



Documentation

Getting Started

What is Segment?
[How Segment Works](#)
Getting Started Guide
A Basic Segment Installation
Planning a Full Installation
A Full Segment Installation
Sending Data to Destinations
Testing and Debugging
What's Next
Use Cases

Guides

Connections

Unify

Engage

Privacy

Protocols

Segment App

API

Partners

Glossary

Config API

Help

<input type="radio"/> Web	<input checked="" type="checkbox"/> Web
<input type="radio"/> Mobile	<input checked="" type="checkbox"/> Mobile
<input type="radio"/> Server	<input checked="" type="checkbox"/> Server

Segment makes it easy to send your data to Amazon Personalize (and lots of other destinations). Once you collect your data using Segment's [open source libraries](#), Segment translates and routes your data to Amazon Personalize in the format it can use. [Amazon Personalize](#) is a machine learning service that makes it easy for developers to create individualized recommendations for customers using their applications. AWS Personalize enables:

- Media companies to provide recommended content for viewers based on their viewing history
- Retailers to provide personalized product recommendations based on shopping behavior
- Any company to provide personalized search results and targeted marketing promotions based on the latest machine-learning capabilities developed at Amazon

Developing the machine-learning capabilities necessary to produce these recommendation systems has been beyond the reach of most organizations today due to the complexity of developing machine learning functionality. Amazon Personalize allows developers with no prior machine learning experience to build sophisticated personalization capabilities into their applications, using machine learning technology perfected from years of use on Amazon.com.

Getting Started

These are the pre-requisites you need before getting started:

- 1 Segment data flowing into an S3 destination, a Snowflake warehouse, or Amazon Redshift warehouse.
- 2 You have the ability to create AWS Glue jobs (only required if using S3 to [train your model](#))
- 3 You have the ability to deploy Lambda functions in Amazon Web Services
- 4 You have access to AWS Personalize

If you don't have S3, Redshift warehouse, or Snowflake warehouse configured, you can read more about setting up [S3](#), [Redshift](#), and [Snowflake](#).

If you're a Segment business tier customer, contact your Success contact to initiate a replay to S3 or your Warehouse.

There are three main parts to using Amazon Personalize with Segment:

- 1 [Train your model](#) on historical data in S3 or a Warehouse.
- 2 [Create a Personalize Dataset Group](#) and Campaign
- 3 [Connect Recommendations](#) and Live Event Updates to your Campaign and Segment

Train Your Model

S3 Bucket Permissions

Whatever method you choose to train your model will result in placing a CSV into an S3 bucket. Be sure to update the policies of the bucket to include [these permissions](#) to allow Personalize to access your CSV:

```
{
  "Version": "2012-10-17",
  "Id": "PersonalizeS3BucketAccessPolicy",
  "Statement": [
    {
      "Sid": "PersonalizeS3BucketAccessPolicy",
      "Effect": "Allow",
      "Principal": {
        "Service": "personalize.amazonaws.com"
      },
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::bucket-name",
        "arn:aws:s3:::bucket-name/*"
      ]
    }
  ]
}
```

Define a Schema

To train a Personalize model, you'll need to define the event schema for the event names and properties that your model uses as features. For the examples below, Segment is using the following Personalize Dataset schema to train the model. You'll want to modