

Overview

The Ephemeris API allows custom ephemerides to be uploaded to AWS Ground Station for use with a satellite. These ephemerides override the default ephemerides from Space-Track (see: Default ephemeris data). We support receiving ephemeris data in Orbit Ephemeris Message (OEM), two-line element (TLE), and azimuth elevation formats.

AWS Ground Station treats ephemerides as Individualized Usage Data. If you use this optional feature, AWS will use your ephemeris data to provide troubleshooting support.

Uploading custom ephemerides can improve the quality of tracking, handle operations where no Space-Track ephemerides are available to AWS Ground Station, and account for maneuvers.

To troubleshoot an invalid ephemeris see: Troubleshoot invalid ephemerides

Example: Using customer-provided ephemerides with AWS Ground Station

For more detailed instructions for using customer-provided ephemerides with AWS Ground Station, see Using customer-provided ephemerides with AWS Ground Station and it's associated GitHub repository aws-samples/aws-groundstation-cpe.

Provide TLE ephemeris data

> **Important**
>
> The ephemeris API is currently in a Preview state

Access to the Ephemeris API is provided only on an as-needed basis. If you require the ability to upload custom ephemeris data, please open an AWS Support ticket through the AWS Support Center Console. Our team will work with you to enable this capability for your specific requirements.

Overview

Two-line element (TLE) sets are a standardized format for describing satellite orbits. The Ephemeris API allows TLE ephemerides to be uploaded to AWS Ground Station for use with a satellite. These ephemerides override the default ephemerides from Space-Track (see: Default ephemeris data).

AWS Ground Station treats ephemerides as Individualized Usage Data. If you use this optional feature, AWS will use your ephemeris data to provide troubleshooting support.

Uploading custom TLE ephemerides can improve the quality of tracking, handle early operations where no Space-Track ephemerides are available to AWS Ground Station, and account for maneuvers.

> **Note**
>
> When providing custom ephemeris before a satellite catalog number is assigned for your satellite, you can use `00000` for the satellite catalog number field of the TLE, and `000` for the launch number portion of the international designator field of the TLE (e.g. `24000A` for a vehicle launched in 2024).

For more information about the format of TLEs, see Two-line element set.

Creating a TLE ephemeris

A TLE ephemeris can be created using the CreateEphemeris action in the AWS Ground Station API. This action will upload an ephemeris using data either in the request body or from a specified S3 bucket.

It is important to note that uploading an ephemeris sets the ephemeris to `VALIDATING` and starts an asynchronous workflow that will validate and generate potential contacts from your ephemeris. Only once an ephemeris has passed this workflow and become `ENABLED` will it be used for contacts. You should poll DescribeEphemeris for the ephemeris status or use CloudWatch events to track the ephemeris' status changes.

To troubleshoot an invalid ephemeris see: Troubleshoot invalid ephemerides

Example: Create a two-line element (TLE) set ephemeris via API

The AWS SDKs, and CLI can be used to upload a two line element (TLE) set ephemeris to AWS Ground Station via the CreateEphemeris call. This ephemeris will be used in place of the default ephemeris data for a satellite (see Default ephemeris data). This example shows how to do this using the AWS SDK for Python (Boto3).

A TLE set is a JSON formatted object that strings one or more TLEs together to construct a continuous trajectory. The TLEs in the TLE set must form a continuous set that we can use to construct a trajectory (i.e. no gaps in time between TLEs in a TLE set). An example TLE set is shown below:

```
[
    {
        "tleLine1": "1 25994U 99068A   20318.54719794  .00000075  00000-0  26688-4 0
 9997",
        "tleLine2": "2 25994  98.2007  30.6589 0001234  89.2782  18.9934
 14.57114995111906",
        "validTimeRange": {
            "startTime": 12345,
            "endTime": 12346
        }
    },
    {
```

```
        "tleLine1": "1 25994U 99068A   20318.54719794  .00000075  00000-0  26688-4 0
 9997",
        "tleLine2": "2 25994  98.2007  30.6589 0001234  89.2782  18.9934
 14.57114995111906",
        "validTimeRange": {
            "startTime": 12346,
            "endTime": 12347
        }
    }
]
```

> **Note**
>
> The time ranges of the TLEs in a TLE set must match up exactly to be a valid, continuous trajectory.

A TLE set can be uploaded via the AWS Ground Station boto3 client as follows:

```
import boto3
from datetime import datetime, timedelta, timezone

# Create AWS Ground Station client
ground_station_client = boto3.client("groundstation")

# Create TLE ephemeris
tle_ephemeris = ground_station_client.create_ephemeris(
    name="Example Ephemeris",
    satelliteId="2e925701-9485-4644-b031-EXAMPLE01",
    enabled=True,
    expirationTime=datetime.now(timezone.utc) + timedelta(days=3),
    priority=2,
    ephemeris={
        "tle": {
            "tleData": [
                {
                    "tleLine1": "1 25994U 99068A   20318.54719794  .00000075  00000-0
 26688-4 0  9997",
                    "tleLine2": "2 25994  98.2007  30.6589 0001234  89.2782  18.9934
 14.57114995111906",
                    "validTimeRange": {
                        "startTime": datetime.now(timezone.utc),
```

```
                        "endTime": datetime.now(timezone.utc) + timedelta(days=7),
                    },
                }
            ]
        }
    },
)

print(f"Created TLE ephemeris with ID: {tle_ephemeris['ephemerisId']}")
```

This call will return an ephemerisId that can be used to reference the ephemeris in the future. For example, we can use the provided ephemerisId from the call above to poll for the status of the ephemeris:

```
import boto3
from datetime import datetime, timedelta, timezone
import time

# Create AWS Ground Station client
ground_station_client = boto3.client("groundstation")

# First, create a TLE ephemeris
print("Creating TLE ephemeris...")

tle_ephemeris = ground_station_client.create_ephemeris(
    name="Example TLE Ephemeris for Description",
    satelliteId="2e925701-9485-4644-b031-EXAMPLE01",
    enabled=True,
    expirationTime=datetime.now(timezone.utc) + timedelta(days=3),
    priority=2,
    ephemeris={
        "tle": {
            "tleData": [
                {
                    "tleLine1": "1 25994U 99068A   20318.54719794  .00000075  00000-0
 26688-4 0  9997",
                    "tleLine2": "2 25994  98.2007  30.6589 0001234  89.2782  18.9934
 14.57114995111906",
                    "validTimeRange": {
                        "startTime": datetime.now(timezone.utc),
                        "endTime": datetime.now(timezone.utc) + timedelta(days=7),
                    },
                }
```

```
            ]
        }
    },
)

ephemeris_id = tle_ephemeris["ephemerisId"]
print(f"Created TLE ephemeris with ID: {ephemeris_id}")

# Describe the ephemeris immediately to check initial status
print("Describing ephemeris...")

response = ground_station_client.describe_ephemeris(ephemerisId=ephemeris_id)

print(f"Ephemeris ID: {response['ephemerisId']}")
print(f"Name: {response['name']}")
print(f"Status: {response['status']}")
```

An example response from the DescribeEphemeris action is provided below

```
{
  "creationTime": 1620254718.765,
  "enabled": true,
  "name": "Example Ephemeris",
  "ephemerisId": "fde41049-14f7-413e-bd7b-EXAMPLE01",
  "priority": 2,
  "status": "VALIDATING",
  "suppliedData": {
    "tle": {
      "ephemerisData": "[{\"tleLine1\": \"1 25994U 99068A   20318.54719794  .00000075
 00000-0  26688-4 0  9997\",\"tleLine2\": \"2 25994  98.2007  30.6589 0001234  89.2782
  18.9934 14.57114995111906\",\"validTimeRange\": {\"startTime\": 1620254712000,
\"endTime\": 1620859512000}}]"
    }
  }
}
```

It is recommended to poll the DescribeEphemeris route or use CloudWatch events to track the status of the uploaded ephemeris as it must go through an asynchronous validation workflow before it is set to ENABLED and becomes usable for scheduling and executing contacts.

Note that the NORAD ID in all TLEs in the TLE set, 25994 in the examples above, must match the NORAD ID your satellite has been assigned in the Space-Track database.

Example: Uploading TLE ephemeris data from an S3 bucket

It is also possible to upload a TLE ephemeris file directly from an S3 bucket by pointing to the bucket and object key. AWS Ground Station will retrieve the object on your behalf. Information about the encryption of data at rest in AWS Ground Station is detailed in: Data encryption at rest for AWS Ground Station.

Below is an example of uploading a TLE ephemeris file from an S3 bucket

```
import boto3
from datetime import datetime, timedelta, timezone
import json

# Create AWS clients
s3_client = boto3.client("s3")
ground_station_client = boto3.client("groundstation")

# Define S3 bucket and key
bucket_name = "ephemeris-bucket"
object_key = "test_data.tle"

# Create sample TLE set data
# Note: For actual satellites, use real TLE data from sources like Space-Track
tle_set_data = [
    {
        "tleLine1": "1 25994U 99068A   20318.54719794  .00000075  00000-0  26688-4 0
 9997",
        "tleLine2": "2 25994  98.2007  30.6589 0001234  89.2782  18.9934
 14.57114995111906",
        "validTimeRange": {
            "startTime": datetime.now(timezone.utc),
            "endTime": datetime.now(timezone.utc) + timedelta(days=3),
        },
    },
    {
        "tleLine1": "1 25994U 99068A   20321.54719794  .00000075  00000-0  26688-4 0
 9998",
        "tleLine2": "2 25994  98.2007  33.6589 0001234  89.2782  18.9934
 14.57114995112342",
        "validTimeRange": {
            "startTime": datetime.now(timezone.utc) + timedelta(days=3),
            "endTime": datetime.now(timezone.utc) + timedelta(days=7),
        },
```

```
    },
]

# Convert to JSON string for upload
tle_json = json.dumps(tle_set_data, indent=2)

# Upload sample TLE data to S3
print(f"Uploading TLE set data to s3://{bucket_name}/{object_key}")

s3_client.put_object(
    Bucket=bucket_name, Key=object_key, Body=tle_json, ContentType="application/json"
)
print("TLE set data uploaded successfully to S3")
print(f"Uploaded {len(tle_set_data)} TLE entries covering 7 days")

# Create TLE ephemeris from S3
print("Creating TLE ephemeris from S3...")

s3_tle_ephemeris = ground_station_client.create_ephemeris(
    name="2022-11-05 S3 TLE Upload",
    satelliteId="fde41049-14f7-413e-bd7b-EXAMPLE01",
    enabled=True,
    expirationTime=datetime.now(timezone.utc) + timedelta(days=5),
    priority=2,
    ephemeris={"tle": {"s3Object": {"bucket": bucket_name, "key": object_key}}},
)

print(f"Created TLE ephemeris with ID: {s3_tle_ephemeris['ephemerisId']}")
```

Provide OEM ephemeris data

> **Important**
>
> The ephemeris API is currently in a Preview state

Access to the Ephemeris API is provided only on an as-needed basis. If you require the ability to upload custom ephemeris data, please open an AWS Support ticket through the AWS Support Center Console. Our team will work with you to enable this capability for your specific requirements.

Overview

Orbit Ephemeris Message (OEM) is a standardized format for representing spacecraft trajectory data. The Ephemeris API allows OEM ephemerides to be uploaded to AWS Ground Station for use with a satellite. These ephemerides override the default ephemerides from Space-Track (see: Default ephemeris data).

AWS Ground Station treats ephemerides as Individualized Usage Data. If you use this optional feature, AWS will use your ephemeris data to provide troubleshooting support.

Uploading custom OEM ephemerides can improve the quality of tracking, handle early operations where no Space-Track ephemerides are available to AWS Ground Station, and account for maneuvers.

> **Note**
>
> When providing custom ephemeris before a satellite catalog number is assigned for your satellite, you can use `satelliteId` for the `OBJECT_ID` portion of the OEM.
> For more information about the format of OEMs, see OEM ephemeris format.

OEM ephemeris format

AWS Ground Station processes OEM Customer Provided Ephemerides according to the CCSDS standard with some extra restrictions. OEM files should be in KVN format. The following table outlines the different fields in an OEM and how AWS Ground Station differs from the CCSDS standard.

Section	Field	CCSDS required	AWS Ground Station required	Notes
Header	CCSDS_OEM _VERS	Yes	Yes	Required value: 2.0
	COMMENT	No	No	
	CLASSIFICATION	No	No	
	CREATION_DATE	Yes	Yes	

Section	Field	CCSDS required	AWS Ground Station required	Notes
	ORIGINATOR	Yes	Yes	
	MESSAGE_ID	No	No	
Metadata	META_START	Yes	Yes	
	COMMENT	No	No	
	OBJECT_NAME	Yes	Yes	
	OBJECT_ID	Yes	Yes	
	CENTER_NAME	Yes	Yes	Required value: Earth
	REF_FRAME	Yes	Yes	Accepted values: EME2000, ITRF2000
	REF_FRAME _EPOCH	No	Not supported*	Not needed because the accepted REF_FRAMEs have an implicit epoch
	TIME_SYSTEM	Yes	Yes	Required value: UTC
	START_TIME	Yes	Yes	
	USEABLE_S TART_TIME	No	No	
	USEABLE_S TOP_TIME	No	No	

Section	Field	CCSDS required	AWS Ground Station required	Notes
	STOP_TIME	Yes	Yes	
	INTERPOLATION	No	Yes	Required so AWS Ground Station can generate accurate pointing angles for contacts.
	INTERPOLATION_DEGREE	No	Yes	Required so AWS Ground Station can generate accurate pointing angles for contacts. The specified degree will be used if possible, but a lower degree will be used if there is not enough data in the segment.
	META_STOP	Yes	Yes	
Data	X	Yes	Yes	Represented in `km`
	Y	Yes	Yes	Represented in `km`
	Z	Yes	Yes	Represented in `km`

Section	Field	CCSDS required	AWS Ground Station required	Notes
	X_DOT	Yes	Yes	Represented in `km/s`
	Y_DOT	Yes	Yes	Represented in `km/s`
	Z_DOT	Yes	Yes	Represented in `km/s`
	X_DDOT	No	No	Represented in `km/s^2`
	Y_DDOT	No	No	Represented in `km/s^2`
	Z_DDOT	No	No	Represented in `km/s^2`
Covariance matrix	COVARIANCE_START	No	No	
	EPOCH	No	No	
	COV_REF_FRAME	No	No	
	COVARIANCE_STOP	No	No	

* If any rows that aren't supported by AWS Ground Station are included in the provided OEM, the OEM will fail validation.

The important deviations from the CCSDS standard for AWS Ground Station are:

- `CCSDS_OEM_VERS` is required to be `2.0`.
- `REF_FRAME` is required to be either `EME2000` or `ITRF2000`.

- REF_FRAME_EPOCH is not supported by AWS Ground Station.
- CENTER_NAME is required to be Earth.
- TIME_SYSTEM is required to be UTC.
- INTERPOLATION and INTERPOLATION_DEGREE are both required for AWS Ground Station customer provided ephemeris.
- AWS Ground Station deviates from CCSDS 5.2.4.7 by allowing OEM data blocks that do not contain enough ephemeris data records to perform interpolation at the specified INTERPOLATION_DEGREE. In this case, AWS Ground Station will use the highest interpolation degree possible that is less than or equal to the specified INTERPOLATION_DEGREE.

Example OEM ephemeris in KVN format

Following is a truncated example of an OEM ephemeris in KVN format for the JPSS-1 public broadcaster satellite.

```
CCSDS_OEM_VERS = 2.0

COMMENT Orbit data are consistent with planetary ephemeris DE-430

CREATION_DATE  = 2024-07-22T05:20:59
ORIGINATOR     = Raytheon-JPSS/CGS

META_START
OBJECT_NAME          = J1
OBJECT_ID            = 2017-073A
CENTER_NAME          = Earth
REF_FRAME            = EME2000
TIME_SYSTEM          = UTC
START_TIME           = 2024-07-22T00:00:00.000000
STOP_TIME            = 2024-07-22T00:06:00.000000
INTERPOLATION        = Lagrange
INTERPOLATION_DEGREE = 5
META_STOP

2024-07-22T00:00:00.000000   5.905147360000000e+02  -1.860082793999999e+03
  -6.944807075000000e+03  -5.784245796000000e+00   4.347501391999999e+00
 -1.657256863000000e+00
2024-07-22T00:01:00.000000   2.425572045154201e+02  -1.595860765983339e+03
  -7.030938457373539e+03  -5.810660250794190e+00   4.457103652219009e+00
 -1.212889340333023e+00
```

```
2024-07-22T00:02:00.000000  -1.063224256538050e+02  -1.325569732497146e+03
  -7.090262617183503e+03  -5.814973972202444e+00   4.549739160042560e+00
 -7.639633689161465e-01
2024-07-22T00:03:00.000000  -4.547973959231161e+02  -1.050238305712201e+03
  -7.122556683227951e+03  -5.797176562437553e+00   4.625064829516728e+00
 -3.121687831090774e-01
2024-07-22T00:04:00.000000  -8.015427368657785e+02  -7.709137891269565e+02
  -7.127699477194810e+03  -5.757338007808417e+00   4.682800822515077e+00
 1.407953645161997e-01
2024-07-22T00:05:00.000000  -1.145240083085062e+03  -4.886583601179489e+02
  -7.105671911254255e+03  -5.695608435738609e+00   4.722731329786999e+00
 5.932259682105052e-01
2024-07-22T00:06:00.000000  -1.484582479061495e+03  -2.045451985605701e+02
  -7.056557069672793e+03  -5.612218005854990e+00   4.744705579872771e+00
 1.043421397392599e+00
```

Creating an OEM ephemeris

An OEM ephemeris can be created using the CreateEphemeris action in the AWS Ground Station API. This action will upload an ephemeris using data either in the request body or from a specified S3 bucket.

It is important to note that uploading an ephemeris sets the ephemeris to `VALIDATING` and starts an asynchronous workflow that will validate and generate potential contacts from your ephemeris. Only once an ephemeris has passed this workflow and become `ENABLED` will it be used for contacts. You should poll DescribeEphemeris for the ephemeris status or use CloudWatch events to track the ephemeris' status changes.

To troubleshoot an invalid ephemeris see: Troubleshoot invalid ephemerides

Example: Uploading OEM ephemeris data from an S3 bucket

It is also possible to upload an OEM ephemeris file directly from an S3 bucket by pointing to the bucket and object key. AWS Ground Station will retrieve the object on your behalf. Information about the encryption of data at rest in AWS Ground Station is detailed in: Data encryption at rest for AWS Ground Station.

Below is an example of uploading an OEM ephemeris file from an S3 bucket

```
import boto3
from datetime import datetime, timedelta, timezone
```

```
# Create AWS clients
s3_client = boto3.client("s3")
ground_station_client = boto3.client("groundstation")

# Define S3 bucket and key
bucket_name = "ephemeris-bucket"
object_key = "test_data.oem"

# Create sample OEM data in KVN format
oem_data = """CCSDS_OEM_VERS = 2.0

COMMENT Orbit data are consistent with planetary ephemeris DE-430

CREATION_DATE  = 2024-07-22T05:20:59
ORIGINATOR     = Raytheon-JPSS/CGS

META_START
OBJECT_NAME          = J1
OBJECT_ID            = 2017-073A
CENTER_NAME          = Earth
REF_FRAME            = EME2000
TIME_SYSTEM          = UTC
START_TIME           = 2024-07-22T00:00:00.000000
STOP_TIME            = 2024-07-22T00:06:00.000000
INTERPOLATION        = Lagrange
INTERPOLATION_DEGREE = 5
META_STOP

2024-07-22T00:00:00.000000   5.905147360000000e+02  -1.860082793999999e+03
  -6.944807075000000e+03  -5.784245796000000e+00   4.347501391999999e+00
 -1.657256863000000e+00
2024-07-22T00:01:00.000000   2.425572045154201e+02  -1.595860765983339e+03
  -7.030938457373539e+03  -5.810660250794190e+00   4.457103652219009e+00
 -1.212889340333023e+00
2024-07-22T00:02:00.000000  -1.063224256538050e+02  -1.325569732497146e+03
  -7.090262617183503e+03  -5.814973972202444e+00   4.549739160042560e+00
 -7.639633689161465e-01
2024-07-22T00:03:00.000000  -4.547973959231161e+02  -1.050238305712201e+03
  -7.122556683227951e+03  -5.797176562437553e+00   4.625064829516728e+00
 -3.121687831090774e-01
2024-07-22T00:04:00.000000  -8.015427368657785e+02  -7.709137891269565e+02
  -7.127699477194810e+03  -5.757338007808417e+00   4.682800822515077e+00
 1.407953645161997e-01
```

```
2024-07-22T00:05:00.000000  -1.145240083085062e+03  -4.886583601179489e+02
  -7.105671911254255e+03  -5.695608435738609e+00   4.722731329786999e+00
 5.932259682105052e-01
2024-07-22T00:06:00.000000  -1.484582479061495e+03  -2.045451985605701e+02
  -7.056557069672793e+03  -5.612218005854990e+00   4.744705579872771e+00
 1.043421397392599e+00
"""

# Upload sample OEM data to S3
print(f"Uploading OEM data to s3://{bucket_name}/{object_key}")

s3_client.put_object(
    Bucket=bucket_name, Key=object_key, Body=oem_data, ContentType="text/plain"
)

print("OEM data uploaded successfully to S3")

# Create OEM ephemeris from S3
print("Creating OEM ephemeris from S3...")

s3_oem_ephemeris = ground_station_client.create_ephemeris(
    name="2024-07-22 S3 OEM Upload",
    satelliteId="fde41049-14f7-413e-bd7b-EXAMPLE01",
    enabled=True,
    expirationTime=datetime.now(timezone.utc) + timedelta(days=5),
    priority=2,
    ephemeris={"oem": {"s3Object": {"bucket": bucket_name, "key": object_key}}},
)

print(f"Created OEM ephemeris with ID: {s3_oem_ephemeris['ephemerisId']}")
```

Below is an example returned data from the DescribeEphemeris action being called for the OEM ephemeris uploaded in the previous block of example code.

```
{
  "creationTime": 1620254718.765,
  "enabled": true,
  "name": "Example Ephemeris",
  "ephemerisId": "fde41049-14f7-413e-bd7b-EXAMPLE02",
  "priority": 2,
  "status": "VALIDATING",
  "suppliedData": {
    "oem": {
```

```
      "sourceS3Object": {
          "bucket": "ephemeris-bucket-for-testing",
          "key": "test_data.oem"
      }
    }
  }
}
```

Provide azimuth elevation ephemeris data

Important

The azimuth elevation ephemeris feature is currently in a Preview state and requires explicit onboarding.
Azimuth elevation ephemeris functionality is under strict access control for a limited number of pre-determined, specialized use cases. Access is significantly more restrictive than for standard customer-provided ephemeris capabilities. For more information about approved use cases and the access request process, please open an AWS Support ticket through the AWS Support Center Console. Our team will guide you through the approval process for specialized use cases.

Overview

Azimuth elevation ephemeris provides a way to directly specify antenna pointing directions without providing satellite orbital information. Instead of uploading ephemeris data that describes a satellite's orbit, you provide time-tagged azimuth and elevation angles that tell the antenna exactly where to point throughout a contact.

AWS Ground Station treats ephemerides as Individualized Usage Data. If you use this optional feature, AWS will use your ephemeris data to provide troubleshooting support.

This approach is particularly useful for the following scenarios:

- *Early operations support:* During Launch and Early Orbit Phase (LEOP) when precise orbital data is unavailable, or orbital parameters are rapidly changing.
- *Custom pointing patterns:* Implementing specific pointing sequences for antenna testing or non-standard operations.

Note

When using azimuth elevation ephemeris, the satellite ARN may be omitted from the contact reservation request. If the satellite ARN is not omitted, it will still be included as part of the contact data, but the azimuth elevation ephemeris will be used for antenna pointing rather than performing ephemeris priority resolution. The azimuth elevation ephemeris is associated with a specific ground station and defines the antenna pointing directions for that location.

Azimuth elevation ephemeris data format

Azimuth elevation ephemeris data consists of time-tagged azimuth and elevation values organized into segments. Each segment contains a series of azimuth and elevation angles that cover a specific time range.

The key components of azimuth elevation ephemeris data are:

- *Ground Station:* The specific ground station where this azimuth elevation ephemeris will be used.
- *Angle Unit:* The unit of measurement for angles (`DEGREE_ANGLE` or `RADIAN`).
- *Segments:* One or more time-bounded collections of azimuth and elevation angles.
- *Time-tagged angles:* Individual azimuth and elevation values with associated timestamps.

Each segment requires:

- A reference epoch (the base time for the segment)
- A valid time range (start and end times for the segment)
- At least 5 time-tagged azimuth/elevation pairs

Azimuth elevation constraints:

- Azimuth in degrees: -180° to 360°
- Azimuth in radians: -π to 2π
- Elevation in degrees: -90° to 90°
- Elevation in radians: -π/2 to π/2
- Time values must be in ascending order within each segment

- Segments must not overlap in time

For more information, see the CreateEphemeris API documentation and the TimeAzEl data type.

Creating azimuth elevation ephemeris

Azimuth elevation ephemeris is created using the same CreateEphemeris API action, but with the `azEl` ephemeris type. The key differences from TLE and OEM ephemeris are:

- You must specify a `groundStation` parameter
- The `satelliteId` parameter must be omitted from the request
- Priority settings do not apply (each azimuth elevation ephemeris is specific to a ground station)
- Each segment must contain at least 5 azimuth/elevation points to support 4th order Lagrange interpolation
- Additional limits and requirements are detailed in the CreateEphemeris API documentation

It is important to note that uploading an ephemeris sets the ephemeris to `VALIDATING` and starts an asynchronous workflow that will validate and generate potential contacts from your ephemeris. An ephemeris will only be used for contacts after it has passed this workflow and its status becomes `ENABLED`. You should poll DescribeEphemeris for the ephemeris status or use CloudWatch events to track the ephemeris' status changes.

To troubleshoot an invalid ephemeris see: Troubleshoot invalid ephemerides

Example: Create azimuth elevation ephemeris via API

The following example shows how to create an azimuth elevation ephemeris using the AWS SDK for Python (Boto3):

```
import boto3

# Create AWS Ground Station client
ground_station_client = boto3.client("groundstation")

# Create azimuth elevation ephemeris
azimuth_elevation_ephemeris = ground_station_client.create_ephemeris(
    name="Azimuth Elevation for Ohio Ground Station",
    ephemeris={
```

```
        "azEl": {
            "groundStation": "Ohio 1",
            "data": {
                "azElData": {
                    "angleUnit": "DEGREE_ANGLE",
                    "azElSegmentList": [
                        {
                            "referenceEpoch": "2024-03-15T10:00:00Z",
                            "validTimeRange": {
                                "startTime": "2024-03-15T10:00:00Z",
                                "endTime": "2024-03-15T10:15:00Z",
                            },
                            "azElList": [
                                {"dt": 0.0, "az": 45.0, "el": 10.0},
                                {"dt": 180.0, "az": 50.0, "el": 15.0},
                                {"dt": 360.0, "az": 55.0, "el": 20.0},
                                {"dt": 540.0, "az": 60.0, "el": 25.0},
                                {"dt": 720.0, "az": 65.0, "el": 30.0},
                                {"dt": 900.0, "az": 70.0, "el": 35.0},
                            ],
                        }
                    ],
                }
            },
        }
    },
)

print(f"Created ephemeris with ID: {azimuth_elevation_ephemeris['ephemerisId']}")
```

In this example:

- The azimuth elevation data is associated with the "Ohio 1" ground station
- Angles are specified in degrees
- The segment covers a 15-minute period
- The `dt` values are atomic seconds offset from the reference epoch
- Six azimuth/elevation pairs are provided (minimum is 5)

Example: Upload azimuth elevation data from S3

For larger datasets, you can upload azimuth elevation data from an S3 bucket:

```
import boto3
import json

# Create AWS clients
s3_client = boto3.client("s3")
ground_station_client = boto3.client("groundstation")

# Define S3 bucket and key
bucket_name = "azimuth-elevation-bucket"
object_key = "singapore-azimuth-elevation.json"

# Create sample azimuth elevation data
azimuth_elevation_data = {
    "angleUnit": "DEGREE_ANGLE",
    "azElSegmentList": [
        {
            "referenceEpoch": "2024-03-15T10:00:00Z",
            "validTimeRange": {
                "startTime": "2024-03-15T10:00:00Z",
                "endTime": "2024-03-15T10:15:00Z",
            },
            "azElList": [
                {"dt": 0.0, "az": 45.0, "el": 10.0},
                {"dt": 180.0, "az": 50.0, "el": 15.0},
                {"dt": 360.0, "az": 55.0, "el": 20.0},
                {"dt": 540.0, "az": 60.0, "el": 25.0},
                {"dt": 720.0, "az": 65.0, "el": 30.0},
                {"dt": 900.0, "az": 70.0, "el": 35.0},
            ],
        },
        {
            "referenceEpoch": "2024-03-15T10:15:00Z",
            "validTimeRange": {
                "startTime": "2024-03-15T10:15:00Z",
                "endTime": "2024-03-15T10:30:00Z",
            },
            "azElList": [
                {"dt": 0.0, "az": 70.0, "el": 35.0},
                {"dt": 180.0, "az": 75.0, "el": 40.0},
                {"dt": 360.0, "az": 80.0, "el": 45.0},
                {"dt": 540.0, "az": 85.0, "el": 50.0},
                {"dt": 720.0, "az": 90.0, "el": 55.0},
                {"dt": 900.0, "az": 95.0, "el": 50.0},
```

```
            ],
        },
    ],
}

# Upload sample data to S3
print(f"Uploading azimuth elevation data to s3://{bucket_name}/{object_key}")

s3_client.put_object(
    Bucket=bucket_name,
    Key=object_key,
    Body=json.dumps(azimuth_elevation_data, indent=2),
    ContentType="application/json",
)
print("Sample data uploaded successfully to S3")

# Create azimuth elevation ephemeris from S3
print("Creating azimuth elevation ephemeris from S3...")

s3_azimuth_elevation_ephemeris = ground_station_client.create_ephemeris(
    name="Large Azimuth Elevation Dataset",
    ephemeris={
        "azEl": {
            "groundStation": "Singapore 1",
            "data": {"s3Object": {"bucket": bucket_name, "key": object_key}},
        }
    },
)

print(f"Created ephemeris with ID: {s3_azimuth_elevation_ephemeris['ephemerisId']}")
```

The S3 object should contain a JSON structure with the azimuth elevation data in the same format as shown in the direct upload example.

Reserving contacts with azimuth elevation ephemeris

When using an azimuth elevation ephemeris to reserve a contact, the process differs from TLE and OEM ephemeris:

1. Create the azimuth elevation ephemeris using CreateEphemeris
2. Wait for the ephemeris to reach ENABLED status
3. Reserve the contact using ReserveContact with tracking overrides

Example of reserving a contact with azimuth elevation ephemeris:

```
import boto3
from datetime import datetime
import time

# Create AWS Ground Station client
ground_station_client = boto3.client("groundstation")

# First, create an azimuth elevation ephemeris
print("Creating azimuth elevation ephemeris...")

create_ephemeris_response = ground_station_client.create_ephemeris(
    name="Azimuth Elevation for Contact Reservation",
    ephemeris={
        "azEl": {
            "groundStation": "Ohio 1",
            "data": {
                "azElData": {
                    "angleUnit": "DEGREE_ANGLE",
                    "azElSegmentList": [
                        {
                            "referenceEpoch": "2024-03-15T10:00:00Z",
                            "validTimeRange": {
                                "startTime": "2024-03-15T10:00:00Z",
                                "endTime": "2024-03-15T10:15:00Z",
                            },
                            "azElList": [
                                {"dt": 0.0, "az": 45.0, "el": 10.0},
                                {"dt": 180.0, "az": 50.0, "el": 15.0},
                                {"dt": 360.0, "az": 55.0, "el": 20.0},
                                {"dt": 540.0, "az": 60.0, "el": 25.0},
                                {"dt": 720.0, "az": 65.0, "el": 30.0},
                                {"dt": 900.0, "az": 70.0, "el": 35.0},
                            ],
                        }
                    ],
                }
            },
        }
    },
)

ephemeris_id = create_ephemeris_response["ephemerisId"]
```

```
print(f"Created ephemeris with ID: {ephemeris_id}")

# Wait for ephemeris to become ENABLED
print("Waiting for ephemeris to become ENABLED...")

while True:
    status = ground_station_client.describe_ephemeris(ephemerisId=ephemeris_id)[
        "status"
    ]
    if status == "ENABLED":
        print("Ephemeris is ENABLED")
        break
    elif status in ["INVALID", "ERROR"]:
        raise RuntimeError(f"Ephemeris failed: {status}")
    time.sleep(5)

# Reserve contact with azimuth elevation ephemeris
print("Reserving contact...")

contact = ground_station_client.reserve_contact(
    # Note: satelliteArn is omitted when using azimuth elevation ephemeris
    missionProfileArn="arn:aws:groundstation:us-east-2:111122223333:mission-profile/
example-mission-profile",
    groundStation="Ohio 1",
    startTime=datetime(2024, 3, 15, 10, 0, 0),
    endTime=datetime(2024, 3, 15, 10, 15, 0),
    trackingOverrides={"programTrackSettings": {"azEl": {"ephemerisId":
 ephemeris_id}}},
)

print(f"Reserved contact with ID: {contact['contactId']}")
```

Note

The `satelliteArn` parameter may be omitted when reserving a contact with azimuth elevation ephemeris. The antenna will follow the specified azimuth and elevation angles during the contact.

Listing available contacts

When using azimuth elevation ephemeris, the ListContacts API requires specific parameters:

- The `satelliteArn` parameter may be omitted from the request
- You must provide an `ephemeris` parameter with the azimuth elevation ephemeris ID to specify which ephemeris to use
- Available contact windows show when the provided azimuth and elevation angles are above the site mask of the requested ground station
- You must still provide `groundStation` and `missionProfileArn`

Example of creating an azimuth elevation ephemeris and listing available contacts with it:

```
import boto3
from datetime import datetime, timezone
import time

# Create AWS Ground Station client
ground_station_client = boto3.client("groundstation")

# Step 1: Create azimuth elevation ephemeris
print("Creating azimuth elevation ephemeris...")
ephemeris_response = ground_station_client.create_ephemeris(
    name="Stockholm AzEl Ephemeris",
    ephemeris={
        "azEl": {
            "groundStation": "Stockholm 1",
            "data": {
                "azElData": {
                    "angleUnit": "DEGREE_ANGLE",
                    "azElSegmentList": [
                        {
                            "referenceEpoch": "2024-04-01T12:00:00Z",
                            "validTimeRange": {
                                "startTime": "2024-04-01T12:00:00Z",
                                "endTime": "2024-04-01T12:30:00Z",
                            },
                            "azElList": [
                                {"dt": 0.0, "az": 30.0, "el": 15.0},
                                {"dt": 360.0, "az": 45.0, "el": 30.0},
                                {"dt": 720.0, "az": 60.0, "el": 45.0},
                                {"dt": 1080.0, "az": 75.0, "el": 35.0},
                                {"dt": 1440.0, "az": 90.0, "el": 20.0},
                                {"dt": 1800.0, "az": 105.0, "el": 10.0},
                            ],
```

```
                        }
                    ],
                }
            },
        }
    },
)

ephemeris_id = ephemeris_response["ephemerisId"]
print(f"Created ephemeris: {ephemeris_id}")

# Step 2: Wait for ephemeris to become ENABLED
print("Waiting for ephemeris to become ENABLED...")
while True:
    describe_response = ground_station_client.describe_ephemeris(
        ephemerisId=ephemeris_id
    )
    status = describe_response["status"]

    if status == "ENABLED":
        print("Ephemeris is ENABLED")
        break
    elif status in ["INVALID", "ERROR"]:
        # Check for validation errors
        if "invalidReason" in describe_response:
            print(f"Ephemeris validation failed: {describe_response['invalidReason']}")
        raise RuntimeError(f"Ephemeris failed with status: {status}")

    print(f"Current status: {status}, waiting...")
    time.sleep(5)

# Step 3: List available contacts using the azimuth elevation ephemeris
print("Listing available contacts with azimuth elevation ephemeris...")

# Convert epoch timestamps to datetime objects
start_time = datetime.fromtimestamp(1760710513, tz=timezone.utc)
end_time = datetime.fromtimestamp(1760883313, tz=timezone.utc)

contacts_response = ground_station_client.list_contacts(
    startTime=start_time,
    endTime=end_time,
    groundStation="Stockholm 1",
    statusList=["AVAILABLE"],
    ephemeris={"azEl": {"id": ephemeris_id}},
```

```
    # satelliteArn is optional
    satelliteArn="arn:aws:groundstation::111122223333:satellite/a88611b0-f755-404e-
b60d-57d8aEXAMPLE",
    missionProfileArn="arn:aws:groundstation:eu-north-1:111122223333:mission-
profile/966b72f6-6d82-4e7e-b072-f8240EXAMPLE",
)

# Process the results
if contacts_response["contactList"]:
    print(f"Found {len(contacts_response['contactList'])} available contacts:")
    for contact in contacts_response["contactList"]:
        print(f"  - Contact from {contact['startTime']} to {contact['endTime']}")
        print(
            f"    Max elevation: {contact.get('maximumElevation', {}).get('value', 'N/
A')}°"
        )
else:
    print("No available contacts found for the specified azimuth elevation ephemeris")
```

Note

The `ephemeris` parameter with the azimuth elevation ID must be provided when listing contacts to specify which azimuth elevation ephemeris should be used for determining contact windows. If the `satelliteArn` is included, it will be associated with the contact data, but the azimuth elevation ephemeris will be used for antenna pointing rather than performing ephemeris priority resolution.

Reserve contacts with custom ephemeris

Overview

When using custom ephemeris (TLE, OEM, or azimuth elevation), you can reserve contacts using the ReserveContact API. This section describes two common workflows for reserving contacts and important considerations for ensuring successful contact scheduling.

AWS Ground Station antennas are shared resources among multiple customers. This means that even if a contact window appears available when you list contacts, another customer might reserve it before you do. Therefore, it's crucial to verify that your contact reaches the `SCHEDULED` state after reservation and to implement proper monitoring for contact state changes.

Important

For azimuth elevation ephemeris, the `satelliteArn` parameter may be omitted from the `ReserveContact` request, and you must provide `trackingOverrides` with the ephemeris ID. For TLE and OEM ephemeris, you still need to provide the `satelliteArn`.

Contact reservation workflows

There are two primary workflows for reserving contacts with custom ephemeris:

1. *List-then-reserve workflow:* First list available contact windows using ListContacts, then select and reserve a specific window. This approach is useful when you want to see all available opportunities before making a selection.
2. *Direct reservation workflow:* Directly reserve a contact for a specific time window without first listing available contacts. This approach is useful when you already know your desired contact time or are working with predetermined schedules.

Both workflows are valid and the choice depends on your operational requirements. The following sections provide examples of each approach.

Workflow 1: List available contacts then reserve

This workflow first queries for available contact windows, then reserves a specific window. This is useful when you want to see all available opportunities before making a selection.

Example: List and reserve with azimuth elevation ephemeris

```
import boto3
from datetime import datetime, timezone
import time

# Create AWS Ground Station client
ground_station_client = boto3.client("groundstation")

# Create azimuth elevation ephemeris
print("Creating azimuth elevation ephemeris...")
ephemeris_response = ground_station_client.create_ephemeris(
    name="AzEl Ephemeris for Contact",
```

```
    ephemeris={
        "azEl": {
            "groundStation": "Ohio 1",
            "data": {
                "azElData": {
                    "angleUnit": "DEGREE_ANGLE",
                    "azElSegmentList": [
                        {
                            "referenceEpoch": "2024-03-15T10:00:00Z",
                            "validTimeRange": {
                                "startTime": "2024-03-15T10:00:00Z",
                                "endTime": "2024-03-15T10:15:00Z",
                            },
                            "azElList": [
                                {"dt": 0.0, "az": 45.0, "el": 10.0},
                                {"dt": 180.0, "az": 50.0, "el": 15.0},
                                {"dt": 360.0, "az": 55.0, "el": 20.0},
                                {"dt": 540.0, "az": 60.0, "el": 25.0},
                                {"dt": 720.0, "az": 65.0, "el": 30.0},
                                {"dt": 900.0, "az": 70.0, "el": 35.0},
                            ],
                        }
                    ],
                }
            },
        }
    },
)

ephemeris_id = ephemeris_response["ephemerisId"]
print(f"Created ephemeris: {ephemeris_id}")

# Wait for ephemeris to become ENABLED
while True:
    status = ground_station_client.describe_ephemeris(ephemerisId=ephemeris_id)[
        "status"
    ]
    if status == "ENABLED":
        print("Ephemeris is ENABLED")
        break
    elif status in ["INVALID", "ERROR"]:
        raise RuntimeError(f"Ephemeris failed: {status}")
    time.sleep(5)
```

```
# List available contacts
print("Listing available contacts...")
contacts = ground_station_client.list_contacts(
    # Note: satelliteArn is omitted for azimuth elevation ephemeris
    groundStation="Ohio 1",
    missionProfileArn="arn:aws:groundstation:us-east-2:111122223333:mission-profile/
example-profile",
    startTime=datetime(2024, 3, 15, 10, 0, 0, tzinfo=timezone.utc),
    endTime=datetime(2024, 3, 15, 10, 15, 0, tzinfo=timezone.utc),
    statusList=["AVAILABLE"],
    ephemeris={"azEl": {"id": ephemeris_id}},
)

if contacts["contactList"]:
    # Reserve the first available contact
    contact = contacts["contactList"][0]
    print(f"Reserving contact from {contact['startTime']} to {contact['endTime']}...")

    reservation = ground_station_client.reserve_contact(
        # Note: satelliteArn is omitted when using azimuth elevation ephemeris
        missionProfileArn="arn:aws:groundstation:us-east-2:111122223333:mission-
profile/example-profile",
        groundStation="Ohio 1",
        startTime=contact["startTime"],
        endTime=contact["endTime"],
        trackingOverrides={
            "programTrackSettings": {"azEl": {"ephemerisId": ephemeris_id}}
        },
    )

    print(f"Reserved contact: {reservation['contactId']}")
else:
    print("No available contacts found")
```

Example: List and reserve with TLE ephemeris

```
import boto3
from datetime import datetime, timedelta, timezone
import time

# Create AWS Ground Station client
ground_station_client = boto3.client("groundstation")
```

```
satellite_id = "12345678-1234-1234-1234-123456789012"
satellite_arn = f"arn:aws:groundstation::111122223333:satellite/{satellite_id}"

# Create TLE ephemeris
print("Creating TLE ephemeris...")
ephemeris_response = ground_station_client.create_ephemeris(
    name="TLE Ephemeris for Contact",
    satelliteId=satellite_id,
    enabled=True,
    expirationTime=datetime.now(timezone.utc) + timedelta(days=7),
    priority=1,  # Higher priority than default ephemeris
    ephemeris={
        "tle": {
            "tleData": [
                {
                    "tleLine1": "1 25994U 99068A   24075.54719794  .00000075  00000-0
 26688-4 0  9997",
                    "tleLine2": "2 25994  98.2007  30.6589 0001234  89.2782  18.9934
 14.57114995111906",
                    "validTimeRange": {
                        "startTime": datetime.now(timezone.utc),
                        "endTime": datetime.now(timezone.utc) + timedelta(days=7),
                    },
                }
            ]
        }
    },
)

ephemeris_id = ephemeris_response["ephemerisId"]
print(f"Created ephemeris: {ephemeris_id}")

# Wait for ephemeris to become ENABLED
while True:
    status = ground_station_client.describe_ephemeris(ephemerisId=ephemeris_id)[
        "status"
    ]
    if status == "ENABLED":
        print("Ephemeris is ENABLED")
        break
    elif status in ["INVALID", "ERROR"]:
        raise RuntimeError(f"Ephemeris failed: {status}")
    time.sleep(5)
```

```
# List available contacts
print("Listing available contacts...")
start_time = datetime.now(timezone.utc) + timedelta(hours=1)
end_time = start_time + timedelta(days=1)

contacts = ground_station_client.list_contacts(
    satelliteArn=satellite_arn,  # Required for TLE/OEM ephemeris
    groundStation="Hawaii 1",
    missionProfileArn="arn:aws:groundstation:us-west-2:111122223333:mission-profile/
example-profile",
    startTime=start_time,
    endTime=end_time,
    statusList=["AVAILABLE"],
)

if contacts["contactList"]:
    # Reserve the first available contact
    contact = contacts["contactList"][0]
    print(f"Reserving contact from {contact['startTime']} to {contact['endTime']}...")

    reservation = ground_station_client.reserve_contact(
        satelliteArn=satellite_arn,  # Required for TLE/OEM ephemeris
        missionProfileArn="arn:aws:groundstation:us-west-2:111122223333:mission-
profile/example-profile",
        groundStation="Hawaii 1",
        startTime=contact["startTime"],
        endTime=contact["endTime"],
        # Note: trackingOverrides is optional for TLE/OEM
        # The system will use the highest priority ephemeris automatically
    )

    print(f"Reserved contact: {reservation['contactId']}")
else:
    print("No available contacts found")
```

Workflow 2: Direct contact reservation

This workflow directly reserves a contact without first listing available windows. This approach is useful when you already know your desired contact time or are implementing automated scheduling.

Example: Direct reservation with azimuth elevation ephemeris

```
import boto3
from datetime import datetime, timezone
import time

# Create AWS Ground Station client
ground_station_client = boto3.client("groundstation")

# Define contact window
contact_start = datetime(2024, 3, 20, 14, 0, 0, tzinfo=timezone.utc)
contact_end = datetime(2024, 3, 20, 14, 15, 0, tzinfo=timezone.utc)

# Create azimuth elevation ephemeris for the specific contact time
print("Creating azimuth elevation ephemeris...")
ephemeris_response = ground_station_client.create_ephemeris(
    name="Direct Contact AzEl Ephemeris",
    ephemeris={
        "azEl": {
            "groundStation": "Ohio 1",
            "data": {
                "azElData": {
                    "angleUnit": "DEGREE_ANGLE",
                    "azElSegmentList": [
                        {
                            "referenceEpoch": contact_start.isoformat(),
                            "validTimeRange": {
                                "startTime": contact_start.isoformat(),
                                "endTime": contact_end.isoformat(),
                            },
                            "azElList": [
                                {"dt": 0.0, "az": 45.0, "el": 10.0},
                                {"dt": 180.0, "az": 50.0, "el": 15.0},
                                {"dt": 360.0, "az": 55.0, "el": 20.0},
                                {"dt": 540.0, "az": 60.0, "el": 25.0},
                                {"dt": 720.0, "az": 65.0, "el": 30.0},
                                {"dt": 900.0, "az": 70.0, "el": 35.0},
                            ],
                        }
                    ],
                }
            },
        }
```

```
    },
)

ephemeris_id = ephemeris_response["ephemerisId"]
print(f"Created ephemeris: {ephemeris_id}")

# Wait for ephemeris to become ENABLED
while True:
    status = ground_station_client.describe_ephemeris(ephemerisId=ephemeris_id)[
        "status"
    ]
    if status == "ENABLED":
        print("Ephemeris is ENABLED")
        break
    elif status in ["INVALID", "ERROR"]:
        raise RuntimeError(f"Ephemeris failed: {status}")
    time.sleep(5)

# Directly reserve the contact
print(f"Reserving contact from {contact_start} to {contact_end}...")

reservation = ground_station_client.reserve_contact(
    # Note: satelliteArn is omitted for azimuth elevation
    missionProfileArn="arn:aws:groundstation:us-east-2:111122223333:mission-profile/
example-profile",
    groundStation="Ohio 1",
    startTime=contact_start,
    endTime=contact_end,
    trackingOverrides={"programTrackSettings": {"azEl": {"ephemerisId":
 ephemeris_id}}},
)

print(f"Reserved contact: {reservation['contactId']}")
```

Example: Direct reservation with TLE ephemeris

```
import boto3
from datetime import datetime, timedelta, timezone
import time

# Create AWS Ground Station client
ground_station_client = boto3.client("groundstation")
```

```
satellite_id = "12345678-1234-1234-1234-123456789012"
satellite_arn = f"arn:aws:groundstation::111122223333:satellite/{satellite_id}"

# Define contact window (based on predicted pass)
contact_start = datetime(2024, 3, 21, 10, 30, 0, tzinfo=timezone.utc)
contact_end = datetime(2024, 3, 21, 10, 42, 0, tzinfo=timezone.utc)

# Create TLE ephemeris
print("Creating TLE ephemeris...")
ephemeris_response = ground_station_client.create_ephemeris(
    name="Direct Contact TLE Ephemeris",
    satelliteId=satellite_id,
    enabled=True,
    expirationTime=contact_end + timedelta(days=1),
    priority=1,
    ephemeris={
        "tle": {
            "tleData": [
                {
                    "tleLine1": "1 25994U 99068A   24080.50000000  .00000075  00000-0
 26688-4 0  9999",
                    "tleLine2": "2 25994  98.2007  35.6589 0001234  89.2782  18.9934
 14.57114995112000",
                    "validTimeRange": {
                        "startTime": (contact_start - timedelta(hours=1)).isoformat(),
                        "endTime": (contact_end + timedelta(hours=1)).isoformat(),
                    },
                }
            ]
        }
    },
)

ephemeris_id = ephemeris_response["ephemerisId"]
print(f"Created ephemeris: {ephemeris_id}")

# Wait for ephemeris to become ENABLED
while True:
    status = ground_station_client.describe_ephemeris(ephemerisId=ephemeris_id)[
        "status"
    ]
    if status == "ENABLED":
        print("Ephemeris is ENABLED")
        break
```

```
    elif status in ["INVALID", "ERROR"]:
        raise RuntimeError(f"Ephemeris failed: {status}")
    time.sleep(5)

# Directly reserve the contact
print(f"Reserving contact from {contact_start} to {contact_end}...")

reservation = ground_station_client.reserve_contact(
    satelliteArn=satellite_arn,  # Required for TLE ephemeris
    missionProfileArn="arn:aws:groundstation:us-west-2:111122223333:mission-profile/
example-profile",
    groundStation="Hawaii 1",
    startTime=contact_start,
    endTime=contact_end,
    # Note: trackingOverrides is optional for TLE
    # The system will use the highest priority ephemeris automatically
)

print(f"Reserved contact: {reservation['contactId']}")
```

Monitoring contact state changes

After reserving a contact, it's important to monitor its state to ensure it successfully transitions to SCHEDULED and to be notified of any issues. AWS Ground Station emits events to Amazon EventBridge for all contact state changes.

Contact states follow this lifecycle:

- SCHEDULING - The contact is being processed for scheduling
- SCHEDULED - The contact was successfully scheduled and will execute
- FAILED_TO_SCHEDULE - The contact could not be scheduled (terminal state)

For more information on contact states and lifecycle, see Understand contact lifecycle.

Implementing contact state monitoring with EventBridge

To monitor contact state changes in real-time, you can set up an Amazon EventBridge rule that triggers a Lambda function whenever a Ground Station contact changes state. This approach is more efficient and scalable than polling the contact status.

Implementation steps

1. Create a Lambda function to process contact state change events
2. Create an EventBridge rule that matches Ground Station contact state change events
3. Add the Lambda function as a target for the rule

Example Lambda function handler

For a complete example of a Lambda function that processes contact state change events, see the `GroundStationCloudWatchEventHandlerLambda` resource in the `AquaSnppJpssTerraDigIF.yml` CloudFormation template. This template is available in the AWS Ground Station customer onboarding Amazon S3 bucket. For instructions on accessing this template, see the Putting it together section of the dataflow endpoint example.

EventBridge rule configuration

The EventBridge rule should use the following event pattern to match all Ground Station contact state changes:

```
{
  "source": ["aws.groundstation"],
  "detail-type": ["Ground Station Contact State Change"]
}
```

To filter for specific states only (e.g., failures), you can add a detail filter:

```
{
  "source": ["aws.groundstation"],
  "detail-type": ["Ground Station Contact State Change"],
  "detail": {
    "contactStatus": [
      "FAILED_TO_SCHEDULE",
      "FAILED",
      "AWS_FAILED",
      "AWS_CANCELLED"
    ]
  }
}
```

For detailed instructions on creating EventBridge rules with Lambda targets, see Creating rules that react to events in the Amazon EventBridge User Guide.

Setting up EventBridge rules for automation

You can create EventBridge rules to automatically respond to contact state changes. For example:

- Send notifications when a contact fails to schedule
- Trigger Lambda functions to prepare resources when a contact enters PREPASS
- Log contact completions for auditing purposes

For detailed information on setting up EventBridge rules for AWS Ground Station events, see Automate AWS Ground Station with Events.

Best practices and considerations

Handling scheduling conflicts

Since AWS Ground Station antennas are shared resources, a contact window that appears available in `ListContacts` might be reserved by another customer before you can reserve it. To handle this:

1. Always check the contact status after reservation
2. Implement retry logic with alternative time windows
3. Consider reserving contacts well in advance when possible
4. Use EventBridge events to monitor for `FAILED_TO_SCHEDULE` states

Ephemeris validation timing

Remember that ephemeris must be in `ENABLED` state before you can use it to reserve contacts. The validation process typically takes a few seconds to a few minutes depending on the ephemeris type and size. Always verify the ephemeris status before attempting to reserve contacts.

Contact timing considerations

When using custom ephemeris:

- Ensure your ephemeris covers the entire contact duration
- For azimuth elevation ephemeris, verify that the angles keep the antenna above the site mask throughout the contact
- Consider ephemeris expiration times when scheduling future contacts

API differences by ephemeris type

The `ReserveContact` API behaves differently depending on the ephemeris type:

Ephemeris Type	satelliteArn Required	trackingOverrides Required
TLE	Yes	No (optional)
OEM	Yes	No (optional)
Azimuth Elevation	No (optional)	Yes

Understand which ephemeris is used

Ephemerides have a *priority*, *expiration time*, and *enabled* flag. Together, these determine which ephemeris is used for tracking during a contact.

TLE and OEM ephemerides

For OEM and TLE ephemerides, only one ephemeris can be active for each satellite. The ephemeris that will be used is the highest-priority enabled ephemeris whose expiration time is in the future. A larger priority value indicates a higher priority. The available contact times returned by ListContacts are based on this ephemeris. If multiple `ENABLED` ephemerides have the same priority, the most recently created or updated ephemeris will be used.

> **Note**
>
> AWS Ground Station has a service quota on the number of `ENABLED` customer-provided ephemerides per satellite (see: Service Quotas). To upload ephemeris data after reaching this quota, delete (using DeleteEphemeris) or disable (using UpdateEphemeris) the lowest-priority/earliest created customer-provided ephemerides.

If no ephemeris has been created, or if no ephemerides have ENABLED status, AWS Ground Station will use a default ephemeris for the satellite (from Space-Track), if available. This default ephemeris has priority 0.

Azimuth elevation ephemerides

Azimuth elevation ephemerides work differently from OEM and TLE ephemerides. Each azimuth elevation ephemeris is associated with a specific ground station and does not have a priority. When you reserve a contact with azimuth elevation ephemeris, you explicitly specify which azimuth elevation ephemeris to use through the `trackingOverrides` parameter.

Key differences for azimuth elevation ephemerides:

- No priority system - you explicitly select the ephemeris for each contact
- Ground station specific - each ephemeris is associated with a particular ground station
- No automatic fallback - if the specified ephemeris is unavailable, the contact will fail

> **Note**
>
> Azimuth elevation ephemerides do not compete with OEM and TLE ephemerides. They are selected explicitly when reserving a contact and are only used when tracking overrides are specified.

Effect of new ephemerides on previously scheduled contacts

Use the DescribeContact API to view the effects of new ephemerides on previously scheduled contacts by returning the active visibility times.

For OEM and TLE ephemerides, contacts scheduled prior to uploading a new ephemeris will retain the originally scheduled contact time, while the antenna tracking will use the active ephemeris. If the spacecraft's position, based on the active ephemeris, differs greatly from the prior ephemeris, this may result in reduced satellite contact time with the antenna due to the spacecraft operating outside the transmit/receive site mask. Therefore, we recommend that you cancel and reschedule your future contacts after you upload a new ephemeris that differs greatly from the previous ephemeris.

With the DescribeContact API, you can determine the portion of your future contact that is unusable due to the spacecraft operating outside the transmit/receive site mask by comparing your scheduled contact `startTime` and `endTime` with the returned `visibilityStartTime` and `visibilityEndTime`. If you choose to cancel and reschedule your future contact(s), the contact time range must not be outside the visibility time range by more than 30 seconds. Cancelled contacts may incur costs when cancelled too close to the time of contact. For more information on cancelled contacts see: Ground Station FAQs.

For azimuth elevation ephemerides, scheduled contacts will use the specific ephemeris that was selected when the contact was reserved. If you need to update the azimuth elevation data for a scheduled contact, you can cancel and reschedule the contact with a new ephemeris.

Get the current ephemeris for a satellite

The current ephemeris in use by AWS Ground Station for a specific satellite can be retrieved by calling the GetSatellite or ListSatellites actions. Both of these methods will return metadata for the ephemeris currently in use. This ephemeris metadata is different for custom ephemerides uploaded to AWS Ground Station and for default ephemerides.

> **Note**
>
> Azimuth elevation ephemerides are not associated with satellites and therefore are not returned by GetSatellite or ListSatellites. To retrieve information about azimuth elevation ephemerides, use the DescribeEphemeris API with the specific ephemeris ID, or use ListEphemerides to see all available ephemerides for your account.

Default Ephemerides will only include `source` and `epoch` fields. The `epoch` is the epoch of the two-line element set that was pulled from Space-Track, and it is currently being used for computing the trajectory of the satellite.

A custom ephemeris will have a `source` value of `CUSTOMER_PROVIDED` and will include a unique identifier in the `ephemerisId` field. This unique identifier can be used to query for the ephemeris via the DescribeEphemeris action. An optional `name` field will be returned if the ephemeris was assigned a name during upload to AWS Ground Station via the CreateEphemeris action.

It is important to note that ephemerides are updated dynamically by AWS Ground Station so the returned data is only a snapshot of the ephemeris being used at the time of the call to the API.

Example GetSatellite return for a satellite using a default ephemeris

```
{
    "satelliteId": "e1cfe0c7-67f9-4d98-bad2-EXAMPLE",
    "satelliteArn": "arn:aws:groundstation::111122223333:satellite/e1cfe0c7-67f9-4d98-
bad2-EXAMPLE",
    "noradSatelliteID": 25994,
    "groundStations": [
        "Ohio 1",
        "Oregon 1"
    ],
    "currentEphemeris": {
        "source": "SPACE_TRACK",
        "epoch": 1528245583.619
    }
}
```

Example GetSatellite for a satellite using a custom ephemeris

```
{
      "satelliteId": "e1cfe0c7-67f9-4d98-bad2-EXAMPLE",
      "satelliteArn": "arn:aws:groundstation::111122223333:satellite/
e1cfe0c7-67f9-4d98-bad2-EXAMPLE",
      "noradSatelliteID": 25994,
      "groundStations": [
            "Ohio 1",
            "Oregon 1"
      ],
      "currentEphemeris": {
          "source": "CUSTOMER_PROVIDED",
          "ephemerisId": "e1cfe0c7-67f9-4d98-bad2-EXAMPLE",
          "name": "My Ephemeris"
      }
  }
```

Listing azimuth elevation ephemerides

Since azimuth elevation ephemerides are not associated with satellites, you need to use different APIs to discover and retrieve information about them:

1. Use ListEphemerides to list all ephemerides in your account, including azimuth elevation ephemerides. You can filter by status and ephemeris type.
2. Use DescribeEphemeris with a specific ephemeris ID to get detailed information about an azimuth elevation ephemeris.
3. Use DescribeContact with a specific contact ID to get detailed information about an ephemeris used for the contact.

Example ListEphemerides response including an azimuth elevation ephemeris:

```
{
    "ephemerides": [
        {
            "ephemerisId": "abc12345-6789-def0-1234-5678EXAMPLE",
            "ephemerisType": "AZ_EL",
            "name": "Azimuth Elevation for Ohio Ground Station",
            "status": "ENABLED",
            "creationTime": 1620254718.765
        },
        {
            "ephemerisId": "def45678-9012-abc3-4567-8901EXAMPLE",
            "ephemerisType": "TLE",
            "name": "TLE for Satellite 12345",
            "status": "ENABLED",
            "creationTime": 1620254700.123
        }
    ]
}
```

Note

In the ListEphemerides response, azimuth elevation ephemerides will have a `groundStation` field instead of a `satelliteId` field, making them easy to identify.

Revert to default ephemeris data

When you upload custom ephemeris data it will override the default ephemerides AWS Ground Station uses for that particular satellite. AWS Ground Station does not use the default ephemeris again until there are no currently enabled, unexpired customer-provided ephemerides available

for use. AWS Ground Station also does not list contacts past the expiration time of the current customer-provided ephemeris, even if there is a default ephemeris available past that expiration time.

> **Note**
>
> Azimuth elevation ephemerides do not have default values and do not override satellite ephemerides. They are explicitly selected when reserving a contact using the `trackingOverrides` parameter. If you no longer wish to use azimuth elevation ephemeris, simply reserve contacts without specifying tracking overrides, and the system will use the active satellite ephemeris instead.

Reverting TLE and OEM ephemerides

To revert back to the default Space-Track ephemerides for a satellite, you will need to do one of the following:

- Delete (using DeleteEphemeris) or disable (using UpdateEphemeris) all enabled customer-provided ephemerides. You can list the customer-provided ephemerides for a satellite using ListEphemerides.
- Wait for all existing customer-provided ephemerides to expire.

You can confirm that the default ephemeris is being used by calling GetSatellite and verifying that the `source` of the current ephemeris for the satellite is SPACE_TRACK. See Default ephemeris data for more information on default ephemerides.

Managing azimuth elevation ephemerides

Since azimuth elevation ephemerides are explicitly selected for each contact and are not associated with satellites, there is no concept of "reverting" to a default. Instead, you can manage azimuth elevation ephemerides as follows:

- *To stop using azimuth elevation ephemeris:* Simply reserve new contacts without specifying `trackingOverrides` and specifying a `satelliteArn`. The contact will use the active ephemeris for the specified satellite instead.

- *To remove unused azimuth elevation ephemerides:* Use DeleteEphemeris to delete azimuth elevation ephemerides that are no longer needed. Note that you cannot delete an ephemeris that is currently being used by a scheduled contact.

To list all azimuth elevation ephemerides in your account, use ListEphemerides. Azimuth elevation ephemerides can be identified by the `ephemerisType` field, or by the presence of a `groundStation` field instead of a `satelliteId` field in the response.

Work with dataflows

AWS Ground Station uses a *node* and *edge* relationship to construct *dataflows* to enable stream processing of your data. Each node is represented by a *config* which describes its expected processing. To illustrate this concept, consider a dataflow of `antenna-downlink` to a `s3-recording`. The `antenna-downlink` node represents the analog to digital transformation of the radio frequency spectrum per the defined parameters on the config. The `s3-recording` represents a compute node which will receive incoming data and store it in your S3 bucket. The resulting dataflow is an asynchronous data delivery of digitized RF data to an S3 bucket based on your specifications.

Within your mission profile, you can create many dataflows to meet your needs. The following sections describe how to set up your other AWS resources to be used with AWS Ground Station and offers recommendations for constructing dataflows. For detailed information on how each node behaves, including if it is considered a source or destination node, please see Use AWS Ground Station Configs.

Topics

AWS Ground Station data plane interfaces

The resulting data structure of your chosen dataflow depends on the source of the dataflow. Details of these formats are provided to you during the onboarding of your satellites. The following summarizes the formats used for each type of dataflow.

- **antenna-downlink**
  - (Bandwidth less-than-or-equal-to 40MHz) data is delivered as VITA-49 Signal Data/IP Format packets.
  - (Bandwidth greater-than 40MHz) data is delivered as AWS Ground Station Class 2 packets.
- **antenna-downlink-demod-decode**

- Data is delivered as Demodulated/Decoded Data/IP Format packets.

- **antenna-uplink**
  - Data must be delivered as VITA-49 Signal Data/IP Format packets.
- **antenna-uplink-echo**
  - Data is delivered as VITA-49 Signal Data/IP Format packets.

Use cross-region data delivery

The AWS Ground Station cross-region data delivery feature gives you the flexibility to send your data from an antenna to any AWS Ground Station supported AWS Region. This means you can maintain your infrastructure in a single AWS Region and schedule contacts on any AWS Ground Station Locations you are onboarded to.

When receiving your contact data in an Amazon S3 Bucket, AWS Ground Station will manage all delivery aspects for you.

To use cross-region data delivery to an Amazon EC2 instance (using either the AWS Ground Station Agent or a dataflow endpoint), the *dataflow-endpoint* must be created in your current AWS Region and your *dataflow-endpoint-config* must specify the same region. AWS Ground Station will manage delivering the data cross-region for you.

Set up and configure Amazon S3

You can utilize a Amazon S3 bucket to receive your downlink signals using AWS Ground Station. To create the destination *s3-recording-config*, you must be able to specify a Amazon S3 bucket and an IAM role which authorizes AWS Ground Station to write files to the bucket.

See Amazon S3 Recording Config for restrictions on the Amazon S3 bucket, IAM role, or AWS Ground Station config creation.

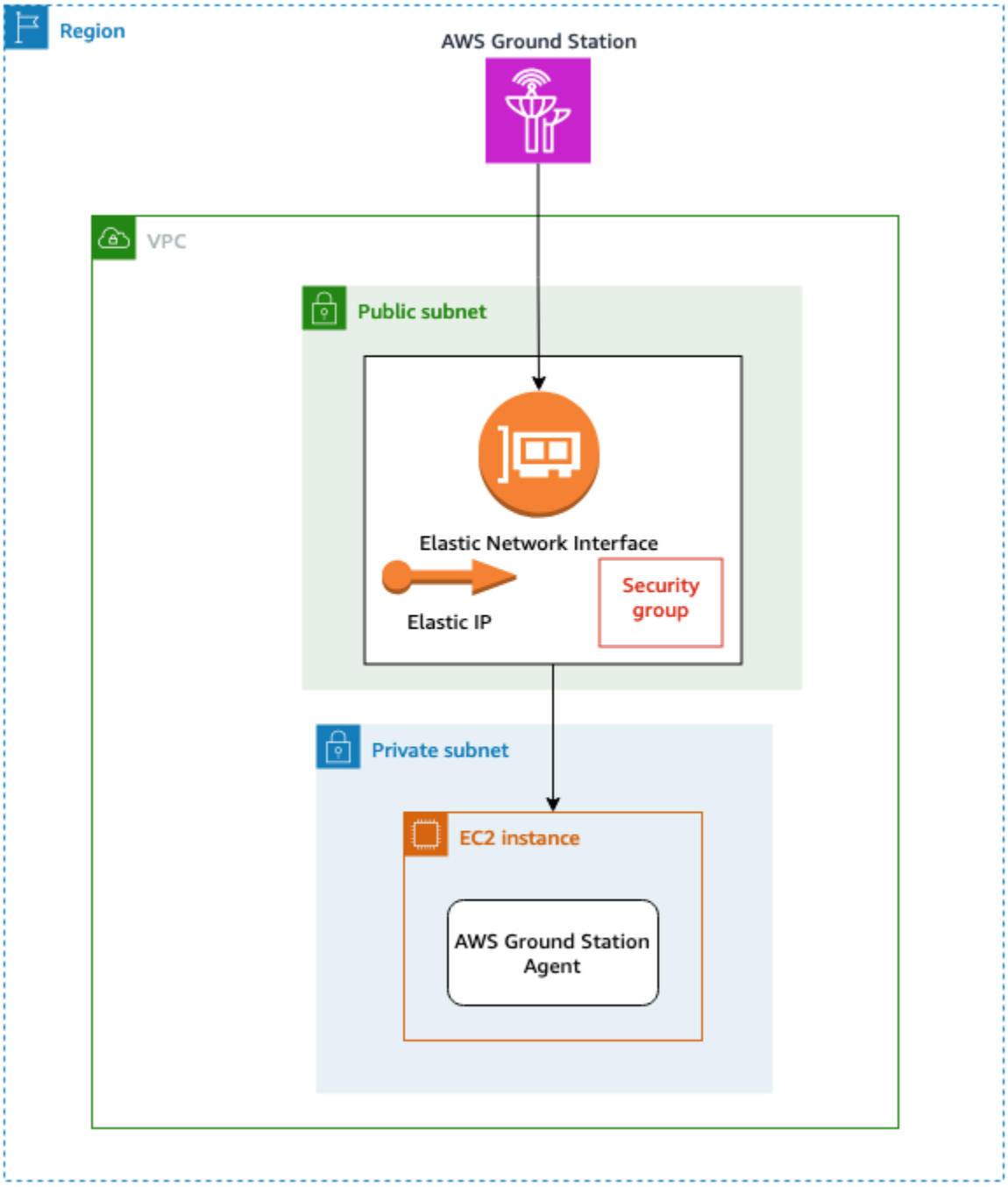
Set up and configure Amazon VPC

A full guide to set up a VPC is beyond the scope of this guide. For an in-depth understanding please refer to the Amazon VPC User Guide.

In this section, it is described how your Amazon EC2 and dataflow endpoint may exist within a VPC. AWS Ground Station does not support multiple delivery points for a given dataflow - it is

expected that each dataflow terminates to a single EC2 receiver. As we expect a single EC2 receiver, the configuration is not multi-AZ redundant. For full examples which will use your VPC, please see Example mission profile configurations.

VPC Configuration with AWS Ground Station Agent



Your satellite data is provided to an AWS Ground Station Agent instance that is proximate to the antenna. The AWS Ground Station Agent will stripe and then encrypt your data using the AWS KMS

key you provide. Each stripe is sent to your Amazon EC2 Elastic IP (EIP) from the source antenna across the AWS Network backbone. The data arrives at your EC2 instance via the Amazon EC2 Elastic Network Interface (ENI) attached. Once on your EC2 instance, the installed AWS Ground Station Agent will decrypt your data and perform forward error correction (FEC) to recover any dropped data, then forward it to the IP and port you specified in your setup.

The below list calls out unique setup considerations when setting up your VPC for AWS Ground Station Agent delivery.

Security Group - It is recommended you set up a security group dedicated to only AWS Ground Station traffic. This security group should allow UDP ingress traffic on the same port range you specify in your Dataflow Endpoint Group. AWS Ground Station maintains an AWS-managed prefix list to restrict your permissions to only AWS Ground Station IP addresses. See AWS Managed Prefix Lists for details on how to replace the *PrefixListId* for your deployment regions.

Elastic Network Interface (ENI) - You will need to associate the above security group with this ENI and place it in your public subnet.

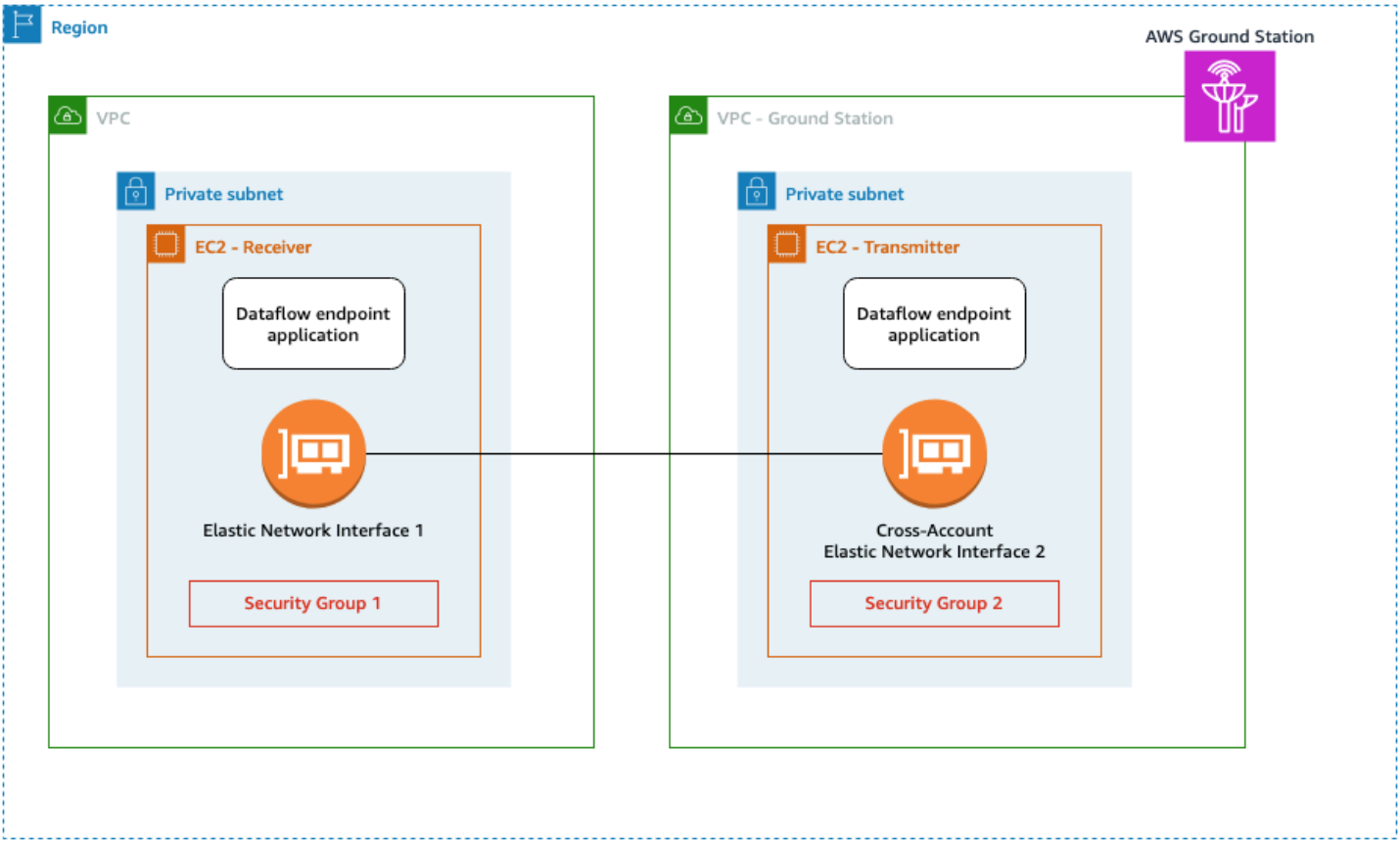
> **Note**
>
> The default quota for number of security groups attached per ENI is 5. This is an adjustable limit up to 16, see Amazon VPC Quotas.

The following CloudFormation template demonstrates how to create the infrastructure described in this section.

```
ReceiveInstanceEIP:
  Type: AWS::EC2::EIP
  Properties:
    Domain: 'vpc'

InstanceSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: AWS Ground Station receiver instance security group.
    VpcId:YourVpcId
    SecurityGroupIngress:
      # Add additional items here.
      - IpProtocol: udp
        FromPort: your-port-start-range
```

```
        ToPort: your-port-end-range
        PrefixListIds:
          - PrefixListId: com.amazonaws.global.groundstation
        Description: "Allow AWS Ground Station Downlink ingress."

InstanceNetworkInterface:
  Type: AWS::EC2::NetworkInterface
  Properties:
    Description: ENI for AWS Ground Station to connect to.
    GroupSet:
      - !Ref InstanceSecurityGroup
    SubnetId: A Public Subnet

ReceiveInstanceEIPAllocation:
  Type: AWS::EC2::EIPAssociation
  Properties:
    AllocationId:
      Fn::GetAtt: [ ReceiveInstanceEIP, AllocationId ]
    NetworkInterfaceId:
      Ref: InstanceNetworkInterface
```

VPC configuration with a dataflow endpoint



Your satellite data is provided to a dataflow endpoint application instance that is proximate to the antenna. The data is then sent through cross-account Amazon EC2 Elastic Network Interface (ENI) from a VPC owned by AWS Ground Station. The data then arrives at your EC2 instance via the ENI attached to your Amazon EC2 instance. The installed dataflow endpoint application will then forward it to the IP and port you specified in your setup. The reverse of this flow occurs for uplink connections.

The below list calls out unique setup considerations when setting up your VPC for dataflow endpoint delivery.

> **Note**
>
> The default quota for number of security groups attached per ENI is 5. This is an adjustable limit up to 16, see Amazon VPC Quotas.

IAM Role - The IAM Role is part of the Dataflow Endpoint and is not shown in the diagram. The IAM role that is used to create and attach the cross-account ENI to the AWS Ground Station Amazon EC2 instance.

Security Group 1 - This security group is attached to the ENI which will be associated to the Amazon EC2 instance in your account. It needs to allow UDP traffic from Security Group 2 on the ports specified in your *dataflow-endpoint-group*.

Elastic Network Interface (ENI) 1 - You will need to associate Security Group 1 with this ENI and place it in a subnet.

Subnet - You will need to ensure that there is at least one available IP address per dataflow for the Amazon EC2 instance in your account. For more details on subnet sizing see, Subnet CIDR blocks

Security Group 2 - This security group is referenced in the Dataflow Endpoint. This security group will be attached to the ENI that AWS Ground Station will use to place data into your account.

Region - For more information on the supported regions for cross-region connections, see Use cross-region data delivery.

The following CloudFormation template demonstrates how to create the infrastructure described in this section.

```
DataflowEndpointSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Security Group for AWS Ground Station registration of Dataflow
 Endpoint Groups
    VpcId: YourVpcId

AWSGroundStationSecurityGroupEgress:
  Type: AWS::EC2::SecurityGroupEgress
  Properties:
    GroupId: !Ref: DataflowEndpointSecurityGroup
    IpProtocol: udp
    FromPort: 55555
    ToPort: 55555
    CidrIp: 10.0.0.0/8
    Description: "Allow AWS Ground Station to send UDP traffic on port 55555 to the
 10/8 range."

InstanceSecurityGroup:
  Type: AWS::EC2::SecurityGroup
```

```
  Properties:
    GroupDescription: AWS Ground Station receiver instance security group.
    VpcId: YourVpcId
    SecurityGroupIngress:
      - IpProtocol: udp
        FromPort: 55555
        ToPort: 55555
        SourceSecurityGroupId: !Ref DataflowEndpointSecurityGroup
        Description: "Allow AWS Ground Station Ingress from
 DataflowEndpointSecurityGroup"

ReceiverSubnet:
  Type: AWS::EC2::Subnet
  Properties:
    # Ensure your CidrBlock will always have at least one available IP address per
 dataflow endpoint.
    # See https://docs.aws.amazon.com/vpc/latest/userguide/subnet-sizing.html for
 subent sizing guidelines.
    CidrBlock: "10.0.0.0/24"
    Tags:
      - Key: "Name"
        Value: "AWS Ground Station - Dataflow endpoint Example Subnet"
      - Key: "Description"
        Value: "Subnet for EC2 instance receiving AWS Ground Station data"
    VpcId: !Ref ReceiverVPC
```

Set up and configure Amazon EC2

Properly configuring your Amazon EC2 instance is required for synchronous delivery of VITA-49 Signal/IP data or VITA-49 Extension data/IP to be delivered via the AWS Ground Station Agent or a dataflow endpoint. Depending on your specific needs, you may perform the Front End (FE) processor or Software Defined Radio (SDR) directly on the same instance, or you may need to utilize additional EC2 instances. Selection and installation of your FE or SDR is beyond the scope of this user guide. For more information on the specific data formats, see AWS Ground Station data plane interfaces.

For information about our service terms, please see AWS Service Terms.

Supplied Common Software

AWS Ground Station provides common software to ease setup of your Amazon EC2 instance.

AWS Ground Station Agent

The AWS Ground Station Agent receives Digital Intermediate Frequency (DigIF) downlink data and egresses decrypted data that enables the following:

- DigIF downlink capability from 40 MHz to 400 MHz of bandwidth.
- High rate, low jitter DigIF data delivery to any public IP (AWS Elastic IP) on the AWS network.
- Reliable data delivery using Forward Error Correction (FEC).
- Secure data delivery using a customer managed AWS KMS key for encryption.

For more information, see AWS Ground Station Agent User Guide.

Dataflow endpoint application

A networking application that is used by AWS Ground Station to send and receive data between the AWS Ground Station antenna locations, and your Amazon EC2 instances. It can be used for the uplink and downlink of data.

Software Defined Radio (SDR)

A software defined radio (SDR) that can be used to modulate/demodulate the signal used to communicate with your satellite.

AWS Ground Station Amazon Machine Images (AMIs)

To reduce the build and configuration times of these installs, AWS Ground Station also offers preconfigured AMIs. The AMIs with a dataflow endpoint networking application and a software defined radio (SDR) are made available to your account after your onboarding is complete. They can be found in the Amazon EC2 console by searching for *groundstation* in private Amazon Machine Images (AMIs). The AMIs with AWS Ground Station Agent are public and can be found in the Amazon EC2 console by searching for *groundstation* in public Amazon Machine Images (AMIs).

Work with telemetry

AWS Ground Station telemetry delivers near real-time metrics from AWS Ground Station antennas during your satellite contacts. You can use telemetry data to monitor contact performance, detect anomalies, and make informed decisions about your satellite communications.

Topics

How telemetry works

To use telemetry, you configure a *TelemetrySinkConfig* that specifies where AWS Ground Station should deliver telemetry data. You then add this config to your mission profile using the `telemetrySinkConfigArn` field. During contacts that use a telemetry-enabled mission profile, AWS Ground Station streams telemetry data to your account.

The telemetry delivery process works as follows:

1. You create a Kinesis Data Streams stream in your AWS account to receive telemetry data. The stream must be created in the same account and region from which you schedule your contacts.
2. You create an IAM role that grants AWS Ground Station permission to write data to your stream.
3. You create a TelemetrySinkConfig that references your stream and IAM role.
4. You add the TelemetrySinkConfig to your mission profile.
5. You list and reserve contacts using the new telemetry-enabled mission profile.
6. During contacts using this mission profile, AWS Ground Station streams telemetry data to your Kinesis Data Streams stream in near real-time.
7. You consume and process the telemetry data from your stream using AWS services or your own applications.

Available telemetry types

AWS Ground Station provides the following telemetry types during contacts:

> **Note**
>
> AWS Ground Station is working on expanding the number of supported telemetry types

Pointing telemetry

: Provides information about antenna pointing direction during satellite contacts. This telemetry type is always sent during a contact and includes actual and commanded azimuth and elevation angles. For more information, see Pointing telemetry.

Tracking telemetry

: Provides information about antenna tracking status and tracking errors. This telemetry type is sent when autotracking is enabled in your tracking config. For more information, see Tracking telemetry.

Regional availability

Telemetry is available in all AWS Regions where AWS Ground Station operates. During contact execution, telemetry will be delivered from the AWS Ground Station antenna to the region you scheduled your contact from, providing cross-region support.

For a complete list of AWS Ground Station Regions and ground station locations, see AWS Ground Station Locations.

Set up telemetry

Follow these steps to configure telemetry for your AWS Ground Station contacts. After completing this setup, telemetry data will be delivered to your Kinesis Data Streams stream during contacts that use a telemetry-enabled mission profile. For an in-depth understanding of Kinesis Data Streams please refer to the Kinesis Data Streams User Guide.

Step 1: Create prerequisite AWS resources

The following CloudFormation snippet demonstrates how to create the prerequisite AWS resources for telemetry delivery. This snippet creates a Kinesis Data Streams stream and an IAM role that grants AWS Ground Station permission to write telemetry data to the stream.

```
TelemetryStream:
  Type: AWS::Kinesis::Stream
  Properties:
    Name: GroundStationTelemetryStream
    StreamModeDetails:
      StreamMode: ON_DEMAND
    RetentionPeriodHours: 24

TelemetryRole:
  Type: AWS::IAM::Role
  Properties:
    RoleName: GroundStationTelemetryRole
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Principal:
            Service: groundstation.amazonaws.com
          Action: sts:AssumeRole
    Policies:
      - PolicyName: KinesisWritePolicy
        PolicyDocument:
          Version: '2012-10-17'
          Statement:
            - Effect: Allow
              Action:
                - kinesis:DescribeStream
                - kinesis:PutRecord
                - kinesis:PutRecords
              Resource: !GetAtt TelemetryStream.Arn
```

The below list calls out unique setup considerations when configuring telemetry delivery for AWS Ground Station.

Kinesis Data Streams stream - The stream uses on-demand capacity mode, which automatically scales based on throughput. This is recommended for most use cases. The stream is configured

to retain data for 24 hours. By default, the stream uses AWS managed encryption. To use customer-managed encryption with AWS Key Management Service, add the `StreamEncryption` property and update the IAM role policy to include `kms:GenerateDataKey` permission. For more information, see Data Protection in Amazon Kinesis Data Streams.

IAM Role - The IAM role allows the `groundstation.amazonaws.com` service principal to assume the role and write telemetry data to your Kinesis Data Streams stream. The role policy grants permissions for `kinesis:DescribeStream`, `kinesis:PutRecord`, and `kinesis:PutRecords` actions on the stream. See Telemetry Sink Config for guidance on setting up the trust policy and role policy.

Additional configuration - Add `iam:PassRole` permissions to the IAM user or role you use for AWS Ground Station API calls. This allows you to pass the telemetry role to AWS Ground Station when creating a TelemetrySinkConfig.

Example PassRole Policy

For more information on how to update or attach a role policy, see Managing IAM policies in the IAM User Guide. For more information on the `iam:PassRole` permission, see Grant a user permissions to pass a role to an AWS service

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:GetRole",
        "iam:PassRole"
      ],
      "Resource": "arn:aws:iam::999999999999:role/your-telemetry-delivery-role-name"
    }
  ]
}
```

Step 2: Create a TelemetrySinkConfig

Create a *TelemetrySinkConfig* that defines how AWS Ground Station will deliver telemetry data to your Kinesis Data Streams stream. Use the stream ARN and role ARN from the CloudFormation stack outputs in Step 1.

> **Note**
>
> When you create a *TelemetrySinkConfig*, AWS Ground Station will verify access to your Kinesis Data Streams stream by delivering an empty test record with a partition key of `test`.

For more information about creating a *TelemetrySinkConfig*, see Telemetry Sink Config.

Step 3: Add telemetry to your mission profile

Create a mission profile. For more information about creating mission profiles, see Use AWS Ground Station Mission Profiles. Add the `telemetrySinkConfigArn` to your mission profile to enable telemetry delivery during contacts. Use the ARN of the *TelemetrySinkConfig* created in Step 2.

Step 4: Schedule a contact

Schedule a contact using your telemetry-enabled mission profile. During the contact, AWS Ground Station will stream telemetry data to your Kinesis Data Streams stream.

What to expect during contacts

- **Telemetry start** - Data begins streaming as the contact starts.
- **Near real-time delivery** - Telemetry arrives in your Kinesis Data Streams stream in near real-time.
- **Contact duration** - Data continues throughout the entire contact.
- **Automatic stop** - Telemetry stops streaming when the contact ends.

Monitoring delivery

You can monitor telemetry delivery using:

- **Kinesis Data Streams stream metrics** - Check incoming records in CloudWatch. For more information, see Monitoring Amazon Kinesis Data Streams.
- **Application logs** - Verify data processing in your applications that consume from the stream.
- **Kinesis Data Viewer** - Use the Kinesis Data Streams stream console to view sample records from your stream.

Next steps

After completing the setup, you can:

- Learn about the telemetry data format and available telemetry types. See Understand telemetry data.
- Build applications to process telemetry data from your Kinesis Data Streams stream. For more information, see Building Consumers for Amazon Kinesis Data Streams.
- Create dashboards and alerts using CloudWatch and other AWS services.
- Review troubleshooting guidance if you encounter issues. See Troubleshoot telemetry.

Understand telemetry data

Telemetry data is delivered as Base64-encoded JSON records to your Kinesis Data Streams stream. Each record contains information collected during your satellite contact, including metadata about the contact and the sampled telemetry measurements.

Data format overview

Each telemetry record contains the following components:

Telemetry type and version

Identifies the specific type of telemetry data and its schema version. This allows you to parse different telemetry types appropriately. For more information about schema versioning, see Schema versioning and evolution.

Scope ID

A unique identifier for the scope of the telemetry. This allows you to correlate telemetry data with specific contacts.

Metadata

Contextual information about the telemetry.

Data

The sampled telemetry measurements specific to the telemetry type.

Partition key

Telemetry records are delivered to your Kinesis Data Streams stream with a partition key in the format:

```
SCOPE#scopeId#TELEMETRY_ID#telemetryId#TELEMETRY_VERSION#telemetryVersion
```

This partition key ensures that all telemetry of a given type for a single contact is delivered to the same shard within your Kinesis Data Streams stream, providing best effort ordering for that contact's telemetry stream.

Pointing telemetry

Pointing telemetry provides information about antenna pointing direction during satellite contacts. This telemetry type is always sent during a contact.

Data fields

sampleTimestamp

Time when the telemetry data was sampled, in ISO-8601 format in UTC with millisecond precision.

azimuth

Actual azimuth angle of the antenna in degrees.

elevation

Actual elevation angle of the antenna in degrees.

commandedAzimuth

Commanded azimuth angle in degrees. This is the target azimuth angle the antenna is attempting to achieve.

commandedElevation

Commanded elevation angle in degrees. This is the target elevation angle the antenna is attempting to achieve.

> **Note**
>
> The actual antenna position may differ from the commanded position due to physical limitations or mechanical delays during the contact.

Metadata fields

groundStation

Name of the ground station (for example, "Ohio 1").

satelliteId

Identifier of the satellite resource in AWS Ground Station.

contactId

Identifier of the contact.

Example JSON

```
{
  "telemetryTypeAndVersion": "POINTING#1.0.0",
  "telemetryType": "POINTING",
  "telemetryVersion": "1.0.0",
  "scopeId": "12345678-1234-1234-1234-123456789012",
  "metadata": {
    "groundStation": "Ohio 1",
    "satelliteId": "87654321-4321-4321-4321-210987654321",
    "contactId": "12345678-1234-1234-1234-123456789012"
  },
  "data": {
    "sampleTimestamp": "2025-12-08T12:00:00.123Z",
    "azimuth": 180.5,
    "elevation": 45.2,
    "commandedAzimuth": 180.0,
    "commandedElevation": 45.0
  }
}
```

Tracking telemetry

Tracking telemetry provides information about antenna tracking status and tracking errors. This telemetry type is sent when autotracking is enabled in your tracking config and when the antenna is actively using autotrack.

> **Note**
>
> If the `autotrack` parameter in your TrackingConfig is set to `REMOVED`, no tracking telemetry will be delivered. For more information about tracking configs, see Tracking Config.

Data fields

sampleTimestamp

Time when the telemetry data was sampled, in ISO-8601 format in UTC with millisecond precision.

trackingStatus

Current tracking status of the antenna. Possible values are:

- `TRACKING` — The antenna has successfully locked onto a signal that matches the mission profile and is actively following it across the sky. This is the nominal operational state during a contact.
- `ACQUIRING` — The antenna is in the process of locating and locking onto the signal. The system is currently using programmatic tracking, pointing based on ephemeris data.
- `MASKED` — The satellite's predicted position is behind an autotrack mask, meaning the antenna cannot reliably utilize autotrack at that specific pointing direction. This typically occurs at areas of high RF interference such as low elevations.

trackingErrorAzimuth

Tracking error in the azimuth axis, measured in degrees.

trackingErrorElevation

Tracking error in the elevation axis, measured in degrees.

> **Note**
>
> The tracking error values represent adjustments from the ephemeris-based program track that AWS Ground Station applies during autotracking to maximize signal strength.

Metadata fields

Tracking telemetry includes the same metadata fields as pointing telemetry: `groundStation`, `satelliteId`, and `contactId`.

Example JSON

```
{
  "telemetryTypeAndVersion": "TRACKING#1.0.0",
  "telemetryType": "TRACKING",
  "telemetryVersion": "1.0.0",
  "scopeId": "12345678-1234-1234-1234-123456789012",
  "metadata": {
    "groundStation": "Ohio 1",
    "satelliteId": "87654321-4321-4321-4321-210987654321",
    "contactId": "12345678-1234-1234-1234-123456789012"
  },
  "data": {
    "sampleTimestamp": "2025-12-08T12:00:00.123Z",
    "trackingStatus": "TRACKING",
    "trackingErrorAzimuth": 0.2,
    "trackingErrorElevation": 0.1
  }
}
```

Reading data from Kinesis Data Streams stream

Telemetry data is delivered to your Kinesis Data Streams stream and can be consumed using standard stream consumption patterns. When reading data from your stream, keep the following considerations in mind.

Base64 decoding

Data in Kinesis Data Streams stream is Base64-encoded. You must decode the data before parsing it as JSON. For more information, see Working with Amazon Kinesis Data Streams.

Using the Kinesis Data Viewer

For quick access to your telemetry data, the Kinesis Data Streams stream console offers a Data Viewer feature. When using this feature:

- Telemetry delivery can occur to any shard within your stream.

- The default starting position reads from the latest records in the shard.
- You may need to adjust the selected shard and use the "At timestamp" starting position to view received records.

Using the Kinesis Client Library

The Kinesis Client Library (KCL) manages many of the complexities associated with consuming data from Kinesis Data Streams stream, including shard management, checkpointing, and load balancing. We recommend using KCL for production telemetry consumption applications.

For more information, see Developing Consumers Using the Kinesis Client Library.

Best practices for consumption

- **Minimize latency** - Use Enhanced Fan-Out to read from Kinesis Data Streams stream with dedicated throughput and lower latency compared to polling. For more information, see Developing Enhanced Fan-Out Consumers.
- **Dedicated stream** - Use a dedicated Kinesis Data Streams stream for your AWS Ground Station telemetry integration. Sharing a stream with other applications can cause write throughput saturation and telemetry delivery failures.
- **On-demand capacity** - Deploy your Kinesis Data Streams stream in on-demand provisioning mode to allow automatic scaling of shards based on throughput.
- **Monitor throughput** - Monitor your stream for throttling using CloudWatch metrics. For more information, see Monitoring Amazon Kinesis Data Streams.

Schema versioning and evolution

Telemetry schemas are versioned to support evolution over time. The `telemetryVersion` field in each record indicates the schema version.

Handling schema changes

- New telemetry types may be introduced in the future.
- Existing telemetry types may receive new versions with breaking changes.
- Your applications should be tolerant of unknown telemetry types and versions.
- Parse the `telemetryTypeAndVersion`, `telemetryType`, and `telemetryVersion` fields to determine how to process each record.

We recommend implementing version-aware payload serialization that can handle multiple schema versions gracefully, allowing your applications to continue functioning when new versions are introduced.

Work with contacts

You can enter satellite data, identify antenna locations, communicate, and schedule antenna time for selected satellites by using the AWS Ground Station console, AWS CLI, or the AWS SDK in the language of your choice. You can review, cancel, and reschedule contact reservations up to 15 minutes before contact start*. You can also update a contact to specify an ephemeris override — including azimuth/elevation, OEM, or TLE tracking data — or change the target satellite. For more information, see Update contacts and contact versioning. In addition, you can view the details of your reserved minutes pricing plan if you are using the AWS Ground Station reserved minutes pricing model.

AWS Ground Station supports cross-region data delivery. The dataflow endpoint configs that are part of the mission profile you select determine to which region(s) the data is delivered. For more information about using cross-region data delivery, see Use cross-region data delivery.

To schedule contacts, your resources must be configured. If you have not configured your resources, see Get started. When ReserveContact is called, AWS Ground Station takes a snapshot of the mission profile and config resources for use throughout the contact's lifecycle. Changes to these resources using the UpdateMissionProfile and UpdateConfig APIs will not be reflected in contacts reserved prior to the updates. If you need the resource changes applied to an already scheduled contact, you must first cancel the contact using CancelContact, and then reschedule it using ReserveContact.

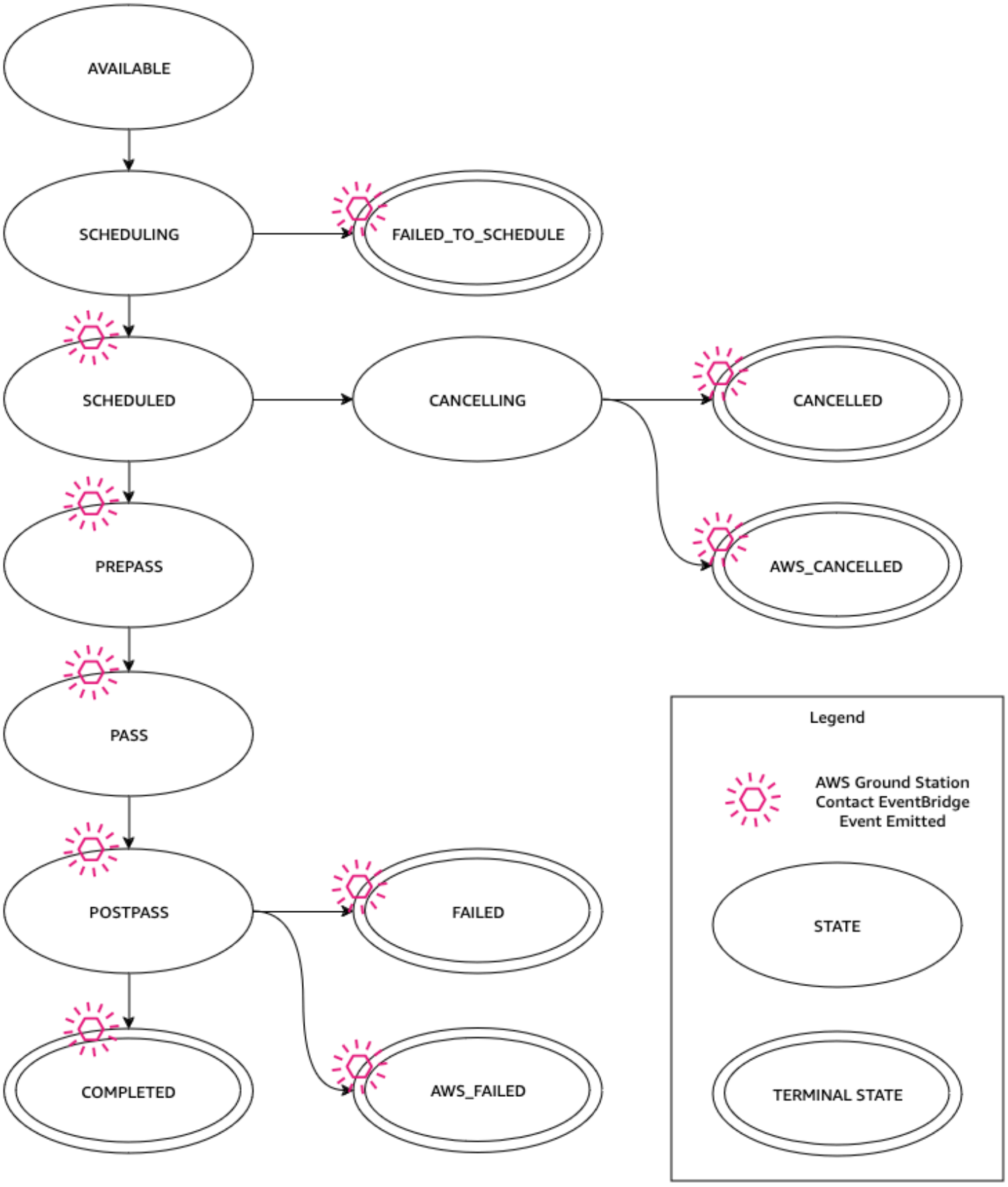
* Cancelled contacts may incur costs when cancelled too close to the time of contact. For more information on cancelled contacts see: Ground Station FAQs.

Topics

Understand contact lifecycle

Understanding the contact lifecycle can help you to automate and troubleshoot various problems while using AWS Ground Station. The following diagram shows the AWS Ground Station contact lifecycle as well as Event Bridge Events emitted during the lifecycle. It is important to note that the COMPLETED, FAILED, FAILED_TO_SCHEDULE, CANCELLED, AWS_CANCELLED, and AWS_FAILED are

terminal states. Contacts will not transition out of a terminal state. See the AWS Ground Station contact statuses for details on what each status indicates and whether it is stoppable or cancellable using CancelContact.


AVAILABLE
SCHEDULING
FAILED_TO_SCHEDULE
SCHEDULED
CANCELLING
CANCELLED
PREPASS
AWS_CANCELLED
PASS
POSTPASS
FAILED
COMPLETED
AWS_FAILED
Legend
AWS Ground Station Contact EventBridge Event Emitted
STATE
TERMINAL STATE

AWS Ground Station contact statuses

The status of an AWS Ground Station contact provides insight into what is happening to that contact at a given time.

Contact statuses

The following table describes the statuses that a contact can have:

Status	Description	Terminal	Cancelable	Stoppable
AVAILABLE	The contact is available to be reserved.	No	N/A	N/A
SCHEDULING	The contact is in the process of scheduling.	No	Yes	No
SCHEDULED	The contact was successfully scheduled.	No	Yes	No
FAILED_TO _SCHEDULE	The contact failed to schedule.	Yes	No	No
PREPASS	The contact is starting soon and resources are being prepared.	No	Yes	No
PASS	The contact is currently executing and the satellite is being communicated with.	No	No	Yes
POSTPASS	The communication has completed and resources used are being cleaned up.	No	No	No
COMPLETED	The contact completed without error.	Yes	No	No
FAILED	The contact failed because of an issue with your resource configuration.	Yes	No	No

Status	Description	Terminal	Cancelable	Stoppable
AWS_FAILED	The contact failed because of a problem in the AWS Ground Station service.	Yes	No	No
CANCELLING	The contact is in the process of being cancelled.	No	No	No
AWS_CANCELLED	The contact was cancelled by the AWS Ground Station service. Antenna or site maintenance, and ephemeris drift are examples of when this could happen.	Yes	No	No
CANCELLED	The contact was cancelled by you.	Yes	No	No

Note

For information about billing implications of cancelled or stopped contacts, see Understand contact billing.

Contact Data Retention

AWS Ground Station retains contact data for 1 year after a ReserveContact request is made to reserve a contact. After the 1 year period, the contact data is deleted.

If you need to retain contact data beyond one year, it is recommended to export your data before the retention period expires. For more information on how to access and export contact data, refer to:

- AWS Ground Station API Reference
- AWS Ground Station CLI Command Reference

Understand contact billing

With AWS Ground Station, you pay only for the antenna time you use. AWS Ground Station meters contact usage on a per-minute basis. For each contact, the service calculates the contact duration from start to end time and rounds up to the nearest minute. This metered duration determines your charges for that contact.

Your rate depends on two main factors:

- **Bandwidth** – The amount of bandwidth reserved for the contact (narrowband or wideband)
- **Ground station location** – Rates vary by ground station location

Bandwidth definitions

AWS Ground Station categorizes contacts into two bandwidth tiers based on the instantaneous bandwidth:

- **Narrowband** – Any contact where the instantaneous bandwidth is less than or equal to 40 MHz
- **Wideband** – Any contact where the instantaneous bandwidth is greater than 40 MHz

Scheduling modes

AWS Ground Station offers two scheduling modes:

- **On-Demand** – Pay for antenna access with no long-term commitments
- **Reserved** – Provides a discounted rate and improved scheduling compared to On-Demand, with a monthly commitment. Reserved minute pricing is available for customers who commit to monthly usage for a certain period of time.

For specific pricing information for your account or to learn more about the Reserved scheduling mode, contact your AWS representative.

CancelContact

Use of the CancelContact API varies based on the contact state when you call it:

- **Before contact start** - Cancels the contact entirely

- **After contact start and before contact end** - Stops the contact in progress

When you cancel a contact, billing depends on your scheduling mode and when you cancel. For more information, reach out to your AWS representative.

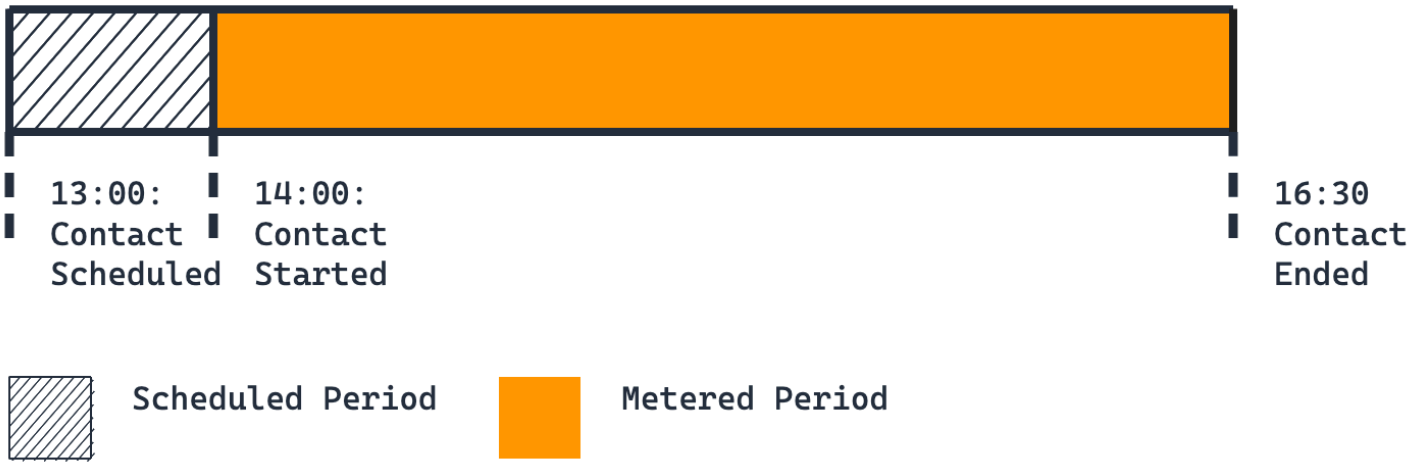
When you stop a contact, you are billed for the portion of the contact that executed, and the remaining time that is not covered by duplicate contacts. A duplicate contact in this context has been:

- Scheduled on the **same** ground station as the original stopped contact
- Scheduled by the **same** AWS account ID as the original stopped contact
- Reserved **after** the command was issued to stop the original contact

The following scenarios demonstrate how this metering works in practice.

Scenario 1: Single contact

You schedule a 150-minute contact on Ground Station Anytown 1 to begin at 14:00 and end at 16:30.



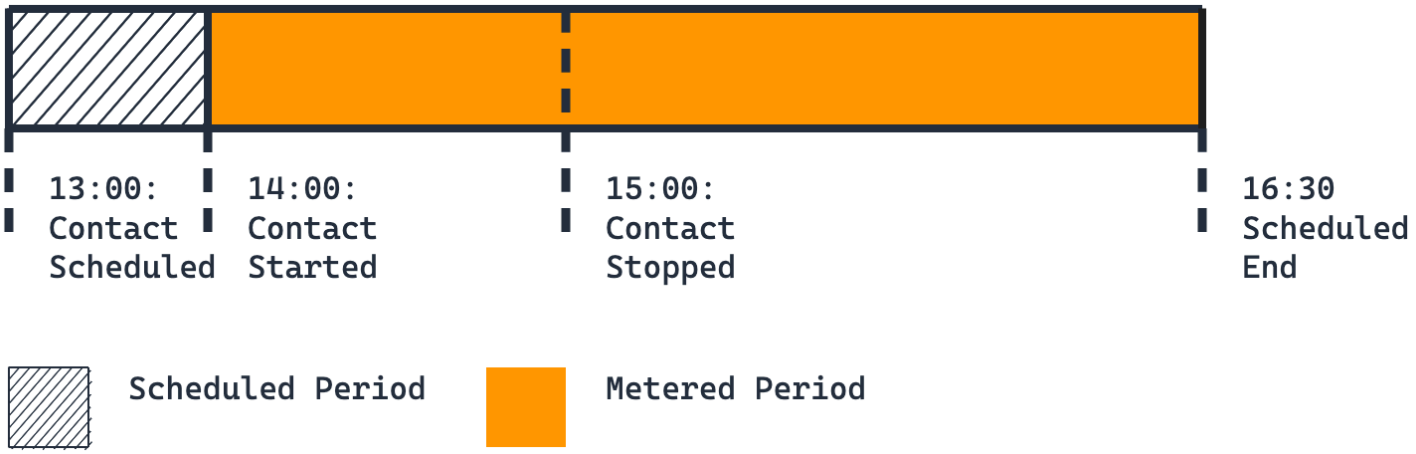
Billing breakdown:

- First contact: 150 minutes (full duration)

You're billed for 150 minutes. This is the baseline scenario where a contact runs to its scheduled completion without any stops or cancellations.

Scenario 2: Single stopped contact

You schedule a 150-minute contact on Ground Station Anytown 1 to begin at 14:00 and end at 16:30. At 15:00, you call the CancelContact API to stop your contact.



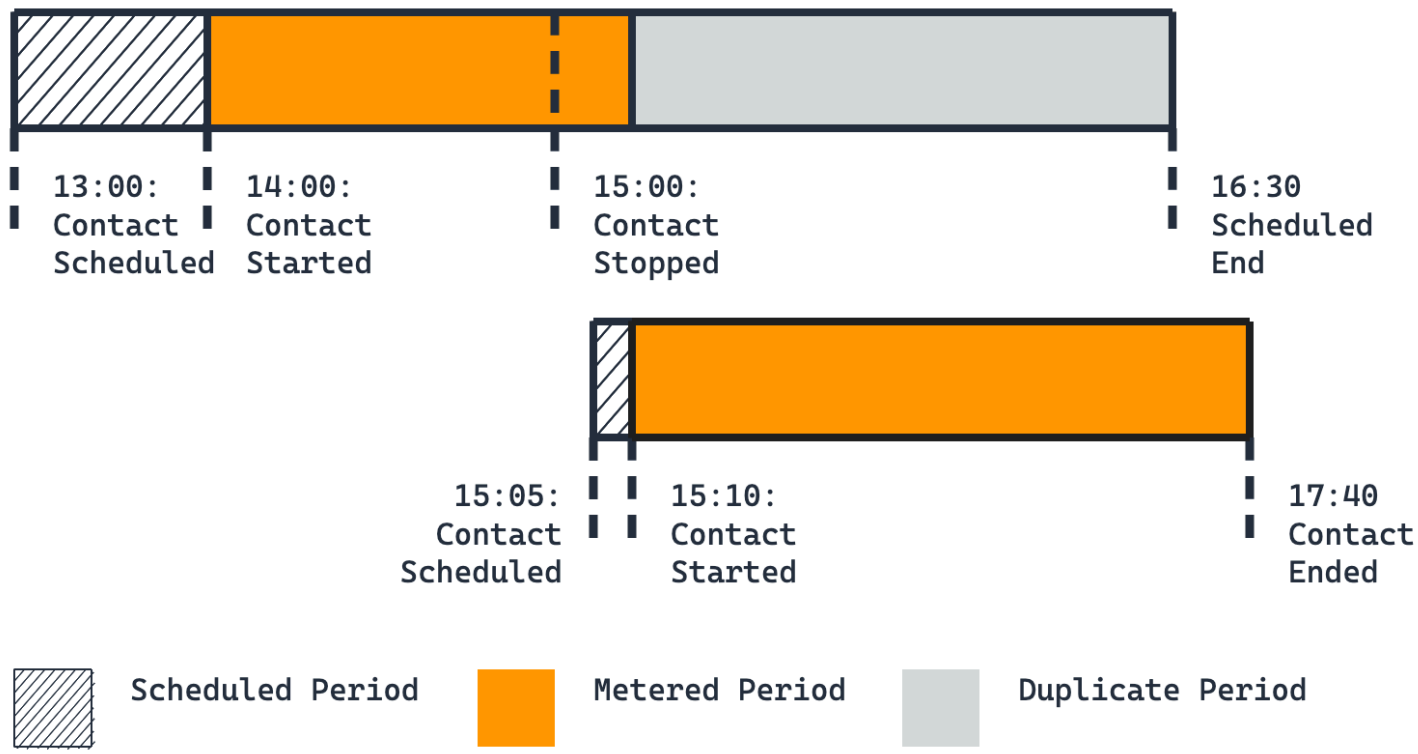
Billing breakdown:

- First contact: 150 minutes (full original duration)

You're billed for the full 150 minutes because you stopped the contact but didn't schedule any duplicate contacts to cover the remaining time (15:00-16:30). When you stop a contact without scheduling duplicates, you remain responsible for the entire originally scheduled duration.

Scenario 3: Single duplicate

You schedule a 150-minute contact on Ground Station Anytown 1 to begin at 14:00 and end at 16:30. At 15:00, you call the CancelContact API to stop your first contact. After calling CancelContact, you schedule another contact on the same Ground Station starting at 15:10 for 150 minutes.



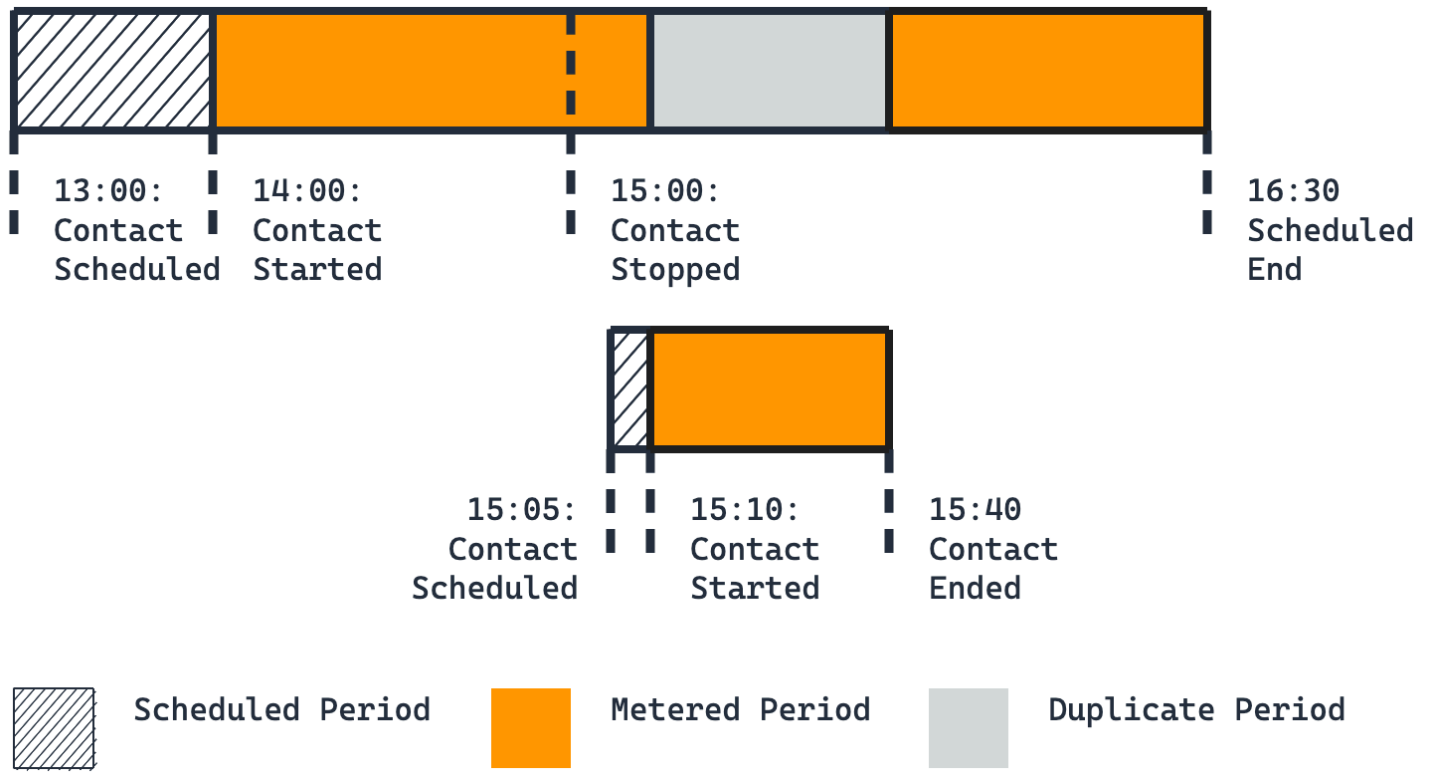
Billing breakdown:

- First contact: 70 minutes (60 minutes executed + 10 minutes of downtime before the second contact starts)
- Second contact: 150 minutes (full duration)

The second contact is a duplicate because you scheduled it after stopping the first contact. The duplicate covers the remaining time from 15:10 to 16:30, so you are only billed for the time the first contact actually ran plus the 10-minute gap between stopping and restarting.

Scenario 4: Short duplicate

You schedule a 150-minute contact on Ground Station Anytown 1 to begin at 14:00 and end at 16:30. At 15:00, you call the CancelContact API to stop your first contact. After calling CancelContact, you schedule a 30-minute contact on the same Ground Station starting at 15:10.



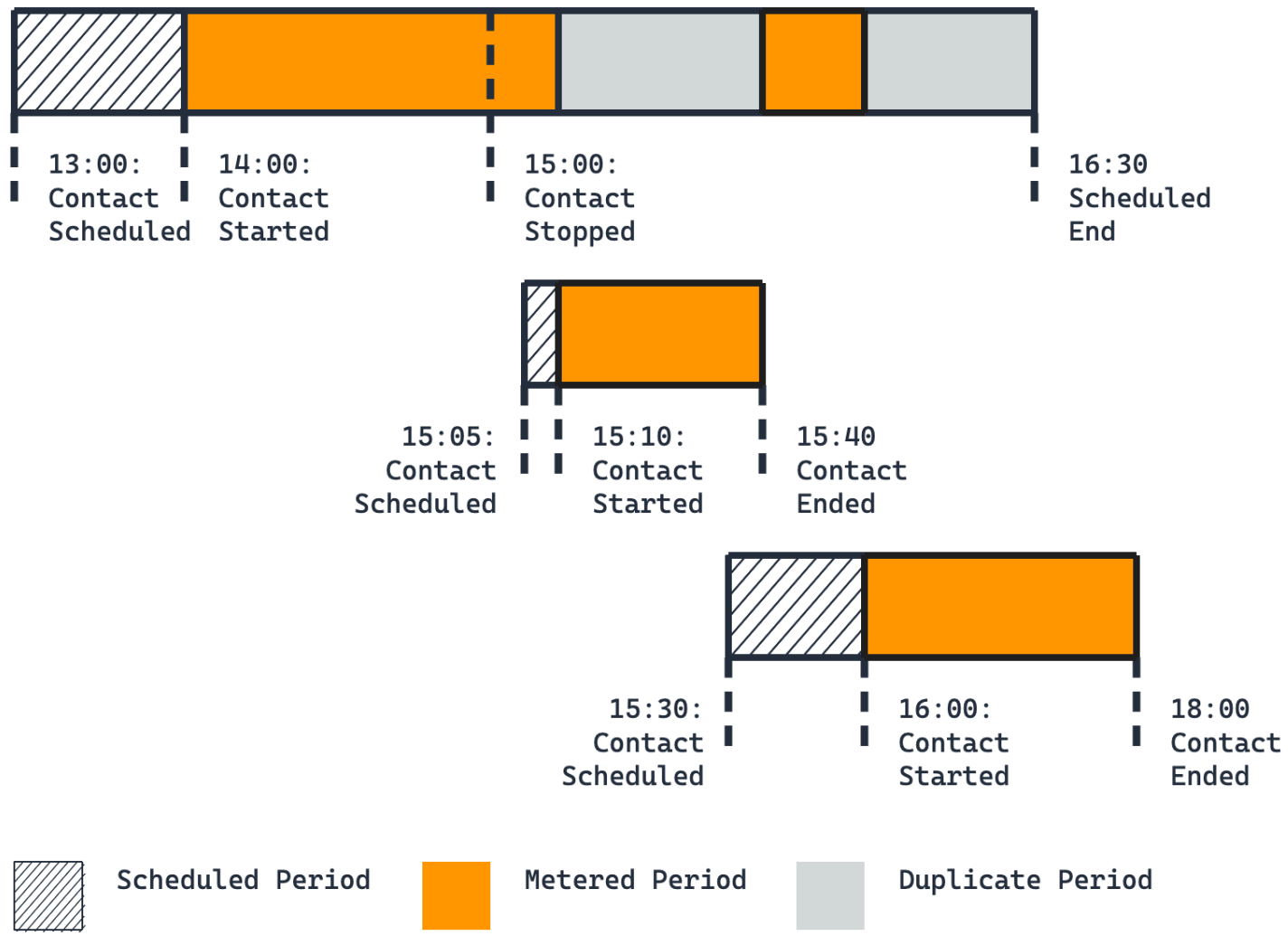
Billing breakdown:

- First contact: 120 minutes (60 minutes executed + 10 minutes of downtime before the second contact starts + 50 minutes of remaining time that the duplicate didn't cover)
- Second contact: 30 minutes (full duration)

The duplicate contact only covers 30 minutes (15:10-15:40) of the 90 minutes remaining after you stopped the first contact. You're billed for both the 10-minute gap before the duplicate starts and the 50 minutes of uncovered time after the duplicate ends (15:40-16:30).

Scenario 5: Multiple duplicates

You schedule a 150-minute contact on Ground Station Anytown 1 to begin at 14:00 and end at 16:30. At 15:00, you call the CancelContact API to stop your first contact. After calling CancelContact, you schedule a 30-minute contact on the same Ground Station starting at 15:10. Later, at 15:30, you schedule another contact starting at 16:00 for 120 minutes.



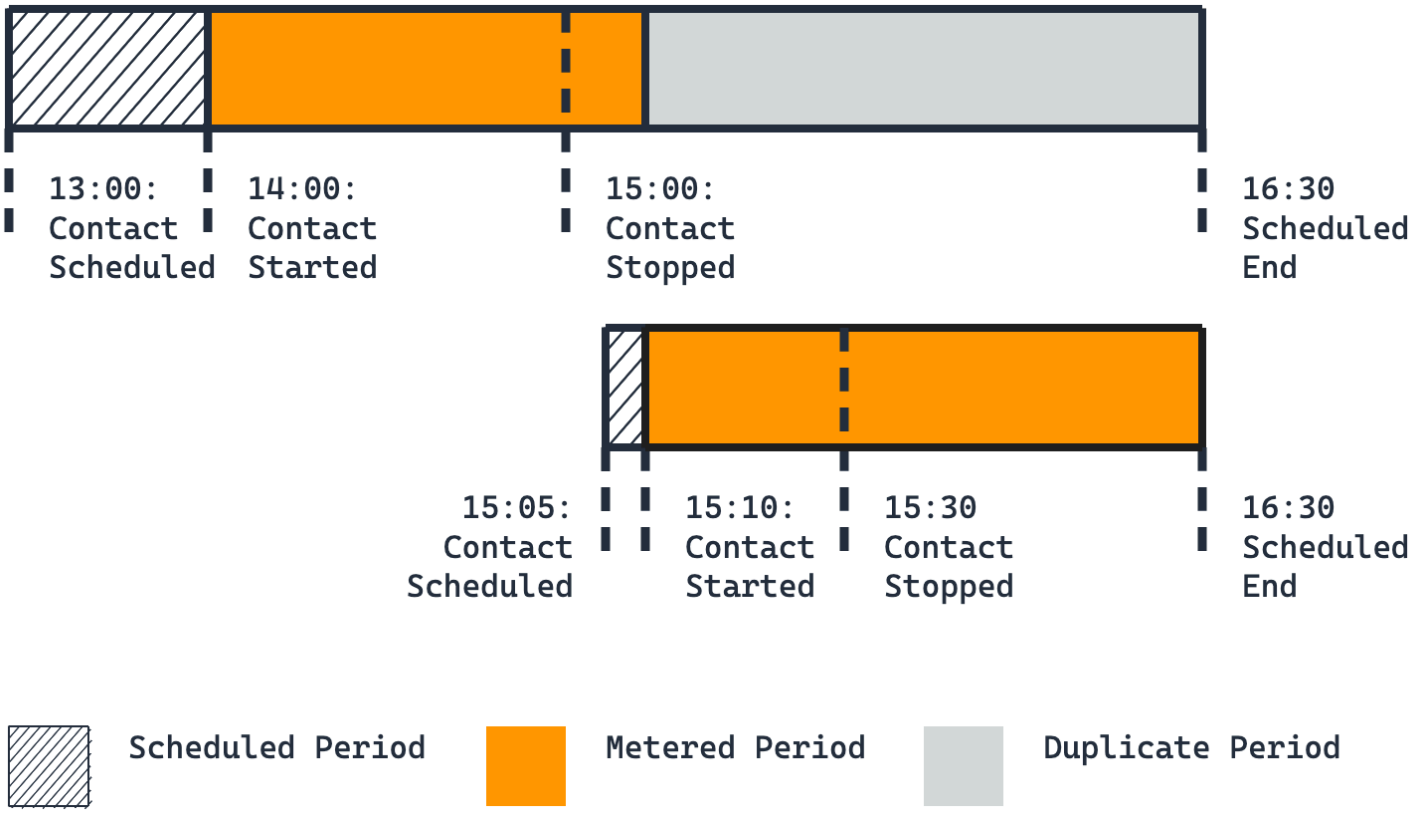
Billing breakdown:

- First contact: 90 minutes (60 minutes executed + 10 minutes of downtime before the second contact starts + 20 minutes of downtime between the second and third contacts)
- Second contact: 30 minutes (full duration)
- Third contact: 120 minutes (full duration)

Both the second and third contacts count as duplicates because you scheduled them after stopping the first contact. However, you are still billed for the gaps between contacts: 10 minutes between the first stop (15:00) and second start (15:10), and 20 minutes between the second end (15:40) and third start (16:00).

Scenario 6: Multiple stops

You schedule a 150-minute contact on Ground Station Anytown 1 to begin at 14:00 and end at 16:30. At 15:00, you call the CancelContact API to stop your first contact. After calling CancelContact, you schedule an 80-minute contact on Ground Station Anytown 1 that begins at 15:10 and ends at 16:30. At 15:30, you call the CancelContact API again, stopping your duplicate contact.



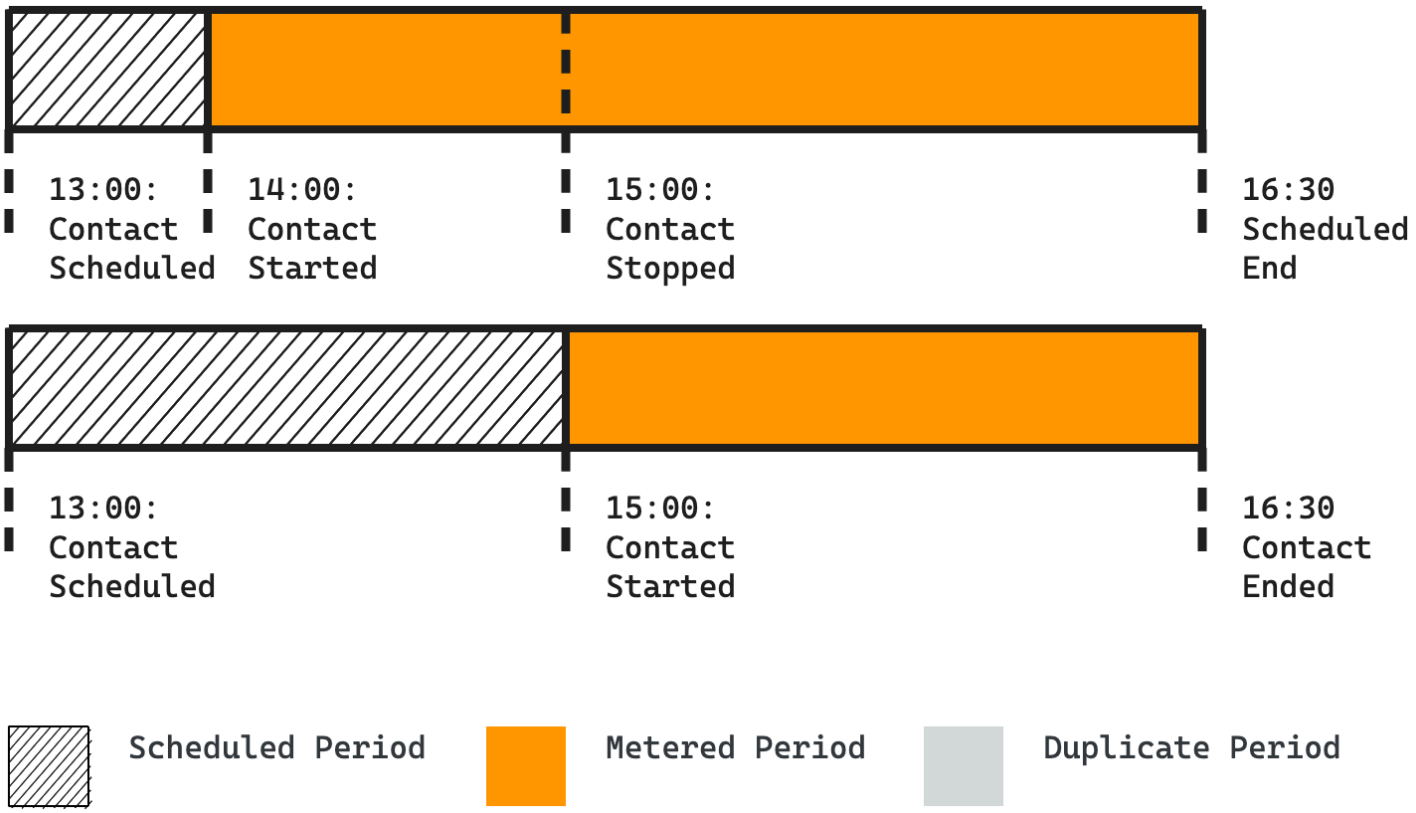
Billing breakdown:

- First contact: 70 minutes (60 minutes executed + 10 minutes of downtime before the second contact starts)
- Second contact: 80 minutes (full original duration)

The second contact is billed for its full 80-minute duration because you stopped it at 15:30, leaving 60 minutes of originally scheduled time (15:30-16:30) unfilled. Unless you schedule another duplicate contact to cover the remaining time, you are responsible for the entire duration of any stopped contact.

Scenario 7: Multi-antenna ground station with no duplicate

At 13:00, you schedule two contacts on Ground Station Anytown 1. The first is a 150-minute contact beginning at 14:00 and ending at 16:30. The second is a 90-minute contact beginning at 15:00 and ending at 16:30. At 15:00, you call the CancelContact API to stop your first contact. Ground Station Anytown 1 is a multi-antenna ground station, which allows both contacts to run simultaneously.



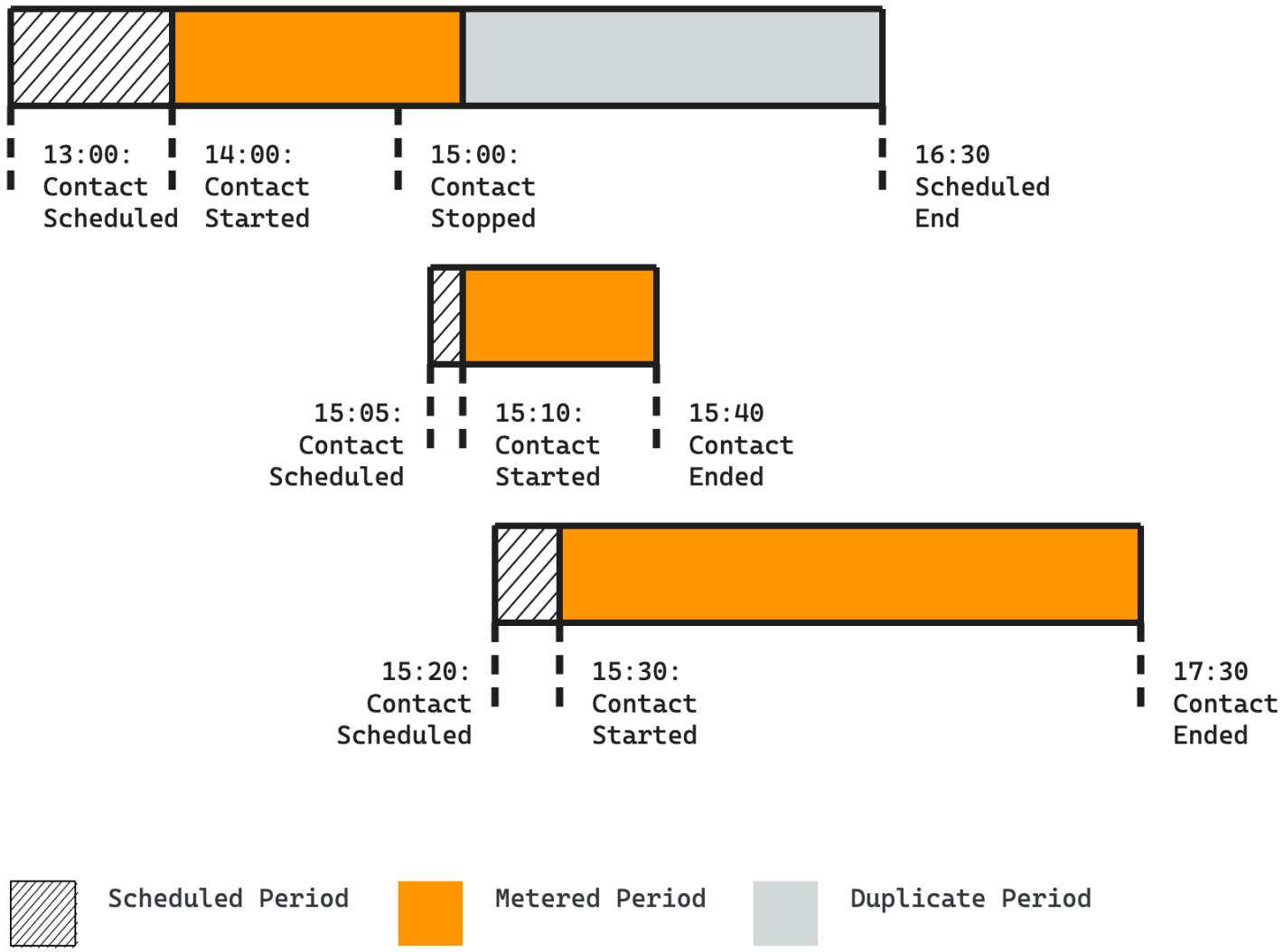
Billing breakdown:

- First contact: 150 minutes (full original duration)
- Second contact: 90 minutes (full duration)

Although the second contact overlaps with the stopped portion of the first contact, it doesn't count as a duplicate. The second contact fails to meet the first criterion for duplicates: it was scheduled at 13:00, **before** you stopped the first contact at 15:00. Because it's not a duplicate, you are billed for the full original duration of the first contact, regardless of when you stopped it.

Scenario 8: Multi-antenna ground station with duplicate contacts

You schedule a 150-minute contact on Ground Station Anytown 1 to begin at 14:00 and end at 16:30. At 15:00, you call the CancelContact API to stop your first contact. After calling CancelContact, you schedule a 30-minute contact on Ground Station Anytown 1 beginning at 15:10 and ending at 15:40. Later, you schedule another 90-minute contact on Ground Station Anytown 1 starting at 15:30 and ending at 17:00. Ground Station Anytown 1 is a multi-antenna ground station, which allows both duplicate contacts to run simultaneously with overlapping times.



Billing breakdown:

- First contact: 70 minutes (60 minutes executed + 10 minutes of downtime before the second contact starts)
- Second contact: 30 minutes (full duration)

- Third contact: 90 minutes (full duration)

Both the second and third contacts count as duplicates because you scheduled them after stopping the first contact. The 10-minute gap between stopping the first contact (15:00) and starting the second contact (15:10) represents downtime that you are billed for against the original contact.

Update contacts and contact versioning

AWS Ground Station supports updating contacts with a SCHEDULED, PREPASS, or PASS contact status. You can use the UpdateContact API to specify an ephemeris override for a contact — including azimuth/elevation, OEM, or TLE tracking data — without cancelling and rescheduling it. This is useful for geosynchronous (GEO) satellite operations where you need to retask an antenna to a different satellite during a contact, or for Launch and Early Operations (LEOPs) where pointing adjustments are required.

Each time you reserve or update a contact, AWS Ground Station creates a new contact version. Contact versions provide a history of changes made to a contact and allow you to track the status of each update.

How contact versioning works

When you call ReserveContact, AWS Ground Station creates the first version of the contact (version 1) and returns the `versionId` in the response. Each subsequent call to UpdateContact creates a new version with an incremented version number.

The DescribeContact API returns the currently `ACTIVE` contact version, including version information in the `version` field of the response. The ListContacts API also includes version information for each contact.

To view a specific version of a contact, use the DescribeContactVersion API. To list all versions of a contact, use the ListContactVersions API.

Updating a contact

You can call UpdateContact when a contact is in the SCHEDULED, PREPASS, or PASS state. The API accepts the following parameters:

- **contactId** — The identifier of the contact to update.

- **clientToken** — An idempotency token that ensures the request is processed only once. If you retry a request with the same client token, AWS Ground Station returns the original response without performing the update again. Many AWS SDKs automatically generate a client token for you if one is not provided.
- **trackingOverrides** — The new tracking configuration for the contact. This includes program track settings (azimuth/elevation, TLE, or OEM ephemeris).
- **satelliteArn** — The ARN of the satellite for the contact. When changing the target satellite along with the program track settings, provide the ARN of the new satellite. Only customers approved for azimuth/elevation pointing angles can set this value to null. All other customers must include the satellite ARN of the contact.

> **Important**
>
> The `UpdateContact` API applies all parameters in the request. Any parameter that is omitted or explicitly set to null is treated as a request to clear that value, not to leave it unchanged. For example, if you provide `trackingOverrides` but omit `satelliteArn`, the satellite ARN is cleared. Make sure to include all desired values in each update request.

You can change the target satellite during a contact by providing a new `satelliteArn` along with the corresponding `trackingOverrides`. The new satellite must be visible from the ground station for the duration of the contact, because the contact start and end time do not change with this API. The new satellite must also be onboarded to the ground station and have the licensing required by the mission profile. The mission profile of the contact cannot be changed, so switching satellites is only applicable when both satellites use the same mission profile.

> **Important**
>
> The `UpdateContact` API does not support changing the start time, end time, or mission profile of a contact. To change these values, cancel the contact and reserve a new one. The `UpdateContact` API is designed for retasking the antenna pointing configuration, such as switching between satellites or updating ephemeris data.

Important

The `UpdateContact` API does not support contacts that have a mission profile that uses Antenna Downlink Demod Decode Config configs. To change the configuration of these contacts, cancel the contact and reserve a new one.

The `UpdateContact` API returns the `contactId` and the new `versionId`. The update is processed asynchronously. Use DescribeContactVersion to check the status of the update. Some AWS SDKs and the AWS Command Line Interface provide a `ContactUpdated` waiter that polls until the version reaches ACTIVE or FAILED_TO_UPDATE status.

Note

Only one update can be in progress at a time. If the latest contact version is in the UPDATING state, the API returns a `ConflictException`. Wait for the current update to reach ACTIVE or FAILED_TO_UPDATE status before submitting another update.

Contact version statuses

Each contact version has one of the following statuses:

Status	Description
UPDATING	The version is being applied to the contact. The update has been submitted and is being processed by AWS Ground Station.
ACTIVE	The version is the currently active configuration for the contact. The ground station is using this version's settings.
SUPERSEDED	The version was previously active but has been replaced by a newer version.
FAILED_TO_UPDATE	The update could not be applied. The contact reverts to the previously active version. Check the `failureCodes` and `failureMessage` fields for details.

Code examples

The following examples demonstrate how to use the contact versioning APIs with the AWS SDK for Python (Boto3).

Example: Update a contact

The following example updates a contact with new tracking overrides and waits for the update to complete using the Boto3 `ContactUpdated` waiter.

```
import boto3
import uuid

# Create AWS Ground Station client
ground_station_client = boto3.client("groundstation")

# The contact ID of an existing scheduled contact to update
contact_id = "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"

# Generate a unique client token for idempotency.
# If you retry the same request with the same client token,
# the API returns the same response without creating a duplicate version.
client_token = str(uuid.uuid4())

# Update the contact to use a different TLE ephemeris for tracking.
# The UpdateContact API applies all parameters in the request.
# Any parameter set to null is treated as a request to clear that value,
# not to leave it unchanged. Include all desired values in each request.
print(f"Updating contact {contact_id}...")

update_response = ground_station_client.update_contact(
    contactId=contact_id,
    clientToken=client_token,
    satelliteArn="arn:aws:groundstation::111122223333:satellite/a88611b0-f755-404e-
b60d-57d8aEXAMPLE",
    trackingOverrides={
        "programTrackSettings": {
            "tle": {"ephemerisId": "b2c3d4e5-6789-01ab-cdef-EXAMPLE22222"}
        }
    },
)

contact_id = update_response["contactId"]
```

```
version_id = update_response["versionId"]
print(f"Update submitted. Contact: {contact_id}, Version: {version_id}")

# Wait for the update to complete using the ContactUpdated waiter.
# The waiter polls DescribeContactVersion until the version reaches
# ACTIVE (success) or FAILED_TO_UPDATE (failure) status.
# The waiter raises WaiterError if the version reaches FAILED_TO_UPDATE
# or if MaxAttempts is exceeded, so we use try/except to handle both cases.
print("Waiting for update to complete...")

from botocore.exceptions import WaiterError

waiter = ground_station_client.get_waiter("contact_updated")

try:
    waiter.wait(
        contactId=contact_id,
        versionId=version_id,
        WaiterConfig={
            "Delay": 5,
            "MaxAttempts": 180,
        },
    )
    print(f"Contact updated successfully. Version {version_id} is now active.")
except WaiterError as e:
    # WaiterError is raised when the version reaches FAILED_TO_UPDATE
    # or when MaxAttempts is exceeded. Retrieve the current version to inspect.
    version_response = ground_station_client.describe_contact_version(
        contactId=contact_id,
        versionId=version_id,
    )
    version_status = version_response["version"]["status"]
    if version_status == "FAILED_TO_UPDATE":
        failure_codes = version_response["version"].get("failureCodes", [])
        failure_message = version_response["version"].get("failureMessage", "")
        print(f"Update failed. Codes: {failure_codes}, Message: {failure_message}")
    else:
        print(f"Waiter timed out. Current version status: {version_status}. Error:
 {e}")
```

Example: Describe a contact version

The following example retrieves the details of a specific contact version, including its status, configuration, and any failure information.

```
import boto3

# Create AWS Ground Station client
ground_station_client = boto3.client("groundstation")

contact_id = "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
version_id = 2

# Describe a specific version of a contact.
# Use this API to check the status of an update or to view the
# configuration that was active at a specific point in time.
print(f"Describing version {version_id} of contact {contact_id}...")

response = ground_station_client.describe_contact_version(
    contactId=contact_id,
    versionId=version_id,
)

# Display version details
version = response["version"]
print(f"Version ID: {version['versionId']}")
print(f"Status: {version['status']}")
print(f"Created: {version.get('created', 'N/A')}")

if version.get("activated"):
    print(f"Activated: {version['activated']}")

if version.get("superseded"):
    print(f"Superseded: {version['superseded']}")

# Display contact details for this version
print(f"\nContact ID: {response['contactId']}")
print(f"Contact Status: {response['contactStatus']}")
print(f"Ground Station: {response['groundStation']}")
print(f"Start Time: {response['startTime']}")
print(f"End Time: {response['endTime']}")

if response.get("satelliteArn"):
```

```
    print(f"Satellite ARN: {response['satelliteArn']}")

if response.get("trackingOverrides"):
    print(f"Tracking Overrides: {response['trackingOverrides']}")

# Check for failure details if the version failed to update
if version["status"] == "FAILED_TO_UPDATE":
    failure_codes = version.get("failureCodes", [])
    failure_message = version.get("failureMessage", "")
    print(f"\nFailure Codes: {failure_codes}")
    print(f"Failure Message: {failure_message}")
```

Example: List contact versions

The following example lists all versions of a contact to view the full history of changes, using pagination to handle large result sets.

```
import boto3

# Create AWS Ground Station client
ground_station_client = boto3.client("groundstation")

contact_id = "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"

# List all versions of a contact to view the full history of changes.
# Results are paginated. Use the nextToken to retrieve additional pages.
print(f"Listing versions for contact {contact_id}...")

paginator = ground_station_client.get_paginator("list_contact_versions")
page_iterator = paginator.paginate(
    contactId=contact_id,
    PaginationConfig={
        "MaxItems": 100,
        "PageSize": 20,
    },
)

for page in page_iterator:
    for version in page["contactVersionsList"]:
        version_id = version["versionId"]
        status = version["status"]
        created = version.get("created", "N/A")
```

```
        print(f"  Version {version_id}: status={status}, created={created}")

        if version.get("activated"):
            print(f"    Activated: {version['activated']}")

        if version.get("superseded"):
            print(f"    Superseded: {version['superseded']}")

        if status == "FAILED_TO_UPDATE":
            failure_codes = version.get("failureCodes", [])
            failure_message = version.get("failureMessage", "")
            print(f"    Failure Codes: {failure_codes}")
            print(f"    Failure Message: {failure_message}")

        if status == "UPDATING":
            print(f"    Update is currently in progress.")
```

Considerations

- Contacts created before the contact versioning feature was introduced do not have version information. Calling DescribeContactVersion or ListContactVersions for these contacts returns a `ResourceNotFoundException`.
- When you update the tracking overrides of an in-progress contact, there is a brief transition period while the antenna adjusts to the new pointing configuration. During this time, signal reception or transmission may be interrupted.
- Contact versions cannot be deleted. Use ListContactVersions to view the full history of changes made to a contact.
- The `UpdateContact` API can only be called from the scheduling region of the contact.

Use the AWS Ground Station digital twin feature

The digital twin feature for AWS Ground Station provides you with an environment where you can test and integrate your satellite mission management and command and control software. The digital twin feature allows you to test scheduling, verification of configurations, and proper error handling without using production antenna capacity. Testing your AWS Ground Station integration with the digital twin feature enables you to have increased confidence in your system's ability to manage your satellite operations smoothly. It also allows you to test AWS Ground Station APIs without using production capacity or requiring spectrum licensing.

To get started, follow Onboard satellite, requesting to be onboarded to the digital twin feature. Once your satellite is onboarded to the digital twin feature, you can schedule contacts against digital twin ground stations. The list of ground stations that you have access to can be retrieved via the AWS SDK ListGroundStations response. Digital twin ground stations are exact copies of the ground stations listed in AWS Ground Station Locations with a modifying prefix to Ground Station Name of “Digital Twin ”. This includes their antenna capabilities and metadata, including, but not limited to, site mask and actual GPS coordinates. At this time, the digital twin feature does not support data delivery as described in Work with dataflows.

Once onboarded, the digital twin feature emits the same Amazon EventBridge events and API responses as the production service as described in Automate AWS Ground Station with Events. These events will allow you to fine tune your configurations and dataflow endpoint groups.

AWS Ground Station Dedicated Antennas

AWS Ground Station Dedicated Antennas are custom-built antenna systems that AWS manages on your behalf. Unlike the public AWS Ground Station antennas where you share antenna time with other customers, a Dedicated Antenna provides you with dedicated access to an antenna built to your specifications. You can have one or more Dedicated Antennas. A Dedicated Antenna is connected to the global AWS network, and you interact with it using the same AWS Ground Station APIs and workflows that you use with public antennas, with the addition of enhanced visibility into antenna utilization.

What is a Dedicated Antenna

A Dedicated Antenna is a physical antenna system that is custom-built for your organization and fully managed by AWS. You can have one or more Dedicated Antennas. A Dedicated Antenna differs from the public AWS Ground Station antennas in the following ways:

- **Custom-built** — A Dedicated Antenna is built to your specifications. The capabilities of a Dedicated Antenna are not limited to the capabilities of public antennas as described in AWS Ground Station Site Capabilities.
- **Flexible location** — Dedicated Antennas are not restricted to existing AWS Ground Station antenna locations. A Dedicated Antenna can be connected to any existing AWS Region, including regions where AWS Ground Station is not currently available. You can work with AWS to determine the location and region that meet your requirements.
- **Dedicated access** — You have dedicated access to the antenna rather than sharing antenna time with other AWS Ground Station customers on a per-contact basis.

All AWS Ground Station antennas, including Dedicated Antennas, are fully managed by AWS. This includes maintenance and connectivity to the global AWS network. You interact with a Dedicated Antenna using the same AWS Ground Station APIs and workflows that you use with public antennas. You schedule contacts, configure mission profiles, and deliver data in the same way.

A Dedicated Antenna can be shared by multiple AWS accounts, where the customer who holds the Dedicated Antenna contract chooses which accounts to onboard. Each onboarded account can schedule contacts on the antenna independently, and has visibility into reservations across all accounts that share the antenna.

To learn more about Dedicated Antennas or to get started, contact AWS Support through the AWS Support Center Console.

Enhanced reservation visibility

When you use the ListGroundStationReservations API against your Dedicated Antenna, you see additional information that is not available on public antennas. The following table summarizes the behavioral differences of the ListGroundStationReservations API for Dedicated Antennas in comparison to public AWS Ground Station antennas.

Behavior	Dedicated Antenna	Public antenna
Your own contacts	Visible with full details, including `contactId`	Visible with full details, including `contactId`
Other accounts' contacts	Visible with time slots only, without `contactId`	Not visible
Maintenance windows	Visible with `PLANNED` or `UNPLANNED` `maintenanceType`	Not visible

Maintenance windows represent periods when the antenna is unavailable for satellite communication. The `maintenanceType` field indicates whether the maintenance was `PLANNED` or `UNPLANNED`. When unplanned maintenance is scheduled, contacts that overlap with the maintenance window may be cancelled by AWS Ground Station.

When you view contacts from other AWS accounts that share your Dedicated Antenna, the reservation includes the time slot and antenna information, but the `contactId` is not included.

Important

Enhanced reservation visibility applies only to your Dedicated Antenna. When you use public AWS Ground Station antennas, you have the same visibility as any other customer. You do not see maintenance windows or reservations from other accounts on public antennas.

For more information about listing reservations, see View ground station reservations.

Related resources

- View ground station reservations
- AWS Ground Station Locations
- AWS Ground Station Site Capabilities
- ListGroundStationReservations in the *AWS Ground Station API Reference*

Understand monitoring with AWS Ground Station

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS Ground Station. AWS provides the following monitoring tools to watch AWS Ground Station, report when something is wrong, and take automatic actions when appropriate.

- *Amazon EventBridge Events* delivers a near real-time stream of system events that describe changes in AWS resources. EventBridge Events enables automated event-driven computing, as you can write rules that watch for certain events and trigger automated actions in other AWS services when these events happen. For more information about EventBridge Events, see the Amazon EventBridge Events User Guide.
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information about AWS CloudTrail, see the AWS CloudTrail User Guide.
- *Amazon CloudWatch Metrics* captures metrics for your scheduled contacts when using AWS Ground Station. CloudWatch Metrics enables you to analyze data based on your channel, polarization, and satellite ID to identify signal strength and errors in your contacts. For more information, see Using Amazon CloudWatch metrics.
- *AWS User Notifications* can be used to set up delivery channels to get notified about AWS Ground Station events. You receive a notification when an event matches a rule that you specify. You can receive notifications for events through multiple channels, including email, Amazon Q Developer in chat applications chat notifications, or AWS Console Mobile Application push notifications. You can also see notifications in the AWS Console Notification center. User Notifications support aggregation, which can reduce the number of notifications you receive during specific events.

Use the following topics to monitor AWS Ground Station.

Topics

- Automate AWS Ground Station with Events
- Log AWS Ground Station API calls with AWS CloudTrail
- View metrics with Amazon CloudWatch

Automate AWS Ground Station with Events

> **Note**
>
> This document uses the term “event” throughout. CloudWatch Events and EventBridge are the same underlying service and API. Rules to match incoming events and route them to targets for processing can be built using either service.

Events enable you to automate your AWS services and respond automatically to system events such as application availability issues or resource changes. Events from AWS services are delivered in near real time. You can write simple rules to indicate which events are of interest to you, and what automated actions to take when an event matches a rule. Some of the actions that can be automatically triggered include the following:

- Invoking an AWS Lambda function
- Invoking Amazon EC2 Run Command
- Relaying the event to Amazon Kinesis Data Streams
- Activating an AWS Step Functions state machine
- Notifying an Amazon SNS topic or an Amazon SQS queue

Some examples of using events with AWS Ground Station include:

- Invoking a Lambda function to automate the starting and stopping of Amazon EC2 instances based off the event state.
- Publishing to an Amazon SNS topic whenever a contact changes states. These topics can be set up to send out email notices at the beginning or end of contacts.

For more information, see the Amazon EventBridge Events User Guide.

AWS Ground Station Event Types

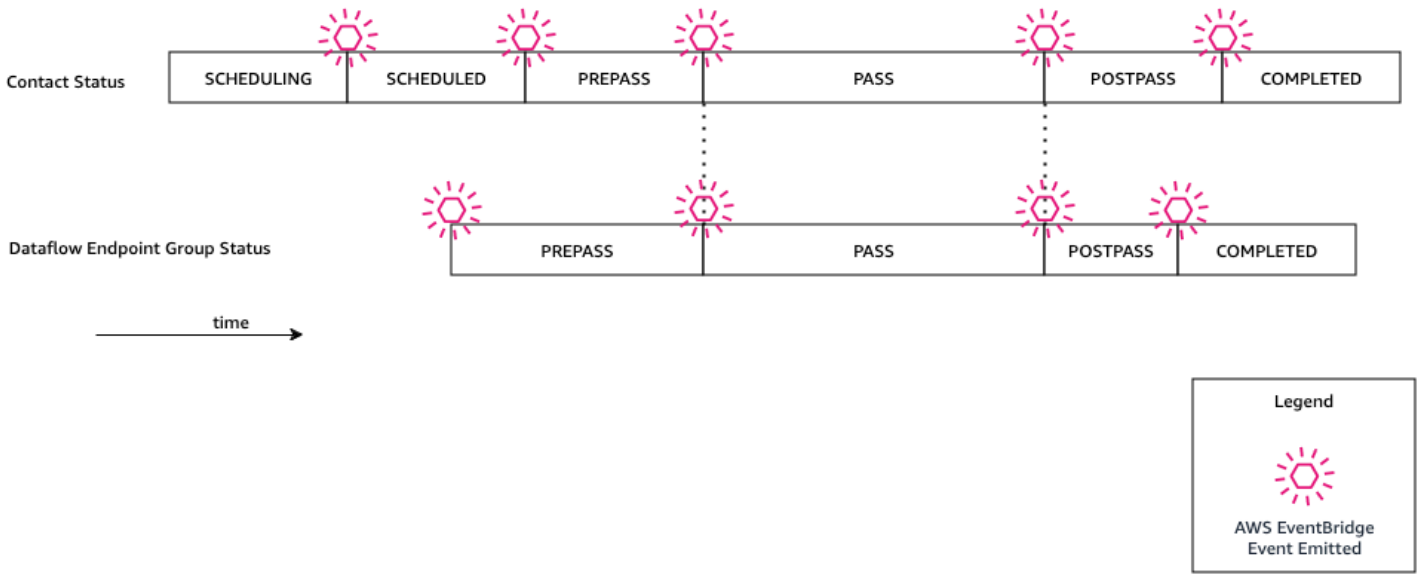
> **Note**
>
> All events generated by AWS Ground Station have "aws.groundstation" as the value for "source".

AWS Ground Station emits events related to state changes to support your ability to customize your automation. Currently, AWS Ground Station supports contact state change events, dataflow endpoint group change events, and ephemeris state change events. The following sections provide detailed information about each type.

Contact Event Timeline

AWS Ground Station emits events when your contact changes states. For more information on what those state changes are, and what the states themselves mean, see Understand contact lifecycle . Any dataflow endpoint groups being used in your contact have an independent set of events that are also emitted. During that same timeframe, we also emit events for your dataflow endpoint group. The precise time of the pre-pass and post-pass events are configurable by you as you set up your mission profile and dataflow endpoint group.

The following diagram shows the statuses and events emitted for a nominal contact and its associated dataflow endpoint group.



Ground Station Contact State Change

If you want to perform a specific action when an upcoming contact is changing states, you can set up a rule to automate this action. This is helpful for when you want to receive notifications about the state changes of your contact. If you would like to change when you receive these events, you can modify your mission profile's contactPrePassDurationSeconds and contactPostPassDurationSeconds. The events are sent to the region that the contact was scheduled from.

An example event is provided below.

```
{
    "version": "0",
    "id": "01234567-0123-0123",
    "account": "123456789012",
    "time": "2019-05-30T17:40:30Z",
    "region": "us-west-2",
    "source": "aws.groundstation",
    "resources": [
        "arn:aws:groundstation:us-
west-2:123456789012:contact/11111111-1111-1111-1111-111111111111"
    ],
    "detailType": "Ground Station Contact State Change",
    "detail": {
        "contactId": "11111111-1111-1111-1111-111111111111",
        "groundstationId": "Ground Station 1",
        "missionProfileArn": "arn:aws:groundstation:us-west-2:123456789012:mission-
profile/11111111-1111-1111-1111-111111111111",
        "satelliteArn":
 "arn:aws:groundstation::123456789012:satellite/11111111-1111-1111-1111-111111111111",
        "contactStatus": "PASS"
    }
}
```

The possible values for `contactStatus` are defined in the section called “AWS Ground Station contact statuses”.

Ground Station Dataflow Endpoint Group State Change

If you want to perform an action when your dataflow endpoint group is being used to receive data, you can set up a rule to automate this action. This will allow you to perform different actions in response to the dataflow endpoint group status changing states. If you would like to change when you receive these events, use a dataflow endpoint group with different contactPrePassDurationSeconds and contactPostPassDurationSeconds. This event will be sent to the region of the dataflow endpoint group.

An example is provided below.

```
{
```

```
    "version": "0",
    "id": "01234567-0123-0123",
    "account": "123456789012",
    "time": "2019-05-30T17:40:30Z",
    "region": "us-west-2",
    "source": "aws.groundstation",
    "resources": [
        "arn:aws:groundstation:us-west-2:123456789012:dataflow-endpoint-group/
bad957a8-1d60-4c45-a92a-39febd98921d",
        "arn:aws:groundstation:us-west-2:123456789012:contact/98ddd10f-f2bc-479c-
bf7d-55644737fb09",
        "arn:aws:groundstation:us-west-2:123456789012:mission-profile/c513c84c-
eb40-4473-88a2-d482648c9234"
    ],
    "detailType": "Ground Station Dataflow Endpoint Group State Change",
    "detail": {
        "dataflowEndpointGroupId": "bad957a8-1d60-4c45-a92a-39febd98921d",
        "groundstationId": "Ground Station 1",
        "contactId": "98ddd10f-f2bc-479c-bf7d-55644737fb09",
        "dataflowEndpointGroupArn": "arn:aws:groundstation:us-
west-2:680367718957:dataflow-endpoint-group/bad957a8-1d60-4c45-a92a-39febd98921d",
        "missionProfileArn": "arn:aws:groundstation:us-west-2:123456789012:mission-
profile/c513c84c-eb40-4473-88a2-d482648c9234",
        "dataflowEndpointGroupState": "PREPASS"
    }
}
```

Possible states for the `dataflowEndpointGroupState` include PREPASS, PASS, POSTPASS, and COMPLETED.

Ephemeris Events

Ground Station Ephemeris State Change

If you want to perform an action when an ephemeris changes state, you can set up a rule to automate this action. This allows you to perform different actions in response to an ephemeris changing state. For example, you can perform an action when an ephemeris has completed validation, and it is now ENABLED. Notification for this event will be sent to the region were the ephemeris was uploaded.

An example is provided below.

```
{
    "id": "7bf73129-1428-4cd3-a780-95db273d1602",
    "detail-type": "Ground Station Ephemeris State Change",
    "source": "aws.groundstation",
    "account": "123456789012",
    "time": "2019-12-03T21:29:54Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:groundstation::123456789012:satellite/10313191-c9d9-4ecb-a5f2-
bc55cab050ec",
        "arn:aws:groundstation::123456789012:ephemeris/111111-cccc-bbbb-a555-
bcccca005000"
    ],
    "detail": {
        "ephemerisStatus": "ENABLED",
        "ephemerisId": "111111-cccc-bbbb-a555-bcccca005000",
        "satelliteId": "10313191-c9d9-4ecb-a5f2-bc55cab050ec"
    }
}
```

Possible states for the `ephemerisStatus` include `ENABLED`, `VALIDATING`, `INVALID`, `ERROR`, `DISABLED`, `EXPIRED`

Log AWS Ground Station API calls with AWS CloudTrail

AWS Ground Station is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS Ground Station. CloudTrail captures all API calls for AWS Ground Station as events. The calls captured include calls from the AWS Ground Station console and code calls to the AWS Ground Station API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS Ground Station. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to AWS Ground Station, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

AWS Ground Station Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS Ground Station, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for AWS Ground Station, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All AWS Ground Station actions are logged by CloudTrail and are documented in the AWS Ground Station API Reference. For example, calls to the `ReserveContact`, `CancelContact` and `ListConfigs` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Understanding AWS Ground Station Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `ReserveContact` action.

Example: ReserveContact


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:sts::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2019-05-15T21:11:59Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::123456789012:role/Alice",
                "accountId": "123456789012",
                "userName": "Alice"
            }
        }
    },
    "eventTime": "2019-05-15T21:14:37Z",
    "eventSource": "groundstation.amazonaws.com",
    "eventName": "ReserveContact",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "Mozilla/5.0 Gecko/20100101 Firefox/123.0",
    "requestParameters": {
```

```
        "satelliteArn":
 "arn:aws:groundstation::123456789012:satellite/11111111-2222-3333-4444-555555555555",
        "groundStation": "Ohio 1",
        "startTime": 1558356107,
        "missionProfileArn": "arn:aws:groundstation:us-east-2:123456789012:mission-
profile/11111111-2222-3333-4444-555555555555",
        "endTime": 1558356886
    },
    "responseElements": {
        "contactId": "11111111-2222-3333-4444-555555555555"
    },
    "requestID": "11111111-2222-3333-4444-555555555555",
    "eventID": "11111111-2222-3333-4444-555555555555",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "11111111-2222-3333-4444-555555555555"
}
```


View metrics with Amazon CloudWatch

During a contact, AWS Ground Station automatically captures and sends data to CloudWatch for analysis. Your data can be viewed in the Amazon CloudWatch console. For more information about accessing and CloudWatch Metrics, see Using Amazon CloudWatch Metrics.

The AWS Ground Station telemetry feature can also be used to receive near real-time metrics during contacts. CloudWatch metrics are not available in near real-time and may have delays in delivery. CloudWatch also aggregates metrics over a one-second period, potentially reducing data granularity. The telemetry feature provides the individual metrics and delivers them in near real-time directly to your AWS account. For more information, see Work with telemetry.

Important

AWS Ground Station emits CloudWatch metrics to the AWS region associated with the contact's ground station location, not the AWS region from which the contact was scheduled. To view metrics for a contact, you must access CloudWatch in the ground station region. For information about which AWS region is associated with each ground station location, see Finding the AWS region for a ground station location. To receive telemetry data in the region from which you schedule your contacts, you can use the AWS Ground Station telemetry feature. See Work with telemetry for more details.

AWS Ground Station Metrics and Dimensions

What metrics are available?

The following metrics are available from AWS Ground Station.

> **Note**
>
> The specific metrics emitted depend on the AWS Ground Station capabilities being used. Depending on your configuration, only a subset of the below metrics may be emitted.

Metric	Metric Dimensions	Description
`AzimuthAngle`	SatelliteId	The azimuth angle of the antenna. True north is 0 degrees and east is 90 degrees. Units: degrees
`BitErrorRate`	Channel, Polarization, SatelliteId	The error rate on bits in a given number of bit transmissions. Bit errors are caused by noise, distortion, or interference Units: Bits errors per unit time
`BlockErrorRate`	Channel, Polarization, SatelliteId	The error rate of blocks in a given number of

Metric	Metric Dimensions	Description
		received blocks. Block errors are caused by interference. Units: Erroneous blocks / Total number of blocks
CarrierFrequencyRecovery_Cn0	Category, Config, SatelliteId	Carrier to noise density ratio per unit bandwidth. Units: decibel-Hertz (dB-Hz)
CarrierFrequencyRecovery_Locked	Category, Config, SatelliteId	Set to 1 when the demodulator carrier frequency recovery loop is locked and 0 when unlocked. Units: unitless

Metric	Metric Dimensions	Description
CarrierFrequencyRecovery_OffsetFrequency_Hz	Category, Config, SatelliteId	The offset between the estimated signal center and ideal center frequency. This is caused by Doppler shift and local oscillator offset between spacecraft and antenna system. Units: hertz (Hz)
ElevationAngle	SatelliteId	The elevation angle of the antenna. The horizon is 0 degrees and zenith is 90 degrees. Units: degrees
Es/N0	Channel, Polarization, SatelliteId	The ratio of energy per symbol to noise power spectral density. Units: decibels (dB)

Metric	Metric Dimensions	Description
ReceivedPower	Polarization, SatelliteId	The measured signal strength in the demodulator/decoder. Units: decibels relative to milliwatts (dBm)
SymbolTimingRecovery_ErrorVectorMagnitude	Category, Config, SatelliteId	The error vector magnitude between received symbols and ideal constellation points. Units: percent
SymbolTimingRecovery_Locked	Category, Config, SatelliteId	Set to 1 when the demodulator symbol timing recovery loop is locked and 0 when unlocked Units: unitless

Metric	Metric Dimensions	Description
SymbolTimingRecovery_OffsetSymbolRate	Category, Config, SatelliteId	The offset between the estimated symbol rate and ideal signal symbol rate. This is caused by Doppler shift and local oscillator offset between spacecraft and antenna system. Units: symbols/second

What dimensions are used for AWS Ground Station?

You can filter AWS Ground Station data using the following dimensions.

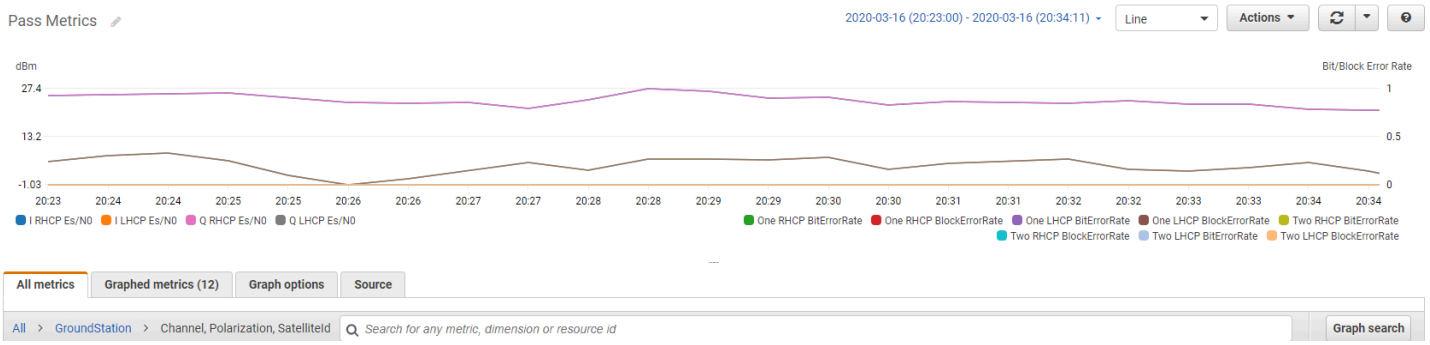
Dimension	Description
Category	Demodulation or Decode.
Channel	The channels for each contact include One, Two, I (in-phase), and Q (quadrature).
Config	An antenna downlink demod decode config arn.
Polarization	The polarization for each contact include LHCP (Left Hand Circular Polarized) or RHCP (Right Hand Circular Polarized).

Dimension	Description
`SatelliteId`	The satellite ID contains the ARN of the satellite for your contacts.

Viewing Metrics

When viewing graphed metrics, it is important to note that the aggregation window determines how your metrics will be displayed. Each metric in a contact can be displayed as data per second for 3 hours after the data is received. Your data will be aggregated by CloudWatch Metrics as data per minute after that 3-hour period has elapsed. If you need to view your metrics on a data per second measurement, it is recommended to view your data within the 3-hour period after the data is received or persist it outside of CloudWatch Metrics. For more information on CloudWatch retention, see Amazon CloudWatch concepts - Metric retention.

In addition, any data captured within the first 60 seconds will not contain enough information to produce meaningful metrics, and will likely not be displayed. In order to view meaningful metrics, it is recommended to view your data after 60 seconds has passed.

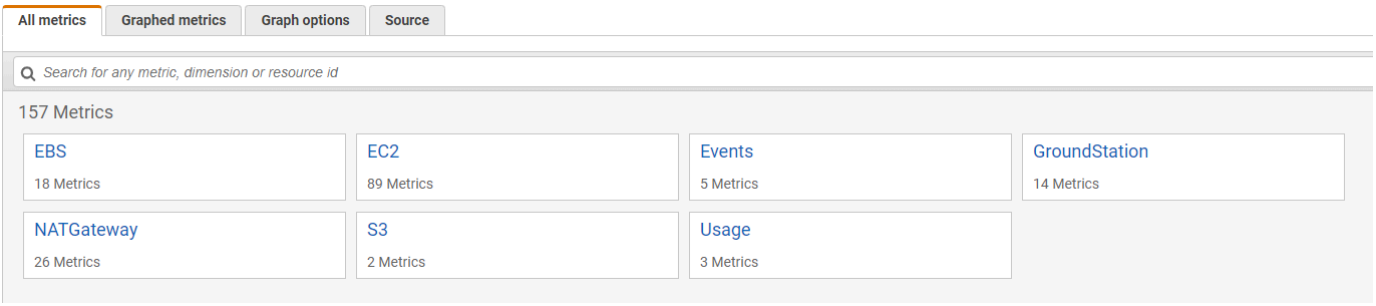


For more information about graphing AWS Ground Station metrics in CloudWatch, see Graphing Metrics.

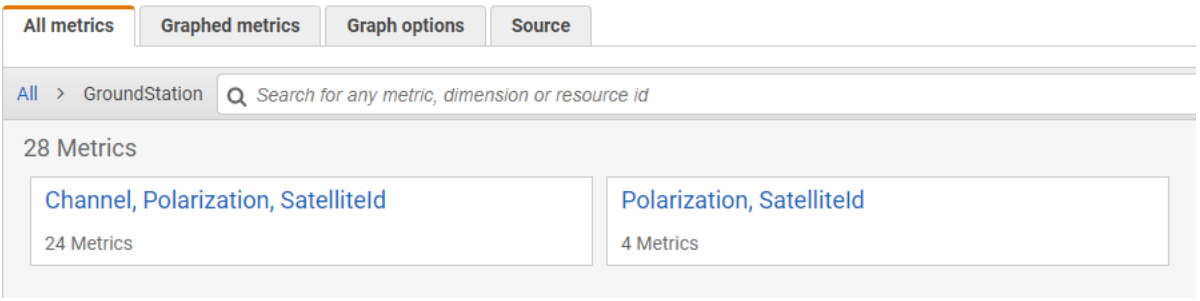
To view metrics using the console

1. Determine the AWS region associated with your ground station location. AWS Ground Station emits CloudWatch metrics in the region associated with your contact's ground station location. For the list of ground station locations and their associated AWS regions, see Finding the AWS region for a ground station location.
2. Open the CloudWatch console.

3. In the navigation pane, choose **Metrics**.
4. Select the **GroundStation** namespace.



5. Select your desired metric dimensions (for example, **Channel, Polarization, SatelliteId**).



6. The **All metrics** tab displays all metrics for that dimension in the namespace. You can do the following:

   a. To sort the table, use the column heading.

   b. To graph a metric, select the checkbox associated with the metric. To select all metrics, select the checkbox in the heading row of the table.

   c. To filter by resource, choose the resource ID and then choose **Add to search**.

   d. To filter by metric, choose the metric name and then choose **Add to search**.

To view metrics using AWS CLI

AWS Ground Station emits CloudWatch metrics in the region associated with your contact's ground station location. For the list of ground station locations their associated AWS regions, Finding the AWS region for a ground station location. Replace *ground-station-region-code* with the AWS region code for your ground station location (for example, `us-west-2` for Oregon 1, Hawaii 1, or Alaska 1). All subsequent AWS CLI commands in this procedure must use the same region.

1. Ensure that AWS CLI is installed. For information about installing AWS CLI, see Installing the AWS CLI version 2.
2. Identify the AWS region associated with your ground station location.
3. Use the get-metric-data method of the CloudWatch CLI to generate a file that can be modified to specify the metrics that you're interested in, and then be used to query for those metrics.

   To do this, run the following: `aws cloudwatch get-metric-data --region` *`ground-station-region-code`* `--generate-cli-skeleton`. This will generate output similar to:

```
{
   "MetricDataQueries": [
      {
         "Id": "",
         "MetricStat": {
            "Metric": {
               "Namespace": "",
               "MetricName": "",
               "Dimensions": [
                  {
                     "Name": "",
                     "Value": ""
                  }
               ]
            },
            "Period": 0,
            "Stat": "",
            "Unit": "Seconds"
         },
         "Expression": "",
         "Label": "",
         "ReturnData": true,
         "Period": 0,
         "AccountId": ""
      } ],
   "StartTime": "1970-01-01T00:00:00",
   "EndTime": "1970-01-01T00:00:00",
   "NextToken": "",
   "ScanBy": "TimestampDescending",
   "MaxDatapoints": 0,
   "LabelOptions": {
      "Timezone": ""
```

```
            }
        }
```

4. List the available CloudWatch metrics by running `aws cloudwatch list-metrics --region` *`ground-station-region-code`* .

   If you've recently used AWS Ground Station, the method should return output that contains entries like:

```
        ...
        {
           "Namespace": "AWS/GroundStation",
           "MetricName": "ReceivedPower",
           "Dimensions": [
              {
                 "Name": "Polarization",
                 "Value": "LHCP"
              },
              {
                 "Name": "SatelliteId",
                 "Value": "arn:aws:groundstation::111111111111:satellite/aaaaaaaa-
bbbb-cccc-dddd-eeeeeeeeeeee"
              }
           ]
       },
       ...
```

   > **Note**
   >
   > If it's been over 2 weeks since you last used AWS Ground Station, then you will need to manually inspect the table of available metrics to find the metric names and dimensions in the `AWS/GroundStation` metric namespace. For more information about CloudWatch limitations see: View available metrics

5. Modify the JSON file you created in step 2 to match the required values from step 3, for example `SatelliteId`, and `Polarization` from your metrics. Also be sure to update the `StartTime`, and `EndTime` values to match your contact. For example:

```
        {
           "MetricDataQueries": [
              {
                 "Id": "receivedPowerExample",
                 "MetricStat": {
                    "Metric": {
                       "Namespace": "AWS/GroundStation",
                       "MetricName": "ReceivedPower",
                       "Dimensions": [
                          {
                             "Name": "SatelliteId",
                             "Value":
 "arn:aws:groundstation::111111111111:satellite/aaaaaaaa-bbbb-cccc-dddd-
eeeeeeeeeeee"
                          },
                          {
                             "Name": "Polarization",
                             "Value": "RHCP"
                          }
                       ]
                    },
                    "Period": 300,
                    "Stat": "Maximum",
                    "Unit": "None"
                 },
                 "Label": "ReceivedPowerExample",
                 "ReturnData": true
              }
           ],
           "StartTime": "2024-02-08T00:00:00",
           "EndTime": "2024-04-09T00:00:00"
        }
```

Note

AWS Ground Station publishes metrics every 1 to 60 seconds, depending on the metric. Metrics will not be returned if the `Period` field has a value less than the publishing period for the metric.

6. Run `aws cloudwatch get-metric-data` with the configuration file created in the previous steps. An example is provided below.

```
aws cloudwatch get-metric-data --region ground-station-region-code --cli-input-json
 file://<nameOfConfigurationFileCreatedInStep2>.json
```

Metrics will be provided with timestamps from your contact. An example output of AWS Ground Station metrics is provided below.


```
{
   "MetricDataResults": [
      {
         "Id": "receivedPowerExample",
         "Label": "ReceivedPowerExample",
         "Timestamps": [
            "2024-04-08T18:35:00+00:00",
            "2024-04-08T18:30:00+00:00",
            "2024-04-08T18:25:00+00:00"
         ],
         "Values": [
            -33.30191555023193,
            -31.46100273132324,
            -32.13915576934814
         ],
         "StatusCode": "Complete"
      }
   ],
   "Messages": []
}
```

Security in AWS Ground Station

Cloud security at AWS is the highest priority. As an AWS customer, you will benefit from a data center and network architecture built to meet the requirements of the most security-sensitive organizations. AWS provides security-specific tools and features to help you meet your security objectives. These tools and features include network security, configuration management, access control, and data security.

When using AWS Ground Station, we recommend that you follow industry best practices and implement end-to-end encryption. AWS provides APIs for you to integrate encryption and data protection. For more information about AWS security, see the Introduction to AWS Security whitepaper.

Use the following topics to learn how to secure your resources.

Topics

Identity and Access Management for AWS Ground Station

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 Ground Station resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Identity-based policy examples for AWS Ground Station
- Troubleshooting AWS Ground Station identity and access

Audience

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see Troubleshooting AWS Ground Station identity and access)
- **Service administrator** - determine user access and submit permission requests (see How AWS Ground Station works with IAM)
- **IAM administrator** - write policies to manage access (see Identity-based policy examples for AWS Ground Station)

Authenticating with identities

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

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, see Choose 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 include 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. You must specify a principal in a resource-based policy.

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.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. 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 Ground Station works with IAM

Before you use IAM to manage access to AWS Ground Station, learn what IAM features are available to use with AWS Ground Station.

IAM features you can use with AWS Ground Station

IAM feature	AWS Ground Station support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Yes
Temporary credentials	Yes
Principal permissions	Yes
Service roles	No
Service-linked roles	Yes

To get a high-level view of how AWS Ground Station and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for AWS Ground Station

Supports identity-based policies: Yes

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 Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for AWS Ground Station

To view examples of AWS Ground Station identity-based policies, see Identity-based policy examples for AWS Ground Station.

Resource-based policies within AWS Ground Station

Supports resource-based policies: No

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.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for AWS Ground Station

Supports policy actions: Yes

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**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

To see a list of AWS Ground Station actions, see Actions defined by AWS Ground Station in the *Service Authorization Reference*.

Policy actions in AWS Ground Station use the following prefix before the action:

```
groundstation
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "groundstation:action1",
      "groundstation:action2"
         ]
```

To view examples of AWS Ground Station identity-based policies, see Identity-based policy examples for AWS Ground Station.

Policy resources for AWS Ground Station

Supports policy resources: Yes

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**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of AWS Ground Station resource types and their ARNs, see Resources defined by AWS Ground Station in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by AWS Ground Station.

To view examples of AWS Ground Station identity-based policies, see Identity-based policy examples for AWS Ground Station.

Policy condition keys for AWS Ground Station

Supports service-specific policy condition keys: Yes

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**.

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of AWS Ground Station condition keys, see Condition keys for AWS Ground Station in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by AWS Ground Station.

To view examples of AWS Ground Station identity-based policies, see Identity-based policy examples for AWS Ground Station.

ACLs in AWS Ground Station

Supports ACLs: No

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.

ABAC with AWS Ground Station

Supports ABAC (tags in policies): Yes

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with AWS Ground Station

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

Cross-service principal permissions for AWS Ground Station

Supports forward access sessions (FAS): Yes

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see Forward access sessions.

Service roles for AWS Ground Station

Supports service roles: No

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 Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break AWS Ground Station functionality. Edit service roles only when AWS Ground Station provides guidance to do so.

Service-linked roles for AWS Ground Station

Supports service-linked roles: Yes

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for AWS Ground Station

By default, users and roles don't have permission to create or modify AWS Ground Station resources. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by AWS Ground Station, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for AWS Ground Station in the *Service Authorization Reference*.

Topics

- Policy best practices
- Using the AWS Ground Station console
- Allow users to view their own permissions

Policy best practices

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

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies

that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.

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

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

Using the AWS Ground Station console

To access the AWS Ground Station console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the AWS Ground Station 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 (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the AWS Ground Station console, also attach the AWS Ground Station *ConsoleAccess* or *ReadOnly* AWS managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

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": "*"
        }
    ]
}
```

Troubleshooting AWS Ground Station identity and access

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

Topics

I am not authorized to perform an action in AWS Ground Station

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `groundstation:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 groundstation:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `groundstation:`*`GetWidget`* action.

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

I am not authorized to perform iam:PassRole

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

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 Ground Station. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

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

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

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

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

AWS managed policies for AWS Ground Station

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

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

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

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

AWS managed policy: AWSGroundStationAgentInstancePolicy

You can attach the `AWSGroundStationAgentInstancePolicy` policy to your IAM identities.

This policy grants AWS Ground Station Agent permissions to your Amazon EC2 instance that allows the instance to send and receive data during Ground Station contacts. All permissions in this policy are from the Ground Station service.

Permissions details

This policy includes the following permissions.

- `groundstation` – Allows dataflow endpoint instances to call the Ground Station Agent APIs.

To view the latest version of the JSON policy document, see AWSGroundStationAgentInstancePolicy in the AWS Managed Policy Reference Guide.

AWS managed policy: AWSServiceRoleForGroundStationDataflowEndpointGroupPolicy

You can not attach AWSServiceRoleForGroundStationDataflowEndpointGroupPolicy to your IAM entities. This policy is attached to a service-linked role that allows AWS Ground Station to perform actions on your behalf. For more information, see Using service linked roles.

This policy grants EC2 permissions that allow AWS Ground Station to find public IPv4 addresses.

Permissions details

This policy includes the following permissions.

- `ec2:DescribeAddresses` – Allows AWS Ground Station to list all IPs associated with EIPs on your behalf.
- `ec2:DescribeNetworkInterfaces` – Allows AWS Ground Station to get information on the network interfaces associated with EC2 instances on your behalf.

To view the latest version of the JSON policy document, see AWSServiceRoleForGroundStationDataflowEndpointGroupPolicy in the AWS Managed Policy Reference Guide.

AWS Ground Station updates to AWS managed policies

View details about updates to AWS managed policies for AWS Ground Station since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the AWS Ground Station Document history page.

Change	Description	Date
AWSGroundStationAgentInstancePolicy – Update to an existing policy	AWS Ground Station added new permissions to allow agents to retrieve task response URLs for enhanced Ground Station operations.	November 13, 2025
AWSGroundStationAgentInstancePolicy – New policy	AWS Ground Station added a new policy to provide the dataflow endpoint instance permissions to use the AWS Ground Station Agent.	April 12, 2023
AWSServiceRoleForGroundStationDataflowEndpointGroupPolicy – New policy	AWS Ground Station added a new policy that grants EC2 permissions to allow AWS Ground Station to find public IPv4 addresses associated with EIPs and network interfaces associated with EC2 instances.	November 02, 2022
AWS Ground Station started tracking changes	AWS Ground Station started tracking changes for AWS managed policies.	March 01, 2021

Use service-linked roles for Ground Station

AWS Ground Station 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 Ground Station. Service-linked roles are predefined by Ground Station and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Ground Station easier because you don't have to manually add the necessary permissions. Ground Station defines the permissions of its service-linked roles, and unless defined otherwise, only Ground Station 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.

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 roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Ground Station

Ground Station uses the service-linked role named **AWSServiceRoleForGroundStationDataflowEndpointGroup** – AWS GroundStation uses this service-linked role to invoke EC2 to find public IPv4 addresses.

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

- `groundstation.amazonaws.com`

The role permissions policy named AWSServiceRoleForGroundStationDataflowEndpointGroupPolicy allows Ground Station to complete the following actions on the specified resources:

- Action: `ec2:DescribeAddresses` on `all AWS resources (*)`

  Action allows Ground Station to list all IPs associated with EIPs.
- Action: `ec2:DescribeNetworkInterfaces` on `all AWS resources (*)`

  Action allows Ground Station to get information on the network interfaces associated with EC2 instances

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 Ground Station

You don't need to manually create a service-linked role. When you create a DataflowEndpointGroup in the AWS CLI or the AWS API, Ground Station 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 create a DataflowEndpointGroup, Ground Station creates the service-linked role for you again.

You can also use the IAM console to create a service-linked role with the **Data Delivery to Amazon EC2** use case. In the AWS CLI or the AWS API, create a service-linked role with the `groundstation.amazonaws.com` service name. For more information, see Creating a service-linked role in the *IAM User Guide*. If you delete this service-linked role, you can use this same process to create the role again.

Editing a service-linked role for Ground Station

Ground Station does not allow you to edit the AWSServiceRoleForGroundStationDataflowEndpointGroup 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 Ground Station

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.

You can delete a service-linked role only after first deleting the DataflowEndpointGroups using the service-linked role. This protects you from inadvertently revoking permissions to your DataflowEndpointGroups. If a service-linked role is used with multiple DataflowEndpointGroups, you must delete all DataflowEndpointGroups that use the service-linked role before you can delete it.

> **Note**
>
> If the Ground Station 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 Ground Station resources used by the AWSServiceRoleForGroundStationDataflowEndpointGroup

- Delete DataflowEndpointGroups via the AWS CLI or the AWS API.

To manually delete the service-linked role using IAM

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

Supported regions for Ground Station service-linked roles

Ground Station supports using service-linked roles in all of the regions where the service is available. For more information, see Region Table.

Troubleshooting

`NOT_AUTHORIZED_TO_CREATE_SLR` - This indicates the role in your account that is being used to call the CreateDataflowEndpointGroup API does not have the `iam:CreateServiceLinkedRole` permission. An administrator with the `iam:CreateServiceLinkedRole` permission must manually create the Service-Linked Role for your account.

Data encryption at rest for AWS Ground Station

AWS Ground Station provides encryption by default to protect your sensitive data at rest using AWS owned encryption keys.

- *AWS owned keys* - AWS Ground Station uses these keys by default to automatically encrypt personal, directly identifiable data and ephemerides. You cannot view, manage, or use AWS-owned keys, or audit their use; however, it is unnecessary to take any action or change programs to protect the keys that encrypt data. For more information, see AWS-owned keys in the AWS Key Management Service Developer Guide.

Encryption of data at rest by default helps by reducing the operational overhead and complexity involved in protecting sensitive data. At the same time, it enables building secure applications that meet strict encryption compliance, as well as regulatory requirements.

AWS Ground Station enforces encryption on all sensitive, at-rest, data, however, for some AWS Ground Station resource, such as ephemerides, you can choose to use a customer managed key in place of the default AWS managed keys.

- *Customer managed keys* -- AWS Ground Station supports the use of a symmetric customer managed key that you create, own, and manage in place of the existing AWS owned encryption. Because you have full control of this layer of encryption, you can perform such tasks as:
  - Establishing and maintaining key policies
  - Establishing and maintaining IAM policies and grants
  - Enabling and disabling key policies
  - Rotating key cryptographic material
  - Adding tags
  - Creating key aliases
  - Scheduling keys for deletion

  For more information, see customer managed key in the AWS Key Management Service Developer Guide.

The following table summarizes resources for which AWS Ground Station supports the use of Customer Managed Keys

Data type	AWS owned key encryption	Customer managed key encryption (Optional)
Ephemeris data used to compute the trajectory of a Satellite	Enabled	Enabled
Azimuth elevation ephemeris used to command antennas	Enabled	Enabled

Note

AWS Ground Station automatically enables encryption at rest using AWS owned keys to protect personally identifiable data at no charge. However, AWS KMS charges apply for using a customer managed key. For more information about pricing, see the AWS Key Management Service pricing.

For more information on AWS KMS, see the AWS Key Management Service Developer Guide.

For information specific to each resource type, see:

- Encryption at rest for TLE and OEM ephemeris data
- Encryption at rest for azimuth elevation ephemeris

Create a customer managed key

You can create a symmetric customer managed key by using the AWS Management Console, or the AWS KMS APIs.

To create a symmetric customer managed key

Follow the steps for creating symmetric customer managed key in the AWS Key Management Service Developer Guide.

Key policy overview

Key policies control access to your customer managed key. Every customer managed key must have exactly one key policy, which contains statements that determine who can use the key and how they can use it. When you create your customer managed key, you can specify a key policy. For more information, see Managing access to customer managed keys in the AWS Key Management Service Developer Guide.

To use your customer managed key with AWS Ground Station resources, you must configure the key policy to grant appropriate permissions to the AWS Ground Station service. The specific permissions and policy configuration depend on the type of resource you're encrypting:

- *For TLE and OEM ephemeris data* - See Encryption at rest for TLE and OEM ephemeris data for specific key policy requirements and examples.
- *For azimuth elevation ephemeris data* - See Encryption at rest for azimuth elevation ephemeris for specific key policy requirements and examples.

> **Note**
>
> The key policy configuration differs between ephemeris types. TLE and OEM ephemeris data uses grants for key access, while azimuth elevation ephemeris uses direct key policy permissions. Ensure you configure your key policy according to the specific resource type you're encrypting.

For more information about specifying permissions in a policy and troubleshooting key access, see the AWS Key Management Service Developer Guide.

Specifying a customer managed key for AWS Ground Station

You can specify a customer managed key to encrypt the following resources:

- Ephemeris (TLE, OEM, and azimuth elevation)

When you create a resource, you can specify the data key by providing a *kmsKeyArn*

- *kmsKeyArn* - A key identifier for an AWS KMS customer managed key

AWS Ground Station encryption context

An encryption context is an optional set of key-value pairs that contain additional contextual information about the data. AWS KMS uses the encryption context as additional authenticated data to support authenticated encryption. When you include an encryption context in a request to encrypt data, AWS KMS binds the encryption context to the encrypted data. To decrypt data, you include the same encryption context in the request.

AWS Ground Station uses different encryption context depending on the resource being encrypted and specifies a specific encryption context for each key grant created.

For resource-specific encryption context details, see:

- Encryption at rest for TLE and OEM ephemeris data
- Encryption at rest for azimuth elevation ephemeris

Encryption at rest for TLE and OEM ephemeris data

Key policy requirements for TLE and OEM ephemeris

To use a customer managed key with ephemeris data, your key policy must grant the following permissions to the AWS Ground Station service:

- `kms:CreateGrant` - Creates an access grant on a customer managed key. Grants AWS Ground Station access to perform grant operations on the customer managed key for reading and storing encrypted data.
- `kms:DescribeKey` - Provides the customer managed key details to allow AWS Ground Station to validate the key before attempting to use the provided key.

For more information about Using Grants, see the AWS Key Management Service Developer Guide.

IAM user permissions for creating ephemeris with customer managed keys

When AWS Ground Station uses a customer managed key in cryptographic operations, it acts on behalf of the user who is creating the ephemeris resource.

To create an ephemeris resource using a customer managed key, a user must have permissions to call the following operations on the customer managed key:

- `kms:CreateGrant` - Allows the user to create grants on the customer managed key on behalf of AWS Ground Station.
- `kms:DescribeKey` - Allows the user to view the customer managed key details to validate the key.

You can specify these required permissions in a key policy, or in an IAM policy if the key policy allows it. These permissions ensure that users can authorize AWS Ground Station to use the customer managed key for encryption operations on their behalf.

How AWS Ground Station uses grants in AWS KMS for ephemeris

AWS Ground Station requires a key grant to use your customer-managed key.

When you upload an ephemeris encrypted with a customer managed key, AWS Ground Station creates a key grant on your behalf by sending a CreateGrant request to AWS KMS. Grants in AWS KMS are used to give AWS Ground Station access to a AWS KMS key in your account.

This allows AWS Ground Station to do the following:

- Call GenerateDataKey to generate an encrypted data key and store it, because the data key isn't immediately used to encrypt.
- Call Decrypt to use the stored encrypted data key to access encrypted data.
- Call Encrypt to use the data key to encrypt data.
- Set up a retiring principal to allow the service to RetireGrant.

You can revoke access to the grant at any time. If you do, AWS Ground Station won't be able to access any of the data encrypted by the customer managed key, which affects operations that are dependent on that data. For example, if you remove a key grant from an ephemeris currently in use for a contact then AWS Ground Station will be unable to use the provided ephemeris data for pointing the antenna during the contact. This will cause the contact to end in a FAILED state.

Ephemeris encryption context

Key grants for encrypting ephemeris resources are bound to a specific satellite ARN.

```
"encryptionContext": {
    "aws:groundstation:arn":
 "arn:aws:groundstation::111122223333:satellite/00a770b0-082d-45a4-80ed-SAMPLE",
    "aws:s3:arn":
 "arn:aws:s3:::customerephemerisbucket/0034abcd-12ab-34cd-56ef-123456SAMPLE"
}
```

> **Note**
>
> Key grants are re-used for the same key-satellite pair.

Using encryption context for monitoring

When you use a symmetric customer managed key to encrypt your ephemerides, you can also use the encryption context in audit records and logs to identify how the customer managed key is being used. The encryption context also appears in logs generated by AWS CloudTrail or Amazon CloudWatch Logs .

Using encryption context to control access to your customer managed key

You can use the encryption context in key policies and IAM policies as `conditions` to control access to your symmetric customer managed key. You can also use encryption context constraints in a grant.

AWS Ground Station uses an encryption context constraint in grants to control access to the customer managed key in your account or region. The grant constraint requires that the operations that the grant allows use the specified encryption context.

The following are example key policy statements to grant access to a customer managed key for a specific encryption context. The condition in this policy statement requires that the grants have an encryption context constraint that specifies the encryption context.

The following example shows a key policy for ephemeris data bound to a satellite:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Allow AWS Ground Station to Describe key",
            "Effect": "Allow",
            "Principal": {
                "Service": "groundstation.us-east-1.amazonaws.com"
            },
            "Action": "kms:DescribeKey",
            "Resource": "*"
        },
        {
            "Sid": "Allow AWS Ground Station to Create Grant on key",
            "Effect": "Allow",
            "Principal": {
                "Service": "groundstation.us-east-1.amazonaws.com"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:aws:groundstation:arn":
 "arn:aws:groundstation::123456789012:satellite/satellite-id"
```

```
                }
            }
        }
    ]
}
```

Monitoring your encryption keys for ephemeris

When you use an AWS Key Management Service customer managed key with your ephemeris resources, you can use AWS CloudTrail or Amazon CloudWatch logs to track requests that AWS Ground Station sends to AWS KMS. The following examples are CloudTrail events for CreateGrant, GenerateDataKey, Decrypt, and DescribeKey to monitor AWS KMS operations called by AWS Ground Station to access data encrypted by your customer managed key.

CreateGrant

When you use an AWS KMS customer managed key to encrypt your ephemeris resources, AWS Ground Station sends a CreateGrant request on your behalf to access the AWS KMS key in your AWS account. The grant that AWS Ground Station creates is specific to the resource associated with the AWS KMS customer managed key. In addition, AWS Ground Station uses the RetireGrant operation to remove a grant when you delete a resource.

The following example event records the CreateGrant operation for an ephemeris:

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "ASIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/SampleUser01",
        "accountId": "111122223333",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "ASIAIOSFODNN7EXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
```

```
            "attributes": {
                "creationDate": "2022-02-22T22:22:22Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2022-02-22T22:22:22Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "ExampleDesktop/1.0 (V1; OS)",
    "requestParameters": {
        "operations": [
            "GenerateDataKeyWithoutPlaintext",
            "Decrypt",
            "Encrypt"
        ],
        "constraints": {
            "encryptionContextSubset": {
                "aws:groundstation:arn":
 "arn:aws:groundstation::111122223333:satellite/00a770b0-082d-45a4-80ed-SAMPLE"
            }
        },
        "granteePrincipal": "groundstation.us-west-2.amazonaws.com",
        "retiringPrincipal": "groundstation.us-west-2.amazonaws.com",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "responseElements": {
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE"
    },
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
```

```
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

DescribeKey

When you use an AWS KMS customer managed key to encrypt your ephemeris resources, AWS Ground Station sends a DescribeKey request on your behalf to validate that the requested key exists in your account.

The following example event records the DescribeKey operation:

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "ASIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:sts::111122223333:assumed-role/User/Role",
        "accountId": "111122223333",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "ASIAIOSFODNN7EXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Role",
                "accountId": "111122223333",
                "userName": "User"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-02-22T22:22:22Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2022-02-22T22:22:22Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DescribeKey",
    "awsRegion": "us-west-2",
```

```
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```


GenerateDataKey

When you use an AWS KMS customer managed key to encrypt your ephemeris resources, AWS Ground Station sends a GenerateDataKey request to in order to generate a data key with which to encrypt your data.

The following example event records the GenerateDataKey operation for an ephemeris:


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2022-02-22T22:22:22Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "AWS Internal",
```

```
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keySpec": "AES_256",
        "encryptionContext": {
            "aws:groundstation:arn":
 "arn:aws:groundstation::111122223333:satellite/00a770b0-082d-45a4-80ed-SAMPLE",
            "aws:s3:arn":
 "arn:aws:s3:::customerephemerisbucket/0034abcd-12ab-34cd-56ef-123456SAMPLE"
        },
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "sharedEventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventCategory": "Management"
}
```


Decrypt

When you use an AWS KMS customer managed key to encrypt your ephemeris resources, AWS Ground Station uses the Decrypt operation to decrypt the ephemeris provided if it is already encrypted with the same customer managed key. For example if an ephemeris is being uploaded from an S3 bucket and is encrypted in that bucket with a given key.

The following example event records the Decrypt operation for an ephemeris:


```
{
    "eventVersion": "1.08",
    "userIdentity": {
```

```
        "type": "AWSService",
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2022-02-22T22:22:22Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "encryptionContext": {
            "aws:groundstation:arn":
 "arn:aws:groundstation::111122223333:satellite/00a770b0-082d-45a4-80ed-SAMPLE",
            "aws:s3:arn":
 "arn:aws:s3:::customerephemerisbucket/0034abcd-12ab-34cd-56ef-123456SAMPLE"
        },
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "sharedEventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventCategory": "Management"
}
```

Encryption at rest for azimuth elevation ephemeris

Key policy requirements for azimuth elevation ephemeris

To use a customer managed key with azimuth elevation ephemeris data, your key policy must grant the following permissions to the AWS Ground Station service. Unlike TLE and OEM ephemeris data which uses grants, azimuth elevation ephemeris uses direct key policy permissions for encryption operations. This is a simpler method to manage the permissions of, and use your keys.

- `kms:GenerateDataKey` - Generates data keys for encrypting your azimuth elevation ephemeris data.
- `kms:Decrypt` - Decrypts the encrypted data keys when accessing your azimuth elevation ephemeris data.

Example key policy granting AWS Ground Station access to a customer managed key

> **Note**
>
> With azimuth elevation ephemeris, you must configure these permissions directly in the key policy. The regional AWS Ground Station service principal (e.g., `groundstation.region.amazonaws.com`) must be granted these permissions in your key policy statements. Without these statements added to the key policy AWS Ground Station will be unable to store or access your custom azimuth elevation ephemeris.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Allow AWS Ground Station to Describe key",
            "Effect": "Allow",
            "Principal": {
                "Service": "groundstation.us-east-1.amazonaws.com"
            },
            "Action": "kms:DescribeKey",
            "Resource": "*"
        },
```

```
        {
            "Sid": "Allow AWS Ground Station to Encrypt and Decrypt with key",
            "Effect": "Allow",
            "Principal": {
                "Service": "groundstation.us-east-1.amazonaws.com"
            },
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": "*"
        }
    ]
}
```

IAM user permissions for creating azimuth elevation ephemeris with customer managed keys

When AWS Ground Station uses a customer managed key in cryptographic operations, it acts on behalf of the user who is creating the azimuth elevation ephemeris resource.

To create an azimuth elevation ephemeris resource using a customer managed key, a user must have permissions to call the following operations on the customer managed key:

- `kms:GenerateDataKey` - Allows the user to generate data keys for encrypting the azimuth elevation ephemeris data.
- `kms:Decrypt` - Allows the user to decrypt data keys when accessing the azimuth elevation ephemeris data.
- `kms:DescribeKey` - Allows the user to view the customer managed key details to validate the key.

You can specify these required permissions in a key policy, or in an IAM policy if the key policy allows it. These permissions ensure that users can authorize AWS Ground Station to use the customer managed key for encryption operations on their behalf.

How AWS Ground Station uses key policies for azimuth elevation ephemeris

When you provide azimuth elevation ephemeris data with a customer managed key, AWS Ground Station uses key policies to access your encryption key. The permissions are granted directly to

AWS Ground Station through key policy statements rather than through grants as with TLE or OEM ephemeris data.

If you remove AWS Ground Station's access to the customer managed key, AWS Ground Station won't be able to access any of the data encrypted by that key, which affects operations that are dependent on that data. For example, if you remove key policy permissions for azimuth elevation ephemeris currently in use for a contact, AWS Ground Station will be unable to use the provided azimuth elevation data for commanding the antenna during the contact. This will cause the contact to end in a FAILED state.

Azimuth elevation ephemeris encryption context

When AWS Ground Station uses your AWS KMS key to encrypt azimuth elevation ephemeris data, the service specifies an encryption context. The encryption context is additional authenticated data (AAD) that AWS KMS uses to ensure data integrity. When an encryption context is specified for an encryption operation, the service must specify the same encryption context for the decryption operation. Otherwise, decryption fails. The encryption context is also written to your CloudTrail logs to help you understand why a given AWS KMS key was used. Your CloudTrail logs might contain many entries describing the use of a AWS KMS key, but the encryption context in each log entry can help you determine the reason for that particular use.

AWS Ground Station specifies the following encryption context when it performs cryptographic operations with your customer managed key on an azimuth elevation ephemeris:

```
{
    "encryptionContext": {
        "aws:groundstation:ground-station-id": "Ohio 1",
        "aws:groundstation:arn": "arn:aws:groundstation:us-
east-2:111122223333:ephemeris/00a770b0-082d-45a4-80ed-SAMPLE",
        "aws:s3:arn": "arn:aws:s3:::customerephemerisbucket/00a770b0-082d-45a4-80ed-
SAMPLE/raw"
    }
}
```

The encryption context contains:

`aws:groundstation:ground-station-id`

The name of the ground station associated with the azimuth elevation ephemeris.

aws:groundstation:arn

The ARN of the ephemeris resource.

aws:s3:arn

The ARN of the ephemeris stored in Amazon S3.

Using encryption context to control access to your customer managed key

You can use IAM condition statements to control AWS Ground Station access to your customer managed key. Adding a condition statement on the `kms:GenerateDataKey` and `kms:Decrypt` actions restricts which ground stations a AWS KMS can be used for.

The following are example key policy statements to grant AWS Ground Station access to your customer managed key in a specific region for a specific ground station. The condition in this policy statement requires that all encrypt and decrypt access to the key that specify an encryption context that matches the condition in the key policy.

Example key policy granting AWS Ground Station access to a customer managed key for a specific ground station

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Allow AWS Ground Station to Describe key",
            "Effect": "Allow",
            "Principal": {
                "Service": "groundstation.us-east-1.amazonaws.com"
            },
            "Action": "kms:DescribeKey",
            "Resource": "*"
        },
        {
            "Sid": "Allow AWS Ground Station to Encrypt and Decrypt with key",
            "Effect": "Allow",
            "Principal": {
                "Service": "groundstation.us-east-1.amazonaws.com"
```

```
            },
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:aws:groundstation:ground-station-id":
 "specific-ground-station-name"
                }
            }
        }
    ]
}
```

Example key policy granting AWS Ground Station access to a customer managed key for multiple ground stations

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Allow AWS Ground Station to Describe key",
            "Effect": "Allow",
            "Principal": {
                "Service": "groundstation.us-east-1.amazonaws.com"
            },
            "Action": "kms:DescribeKey",
            "Resource": "*"
        },
        {
            "Sid": "Allow AWS Ground Station to Encrypt and Decrypt with key",
            "Effect": "Allow",
            "Principal": {
                "Service": "groundstation.us-east-1.amazonaws.com"
            },
            "Action": [
                "kms:GenerateDataKey",
```

```
                "kms:Decrypt"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "kms:EncryptionContext:aws:groundstation:ground-station-id":
 [
                        "specific-ground-station-name-1",
                        "specific-ground-station-name-2"
                    ]
                }
            }
        }
    ]
}
```

Monitoring your encryption keys for azimuth elevation ephemeris

When you use an AWS KMS customer managed key with your azimuth elevation ephemeris resources, you can use CloudTrail or CloudWatch logs to track requests that AWS Ground Station sends to AWS KMS. The following examples are CloudTrail events for GenerateDataKey and Decrypt to monitor AWS KMS operations called by AWS Ground Station to access data encrypted by your customer managed key.

GenerateDataKey

When you use an AWS KMS customer managed key to encrypt your azimuth elevation ephemeris resources, AWS Ground Station sends a GenerateDataKey request to AWS KMS in order to generate a data key with which to encrypt your data.

The following example event records the GenerateDataKey operation for azimuth elevation ephemeris:

```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "ASIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/SampleUser01",
        "accountId": "111122223333",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
```

```
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "ASIAIOSFODNN7EXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "attributes": {
                "creationDate": "2025-08-25T14:45:48Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2025-08-25T14:52:02Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keySpec": "AES_256",
        "encryptionContext": {
            "aws:groundstation:arn": "arn:aws:groundstation:us-
west-2:111122223333:ephemeris/bb650670-7a4b-4152-bd60-SAMPLE",
            "aws:groundstation:ground-station-id": "Ohio 1",
            "aws:s3:arn": "arn:aws:s3:::customerephemerisbucket/bb650670-7a4b-4152-
bd60-SAMPLE/raw"
        },
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "responseElements": null,
    "requestID": "ef6f9a8f-8ef6-46a1-bdcb-123456SAMPLE",
    "eventID": "952842d4-1389-3232-b885-123456SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
```

```
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "sharedEventID": "8424f6b6-2280-4d1d-b9fd-0348b1546cba",
    "eventCategory": "Management"
}
```

Decrypt

When you use an AWS KMS customer managed key to encrypt your azimuth elevation ephemeris resources, AWS Ground Station uses the Decrypt operation to decrypt the azimuth elevation ephemeris data provided if they are already encrypted with the same customer managed key.

The following example event records the Decrypt operation for azimuth elevation ephemeris:

```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "ASIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/SampleUser01",
        "accountId": "111122223333",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "ASIAIOSFODNN7EXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            }
        },
        "attributes": {
            "creationDate": "2025-08-25T14:45:48Z",
            "mfaAuthenticated": "false"
        }
    },
    "invokedBy": "AWS Internal",
    "eventTime": "2025-08-25T14:54:01Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
```

```
    "awsRegion": "us-west-2",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "encryptionContext": {
            "aws:groundstation:arn": "arn:aws:groundstation:us-
west-2:111122223333:ephemeris/bb650670-7a4b-4152-bd60-SAMPLE",
            "aws:groundstation:ground-station-id": "Ohio 1",
            "aws:s3:arn": "arn:aws:s3:::customerephemerisbucket/bb650670-7a4b-4152-
bd60-SAMPLE/raw"
        },
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "a2f46066-49fb-461a-93cb-123456SAMPLE",
    "eventID": "e997b426-e3ad-31c7-a308-123456SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "sharedEventID": "477b568e-7f56-4f04-905c-623ff146f30d",
    "eventCategory": "Management"
}
```


Data encryption during transit for AWS Ground Station

AWS Ground Station provides encryption by default to protect your sensitive data during transit. Data can be streamed between AWS Ground Station antenna locations and your Amazon EC2 instances in two ways, depending on the mission profile configuration.

- AWS Ground Station Agent
- Dataflow endpoint

Each method of streaming data handles encrypting data in transit differently. The following sections describe each method.

AWS Ground Station Agent streams

AWS Ground Station Agent encrypts its streams using customer managed AWS KMS keys. The AWS Ground Station Agent running on your Amazon EC2 instance will automatically decrypt the stream to provide decrypted data.

The AWS KMS key used for encrypting a stream is specified when creating a `MissionProfile` in the streamsKmsKey parameter. All permissions granting AWS Ground Station access to the keys are handled through the AWS KMS key policy attached to `streamsKmsKey`.

Dataflow endpoint streams

Dataflow endpoint streams are encrypted using Datagram Transport Layer Security (DTLS). This is done using self-signed certificates, and doesn't require additional configuration.

Example mission profile configurations

The examples provided show how to take a public broadcast satellite and create a mission profile that supports it. The resulting templates are provided to help you take a public broadcast satellite contact and to help you make decisions about your satellites.

Topics

JPSS-1 - Public broadcast satellite (PBS) - Evaluation

This example section matches the Customer onboarding process overview. It provides a brief compatibility analysis with AWS Ground Station and sets the stage for the specific examples that follow.

As mentioned in the Public broadcast satellites section, you can utilize select satellites, or communication paths of a satellite, that are publicly available. In this section we describe JPSS-1 in the AWS Ground Station terms. For reference, we utilize the Joint Polar Satellite System 1 (JPSS-1) Spacecraft High Rate Data (HRD) to Direct Broadcast Stations (DBS) Radio Frequency (RF) Interface Control Document (ICD) to complete the example. Also, worth noting that JPSS-1 is associated to the NORAD ID 43013.

The JPSS-1 satellite offers one uplink and three direct downlink communication paths, as seen in Figure 1-1 of the ICD. Of these four communication paths, only the single High Rate Data (HRD) downlink communication path is available for public consumption. Based on this, you'll see this path will have much more specific data associated with it as well. The four paths are as follows:

- Command path (uplink) at 2067.27 MHz center frequency with a data rate of 2-128 kbps. This path is not publicly accessible.
- Telemetry path (downlink) at 2247.5 MHz center frequency with a data rate of 1-524 kbps. This path is not publicly accessible.

- SMD path (downlink) at 26.7034 GHz center frequency with a data rate of 150-300 Mbps. This path is not publicly accessible.
- The RF for the HRD path (downlink) at 7812 MHz center frequency with a data rate of 15 Mbps. It has a 30 MHz bandwidth, and is right-hand-circular-polarized. When you onboard JPSS-1 with AWS Ground Station, this is the communication path you get access to. This communication path contains instrument science data, instrument engineering data, instrument telemetry data, and real-time spacecraft housekeeping data.

As we compare the potential data paths, we see that the command (uplink), telemetry (downlink), and HRD (downlink) paths meet the frequency, bandwidth, and multi-channel concurrent use capabilities of AWS Ground Station. The SMD path is not compatible as the center frequency is out of range of the existing receivers. For more information about the supported capabilities, see AWS Ground Station Site Capabilities.

Note

As the SMD path is not compatible with AWS Ground Station it will not be represented in the example configurations.

Note

As the command (uplink) and telemetry (downlink) paths are not defined in the ICD, nor are they available for public use, the values provided when used are notional.

Public broadcast satellite utilizing Amazon S3 data delivery

This example builds off the analysis done in the JPSS-1 - Public broadcast satellite (PBS) - Evaluation section of the user guide.

For this example, you'll need to assume a scenario -- you want to capture the HRD communication path as digital intermediate frequency and store it for future batch processing. This saves off the raw radio frequency (RF) in-phase quadrature (I/Q) samples after it has been digitized. Once the data is in your Amazon S3 bucket, you can demodulate and decode the data using any software you desire. See the MathWorks Tutorial for a detailed example of processing. After using this

example, you may consider adding Amazon EC2 spot pricing components to process the data and lower your overall processing costs.

Communication paths

This section represents Plan your dataflow communication paths of getting started.

All of the following template snippets belong in the Resources section of the CloudFormation template.

```
Resources:
  # Resources that you would like to create should be placed within the Resources
 section.
```

> **Note**
>
> For more information about the contents of a CloudFormation template, see Template sections.

Given our scenario to deliver a single communication path to Amazon S3, you know that you'll have a single asynchronous delivery path. Per the Asynchronous data delivery section, you must define a Amazon S3 bucket.

```
  # The S3 bucket where AWS Ground Station will deliver the downlinked data.
  GroundStationS3DataDeliveryBucket:
    Type: AWS::S3::Bucket
    DeletionPolicy: Retain
    UpdateReplacePolicy: Retain
    Properties:
      # Results in a bucket name formatted like: aws-groundstation-data-{account id}-
{region}-{random 8 character string}
      BucketName: !Join ["-", ["aws-groundstation-data", !Ref AWS::AccountId, !Ref
 AWS::Region, !Select [0, !Split ["-", !Select [2, !Split ["/", !Ref AWS::StackId]]]]]]
```

In addition, you will need to create the appropriate roles and policies in order to allow AWS Ground Station to use the bucket.

```
  # The IAM role that AWS Ground Station will assume to have permission find and write
  # data to your S3 bucket.
  GroundStationS3DataDeliveryRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action:
              - 'sts:AssumeRole'
            Effect: Allow
            Principal:
              Service:
                - groundstation.amazonaws.com
            Condition:
              StringEquals:
                "aws:SourceAccount": !Ref AWS::AccountId
              ArnLike:
                "aws:SourceArn": !Sub "arn:aws:groundstation:${AWS::Region}:
${AWS::AccountId}:config/s3-recording/*"

  # The S3 bucket policy that defines what actions AWS Ground Station can perform on
 your S3 bucket.
  GroundStationS3DataDeliveryBucketPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - 's3:GetBucketLocation'
            Effect: Allow
            Resource:
              - !GetAtt GroundStationS3DataDeliveryBucket.Arn
          - Action:
              - 's3:PutObject'
            Effect: Allow
            Resource:
              - !Join [ "/", [ !GetAtt GroundStationS3DataDeliveryBucket.Arn, "*" ] ]
      PolicyName: GroundStationS3DataDeliveryPolicy
      Roles:
        - !Ref GroundStationS3DataDeliveryRole
```

AWS Ground Station configs

This section represents Create configs of getting started.

You'll need a *tracking-config* to set your preference on using autotrack. Selecting *PREFERRED* as autotrack can improve the signal quality, but it isn't required to meet the signal quality due to sufficient JPSS-1 ephemeris quality.

```
 TrackingConfig:
   Type: AWS::GroundStation::Config
   Properties:
     Name: "JPSS Tracking Config"
     ConfigData:
       TrackingConfig:
         Autotrack: "PREFERRED"
```

Based on the communication path, you'll need to define an *antenna-downlink* config to represent the satellite portion as well as an *s3-recording* to refer to the Amazon S3 bucket you just created.

```
 # The AWS Ground Station Antenna Downlink Config that defines the frequency spectrum
used to
 # downlink data from your satellite.
 JpssDownlinkDigIfAntennaConfig:
   Type: AWS::GroundStation::Config
   Properties:
     Name: "JPSS Downlink DigIF Antenna Config"
     ConfigData:
       AntennaDownlinkConfig:
         SpectrumConfig:
           Bandwidth:
             Units: "MHz"
             Value: 30
           CenterFrequency:
             Units: "MHz"
             Value: 7812
           Polarization: "RIGHT_HAND"

 # The AWS Ground Station S3 Recording Config that defines the S3 bucket and IAM role
to use
```

```
  # when AWS Ground Station delivers the downlink data.
  S3RecordingConfig:
    Type: AWS::GroundStation::Config
    DependsOn: GroundStationS3DataDeliveryBucketPolicy
    Properties:
      Name: "JPSS S3 Recording Config"
      ConfigData:
        S3RecordingConfig:
          BucketArn: !GetAtt GroundStationS3DataDeliveryBucket.Arn
          RoleArn: !GetAtt GroundStationS3DataDeliveryRole.Arn
```

AWS Ground Station mission profile

This section represents Create mission profile of getting started.

Now that you have the associated configs, you can use them to construct the dataflow. You'll use the defaults for the remaining parameters.

```
  # The AWS Ground Station Mission Profile that groups the above configurations to
define how to downlink data.
  JpssAsynchMissionProfile:
    Type: AWS::GroundStation::MissionProfile
    Properties:
      Name: "43013 JPSS Asynchronous Data"
      MinimumViableContactDurationSeconds: 180
      TrackingConfigArn: !Ref TrackingConfig
      DataflowEdges:
        - Source: !Ref JpssDownlinkDigIfAntennaConfig
          Destination: !Ref S3RecordingConfig
```

Putting it together

With the above resources, you now have the ability to schedule JPSS-1 contacts for asynchronous data delivery from any of your onboarded AWS Ground Station AWS Ground Station Locations.

The following is a complete CloudFormation template that includes all resources described in this section combined into a single template that can be directly used in CloudFormation.

The CloudFormation template named `AquaSnppJpss-1TerraDigIfS3DataDelivery.yml` contains an Amazon S3 bucket and the required AWS Ground Station resources to schedule contacts and receive VITA-49 Signal/IP direct broadcast data.

If Aqua, SNPP, JPSS-1/NOAA-20, and Terra are not onboarded to your account, see Onboard satellite.

Note

You can access the template by accessing the customer onboarding Amazon S3 bucket using valid AWS credentials. The links below use a regional Amazon S3 bucket. Change the `us-west-2` region code to represent the corresponding region of which you want to create the CloudFormation stack in.
Additionally, the following instructions use YAML. However, the templates are available in both YAML and JSON format. To use JSON, replace the `.yml` file extension with `.json` when downloading the template.

To download the template using AWS CLI, use the following command:

```
aws s3 cp s3://groundstation-cloudformation-templates-us-west-2/
AquaSnppJpss-1TerraDigIfS3DataDelivery.yml .
```

You can view and download the template in the console by navigating to the following URL in your browser:

```
https://s3.console.aws.amazon.com/s3/object/groundstation-cloudformation-templates-us-
west-2/AquaSnppJpss-1TerraDigIfS3DataDelivery.yml
```

You can specify the template directly in CloudFormation using the following link:

```
https://groundstation-cloudformation-templates-us-west-2.s3.us-west-2.amazonaws.com/
AquaSnppJpss-1TerraDigIfS3DataDelivery.yml
```

Public broadcast satellite utilizing a dataflow endpoint (narrowband)

This example builds off the analysis done in the JPSS-1 - Public broadcast satellite (PBS) - Evaluation section of the user guide.

To complete this example, you'll need to assume a scenario -- you want to capture the HRD communication path as digital intermediate frequency (DigIF) and process it as it's received by a dataflow endpoint application on an Amazon EC2 instance using an SDR.

Communication paths

This section represents Plan your dataflow communication paths of getting started. For this example, you will be creating two sections in your CloudFormation template: Parameters and Resources sections.

> **Note**
>
> For more information about the contents of a CloudFormation template, see Template sections.

For the Parameters section, you're going to add the following parameters. You'll specify values for these when creating the stack via the CloudFormation console.

```
Parameters:
  EC2Key:
    Description: The SSH key used to access the EC2 receiver instance. Choose any
 SSH key if you are not creating an EC2 receiver instance. For instructions on how to
 create an SSH key see https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/create-
key-pairs.html
    Type: AWS::EC2::KeyPair::KeyName
    ConstraintDescription: must be the name of an existing EC2 KeyPair.

  ReceiverAMI:
    Description: The Ground Station DDX AMI ID you want to use. Please note
 that AMIs are region specific. For instructions on how to retrieve an AMI
 see https://docs.aws.amazon.com/ground-station/latest/ug/dataflows.ec2-
configuration.html#dataflows.ec2-configuration.amis
```

```
    Type: AWS::EC2::Image::Id
```

> **Note**
>
> You **need** to create a key pair, and provide the name for the Amazon EC2 EC2Key parameter. See Create a key pair for your Amazon EC2 instance.
> Additionally, you'll **need** to provide the correct **region specific** AMI ID, when creating the CloudFormation stack. See AWS Ground Station Amazon Machine Images (AMIs).

The remaining template snippets belong in the Resources section of the CloudFormation template.

```
Resources:
  # Resources that you would like to create should be placed within the resource
 section.
```

Given our scenario to deliver a single communication path to an EC2 instance, you'll have a single synchronous delivery path. Per the Synchronous data delivery section, you must set up and configure an Amazon EC2 instance with a dataflow endpoint application, and create one or more dataflow endpoint groups.

```
  # The EC2 instance that will send/receive data to/from your satellite using AWS
 Ground Station.
  ReceiverInstance:
    Type: AWS::EC2::Instance
    Properties:
      DisableApiTermination: false
      IamInstanceProfile: !Ref GeneralInstanceProfile
      ImageId: !Ref ReceiverAMI
      InstanceType: m5.4xlarge
      KeyName: !Ref EC2Key
      Monitoring: true
      PlacementGroupName: !Ref ClusterPlacementGroup
      SecurityGroupIds:
        - Ref: InstanceSecurityGroup
      SubnetId: !Ref ReceiverSubnet
      BlockDeviceMappings:
        - DeviceName: /dev/xvda
          Ebs:
```

```
            VolumeType: gp2
            VolumeSize: 40
      Tags:
        - Key: Name
          Value: !Join [ "-" , [ "Receiver" , !Ref "AWS::StackName" ] ]
      UserData:
        Fn::Base64:
          |
          #!/bin/bash
          exec > >(tee /var/log/user-data.log|logger -t user-data -s 2>/dev/console)
2>&1
          echo `date +'%F %R:%S'` "INFO: Logging Setup" >&2

          GROUND_STATION_DIR="/opt/aws/groundstation"
          GROUND_STATION_BIN_DIR="${GROUND_STATION_DIR}/bin"
          STREAM_CONFIG_PATH="${GROUND_STATION_DIR}/customer_stream_config.json"

          echo "Creating ${STREAM_CONFIG_PATH}"
          cat << STREAM_CONFIG > "${STREAM_CONFIG_PATH}"
          {
            "ddx_streams": [
              {
                "streamName": "Downlink",
                "maximumWanRate": 4000000000,
                "lanConfigDevice": "lo",
                "lanConfigPort": 50000,
                "wanConfigDevice": "eth1",
                "wanConfigPort": 55888,
                "isUplink": false
              }
            ]
          }
          STREAM_CONFIG

          echo "Waiting for dataflow endpoint application to start"
          while netstat -lnt | awk '$4 ~ /:80$/ {exit 1}'; do sleep 10; done

          echo "Configuring dataflow endpoint application streams"
          python "${GROUND_STATION_BIN_DIR}/configure_streams.py" --configFileName
"${STREAM_CONFIG_PATH}"
          sleep 2
          python "${GROUND_STATION_BIN_DIR}/save_default_config.py"

          exit 0
```

```
 # The AWS Ground Station Dataflow Endpoint Group that defines the endpoints that AWS
Ground
 # Station will use to send/receive data to/from your satellite.
 DataflowEndpointGroup:
   Type: AWS::GroundStation::DataflowEndpointGroup
   Properties:
     ContactPostPassDurationSeconds: 180
     ContactPrePassDurationSeconds: 120
     EndpointDetails:
       - Endpoint:
           Name: !Join [ "-" , [ !Ref "AWS::StackName" , "Downlink" ] ] # needs to
match DataflowEndpointConfig name
           Address:
             Name: !GetAtt ReceiverInstanceNetworkInterface.PrimaryPrivateIpAddress
             Port: 55888
         SecurityDetails:
           SecurityGroupIds:
             - Ref: "DataflowEndpointSecurityGroup"
           SubnetIds:
             - !Ref ReceiverSubnet
           RoleArn: !GetAtt DataDeliveryServiceRole.Arn

 # The security group for your EC2 instance.
 InstanceSecurityGroup:
   Type: AWS::EC2::SecurityGroup
   Properties:
     GroupDescription: AWS Ground Station receiver instance security group.
     VpcId: !Ref ReceiverVPC
     SecurityGroupIngress:
       # To allow SSH access to the instance, add another rule allowing tcp port 22
from your CidrIp
       - IpProtocol: udp
         FromPort: 55888
         ToPort: 55888
         SourceSecurityGroupId: !Ref DataflowEndpointSecurityGroup
         Description: "AWS Ground Station Downlink Stream"

 # The security group that the ENI created by AWS Ground Station belongs to.
 DataflowEndpointSecurityGroup:
   Type: AWS::EC2::SecurityGroup
   Properties:
     GroupDescription: Security Group for AWS Ground Station registration of Dataflow
Endpoint Groups
```

```
      VpcId: !Ref ReceiverVPC
      SecurityGroupEgress:
        - IpProtocol: udp
          FromPort: 55888
          ToPort: 55888
          CidrIp: 10.0.0.0/8
          Description: "AWS Ground Station Downlink Stream To 10/8"
        - IpProtocol: udp
          FromPort: 55888
          ToPort: 55888
          CidrIp: 172.16.0.0/12
          Description: "AWS Ground Station Downlink Stream To 172.16/12"
        - IpProtocol: udp
          FromPort: 55888
          ToPort: 55888
          CidrIp: 192.168.0.0/16
          Description: "AWS Ground Station Downlink Stream To 192.168/16"

  # The placement group in which your EC2 instance is placed.
  ClusterPlacementGroup:
    Type: AWS::EC2::PlacementGroup
    Properties:
      Strategy: cluster

  ReceiverVPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: "10.0.0.0/16"
      Tags:
        - Key: "Name"
          Value: "AWS Ground Station - PBS to dataflow endpoint Example VPC"
        - Key: "Description"
          Value: "VPC for EC2 instance receiving AWS Ground Station data"

  ReceiverSubnet:
    Type: AWS::EC2::Subnet
    Properties:
      # Ensure your CidrBlock will always have at least one available IP address per
dataflow endpoint.
      # See https://docs.aws.amazon.com/vpc/latest/userguide/subnet-sizing.html for
subent sizing guidelines.
      CidrBlock: "10.0.0.0/24"
      Tags:
        - Key: "Name"
```

```
          Value: "AWS Ground Station - PBS to dataflow endpoint Example Subnet"
        - Key: "Description"
          Value: "Subnet for EC2 instance receiving AWS Ground Station data"
      VpcId: !Ref ReceiverVPC

  # An ENI providing a fixed IP address for AWS Ground Station to connect to.
  ReceiverInstanceNetworkInterface:
    Type: AWS::EC2::NetworkInterface
    Properties:
      Description: Floating network interface providing a fixed IP address for AWS
Ground Station to connect to.
      GroupSet:
        - !Ref InstanceSecurityGroup
      SubnetId: !Ref ReceiverSubnet

  # Attach the ENI to the EC2 instance.
  ReceiverInstanceInterfaceAttachment:
    Type: AWS::EC2::NetworkInterfaceAttachment
    Properties:
      DeleteOnTermination: false
      DeviceIndex: "1"
      InstanceId: !Ref ReceiverInstance
      NetworkInterfaceId: !Ref ReceiverInstanceNetworkInterface
```

In addition, you'll also need to create the appropriate policies and roles to allow AWS Ground Station to create an elastic network interface (ENI) in your account.

```
  # AWS Ground Station assumes this role to create/delete ENIs in your account in order
to stream data.
  DataDeliveryServiceRole:
    Type: AWS::IAM::Role
    Properties:
      Policies:
        - PolicyDocument:
            Statement:
              - Action:
                  - ec2:CreateNetworkInterface
                  - ec2:DeleteNetworkInterface
                  - ec2:CreateNetworkInterfacePermission
                  - ec2:DeleteNetworkInterfacePermission
                  - ec2:DescribeSubnets
```

```
                - ec2:DescribeVpcs
                - ec2:DescribeSecurityGroups
              Effect: Allow
              Resource: '*'
          Version: '2012-10-17'
        PolicyName: DataDeliveryServicePolicy
    AssumeRolePolicyDocument:
      Version: 2012-10-17
      Statement:
        - Effect: Allow
          Principal:
            Service:
            - groundstation.amazonaws.com
          Action:
          - sts:AssumeRole

# The EC2 instance assumes this role.
InstanceRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        - Effect: "Allow"
          Principal:
            Service:
              - "ec2.amazonaws.com"
          Action:
            - "sts:AssumeRole"
    Path: "/"
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
      - arn:aws:iam::aws:policy/service-role/AmazonEC2ContainerServiceforEC2Role
      - arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
      - arn:aws:iam::aws:policy/service-role/AmazonEC2RoleforSSM

# The instance profile for your EC2 instance.
GeneralInstanceProfile:
  Type: AWS::IAM::InstanceProfile
  Properties:
    Roles:
      - !Ref InstanceRole
```

AWS Ground Station configs

This section represents Create configs of getting started.

You'll need a *tracking-config* to set your preference on using autotrack. Selecting *PREFERRED* as autotrack can improve the signal quality, but it isn't required to meet the signal quality due to sufficient JPSS-1 ephemeris quality.

```
 TrackingConfig:
   Type: AWS::GroundStation::Config
   Properties:
     Name: "JPSS Tracking Config"
     ConfigData:
       TrackingConfig:
         Autotrack: "PREFERRED"
```

Based on the communication path, you'll need to define an *antenna-downlink* config to represent the satellite portion, as well as a *dataflow-endpoint* config to refer to the dataflow endpoint group that defines the endpoint details.

```
 # The AWS Ground Station Antenna Downlink Config that defines the frequency spectrum
used to
 # downlink data from your satellite.
 SnppJpssDownlinkDigIfAntennaConfig:
   Type: AWS::GroundStation::Config
   Properties:
     Name: "SNPP JPSS Downlink DigIF Antenna Config"
     ConfigData:
       AntennaDownlinkConfig:
         SpectrumConfig:
           Bandwidth:
             Units: "MHz"
             Value: 30
           CenterFrequency:
             Units: "MHz"
             Value: 7812
           Polarization: "RIGHT_HAND"
```

```
 # The AWS Ground Station Dataflow Endpoint Config that defines the endpoint used to
downlink data
 # from your satellite.
 DownlinkDigIfEndpointConfig:
   Type: AWS::GroundStation::Config
   Properties:
     Name: "Aqua SNPP JPSS Downlink DigIF Endpoint Config"
     ConfigData:
       DataflowEndpointConfig:
         DataflowEndpointName: !Join [ "-" , [ !Ref "AWS::StackName" , "Downlink" ] ]
         DataflowEndpointRegion: !Ref AWS::Region
```

AWS Ground Station mission profile

This section represents Create mission profile of getting started.

Now that you have the associated configs, you can use them to construct the dataflow. You'll use the defaults for the remaining parameters.

```
 # The AWS Ground Station Mission Profile that groups the above configurations to
define how to
 # uplink and downlink data to your satellite.
 SnppJpssMissionProfile:
   Type: AWS::GroundStation::MissionProfile
   Properties:
     Name: "37849 SNPP And 43013 JPSS"
     ContactPrePassDurationSeconds: 120
     ContactPostPassDurationSeconds: 60
     MinimumViableContactDurationSeconds: 180
     TrackingConfigArn: !Ref TrackingConfig
     DataflowEdges:
       - Source: !Ref SnppJpssDownlinkDigIfAntennaConfig
         Destination: !Ref DownlinkDigIfEndpointConfig
```

Putting it together

With the above resources, you now have the ability to schedule JPSS-1 contacts for synchronous data delivery from any of your onboarded AWS Ground Station AWS Ground Station Locations.

The following is a complete CloudFormation template that includes all resources described in this section combined into a single template that can be directly used in CloudFormation.

The CloudFormation template named `AquaSnppJpssTerraDigIF.yml` is designed to give you quick access to start receiving digitized intermediate frequency (DigIF) data for the Aqua, SNPP, JPSS-1/NOAA-20, and Terra satellites. It contains an Amazon EC2 instance and the required CloudFormation resources to receive raw DigIF direct broadcast data.

If Aqua, SNPP, JPSS-1/NOAA-20, and Terra are not onboarded to your account, see Onboard satellite.

Note

You can access the template by accessing the customer onboarding Amazon S3 bucket using valid AWS credentials. The links below use a regional Amazon S3 bucket. Change the `us-west-2` region code to represent the corresponding region of which you want to create the CloudFormation stack in.
Additionally, the following instructions use YAML. However, the templates are available in both YAML and JSON format. To use JSON, replace the `.yml` file extension with `.json` when downloading the template.

To download the template using AWS CLI, use the following command:

```
aws s3 cp s3://groundstation-cloudformation-templates-us-west-2/
AquaSnppJpssTerraDigIF.yml .
```

You can view and download the template in the console by navigating to the following URL in your browser:

```
https://s3.console.aws.amazon.com/s3/object/groundstation-cloudformation-templates-us-
west-2/AquaSnppJpssTerraDigIF.yml
```

You can specify the template directly in CloudFormation using the following link:

```
https://groundstation-cloudformation-templates-us-west-2.s3.us-west-2.amazonaws.com/
AquaSnppJpssTerraDigIF.yml
```

What additional resources does the template define?

The `AquaSnppJpssTerraDigIF` template includes the following additional resources:

- (Optional) **CloudWatch Event Triggers** - AWS Lambda Function that is triggered using CloudWatch Events sent by AWS Ground Station before and after a contact. The AWS Lambda Function will start and optionally stop your Receiver Instance.
- (Optional) **EC2 Verification for Contacts** - The option to use Lambda to set up a verification system of your Amazon EC2 instance(s) for contacts with SNS notification. It is important to note that this may incur charges depending on your current usage.
- **Ground Station Amazon Machine Image Retrieval Lambda** - The option to select what software is installed in your instance and the AMI of your choice. The software options include `DDX 2.6.2 Only` and `DDX 2.6.2 with qRadio 3.6.0`. These options will continue to expand as additional software updates and features are released.
- **Additional mission profiles** - Mission profiles for additional public broadcast satellites (Aqua, SNPP, and Terra).
- **Additional antenna-downlink configs** - Antenna downlink configs for additional public broadcast satellites (Aqua, SNPP, and Terra).

The values and parameters for the satellites in this template are already populated. These parameters make it easy for you to use AWS Ground Station immediately with these satellites. You do not need to configure your own values in order to use AWS Ground Station when using this template. However, you can customize the values to make the template work for your use case.

Where do I receive my data?

The dataflow endpoint group is set up to use the receiver instance network interface that part of the template creates. The receiver instance uses a dataflow endpoint application to receive the data stream from AWS Ground Station on the port defined by the dataflow endpoint. Once received, the data is available for consumption via UDP port 50000 on the loopback adapter of the receiver instance. For more information about setting up a dataflow endpoint group, see AWS::GroundStation::DataflowEndpointGroup.

Public broadcast satellite utilizing a dataflow endpoint (demodulated and decoded)

This example builds off the analysis done in the JPSS-1 - Public broadcast satellite (PBS) - Evaluation section of the user guide.

To complete this example, you'll need to assume a scenario -- you want to capture the HRD communication path as demodulated and decoded direct broadcast data using a dataflow endpoint. This example is a good starting point if you plan to process the data using NASA Direct Readout Labs software (RT-STPS and IPOPP).

Communication paths

This section represents Plan your dataflow communication paths of getting started. For this example, you will be creating two sections in your CloudFormation template: Parameters and Resources sections.

> **Note**
>
> For more information about the contents of a CloudFormation template, see Template sections.

For the Parameters section, you're going to add the following parameters. You'll specify values for these when creating the stack via the CloudFormation console.

```
Parameters:
  EC2Key:
    Description: The SSH key used to access the EC2 receiver instance. Choose any
 SSH key if you are not creating an EC2 receiver instance. For instructions on how to
 create an SSH key see https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/create-
key-pairs.html
    Type: AWS::EC2::KeyPair::KeyName
    ConstraintDescription: must be the name of an existing EC2 KeyPair.

  ReceiverAMI:
    Description: The Ground Station DDX AMI ID you want to use. Please note
 that AMIs are region specific. For instructions on how to retrieve an AMI
 see https://docs.aws.amazon.com/ground-station/latest/ug/dataflows.ec2-
configuration.html#dataflows.ec2-configuration.amis
    Type: AWS::EC2::Image::Id
```

> **Note**
>
> You **need** to create a key pair, and provide the name for the Amazon EC2 EC2Key parameter. See Create a key pair for your Amazon EC2 instance.
> Additionally, you'll **need** to provide the correct **region specific** AMI ID, when creating the CloudFormation stack. See AWS Ground Station Amazon Machine Images (AMIs).

The remaining template snippets belong in the Resources section of the CloudFormation template.

```
Resources:
  # Resources that you would like to create should be placed within the resource
 section.
```

Given our scenario to deliver a single communication path to an EC2 instance, you'll have a single synchronous delivery path. Per the Synchronous data delivery section, you must set up and configure an Amazon EC2 instance with a dataflow endpoint application, and create one or more dataflow endpoint groups.

```
  # The EC2 instance that will send/receive data to/from your satellite using AWS
 Ground Station.
  ReceiverInstance:
    Type: AWS::EC2::Instance
    Properties:
      DisableApiTermination: false
      IamInstanceProfile: !Ref GeneralInstanceProfile
      ImageId: !Ref ReceiverAMI
      InstanceType: m5.4xlarge
      KeyName: !Ref EC2Key
      Monitoring: true
      PlacementGroupName: !Ref ClusterPlacementGroup
      SecurityGroupIds:
        - Ref: InstanceSecurityGroup
      SubnetId: !Ref ReceiverSubnet
      BlockDeviceMappings:
        - DeviceName: /dev/xvda
          Ebs:
            VolumeType: gp2
```

```
            VolumeSize: 40
      Tags:
        - Key: Name
          Value: !Join [ "-" , [ "Receiver" , !Ref "AWS::StackName" ] ]
      UserData:
        Fn::Base64:
          |
          #!/bin/bash
          exec > >(tee /var/log/user-data.log|logger -t user-data -s 2>/dev/console)
2>&1
          echo `date +'%F %R:%S'` "INFO: Logging Setup" >&2

          GROUND_STATION_DIR="/opt/aws/groundstation"
          GROUND_STATION_BIN_DIR="${GROUND_STATION_DIR}/bin"
          STREAM_CONFIG_PATH="${GROUND_STATION_DIR}/customer_stream_config.json"

          echo "Creating ${STREAM_CONFIG_PATH}"
          cat << STREAM_CONFIG > "${STREAM_CONFIG_PATH}"
          {
            "ddx_streams": [
              {
                "streamName": "Downlink",
                "maximumWanRate": 4000000000,
                "lanConfigDevice": "lo",
                "lanConfigPort": 50000,
                "wanConfigDevice": "eth1",
                "wanConfigPort": 55888,
                "isUplink": false
              }
            ]
          }
          STREAM_CONFIG

          echo "Waiting for dataflow endpoint application to start"
          while netstat -lnt | awk '$4 ~ /:80$/ {exit 1}'; do sleep 10; done

          echo "Configuring dataflow endpoint application streams"
          python "${GROUND_STATION_BIN_DIR}/configure_streams.py" --configFileName
"${STREAM_CONFIG_PATH}"
          sleep 2
          python "${GROUND_STATION_BIN_DIR}/save_default_config.py"

          exit 0
```

```
 # The AWS Ground Station Dataflow Endpoint Group that defines the endpoints that AWS
Ground
 # Station will use to send/receive data to/from your satellite.
 DataflowEndpointGroup:
   Type: AWS::GroundStation::DataflowEndpointGroup
   Properties:
     ContactPostPassDurationSeconds: 180
     ContactPrePassDurationSeconds: 120
     EndpointDetails:
       - Endpoint:
           Name: !Join [ "-" , [ !Ref "AWS::StackName" , "Downlink" ] ] # needs to
match DataflowEndpointConfig name
           Address:
             Name: !GetAtt ReceiverInstanceNetworkInterface.PrimaryPrivateIpAddress
             Port: 55888
         SecurityDetails:
           SecurityGroupIds:
             - Ref: "DataflowEndpointSecurityGroup"
           SubnetIds:
             - !Ref ReceiverSubnet
           RoleArn: !GetAtt DataDeliveryServiceRole.Arn

 # The security group that the ENI created by AWS Ground Station belongs to.
 DataflowEndpointSecurityGroup:
   Type: AWS::EC2::SecurityGroup
   Properties:
     GroupDescription: Security Group for AWS Ground Station registration of Dataflow
Endpoint Groups
     VpcId: !Ref ReceiverVPC
     SecurityGroupEgress:
       - IpProtocol: udp
         FromPort: 55888
         ToPort: 55888
         CidrIp: 10.0.0.0/8
         Description: "AWS Ground Station Downlink Stream To 10/8"
       - IpProtocol: udp
         FromPort: 55888
         ToPort: 55888
         CidrIp: 172.16.0.0/12
         Description: "AWS Ground Station Downlink Stream To 172.16/12"
       - IpProtocol: udp
         FromPort: 55888
```

```
          ToPort: 55888
          CidrIp: 192.168.0.0/16
          Description: "AWS Ground Station Downlink Stream To 192.168/16"

  # The placement group in which your EC2 instance is placed.
  ClusterPlacementGroup:
    Type: AWS::EC2::PlacementGroup
    Properties:
      Strategy: cluster

  # The security group for your EC2 instance.
  InstanceSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: AWS Ground Station receiver instance security group.
      VpcId: !Ref ReceiverVPC
      SecurityGroupIngress:
        # To allow SSH access to the instance, add another rule allowing tcp port 22
from your CidrIp
        - IpProtocol: udp
          FromPort: 55888
          ToPort: 55888
          SourceSecurityGroupId: !Ref DataflowEndpointSecurityGroup
          Description: "AWS Ground Station Downlink Stream"

  ReceiverVPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: "10.0.0.0/16"
      Tags:
        - Key: "Name"
          Value: "AWS Ground Station - PBS to dataflow endpoint Demod Decode Example
VPC"
        - Key: "Description"
          Value: "VPC for EC2 instance receiving AWS Ground Station data"

  ReceiverSubnet:
    Type: AWS::EC2::Subnet
    Properties:
      CidrBlock: "10.0.0.0/24"
      Tags:
        - Key: "Name"
          Value: "AWS Ground Station - PBS to dataflow endpoint Demod Decode Example
Subnet"
```

```
        - Key: "Description"
          Value: "Subnet for EC2 instance receiving AWS Ground Station data"
      VpcId: !Ref ReceiverVPC

  # An ENI providing a fixed IP address for AWS Ground Station to connect to.
  ReceiverInstanceNetworkInterface:
    Type: AWS::EC2::NetworkInterface
    Properties:
      Description: Floating network interface providing a fixed IP address for AWS
Ground Station to connect to.
      GroupSet:
        - !Ref InstanceSecurityGroup
      SubnetId: !Ref ReceiverSubnet

  # Attach the ENI to the EC2 instance.
  ReceiverInstanceInterfaceAttachment:
    Type: AWS::EC2::NetworkInterfaceAttachment
    Properties:
      DeleteOnTermination: false
      DeviceIndex: "1"
      InstanceId: !Ref ReceiverInstance
      NetworkInterfaceId: !Ref ReceiverInstanceNetworkInterface

  # The instance profile for your EC2 instance.
  GeneralInstanceProfile:
    Type: AWS::IAM::InstanceProfile
    Properties:
      Roles:
        - !Ref InstanceRole
```

You'll also need the appropriate policies, roles, and profiles to allow AWS Ground Station to create an elastic network interface (ENI) in your account.

```
  # AWS Ground Station assumes this role to create/delete ENIs in your account in order
to stream data.
  DataDeliveryServiceRole:
    Type: AWS::IAM::Role
    Properties:
      Policies:
        - PolicyDocument:
            Statement:
```

```
            - Action:
                - ec2:CreateNetworkInterface
                - ec2:DeleteNetworkInterface
                - ec2:CreateNetworkInterfacePermission
                - ec2:DeleteNetworkInterfacePermission
                - ec2:DescribeSubnets
                - ec2:DescribeVpcs
                - ec2:DescribeSecurityGroups
              Effect: Allow
              Resource: '*'
          Version: '2012-10-17'
        PolicyName: DataDeliveryServicePolicy
    AssumeRolePolicyDocument:
      Version: 2012-10-17
      Statement:
        - Effect: Allow
          Principal:
            Service:
            - groundstation.amazonaws.com
          Action:
          - sts:AssumeRole

# The EC2 instance assumes this role.
InstanceRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        - Effect: "Allow"
          Principal:
            Service:
              - "ec2.amazonaws.com"
          Action:
            - "sts:AssumeRole"
    Path: "/"
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
      - arn:aws:iam::aws:policy/service-role/AmazonEC2ContainerServiceforEC2Role
      - arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
      - arn:aws:iam::aws:policy/service-role/AmazonEC2RoleforSSM
```

AWS Ground Station configs

This section represents Create configs of the user guide.

You'll need a *tracking-config* to set your preference on using autotrack. Selecting *PREFERRED* as autotrack can improve the signal quality, but it isn't required to meet the signal quality due to sufficient JPSS-1 ephemeris quality.

```
TrackingConfig:
  Type: AWS::GroundStation::Config
  Properties:
    Name: "JPSS Tracking Config"
    ConfigData:
      TrackingConfig:
        Autotrack: "PREFERRED"
```

Based on the communication path, you'll need to define an *antenna-downlink-demod-decode* config to represent the satellite portion, as well as a *dataflow-endpoint* config to refer to the dataflow endpoint group that defines the endpoint details.

> **Note**
>
> For details on how to set the values for `DemodulationConfig`, and `DecodeConfig`, please see Antenna Downlink Demod Decode Config.

```
 # The AWS Ground Station Antenna Downlink Config that defines the frequency spectrum
used to
 # downlink data from your satellite.
 JpssDownlinkDemodDecodeAntennaConfig:
   Type: AWS::GroundStation::Config
   Properties:
     Name: "JPSS Downlink Demod Decode Antenna Config"
     ConfigData:
       AntennaDownlinkDemodDecodeConfig:
         SpectrumConfig:
           CenterFrequency:
             Value: 7812
```

```
        Units: "MHz"
      Polarization: "RIGHT_HAND"
      Bandwidth:
        Value: 30
        Units: "MHz"
    DemodulationConfig:
      UnvalidatedJSON: '{
        "type":"QPSK",
        "qpsk":{
          "carrierFrequencyRecovery":{
            "centerFrequency":{
              "value":7812,
              "units":"MHz"
            },
            "range":{
              "value":250,
              "units":"kHz"
            }
          },
          "symbolTimingRecovery":{
            "symbolRate":{
              "value":15,
              "units":"Msps"
            },
            "range":{
              "value":0.75,
              "units":"ksps"
            },
            "matchedFilter":{
              "type":"ROOT_RAISED_COSINE",
              "rolloffFactor":0.5
            }
          }
        }
      }'
    DecodeConfig:
      UnvalidatedJSON: '{
        "edges":[
          {
            "from":"I-Ingress",
            "to":"IQ-Recombiner"
          },
          {
            "from":"Q-Ingress",
```

```
      "to":"IQ-Recombiner"
    },
    {
      "from":"IQ-Recombiner",
      "to":"CcsdsViterbiDecoder"
    },
    {
      "from":"CcsdsViterbiDecoder",
      "to":"NrzmDecoder"
    },
    {
      "from":"NrzmDecoder",
      "to":"UncodedFramesEgress"
    }
  ],
  "nodeConfigs":{
    "I-Ingress":{
      "type":"CODED_SYMBOLS_INGRESS",
      "codedSymbolsIngress":{
        "source":"I"
      }
    },
    "Q-Ingress":{
      "type":"CODED_SYMBOLS_INGRESS",
      "codedSymbolsIngress":{
        "source":"Q"
      }
    },
    "IQ-Recombiner":{
      "type":"IQ_RECOMBINER"
    },
    "CcsdsViterbiDecoder":{
      "type":"CCSDS_171_133_VITERBI_DECODER",
      "ccsds171133ViterbiDecoder":{
        "codeRate":"ONE_HALF"
      }
    },
    "NrzmDecoder":{
      "type":"NRZ_M_DECODER"
    },
    "UncodedFramesEgress":{
      "type":"UNCODED_FRAMES_EGRESS"
    }
  }
```

```
            }'
```

```
 # The AWS Ground Station Dataflow Endpoint Config that defines the endpoint used to
downlink data
 # from your satellite.
 DownlinkDemodDecodeEndpointConfig:
   Type: AWS::GroundStation::Config
   Properties:
     Name: "Aqua SNPP JPSS Downlink Demod Decode Endpoint Config"
     ConfigData:
       DataflowEndpointConfig:
         DataflowEndpointName: !Join [ "-" , [ !Ref "AWS::StackName" , "Downlink" ] ]
         DataflowEndpointRegion: !Ref AWS::Region
```

AWS Ground Station mission profile

This section represents Create mission profile of the user guide.

Now that you have the associated configs, you can use them to construct the dataflow. You'll use the defaults for the remaining parameters.

```
 # The AWS Ground Station Mission Profile that groups the above configurations to
define how to
 # uplink and downlink data to your satellite.
 SnppJpssMissionProfile:
   Type: AWS::GroundStation::MissionProfile
   Properties:
     Name: "37849 SNPP And 43013 JPSS"
     ContactPrePassDurationSeconds: 120
     ContactPostPassDurationSeconds: 60
     MinimumViableContactDurationSeconds: 180
     TrackingConfigArn: !Ref TrackingConfig
     DataflowEdges:
       - Source: !Join [ "/", [ !Ref JpssDownlinkDemodDecodeAntennaConfig,
"UncodedFramesEgress" ] ]
         Destination: !Ref DownlinkDemodDecodeEndpointConfig
```

Putting it together

With the above resources, you now have the ability to schedule JPSS-1 contacts for synchronous data delivery from any of your onboarded AWS Ground Station AWS Ground Station Locations.

The following is a complete CloudFormation template that includes all resources described in this section combined into a single template that can be directly used in CloudFormation.

The CloudFormation template named `AquaSnppJpss.yml` is designed to give you quick access to start receiving data for the Aqua, SNPP, and JPSS-1/NOAA-20 satellites. It contains an Amazon EC2 instance and the required AWS Ground Station resources to schedule contacts and receive demodulated and decoded direct broadcast data.

If Aqua, SNPP, JPSS-1/NOAA-20, and Terra are not onboarded to your account, see Onboard satellite.

> **Note**
>
> You can access the template by accessing the customer onboarding Amazon S3 bucket using valid AWS credentials. The links below use a regional Amazon S3 bucket. Change the `us-west-2` region code to represent the corresponding region of which you want to create the CloudFormation stack in.
> Additionally, the following instructions use YAML. However, the templates are available in both YAML and JSON format. To use JSON, replace the `.yml` file extension with `.json` when downloading the template.

To download the template using AWS CLI, use the following command:

```
aws s3 cp s3://groundstation-cloudformation-templates-us-west-2/AquaSnppJpss.yml .
```

You can view and download the template in the console by navigating to the following URL in your browser:

```
https://s3.console.aws.amazon.com/s3/object/groundstation-cloudformation-templates-us-west-2/AquaSnppJpss.yml
```

You can specify the template directly in CloudFormation using the following link:

```
https://groundstation-cloudformation-templates-us-west-2.s3.us-west-2.amazonaws.com/
AquaSnppJpss.yml
```

What additional resources does the template define?

The `AquaSnppJpss` template includes the following additional resources:

- (Optional) **CloudWatch Event Triggers** - AWS Lambda Function that is triggered using CloudWatch Events sent by AWS Ground Station before and after a contact. The AWS Lambda Function will start and optionally stop your Receiver Instance.
- (Optional) **EC2 Verification for Contacts** - The option to use Lambda to set up a verification system of your Amazon EC2 instance(s) for contacts with SNS notification. It is important to note that this may incur charges depending on your current usage.
- **Ground Station Amazon Machine Image Retrieval Lambda** - The option to select what software is installed in your instance and the AMI of your choice. The software options include `DDX 2.6.2 Only` and `DDX 2.6.2 with qRadio 3.6.0`. If you want to use Wideband DigIF Data Delivery and the AWS Ground Station Agent, please see Public broadcast satellite utilizing AWS Ground Station Agent (wideband). These options will continue to expand as additional software updates and features are released.
- **Additional mission profiles** - Mission profiles for additional public broadcast satellites (Aqua, SNPP, and Terra).
- **Additional antenna-downlink configs** - Antenna downlink configs for additional public broadcast satellites (Aqua, SNPP, and Terra).

The values and parameters for the satellites in this template are already populated. These parameters make it easy for you to use AWS Ground Station immediately with these satellites. You do not need to configure your own values in order to use AWS Ground Station when using this template. However, you can customize the values to make the template work for your use case.

Where do I receive my data?

The dataflow endpoint group is set up to use the receiver instance network interface that part of the template creates. The receiver instance uses a dataflow endpoint application to receive the data stream from AWS Ground Station on the port defined by the dataflow endpoint. Once received, the data is available for consumption via UDP port 50000 on the loopback adapter of the receiver instance. For more information about setting up a dataflow endpoint group, see AWS::GroundStation::DataflowEndpointGroup.

Public broadcast satellite utilizing AWS Ground Station Agent (wideband)

This example builds off the analysis done in the JPSS-1 - Public broadcast satellite (PBS) - Evaluation section of the user guide.

To complete this example, you'll need to assume a scenario -- you want to capture the HRD communication path as wideband digital intermediate frequency (DigIF) and process it as it's received by the AWS Ground Station Agent on an Amazon EC2 instance using an SDR.

> **Note**
>
> The actual JPSS HRD communication path signal has a bandwidth of 30 MHz, but you will configure the *antenna-downlink* config to treat it as a signal with a 100 MHz bandwidth so that it can flow through the correct path to be received by the AWS Ground Station Agent for this example.

Communication paths

This section represents Plan your dataflow communication paths of getting started. For this example, you will need an additional section in your CloudFormation template that hasn't been used in the other examples, the Mappings section.

> **Note**
>
> For more information about the contents of a CloudFormation template, see Template sections.

You'll begin by setting up a Mappings section in your CloudFormation template for the AWS Ground Station prefix lists by region. This allows the prefix lists to be easily referenced by the Amazon EC2 instance security group. For more information about using a prefix list, see VPC Configuration with AWS Ground Station Agent.

```
Mappings:
  PrefixListId:
```

```
    us-east-2:
      groundstation: pl-087f83ba4f34e3bea
    us-west-2:
      groundstation: pl-0cc36273da754ebdc
    us-east-1:
      groundstation: pl-0e5696d987d033653
    eu-central-1:
      groundstation: pl-03743f81267c0a85e
    sa-east-1:
      groundstation: pl-098248765e9effc20
    ap-northeast-2:
      groundstation: pl-059b3e0b02af70e4d
    ap-southeast-1:
      groundstation: pl-0d9b804fe014a6a99
    ap-southeast-2:
      groundstation: pl-08d24302b8c4d2b73
    me-south-1:
      groundstation: pl-02781422c4c792145
    eu-west-1:
      groundstation: pl-03fa6b266557b0d4f
    eu-north-1:
      groundstation: pl-033e44023025215c0
    af-south-1:
      groundstation: pl-0382d923a9d555425
```

For the Parameters section, you're going to add the following parameters. You'll specify values for these when creating the stack via the CloudFormation console.

```
Parameters:
  EC2Key:
    Description: The SSH key used to access the EC2 receiver instance. Choose any
 SSH key if you are not creating an EC2 receiver instance. For instructions on how to
 create an SSH key see https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/create-
key-pairs.html
    Type: AWS::EC2::KeyPair::KeyName
    ConstraintDescription: must be the name of an existing EC2 KeyPair.

  AZ:
    Description: "The AvailabilityZone that the resources of this stack will be created
 in. (e.g. us-east-2a)"
    Type: AWS::EC2::AvailabilityZone::Name
```

```
  ReceiverAMI:
    Description: The Ground Station Agent AMI ID you want to use. Please note
  that AMIs are region specific. For instructions on how to retrieve an AMI
  see https://docs.aws.amazon.com/ground-station/latest/ug/dataflows.ec2-
configuration.html#dataflows.ec2-configuration.amis
    Type: AWS::EC2::Image::Id
```

Note

You **need** to create a key pair, and provide the name for the Amazon EC2 EC2Key parameter. See Create a key pair for your Amazon EC2 instance.
Additionally, you'll **need** to provide the correct **region specific** AMI ID, when creating the CloudFormation stack. See AWS Ground Station Amazon Machine Images (AMIs).

The remaining template snippets belong in the Resources section of the CloudFormation template.

```
Resources:
  # Resources that you would like to create should be placed within the Resources
  section.
```

Given our scenario to deliver a single communication path to an Amazon EC2 instance, you know that you'll have a single synchronous delivery path. Per the Synchronous data delivery section, you must set up and configure an Amazon EC2 instance with AWS Ground Station Agent, and create one or more dataflow endpoint groups. You'll begin by first setting up the Amazon VPC for the AWS Ground Station Agent.

```
  ReceiverVPC:
    Type: AWS::EC2::VPC
    Properties:
      EnableDnsSupport: 'true'
      EnableDnsHostnames: 'true'
      CidrBlock: 10.0.0.0/16
      Tags:
      - Key: "Name"
        Value: "AWS Ground Station Example - PBS to AWS Ground Station Agent VPC"
      - Key: "Description"
        Value: "VPC for EC2 instance receiving AWS Ground Station data"
```

```
 PublicSubnet:
   Type: AWS::EC2::Subnet
   Properties:
     VpcId: !Ref ReceiverVPC
     MapPublicIpOnLaunch: 'true'
     AvailabilityZone: !Ref AZ
     CidrBlock: 10.0.0.0/20
     Tags:
     - Key: "Name"
       Value: "AWS Ground Station Example - PBS to AWS Ground Station Agent Public
Subnet"
     - Key: "Description"
       Value: "Subnet for EC2 instance receiving AWS Ground Station data"

 RouteTable:
   Type: AWS::EC2::RouteTable
   Properties:
     VpcId: !Ref ReceiverVPC
     Tags:
       - Key: Name
         Value: AWS Ground Station Example - RouteTable

 RouteTableAssociation:
   Type: AWS::EC2::SubnetRouteTableAssociation
   Properties:
     RouteTableId: !Ref RouteTable
     SubnetId: !Ref PublicSubnet

 Route:
   Type: AWS::EC2::Route
   DependsOn: InternetGateway
   Properties:
     RouteTableId: !Ref RouteTable
     DestinationCidrBlock: '0.0.0.0/0'
     GatewayId: !Ref InternetGateway

 InternetGateway:
   Type: AWS::EC2::InternetGateway
   Properties:
     Tags:
       - Key: Name
         Value: AWS Ground Station Example - Internet Gateway
```

```
  GatewayAttachment:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      VpcId: !Ref ReceiverVPC
      InternetGatewayId: !Ref InternetGateway
```

> **Note**
>
> For more information about the VPC configurations supported by the AWS Ground Station Agent, see AWS Ground Station Agent Requirements - VPC diagrams.

Next, you'll set up the Receiver Amazon EC2 instance.

```
  # The placement group in which your EC2 instance is placed.
  ClusterPlacementGroup:
    Type: AWS::EC2::PlacementGroup
    Properties:
      Strategy: cluster

  # This is required for the EIP if the receiver EC2 instance is in a private subnet.
  # This ENI must exist in a public subnet, be attached to the receiver and be
associated with the EIP.
  ReceiverInstanceNetworkInterface:
    Type: AWS::EC2::NetworkInterface
    Properties:
      Description: Floating network interface
      GroupSet:
        - !Ref InstanceSecurityGroup
      SubnetId: !Ref PublicSubnet

  # An EIP providing a fixed IP address for AWS Ground Station to connect to. Attach it
to the receiver instance created in the stack.
  ReceiverInstanceElasticIp:
    Type: AWS::EC2::EIP
    Properties:
      Tags:
        - Key: Name
          Value: !Join [ "-" , [ "EIP" , !Ref "AWS::StackName" ] ]

  # Attach the ENI to the EC2 instance if using a separate public subnet.
```

```
  # Requires the receiver instance to be in a public subnet (SubnetId should be the id
 of a public subnet)
  ReceiverNetworkInterfaceAttachment:
    Type: AWS::EC2::NetworkInterfaceAttachment
    Properties:
      DeleteOnTermination: false
      DeviceIndex: 1
      InstanceId: !Ref ReceiverInstance
      NetworkInterfaceId: !Ref ReceiverInstanceNetworkInterface

  # Associate EIP with the ENI if using a separate public subnet for the ENI.
  ReceiverNetworkInterfaceElasticIpAssociation:
    Type: AWS::EC2::EIPAssociation
    Properties:
      AllocationId: !GetAtt [ReceiverInstanceElasticIp, AllocationId]
      NetworkInterfaceId: !Ref ReceiverInstanceNetworkInterface

  # The EC2 instance that will send/receive data to/from your satellite using AWS
 Ground Station.
  ReceiverInstance:
    Type: AWS::EC2::Instance
    DependsOn: PublicSubnet
    Properties:
      DisableApiTermination: false
      IamInstanceProfile: !Ref GeneralInstanceProfile
      ImageId: !Ref ReceiverAMI
      AvailabilityZone: !Ref AZ
      InstanceType: c5.24xlarge
      KeyName: !Ref EC2Key
      Monitoring: true
      PlacementGroupName: !Ref ClusterPlacementGroup
      SecurityGroupIds:
        - Ref: InstanceSecurityGroup
      SubnetId: !Ref PublicSubnet
      Tags:
        - Key: Name
          Value: !Join [ "-" , [ "Receiver" , !Ref "AWS::StackName" ] ]
      # agentCpuCores list in the AGENT_CONFIG below defines the cores that the AWS
 Ground Station Agent is allowed to run on. This list can be changed to suit your use-
case, however if the agent isn't supplied with enough cores data loss may occur.
      UserData:
        Fn::Base64:
          Fn::Sub:
            - |
```

```
              #!/bin/bash
              yum -y update

              AGENT_CONFIG_PATH="/opt/aws/groundstation/etc/aws-gs-agent-config.json"
              cat << AGENT_CONFIG > "$AGENT_CONFIG_PATH"
              {
                "capabilities": [
                  "arn:aws:groundstation:${AWS::Region}:${AWS::AccountId}:dataflow-
endpoint-group/${DataflowEndpointGroupId}"
                ],
                "device": {
                  "privateIps": [
                    "127.0.0.1"
                  ],
                  "publicIps": [
                    "${EIP}"
                  ],
                  "agentCpuCores": [

 24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,72,73,74,75,76,77,78,79,80,81,8
                  ]
                }
              }
              AGENT_CONFIG

              systemctl start aws-groundstation-agent
              systemctl enable aws-groundstation-agent

              # <Tuning Section Start>
              # Visit the AWS Ground Station Agent Documentation in the User Guide for
 more details and guidance updates

              # Set IRQ affinity with list of CPU cores and Receive Side Scaling mask
              # Core list should be the first two cores (and hyperthreads) on each
 socket
              # Mask set to everything currently
              # https://github.com/torvalds/linux/blob/v4.11/Documentation/networking/
scaling.txt#L80-L96
              echo "@reboot sudo /opt/aws/groundstation/bin/set_irq_affinity.sh '0 1 48
 49' 'ffffffff,ffffffff,ffffffff' >>/var/log/user-data.log 2>&1" >>/var/spool/cron/root

              # Reserving the port range defined in the GS agent ingress address in
 the Dataflow Endpoint Group so the kernel doesn't steal any of them from the GS agent.
 These ports are the ports that the GS agent will ingress data
```

```
              # across, so if the kernel steals one it could cause problems ingressing
data onto the instance.
              echo net.ipv4.ip_local_reserved_ports="42000-50000" >> /etc/sysctl.conf

              # </Tuning Section End>

              # We have to reboot for linux kernel settings to apply
              shutdown -r now

            - DataflowEndpointGroupId: !Ref DataflowEndpointGroup
              EIP: !Ref ReceiverInstanceElasticIp
```

```
 # The AWS Ground Station Dataflow Endpoint Group that defines the endpoints that AWS
Ground
 # Station will use to send/receive data to/from your satellite.
 DataflowEndpointGroup:
   Type: AWS::GroundStation::DataflowEndpointGroup
   Properties:
     ContactPostPassDurationSeconds: 180
     ContactPrePassDurationSeconds: 120
     EndpointDetails:
       - AwsGroundStationAgentEndpoint:
           Name: !Join [ "-" , [ !Ref "AWS::StackName" , "Downlink" ] ] # needs to
match DataflowEndpointConfig name
           EgressAddress:
             SocketAddress:
               Name: 127.0.0.1
               Port: 55000
           IngressAddress:
             SocketAddress:
               Name: !Ref ReceiverInstanceElasticIp
               PortRange:
                 Minimum: 42000
                 Maximum: 55000
```

You'll also need the appropriate policies, roles, and profiles to allow AWS Ground Station to create the elastic network interface (ENI) in your account.

```
 # The security group for your EC2 instance.
 InstanceSecurityGroup:
```

```
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: AWS Ground Station receiver instance security group.
      VpcId: !Ref ReceiverVPC
      SecurityGroupEgress:
        - CidrIp: 0.0.0.0/0
          Description: Allow all outbound traffic by default
          IpProtocol: "-1"
      SecurityGroupIngress:
        # To allow SSH access to the instance, add another rule allowing tcp port 22
from your CidrIp
        - IpProtocol: udp
          Description: Allow AWS Ground Station Incoming Dataflows
          ToPort: 50000
          FromPort: 42000
          SourcePrefixListId:
            Fn::FindInMap:
              - PrefixListId
              - Ref: AWS::Region
              - groundstation

  # The EC2 instance assumes this role.
 InstanceRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: "Allow"
            Principal:
              Service:
                - "ec2.amazonaws.com"
            Action:
              - "sts:AssumeRole"
      Path: "/"
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
        - arn:aws:iam::aws:policy/service-role/AmazonEC2ContainerServiceforEC2Role
        - arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
        - arn:aws:iam::aws:policy/service-role/AmazonEC2RoleforSSM
        - arn:aws:iam::aws:policy/AWSGroundStationAgentInstancePolicy
      Policies:
        - PolicyDocument:
            Statement:
```

```
              - Action:
                  - sts:AssumeRole
                Effect: Allow
                Resource: !GetAtt GroundStationKmsKeyRole.Arn
            Version: "2012-10-17"
          PolicyName: InstanceGroundStationApiAccessPolicy

  # The instance profile for your EC2 instance.
  GeneralInstanceProfile:
    Type: AWS::IAM::InstanceProfile
    Properties:
      Roles:
        - !Ref InstanceRole

  # The IAM role that AWS Ground Station will assume to access and use the KMS Key for
 data delivery
  GroundStationKmsKeyRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service:
                - groundstation.amazonaws.com
            Condition:
              StringEquals:
                "aws:SourceAccount": !Ref AWS::AccountId
              ArnLike:
                "aws:SourceArn": !Sub "arn:${AWS::Partition}:groundstation:
${AWS::Region}:${AWS::AccountId}:mission-profile/*"
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              AWS: !Sub "arn:${AWS::Partition}:iam::${AWS::AccountId}:root"

  GroundStationKmsKeyAccessPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - kms:Decrypt
```

```
            Effect: Allow
            Resource: !GetAtt GroundStationDataDeliveryKmsKey.Arn
      PolicyName: GroundStationKmsKeyAccessPolicy
      Roles:
        - Ref: GroundStationKmsKeyRole

  GroundStationDataDeliveryKmsKey:
    Type: AWS::KMS::Key
    Properties:
      KeyPolicy:
        Statement:
          - Action:
              - kms:CreateAlias
              - kms:Describe*
              - kms:Enable*
              - kms:List*
              - kms:Put*
              - kms:Update*
              - kms:Revoke*
              - kms:Disable*
              - kms:Get*
              - kms:Delete*
              - kms:ScheduleKeyDeletion
              - kms:CancelKeyDeletion
              - kms:GenerateDataKey
              - kms:TagResource
              - kms:UntagResource
            Effect: Allow
            Principal:
              AWS: !Sub "arn:${AWS::Partition}:iam::${AWS::AccountId}:root"
            Resource: "*"
          - Action:
              - kms:Decrypt
              - kms:GenerateDataKeyWithoutPlaintext
            Effect: Allow
            Principal:
              AWS: !GetAtt GroundStationKmsKeyRole.Arn
            Resource: "*"
            Condition:
              StringEquals:
                "kms:EncryptionContext:sourceAccount": !Ref AWS::AccountId
              ArnLike:
                "kms:EncryptionContext:sourceArn": !Sub "arn:
${AWS::Partition}:groundstation:${AWS::Region}:${AWS::AccountId}:mission-profile/*"
```

```
          - Action:
              - kms:CreateGrant
            Effect: Allow
            Principal:
              AWS: !Sub "arn:${AWS::Partition}:iam::${AWS::AccountId}:root"
            Resource: "*"
            Condition:
              ForAllValues:StringEquals:
                "kms:GrantOperations":
                  - Decrypt
                  - GenerateDataKeyWithoutPlaintext
                "kms:EncryptionContextKeys":
                  - sourceArn
                  - sourceAccount
              ArnLike:
                "kms:EncryptionContext:sourceArn": !Sub "arn:
${AWS::Partition}:groundstation:${AWS::Region}:${AWS::AccountId}:mission-profile/*"
              StringEquals:
                "kms:EncryptionContext:sourceAccount": !Ref AWS::AccountId
        Version: "2012-10-17"
      EnableKeyRotation: true
```

AWS Ground Station configs

This section represents Create configs of getting started.

You'll need a *tracking-config* to set your preference on using autotrack. Selecting *PREFERRED* as autotrack can improve the signal quality, but it isn't required to meet the signal quality due to sufficient JPSS-1 ephemeris quality.

```
  TrackingConfig:
    Type: AWS::GroundStation::Config
    Properties:
      Name: "JPSS Tracking Config"
      ConfigData:
        TrackingConfig:
          Autotrack: "PREFERRED"
```

Based on the communication path, you'll need to define an *antenna-downlink* config to represent the satellite portion, as well as a *dataflow-endpoint* config to refer to the dataflow endpoint group that defines the endpoint details.

```
 # The AWS Ground Station Antenna Downlink Config that defines the frequency spectrum
used to
 # downlink data from your satellite.
 SnppJpssDownlinkDigIfAntennaConfig:
   Type: AWS::GroundStation::Config
   Properties:
     Name: "SNPP JPSS Downlink WBDigIF Antenna Config"
     ConfigData:
       AntennaDownlinkConfig:
         SpectrumConfig:
           Bandwidth:
             Units: "MHz"
             Value: 100
           CenterFrequency:
             Units: "MHz"
             Value: 7812
           Polarization: "RIGHT_HAND"

 # The AWS Ground Station Dataflow Endpoint Config that defines the endpoint used to
downlink data
 # from your satellite.
 DownlinkDigIfEndpointConfig:
   Type: AWS::GroundStation::Config
   Properties:
     Name: "Aqua SNPP JPSS Terra Downlink DigIF Endpoint Config"
     ConfigData:
       DataflowEndpointConfig:
         DataflowEndpointName: !Join [ "-" , [ !Ref "AWS::StackName" , "Downlink" ] ]
         DataflowEndpointRegion: !Ref AWS::Region
```

AWS Ground Station mission profile

This section represents Create mission profile of getting started.

Now that you have the associated configs, you can use them to construct the dataflow. You'll use the defaults for the remaining parameters.

```
 # The AWS Ground Station Mission Profile that groups the above configurations to
define how to
 # uplink and downlink data to your satellite.
 SnppJpssMissionProfile:
   Type: AWS::GroundStation::MissionProfile
   Properties:
     Name: !Sub 'JPSS WBDigIF gs-agent EC2 Delivery'
     ContactPrePassDurationSeconds: 120
     ContactPostPassDurationSeconds: 120
     MinimumViableContactDurationSeconds: 180
     TrackingConfigArn: !Ref TrackingConfig
     DataflowEdges:
       - Source: !Ref SnppJpssDownlinkDigIfAntennaConfig
         Destination: !Ref DownlinkDigIfEndpointConfig
     StreamsKmsKey:
       KmsKeyArn: !GetAtt GroundStationDataDeliveryKmsKey.Arn
     StreamsKmsRole: !GetAtt GroundStationKmsKeyRole.Arn
```

Putting it together

With the above resources, you now have the ability to schedule JPSS-1 contacts for synchronous data delivery from any of your onboarded AWS Ground Station AWS Ground Station Locations.

The following is a complete CloudFormation template that includes all resources described in this section combined into a single template that can be directly used in CloudFormation.

The CloudFormation template named `DirectBroadcastSatelliteWbDigIfEc2DataDelivery.yml` is designed to give you quick access to start receiving digitized intermediate frequency (DigIF) data for the Aqua, SNPP, JPSS-1/NOAA-20, and Terra satellites. It contains an Amazon EC2 instance and the required CloudFormation resources to receive raw DigIF direct broadcast data using AWS Ground Station Agent.

If Aqua, SNPP, JPSS-1/NOAA-20, and Terra are not onboarded to your account, see Onboard satellite.

> **Note**
>
> You can access the template by accessing the customer onboarding Amazon S3 bucket using valid AWS credentials. The links below use a regional Amazon S3 bucket. Change the `us-west-2` region code to represent the corresponding region of which you want to create the CloudFormation stack in.
> Additionally, the following instructions use YAML. However, the templates are available in both YAML and JSON format. To use JSON, replace the `.yml` file extension with `.json` when downloading the template.

To download the template using AWS CLI, use the following command:

```
aws s3 cp s3://groundstation-cloudformation-templates-us-west-2/agent/ec2_delivery/
DirectBroadcastSatelliteWbDigIfEc2DataDelivery.yml .
```

You can view and download the template in the console by navigating to the following URL in your browser:

```
https://s3.console.aws.amazon.com/s3/object/groundstation-cloudformation-templates-us-
west-2/agent/ec2_delivery/DirectBroadcastSatelliteWbDigIfEc2DataDelivery.yml
```

You can specify the template directly in CloudFormation using the following link:

```
https://groundstation-cloudformation-templates-us-west-2.s3.us-west-2.amazonaws.com/
agent/ec2_delivery/DirectBroadcastSatelliteWbDigIfEc2DataDelivery.yml
```

What additional resources does the template define?

The `DirectBroadcastSatelliteWbDigIfEc2DataDelivery` template includes the following additional resources:

- **Receiver Instance Elastic Network Interface** - (Conditional) An elastic network interface is created in the subnet specified by **PublicSubnetId** if provided. This is required if the receiver instance is in a private subnet. The elastic network interface will be associated with the EIP and attached to the receiver instance.
- **Receiver Instance Elastic IP** - An elastic IP that AWS Ground Station will connect to. This attaches to the receiver instance or elastic network interface.

- One of the following Elastic IP associations:
  - **Receiver Instance to Elastic IP Association** - The association of the Elastic IP to your receiver instance, if **PublicSubnetId** is not specified. This requires that **SubnetId** reference a public subnet.
  - **Receiver Instance Elastic Network Interface to Elastic IP Association** - The association of the elastic IP to the receiver instance elastic network interface, if **PublicSubnetId** is specified.
- (Optional) **CloudWatch Event Triggers** - AWS Lambda Function that is triggered using CloudWatch Events sent by AWS Ground Station before and after a contact. The AWS Lambda Function will start and optionally stop your Receiver Instance.
- (Optional) **Amazon EC2 Verification for Contacts** - The option to use Lambda to set up a verification system of your Amazon EC2 instance(s) for contacts with SNS notification. It is important to note that this may incur charges depending on your current usage.
- **Additional mission profiles** - Mission profiles for additional public broadcast satellites (Aqua, SNPP, and Terra).
- **Additional antenna-downlink configs** - Antenna downlink configs for additional public broadcast satellites (Aqua, SNPP, and Terra).

The values and parameters for the satellites in this template are already populated. These parameters make it easy for you to use AWS Ground Station immediately with these satellites. You do not need to configure your own values in order to use AWS Ground Station when using this template. However, you can customize the values to make the template work for your use case.

Where do I receive my data?

The dataflow endpoint group is set up to use the receiver instance network interface that part of the template creates. The receiver instance uses the AWS Ground Station Agent to receive the data stream from AWS Ground Station on the port defined by the dataflow endpoint. For more information about setting up a dataflow endpoint group, see AWS::GroundStation::DataflowEndpointGroup. For more information about the AWS Ground Station Agent, see What is the AWS Ground Station Agent?

Troubleshooting

The following documentation can help you troubleshoot issues that may occur while using AWS Ground Station.

Topics

Troubleshoot contacts that deliver data to Amazon EC2

If you are unable to successfully complete an AWS Ground Station contact, you'll need to verify that your Amazon EC2 instance is running, verify that your dataflow endpoint application is running, and verify that your dataflow endpoint application's stream is configured properly.

> **Note**
>
> DataDefender (DDX) is an example of a dataflow endpoint application currently supported by AWS Ground Station

Prerequisite

The following procedures assume that an Amazon EC2 instance is already set up. To set up an Amazon EC2 instance in AWS Ground Station, see Getting Started.

Step 1: Verify that your EC2 instance is running

The following procedure shows how to find your Amazon EC2 instance in the console and start it if it's not running.

1. Locate the Amazon EC2 instance that was used for the contact you are troubleshooting. Use the following steps:

   a. In your **CloudFormation** dashboard, select the stack that contains your Amazon EC2 instance.

   b. Choose the **Resources** tab and locate your Amazon EC2 instance in the **Logical ID** column. Verify that the instance is created in the **Status** column.

   c. In the **Physical ID** column, choose the link for your Amazon EC2 instance. This will take you to the Amazon EC2 management console.

2. In the Amazon EC2 management console, ensure that your Amazon EC2 **Instance State** is *running*.

3. If your instance is running, continue to the next step. If your instance is not running, start the instance by using the following step:

   - With your Amazon EC2 instance selected, choose **Actions > Instance State > Start**.

Step 2: Determine type of dataflow application used

If you are using the **AWS Ground Station Agent** for data delivery please redirect to section Troubleshooting AWS Ground Station Agent. Otherwise, if you are using the **DataDefender (DDX)** application continue to the section called “Step 3: Verify that dataflow application is running”.

Step 3: Verify that dataflow application is running

Verifying the status of DataDefender requires you to connect to your instance in Amazon EC2. For more details on connecting to your instance, see Connect to your Linux instance.

The following procedure provides troubleshooting steps using commands in an SSH client.

1. Open a terminal or command prompt and connect to your Amazon EC2 instance by using SSH. Forward port 80 of the remote host in order to view the DataDefender web UI. The following commands demonstrate how to use SSH to connect to an Amazon EC2 instance through a bastion with port forwarding enabled.

> **Note**
>
> You must replace <SSH KEY>, <BASTION HOST>, and <HOST> with your specific ssh key, bastion host name, and Amazon EC2 instance host name.

For Windows

```
ssh -L 8080:localhost:80 -o ProxyCommand="C:\Windows\System32\OpenSSH\ssh.exe -o
 \"ForwardAgent yes\" -W %h:%p -i \"<SSH KEY>\" ec2-user@<BASTION HOST>" -i "<SSH
 KEY>" ec2-user@<HOST>
```

For Mac

```
ssh -L 8080:localhost:80 -o ProxyCommand="ssh -A -o 'ForwardAgent yes' -W %h:%p -i
 <SSH KEY> ec2-user@<BASTION HOST>" -i <SSH  KEY> ec2-user@<HOST>
```

2. Verify that DataDefender (also called DDX) is running by grepping (checking) for a running process named ddx in the output. The command for grepping (checking) for a running process and a successful example output is provided below.

```
[ec2-user@Receiver-Instance ~]$ ps -ef | grep ddx
          Rtlogic   4977      1 10 Oct16 ?        2-00:22:14 /opt/rtlogic/ddx/
bin/ddx -m/opt/rtlogic/ddx/modules -p/opt/rtlogic/ddx/plugins -c/opt/rtlogic/
ddx/bin/ddx.xml -umask=077 -daemon -f installed=true -f security=true -f enable
 HttpsForwarding=true
          Ec2-user 18787 18657  0 16:51 pts/0      00:00:00 grep -color=auto ddx
```

   If DataDefender is running, skip to the section called “Step 4: Verify that your dataflow application stream is configured” Otherwise, continue to the next step.

3. Start DataDefender using the command show below.

```
sudo service rtlogic-ddx start
```

   If DataDefender is running after using the command, skip to the section called “Step 4: Verify that your dataflow application stream is configured” Otherwise, continue to the next step.

4. Inspect the following files using the commands below to see if there were any errors while installing and configuring DataDefender.

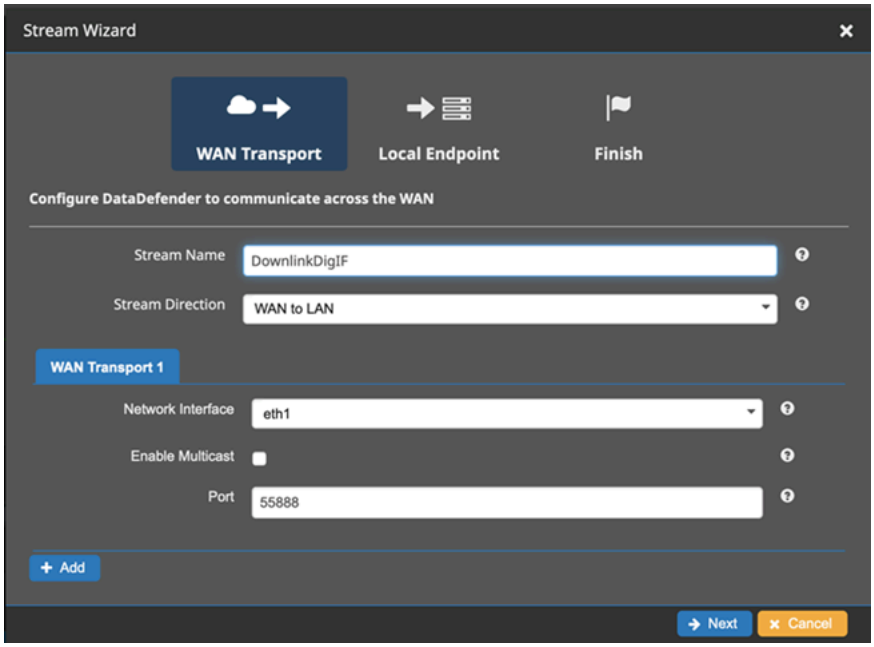
```
cat /var/log/user-data.log
          cat /opt/aws/groundstation/.startup.out
```

> **Note**
>
> A common issue discovered when inspecting these files is that the Amazon VPC that your Amazon EC2 instance is running in does not have access to Amazon S3 to download the installation files. If you discover in your logs that this is the issue, check your EC2 instance's Amazon VPC and security group settings to ensure they are not blocking access to Amazon S3.

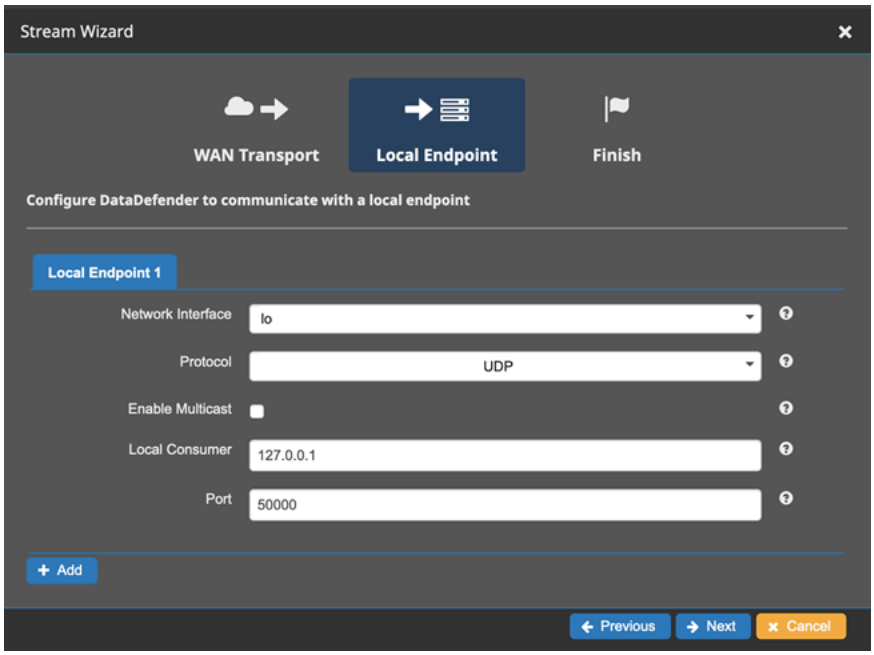
If DataDefender is running after checking your Amazon VPC settings, continue to the section called “Step 4: Verify that your dataflow application stream is configured”. If the problem persists, contact AWS Support and send your log files with a description of your issue.

Step 4: Verify that your dataflow application stream is configured

1. In a web browser, access your DataDefender web user interface by entering the following address in the address bar: *localhost:8080*. Then, press **Enter**.
2. On the **DataDefender** dashboard, choose **Go to Details**.
3. Select your stream from the list of streams, and choose **Edit Stream**.
4. In the **Stream Wizard** dialog box, do the following:
   a. In the **WAN Transport** pane, ensure **WAN to LAN** is selected for **Stream Direction**.
   b. In the **Port** box, ensure the WAN port you have chosen for your dataflow endpoint group is present. By default, this port is 55888. Then, choose **Next**.



c. In the **Local Endpoint** pane, ensure that a valid port is present in the *Port* box. By default, this port is 50000. This is the port on which you'll receive your data after DataDefender has received it from the AWS Ground Station service. Then, choose **Next**.



d. Choose **Finish** in the remaining menu if you have changed any values. Otherwise, you can cancel out of the **Stream Wizard** menu.

You have now ensured that your Amazon EC2 instance and DataDefender are both running and configured properly to receive data from AWS Ground Station. Continue to the section called “Step 5: Ensure you have enough available IP addresses in your receiver instance(s) subnet”.

Step 5: Ensure you have enough available IP addresses in your receiver instance(s) subnet

The following procedure shows how to find the number of available IP addresses in an Amazon EC2 reciever instance in the console.

1. For each Amazon EC2 receiver instance that was used for the contact you are troubleshooting. Use the following steps:

   a. In your **CloudFormation** dashboard, select the stack that contains your Amazon EC2 instance.

   b. Choose the **Resources** tab and locate your Amazon EC2 instance in the **Logical ID** column. Verify that the instance is created in the **Status** column.

   c. In the **Physical ID** column, choose the link for your Amazon EC2 instance. This will take you to the Amazon EC2 management console.
2. In the Amazon EC2 management console, find and click the **Subnet ID** link in your Amazon EC2 receiver instance's **Instance Summary**. This will take you to the corresponding Amazon VPC management console.
3. Select the matching subnet in the Amazon VPC management console and check the **Details** of your subnet for **Available IPv4 addresses**. If this number is not at least as many as dataflow endpoints that use this Amazon EC2 receiver instance do the following:

   a. Update your CloudFormation template's corresponding subnet **CidrBlock** to be sized correctly. For more details on subnet sizing see, Subnet CIDR blocks.

   b. Redeploy your stack with your updated CloudFormation template.

If you continue to experience issues, contact AWS Support.

Troubleshoot FAILED contacts

A contact will have a terminal contact status of **FAILED** when AWS Ground Station detects an issue with your resource configuration. The common use cases that can cause **FAILED** contacts are provided below, along with steps to help troubleshoot.

> **Note**
>
> This guide is specifically for the **FAILED** contact status - and is not intended for other failure statuses, such as **AWS_FAILED**, **AWS_CANCELLED**, or **FAILED_TO_SCHEDULE**. For more information on contact statuses, see the section called "AWS Ground Station contact statuses"

Dataflow endpoint FAILED use cases

The following is the list of common use cases that can result in a **FAILED** contact status for dataflow endpoint based dataflows:

- **Dataflow endpoint never connects** - The connection between AWS Ground Station Antenna and your Dataflow Endpoint Group for one or more dataflows was never established.
- **Dataflow endpoint connects late** - The connection between AWS Ground Station Antenna and your Dataflow Endpoint Group for one or more dataflows was established after the contact start time.
- **Dataflow endpoint's subnet is out of available IP addresses** - AWS Ground Station's data delivery solution is not able to create an ENI in your private network due to not having any available IP address in the reciever instance's subnet.
- **Dataflow endpoint's subnet is invalid** - AWS Ground Station's data delivery solution is not able to create an ENI in your private network due to inability to access the provided subnet specified in the Dataflow Endpoint Group.

For any dataflow endpoint failure cases, it is recommended to look into the following:

- Confirm the receiver Amazon EC2 instance was started successfully, prior to contact start time.
- Confirm the dataflow endpoint software was up and running during the contact.
- Ensure you have at least one available IP address per dataflow endpoint per receiver instance subnet.
- Ensure subnets associated to your Dataflow Endpoint Group, through dataflows configured in Set up and configure Amazon VPC, remain active and available to AWS Ground Station.

See the section on Troubleshoot contacts that deliver data to Amazon EC2 for more specific troubleshooting steps.

AWS Ground Station Agent FAILED use cases

The following is the list of common use cases that can result in a **FAILED** contact status for Agent-based dataflows:

- **AWS Ground Station Agent Never Reported Status** - The Agent responsible for orchestrating data delivery on your Dataflow Endpoint Group for one or more dataflows never successfully reported status to AWS Ground Station. This status update should happen within a few seconds of the contact end time.
- **AWS Ground Station Agent Started Late** - The Agent responsible for orchestrating data delivery on your Dataflow Endpoint Group for one or more dataflows was started late, after the contact start time.

For any AWS Ground Station Agent dataflow failure cases, it is recommended to look into the following:

- Confirm the receiver Amazon EC2 instance was started successfully, prior to contact start time.
- Confirm the Agent application was up and running at the start and during the contact.
- Confirm the Agent application and Amazon EC2 instance were not shut down within 15 seconds of contact end. This provides the Agent sufficient time to report status to AWS Ground Station.

See the section on Troubleshoot contacts that deliver data to Amazon EC2 for more specific troubleshooting steps.

Troubleshoot failed contact updates

When you call the UpdateContact API, AWS Ground Station performs synchronous validation on the request. If validation passes, the update is processed asynchronously to propagate changes to the antenna region. Synchronous validation errors are returned directly in the HTTP response. Asynchronous failures are reported through `failureCodes` and `failureMessage` fields on the contact version, which you can view by calling DescribeContactVersion for the version that failed to update.

For more information about contact versioning, see Update contacts and contact versioning.

Synchronous validation errors

The following errors are returned directly in the HTTP response when the UpdateContact request fails validation.

ResourceNotFoundException: Contact not found

Common cause

The specified `contactId` does not exist or belongs to a different AWS account.

Resolution

1. Verify that the `contactId` is correct.
2. Confirm that you are using credentials for the AWS account that owns the contact.
3. Use ListContacts to find the correct `contactId`.

ConflictException: Cannot update contact

Common cause

The contact is in a state that does not allow updates. The UpdateContact API can only be called when the contact is in the `SCHEDULED`, `PREPASS`, or `PASS` state. This error also occurs if another update is already in progress (the latest contact version is in the `UPDATING` state).

Resolution

1. Call DescribeContact to check the current contact status.
2. If the contact is in a terminal state (for example, `COMPLETED`, `FAILED`, or `CANCELLED`), it cannot be updated. A contact can only be updated when it is in the `SCHEDULED`, `PREPASS`, or `PASS` state. For a complete list of terminal states, see AWS Ground Station contact statuses.
3. If another update is in progress, wait for the current update to reach `ACTIVE` or `FAILED_TO_UPDATE` status before submitting another update. You can poll the DescribeContactVersion API, or use the `ContactUpdated` waiter convenience utilities provided by some AWS SDKs and the AWS Command Line Interface.

InvalidParameterException: Invalid request parameters

Common cause

The request contains invalid parameters. Common causes include:

- Missing or empty `clientToken`.
- Multiple types of `ProgramTrackSettings` (azimuth/elevation, OEM, and TLE) included in a single request. Only one type is allowed per request.
- Setting `satelliteArn` to null without being approved for azimuth elevation ephemeris at the contact's ground station.
- Missing `AzElProgramTrackSettings` when `satelliteArn` is null.
- Providing an `ephemerisId` that is not associated with the specified `satelliteArn`.
- The satellite does not have a valid visibility window from the ground station for the contact time range.
- The satellite is not onboarded to the ground station or does not have the licensing required by the mission profile.
- The mission profile includes Antenna Downlink Demod Decode Config configs, which are not supported for contact updates.

Resolution

1. Review the error message in the response for details about which parameter is invalid.
2. Ensure you provide exactly one type of `ProgramTrackSettings` per request.
3. If using azimuth/elevation pointing angles without a `satelliteArn`, confirm that your account is approved for this capability at the ground station. For more information, see Provide azimuth elevation ephemeris data.
4. Verify that the ephemeris you reference is associated with the correct satellite and covers the contact time range.

ResourceLimitExceededException: Maximum version limit reached

Common cause

The contact has reached the maximum number of versions (128). Each call to UpdateContact creates a new version, and a contact cannot exceed this limit.

Resolution

1. This limit cannot be increased. If you need to make further changes, cancel the contact and reserve a new one.

Asynchronous failure codes

The following failure codes appear in the `failureCodes` field of a contact version with a `FAILED_TO_UPDATE` status. Use DescribeContactVersion to retrieve these details. The `failureMessage` field provides additional context about the failure.

Failure code	Common cause	Resolution
`INTERNAL_ERROR`	An unexpected internal error occurred while processing the update.	Retry the update. If the issue persists, contact AWS Support.
`INVALID_SATELLITE_ARN`	The satellite ARN provided in the update request is not valid or does not exist.	Verify the satellite ARN and confirm that the satellite is registered in your account.
`INVALID_UPDATE_CONTACT_REQUEST`	The update request contains invalid parameters that were not caught during synchronous validation.	Review the `failureMessage` for details and correct the request parameters.
`EPHEMERIS_NOT_FOUND`	The ephemeris referenced in the tracking overrides does not exist.	Verify the `ephemerisId` and confirm that the ephemeris has not been deleted.
`EPHEMERIS_TIME_RANGE_INVALID`	The ephemeris does not cover the time range of the contact.	Upload a new ephemeris that covers the full contact time range. If the ephemeris time range cannot be extended, cancel the contact and reserve a new one during the time span of the ephemeris. For more information, see Provide custom ephemeris data.

Failure code	Common cause	Resolution
EPHEMERIS_NOT_ENABLED	The referenced ephemeris is not in an ENABLED state.	Check the ephemeris status and enable it before retrying the update.
SATELLITE_DOES_NOT_MATCH_EPHEMERIS	The ephemeris is not associated with the satellite specified in the update request.	Ensure the ephemerisId belongs to the satellite specified in satelliteArn .
NOT_ONBOARDED_TO_AZEL_EPHEMERIS	Your account is not approved to use azimuth elevation ephemeris data at the contact's ground station. Azimuth elevation ephemeris is a restricted feature available for a limited number of specialized use cases.	If azimuth elevation ephemeris is required for your use case, open an AWS Support ticket through the AWS Support Center Console to request access. Alternatively, consider using TLE ephemeris data or OEM ephemeris data if they fit your use case.
AZEL_EPHEMERIS_NOT_FOUND	The azimuth elevation ephemeris referenced in the request does not exist.	Verify the ephemerisId and confirm that the azimuth elevation ephemeris has not been deleted.
AZEL_EPHEMERIS_WRONG_GROUND_STATION	The azimuth elevation ephemeris was created for a different ground station than the one used by the contact.	Upload a new azimuth elevation ephemeris for the correct ground station, or use an existing ephemeris that matches the contact's ground station.
AZEL_EPHEMERIS_INVALID_STATUS	The azimuth elevation ephemeris is not in a valid state for use.	Check the ephemeris status. It must be in an ENABLED state. If the ephemeris failed validation, upload a corrected version.

Failure code	Common cause	Resolution
AZEL_EPHEMERIS_TIME_RANGE_INVALID	The azimuth elevation ephemeris does not cover the time range of the contact.	Upload a new azimuth elevation ephemeris that covers the full contact time range. If the ephemeris time range cannot be extended, cancel the contact and reserve a new one during the time span of the ephemeris.

Checking the status of an update

After calling UpdateContact, the new contact version starts in the UPDATING state. During this time, DescribeContact continues to return the previously active version of the contact. The new version does not appear in DescribeContact until it has been propagated to the antenna and reaches ACTIVE status. To check the status of a specific version, use DescribeContactVersion.

To determine whether an update succeeded or failed:

1. Call DescribeContactVersion with the contactId and versionId returned by the UpdateContact response.
2. Check the version.status field. A status of ACTIVE means the update was applied successfully. A status of FAILED_TO_UPDATE means the update failed.
3. If the status is FAILED_TO_UPDATE, check the version.failureCodes and version.failureMessage fields for details about what went wrong.

> **Tip**
>
> Some AWS SDKs and the AWS Command Line Interface support a ContactUpdated waiter that automatically polls DescribeContactVersion until the version reaches ACTIVE or FAILED_TO_UPDATE status. For example, the AWS Command Line Interface provides an aws groundstation wait contact-updated command. Use the waiter instead of implementing your own polling logic.

Troubleshoot FAILED_TO_SCHEDULE contacts

A contact will end in a **FAILED_TO_SCHEDULE** state when AWS Ground Station detects an issue either with your resource configuration or within the internal system. A contact that ends in a **FAILED_TO_SCHEDULE** state will optionally provide an `errorMessage` for additional context. For information about describing contacts, see the DescribeContact API.

The common use cases that can cause **FAILED_TO_SCHEDULE** contacts are provided below, along with steps to help troubleshoot.

> **Note**
>
> This guide is specifically for the **FAILED_TO_SCHEDULE** contact status - and is not intended for other failure statuses, such as **AWS_FAILED**, **AWS_CANCELLED**, or **FAILED**. For more information on contact statuses, see the section called "AWS Ground Station contact statuses"

The settings specified in your Antenna Downlink Demod Decode Config are not supported

The mission profile that was used to schedule this contact had an antenna-downlink-demod-decode config that was not valid.

Previously existing AntennaDownlinkDemodDecode config

- If your antenna-downlink-demod-decode configs have recently been changed - roll back to a previously working version before attempting to schedule.
- If this was an intentional change on an existing config, or a previously existing config that is no longer successfully scheduling - follow the next step on how to onboard a new AntennaDownlinkDemodDecode config.

Newly created AntennaDownlinkDemodDecode config

Contact AWS Ground Station directly to onboard your new config. Create a case with AWS Support including the `contactId` that ended in the **FAILED_TO_SCHEDULE** state

General Troubleshooting Steps

If the preceding troubleshooting steps did not resolve your issue:

- Re-attempt scheduling the contact or schedule another contact using the same mission profile. For information about how to reserve a contact, see ReserveContact.
- If you continue to receive a **FAILED_TO_SCHEDULE** status for this mission profile, contact AWS Support

Troubleshoot DataflowEndpointGroups not in a HEALTHY state

Listed below are the reasons your dataflow endpoint groups may not be in a `HEALTHY` state as well as the appropriate corrective action to take.

- `NO_REGISTERED_AGENT` - Start your EC2 instance, which will register the agent. Note that you must have a valid controller config file for this call to be successful. Refer to the Use AWS Ground Station Agent for details on configuring that file.
- `INVALID_IP_OWNERSHIP` - Use the DeleteDataflowEndpointGroup API to delete the Dataflow Endpoint Group, then use the CreateDataflowEndpointGroup API to recreate the Dataflow Endpoint Group using IP addresses and ports that are associated with the EC2 instance.
- `UNVERIFIED_IP_OWNERSHIP` - IP address has not been validated yet. Validation occurs periodically so this should resolve itself.
- `NOT_AUTHORIZED_TO_CREATE_SLR` - Account is not authorized to create the necessary Service-Linked Role. Check the troubleshooting steps in Use service-linked roles for Ground Station

Troubleshoot invalid ephemerides

When you upload ephemeris data to AWS Ground Station, it goes through an asynchronous validation workflow. If validation fails, the ephemeris status will change to `INVALID`. The error messaging in the DescribeEphemeris response provides detailed information to help you identify and resolve the issue.

Understanding ephemeris validation errors

When an ephemeris fails validation, the DescribeEphemeris API response includes two fields to help diagnose the problem:

errorCode

A machine-readable code identifying the specific validation error. This can be used for programmatic error handling.

errorMessage

A human-readable description of the validation error with specific details about what went wrong and guidance on how to fix it.

Example DescribeEphemeris response for an invalid ephemeris:

```
{
    "ephemerisId": "abc12345-6789-def0-1234-567890abcdef",
    "name": "My Invalid Ephemeris",
    "status": "INVALID",
    "creationTime": 1620254718.765,
    "invalidReason": "METADATA_INVALID",
    "errorCode": "OBJECT_NAME_MISSING",
    "errorMessage": "Metadata field missing: OBJECT_NAME",
    "suppliedData": {
        "tle": {
            "ephemerisData": "[...]"
        }
    }
}
```

Common validation errors for TLE ephemerides

The following are common validation errors encountered when uploading TLE ephemerides:

Mismatched satellite catalog number

Error: "The satellite catalog number present in the ephemeris does not match the associated satellite's satellite catalog number"

Solution: Verify that the NORAD ID/satellite catalog number in your TLE lines matches the satellite catalog number of your satellite. Use `00000` for satellites without an assigned catalog number.

Invalid mean motion

Error: "The mean motion of the provided ephemeris differs too greatly from the most recent reference ephemeris"

Solution: Verify that your TLE data is correct and represents a valid orbit. Ground Station uses Space-Track ephemerides as a reference during validation.

Common validation errors for OEM ephemerides

The following are common validation errors encountered when uploading OEM ephemerides:

Invalid reference frame

Error: "The REF_FRAME is not supported"

Solution: Update your OEM file to use one of the supported reference frames: EME2000 or ITRF2000.

Missing required fields

Error: "Metadata field missing: INTERPOLATION"

Solution: Add the INTERPOLATION and INTERPOLATION_DEGREE fields to your OEM metadata section. These are required for AWS Ground Station to generate accurate antenna pointing angles.

Unsupported time system

Error: "The TIME_SYSTEM is not supported"

Solution: Ensure your OEM file uses UTC as the time system.

Unsupported OEM version

Error: "The CCSDS_OEM_VERS is not supported"

Solution: Ensure your OEM file uses CCSDS OEM version 2.0.

Common validation errors for azimuth elevation ephemerides

The following are common validation errors encountered when uploading azimuth elevation ephemerides:

Missing azimuth/elevation data

Error: "No TimeAzEl fields were present in at least one AzElSegment"

Solution: Ensure each segment in your azimuth elevation data contains at least one time-tagged azimuth/elevation pair.

Invalid azimuth angle range (degrees)

Error: "AzEl az must be greater than or equal to -180 and less or equal to 360 degrees"

Solution: Verify that azimuth angles are within [-180, 360] degrees.

Invalid elevation angle range (degrees)

Error: "AzEl el must be greater than or equal to -90 and less than or equal to 90 degrees"

Solution: Verify that elevation angles are within [-90, 90] degrees.

Invalid azimuth angle range (radians)

Error: "AzEl az must be greater than or equal to -pi and less than or equal to 2pi radians"

Solution: Verify that azimuth angles are within [-π, 2π] radians.

Invalid elevation angle range (radians)

Error: "AzEl el must be greater than or equal to -pi/2 and less than or equal to pi/2 radians"

Solution: Verify that elevation angles are within [-π/2, π/2] radians.

Non-monotonic time values

Error: "The TimeAzEl items within a AzElSegment must be temporally in order"

Solution: Ensure that the time values in each segment are strictly increasing.

Segments out of order

Error: "AzElSegments must be temporally in order"

Solution: Ensure that segments are arranged in chronological order.

Overlapping segments

Error: "The time range of at least one segment overlaps with other segment time ranges"

Solution: Ensure that each segment has a unique, non-overlapping time range. The `endTime` of one segment should not exceed the `startTime` of the next segment.

Troubleshooting steps

If your ephemeris fails validation, follow these steps to resolve the issue:

1. Call DescribeEphemeris with your ephemeris ID to retrieve the `errorCode` and `errorMessage`.
2. Review the error message for specific details about what validation check failed.
3. Correct the identified issues in your ephemeris data.
4. Upload a new ephemeris with the corrected data using CreateEphemeris.
5. Monitor the new ephemeris status until it reaches `ENABLED` state.
6. Delete the invalid ephemeris using DeleteEphemeris if it's no longer needed.

Complete error code reference

The following sections provide a comprehensive mapping of all `errorCode` values that may be returned when ephemeris validation fails, organized by the high-level `invalidReason` category.

Invalid Reason: `METADATA_INVALID`

These errors occur when required metadata fields are missing, incorrectly formatted, or contain unsupported values in the ephemeris data.

Error Code	Error Message
MISMATCHED_SATCAT_ID	The satellite catalog number present in the TLE ephemeris does not match the associated satellite's satellite catalog number
OEM_VERSION_UNSUPPORTED	The `CCSDS_OEM_VERS` in the OEM ephemeris is not supported. Supported values: [`2.0`]
ORIGINATOR_MISSING	The `ORIGINATOR` header field is missing from the OEM ephemeris

Error Code	Error Message
CREATION_DATE_MISSING	The `CREATION_DATE` header field is missing from the OEM ephemeris
OBJECT_NAME_MISSING	The `OBJECT_NAME` metadata field is missing from the OEM ephemeris
OBJECT_ID_MISSING	The `OBJECT_ID` metadata field is missing from the OEM ephemeris
REF_FRAME_UNSUPPORTED	The `REF_FRAME` in the OEM ephemeris is not supported. Supported values: [`EME2000`, `ITRF2000`]
REF_FRAME_EPOCH_UN SUPPORTED	The `REF_FRAME_EPOCH` metadata field in the OEM ephemeris is not supported. Please remove this field from the ephemeris
TIME_SYSTEM_UNSUPPORTED	The `TIME_SYSTEM` in the OEM ephemeris is not supported. Supported values: [`UTC`]
CENTER_BODY_UNSUPPORTED	The `CENTER_BODY` in the OEM ephemeris is not supported. Supported values: [`Earth`]
INTERPOLATION_MISSING	The `INTERPOLATION` metadata field is missing from the OEM ephemeris
INTERPOLATION_DEGREE_INVALID	The interpolation degree in the OEM ephemeris must be larger than 0 for the interpolation method
AZ_EL_SEGMENT_LIST_MISSING	The `azElSegmentList` field is missing
INSUFFICIENT_TIME_AZ_EL	No `TimeAzEl` fields were present in at least one `azElSegmentList`

Invalid Reason: `TIME_RANGE_INVALID`

These errors occur when the ephemeris contains invalid time ranges, including issues with start/end times, segment ordering, overlapping segments, or temporal inconsistencies.

Error Code	Error Message
START_TIME_IN_FUTURE	Ephemeris start time is in the future, but must be in the past
END_TIME_IN_PAST	Ephemeris end time is in the past, but must be in the future
EXPIRATION_TIME_TOO_EARLY	The provided expiration time is earlier than the ephemeris end time
START_TIME_METADATA_TOO_EARLY	The `START_TIME` metadata value is earlier than the earliest time present in the OEM ephemeris data
STOP_TIME_METADATA_TOO_LATE	The `STOP_TIME` metadata value is later than the latest time present in the OEM ephemeris data
AZ_EL_SEGMENT_END_TIME_BEFORE_START_TIME	The `endTime` of at least one data segment is before the segment's `startTime`
AZ_EL_SEGMENT_TIMES_OVERLAP	The time range of at least one segment overlaps with other segment time ranges
AZ_EL_SEGMENTS_OUT_OF_ORDER	The segments are not temporally ordered
TIME_AZ_EL_ITEMS_OUT_OF_ORDER	The `TimeAzEl` items within a `AzElSegment` must be temporally in order
AZ_EL_SEGMENT_REFERENCE_EPOCH_INVALID	The reference epoch for a segment is invalid or incorrectly formatted
AZ_EL_SEGMENT_START_TIME_INVALID	The start time in a segment's valid time range does not start after the first segment
AZ_EL_SEGMENT_END_TIME_INVALID	The end time in a segment's valid time range does not end after the last segment
AZ_EL_SEGMENT_VALID_TIME_RANGE_INVALID	The valid time range for a segment is invalid

Error Code	Error Message
AZ_EL_SEGMENT_END_TIME_TOO_LATE	The end time of a segment exceeds the maximum allowed duration from the reference epoch
AZ_EL_TOTAL_DURATION_EXCEEDED	The total duration across all segments exceeds the maximum allowed pointing angle duration

Invalid Reason: `TRAJECTORY_INVALID`

These errors occur when the ephemeris contains invalid trajectory data, including issues with orbital parameters, angle ranges, or units.

Error Code	Error Message
MEAN_MOTION_INVALID	The mean motion of the provided TLE ephemeris differs too greatly from the most recent reference ephemeris. Note: Ground Station uses Space-Track ephemerides as a reference during validation
TIME_AZ_EL_AZ_RADIAN_RANGE_INVALID	AzEl `az` must be greater than or equal to -π and less than or equal to 2π radians
TIME_AZ_EL_EL_RADIAN_RANGE_INVALID	AzEl `el` must be greater than or equal to -π/2 and less than or equal to π/2 radians
TIME_AZ_EL_AZ_DEGREE_RANGE_INVALID	AzEl `az` must be greater than or equal to -180 and less or equal to 360 degrees
TIME_AZ_EL_EL_DEGREE_RANGE_INVALID	AzEl `el` must be greater than or equal to -90 degrees and less than or equal to 90 degrees
TIME_AZ_EL_ANGLE_UNITS_INVALID	Invalid AzEl angle units

Invalid Reason: `KMS_KEY_INVALID`

These errors occur when there are issues with the AWS Key Management Service (KMS) key used to encrypt the ephemeris data.

Error Code	Error Message
INSUFFICIENT_KMS_PERMISSIONS	Ground Station does not have sufficient permissions to access this ephemeris' KMS key

Invalid Reason: `VALIDATION_ERROR`

These errors occur when there are general validation issues with the ephemeris data that don't fall into the other specific categories.

Error Code	Error Message
INTERNAL_ERROR	Internal error occurred during ephemeris validation
FILE_FORMAT_INVALID	The ephemeris file format is invalid or corrupted. Verify the file conforms to the expected format for the ephemeris type

Troubleshoot contacts that received no data

It is possible for a contact to appear successful, but still did not receive any data. This may mean that you receive PCAP files that are empty, or no PCAP files at all if you are using S3 data delivery. This can happen for a number of reasons. The following discusses some of the causes, and how to address them.

Incorrect downlink config

Each contact that receives data from a satellite will have an associated Antenna Downlink Config or Antenna Downlink Demod Decode Config. If the configuration specified does not agree with the signal being transmitted by a satellite, AWS Ground Station will not be able to receive the transmitted signal. This will result in no data being received by AWS Ground Station.

To fix this, please verify that the configs you are using agree with the signal being transmitted by your satellite. For example, verify that you've set the correct center frequency, bandwidth, polarization, and if needed, demodulation and decoding parameters.

Satellite maneuver

There are times that a satellite may perform a maneuver which temporarily disables some of its communication systems. The maneuver may also significantly change the location of the satellite in the sky. AWS Ground Station will not be able to receive a signal from a satellite that is not transmitting a signal, or if the ephemeris being used causes the AWS Ground Station antenna to point at a location in the sky where the satellite is not present.

If you are trying to communicate with a public broadcast satellite operated by NOAA, you may be able to find a message describing an outage or maneuver on the NOAA Satellite Alert Messages page. The message may include a timeline of when data transmission is expected to resume, or this may be posted in a subsequent message.

If you are communicating with your own satellites, it's your responsibility to understand your satellite operations, and how this might impact communicating with AWS Ground Station. If you are performing a maneuver that will impact the satellite trajectory, this may include providing updated custom ephemeris data. For more information on providing custom ephemeris data, see Understand how AWS Ground Station uses ephemerides.

AWS Ground Station outage

If AWS Ground Station causes a contact to fail, or cancels it, AWS Ground Station will set the contact status to *AWS_FAILED*, or *AWS_CANCELLED*. For more information on contact lifecycle, see Understand contact lifecycle. In some cases, AWS Ground Station may have a failure that prevents data from being delivered to your account, but doesn't result in the contact being in an *AWS_FAILED* or *AWS_CANCELLED* status. When this happens, AWS Ground Station should post an account-specific event to your AWS Health dashboard. For more information about the AWS Health dashboard, see AWS Health User Guide .

Troubleshoot telemetry

Use the following information to troubleshoot common issues with telemetry.

Common setup issues

IAM permission errors

Symptoms

When calling `CreateConfig` to create a TelemetrySinkConfig, you receive an error:

```
Unable to write to Kinesis Data Streams stream. Ensure that Ground Station has
kinesis:PutRecord permissions for the given stream
```

Causes

- The IAM role specified in the TelemetrySinkConfig doesn't have the required permissions to write to the Kinesis Data Streams stream.
- The trust policy on the IAM role doesn't allow AWS Ground Station to assume the role.
- The Kinesis Data Streams stream ARN in the TelemetrySinkConfig is incorrect or the stream doesn't exist.

Solutions

1. Verify the IAM role exists and has the correct permissions. Review Step 2: Create a TelemetrySinkConfig and ensure all steps were followed.
2. Check that AWS Ground Station can assume your IAM role:

   ```
   aws iam get-role --role-name GroundStationTelemetryRole
   ```

   Verify the trust policy includes `groundstation.amazonaws.com` as a trusted service principal.
3. Verify the IAM role has the required Kinesis permissions:

   ```
   aws iam list-attached-role-policies --role-name GroundStationTelemetryRole
   ```

   Ensure the policy includes `kinesis:DescribeStream`, `kinesis:PutRecord`, and `kinesis:PutRecords` permissions for your stream.
4. Verify the Kinesis Data Streams stream exists and the ARN is correct:

```
aws kinesis describe-stream \
    --stream-name your-stream-name \
    --region us-east-2
```

5. If using customer-managed encryption, verify the IAM role has `kms:GenerateDataKey` permission for your AWS KMS key.

PassRole permission errors

Symptoms

When calling `CreateConfig`, you receive an error about not having permission to pass the IAM role.

Solution

Ensure your IAM user or role has `iam:PassRole` permission for the telemetry IAM role. Add the following policy to your user or role:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:GetRole",
        "iam:PassRole"
      ],
      "Resource": "arn:aws:iam::99999999999:role/your-stream-name"
    }
  ]
}
```

Kinesis Data Streams stream configuration issues

Symptoms

Telemetry delivery fails or is intermittent.

Causes

- The Kinesis Data Streams stream has insufficient capacity for the telemetry throughput.
- The stream is being used by other applications, causing write throttling.

Solutions

1. Check the stream status:

```
aws kinesis describe-stream \
    --stream-name your-stream-name \
    --region us-east-2
```

2. Monitor for write throttling using CloudWatch metrics:

```
aws cloudwatch get-metric-statistics \
    --namespace AWS/Kinesis \
    --metric-name WriteProvisionedThroughputExceeded \
    --dimensions Name=StreamName,Value=your-stream-name \
    --start-time 2025-12-08T00:00:00Z \
    --end-time 2025-12-08T23:59:59Z \
    --period 60 \
    --statistics Sum \
    --region us-east-2
```

3. If throttling is detected, consider:

   - Switching to on-demand capacity mode for automatic scaling.
   - Using a dedicated stream for AWS Ground Station telemetry.
   - If using provisioned mode, increasing the number of shards.

Telemetry delivery problems

No telemetry data appearing

Symptoms

After scheduling a contact with a telemetry-enabled mission profile, no telemetry data appears in your Kinesis Data Streams stream.

Possible causes and solutions

Mission profile doesn't have telemetry enabled

Verify the mission profile used for the contact includes a `telemetrySinkConfigArn`:

```
aws groundstation get-mission-profile \
    --mission-profile-id 12345678-1234-1234-1234-123456789012 \
    --region us-east-2
```

Check the output for the `telemetrySinkConfigArn` field. If it's not present, the mission profile doesn't have telemetry enabled.

IAM role permissions issue

Review the IAM permission troubleshooting steps in IAM permission errors.

Kinesis Data Streams stream doesn't exist or is in wrong region

Verify the stream exists in the correct region:

```
aws kinesis describe-stream \
    --stream-name your-stream-name \
    --region us-east-2
```

Contact hasn't started yet

Telemetry delivery begins at contact start time. Verify the contact has started by checking the contact status:

```
aws groundstation describe-contact \
    --contact-id 12345678-1234-1234-1234-123456789012 \
    --region us-east-2
```

Intermittent telemetry data

Symptoms

Telemetry data is delivered inconsistently with gaps or missing records.

Possible causes

- Kinesis Data Streams stream capacity issues or throttling. See Kinesis Data Streams stream configuration issues.

- Network connectivity issues between AWS Ground Station and your Kinesis Data Streams stream.

Solutions

- Monitor Kinesis Data Streams stream metrics in CloudWatch for throttling or errors.
- Ensure your stream is using on-demand capacity mode or has sufficient provisioned capacity.
- Use a dedicated stream for AWS Ground Station telemetry to avoid contention with other applications.

Data format issues

JSON parsing errors

Symptoms

Your application encounters errors when parsing telemetry records as JSON.

Solutions

- **Verify Base64 decoding** - Data in Kinesis Data Streams stream is Base64-encoded. Ensure you decode the data before parsing it as JSON. For more information, see Reading data from Kinesis Data Streams stream.
- **Check for empty records** - AWS Ground Station may send empty validation records when creating a *TelemetrySinkConfig*. Your application should handle empty or malformed records gracefully.
- **Implement version-aware parsing** - Parse the `telemetryTypeAndVersion`, `telemetryType`, and `telemetryVersion` fields first to determine the appropriate schema for each record.

Unknown telemetry types or versions

Symptoms

Your application encounters telemetry types or versions it doesn't recognize.

Solution

This is expected behavior as new telemetry types and schema versions may be introduced over time. Your application should:

- Log unknown types and versions for monitoring.
- Continue processing known types and versions.
- Implement graceful handling for unknown schemas.

For more information about schema versioning, see Schema versioning and evolution.

Getting help

If you continue to experience issues after following the troubleshooting steps, contact AWS Support.

Information to provide

When contacting support, provide the following information:

- Contact IDs experiencing issues
- Mission profile ID used
- TelemetrySinkConfig ARN
- Kinesis Data Streams stream ARN
- IAM role ARN and attached policies
- Error messages from CloudWatch Logs or your application
- Timestamps when issues occurred
- Troubleshooting steps already taken

For general AWS Ground Station support, see the AWS Ground Station User Guide.

Quotas and limits

You can view the supported regions, their associated endpoints, and quotas at AWS Ground Station endpoints and quotas.

You can use the Service Quotas console, the AWS API and the AWS CLI to request quota increases, when needed.

Service terms

For the AWS Ground Station service terms, please refer to AWS Service Terms.

Document History for the AWS Ground Station User Guide

The following table describes the important changes in each release of the AWS Ground Station User Guide.

Change	Description	Date
New Feature	Added documentation for AWS Ground Station Dedicated Antennas. For more information, see AWS Ground Station Dedicated Antennas.	April 14, 2026
New Feature	Added documentation for the UpdateContact API and contact versioning. For more information, see Update contacts and contact versioning.	April 14, 2026
New Feature	Added documentation for the ListAntennas and ListGroundStationReservations APIs. For more information, see AWS Ground Station Locations and View ground station reservations.	April 14, 2026
Documentation Update	Added additional functionality to the CancelContact API, and includes information on said functionality and metering implications. For more information see Understand contact metering.	December 10, 2025

Documentation Update	Clarified that CloudWatch metrics are emitted in the region associated with the contact's ground station. Fixed broken links.	December 2, 2025
Updated AWS managed policy	AWS Ground Station has updated the managed policy `AWSGroundStationAgentInstancePolicy` to include additional permissions for retrieving task response URLs. For information, see AWS Ground Station updates to AWS managed policies.	November 13, 2025
New Feature	Updated the user guide to include azimuth elevation ephemerides. For more information, see Provide azimuth elevation ephemeris data	October 22, 2025
Documentation Update	Cross-region data delivery no longer requires special configuration or approvals. For more information, see Use cross-region data delivery.	September 11, 2025
Documentation Update	Added clarification on contact utilization of configured resources.	April 4, 2025
New Feature	Updated the user guide to include AWS Ground Station digital twin.	August 6, 2024

Documentation Update	Updated many sections of the user guide, including new diagrams, examples, and more.	July 18, 2024
Documentation Update	Added RSS feed to User Guide.	July 18, 2024
Documentation Update	Split AWS Ground Station Agent User Guide into a separate User Guide.	July 18, 2024
New Feature	Contacts can now be scheduled up to 30 seconds outside visibility time ranges. Visibility times are included in DescribeContact responses.	March 26, 2024
Documentation Update	Improved organization and added "EC2 Instance Selection and CPU Planning" section.	March 6, 2024
Documentation Update	Added new best practice to AWS Ground Station Agent User Guide for running services and processes alongside the AWS Ground Station Agent.	February 23, 2024
Documentation Update	Added Agent Release Notes page.	February 21, 2024
Template Update	Added support for separate public subnet in the DirectBroadcastSatelliteWbDigIfEc2DataDelivery template.	February 14, 2024

Documentation Update	Added referral to AWS User Notifications in monitoring documentation.	August 6, 2023
Documentation Update	Added instructions for tagging satellites with a name to be shown in the AWS Ground Station console.	July 26, 2023
New Feature	Added the AWS Ground Station Agent User Guide for the release of Wideband DigIF Data Delivery.	April 12, 2023
New AWS managed policy	AWS Ground Station added a new policy named AWSGroundStationAgentInstancePolicy.	April 12, 2023
New Feature	Updated the user guide for release of CPE Preview.	November 9, 2022
New AWS managed policy	AWS Ground Station added the AWSServiceRoleForGroundStationDataflowEndpointGroup service-linked-role (SLR) that includes a new policy named AWSServiceRoleForGroundStationDataflowEndpointGroupPolicy.	November 2, 2022
New Feature	Updated the user guide to include integration with AWS CLI.	April 17, 2020

New Feature	Updated the user guide to include integration with CloudWatch Metrics.	February 24, 2020
New Template	Public Broadcast Satellites (AquaSnppJpss Template) added to the *AWS Ground Station User Guide*.	February 19, 2020
New Feature	Updated the user guide to include cross-region data delivery.	February 5, 2020
Documentation Update	Updated examples and descriptions for monitorin g AWS Ground Station with CloudWatch Events.	February 4, 2020
Documentation Update	Template locations have been updated and the Getting Started and Troubleshooting sections have been revised.	December 19, 2019
New Troubleshooting Section	Troubleshooting section added to the *AWS Ground Station User Guide*.	November 7, 2019
New Getting Started Topic	Updated the Getting Started topic, which includes the most current CloudFormation templates.	July 1, 2019
Kindle Version	Published Kindle version of the *AWS Ground Station User Guide*.	June 20, 2019

New service and guide	This is the initial release of AWS Ground Station and the *AWS Ground Station User Guide*.	May 23, 2019

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.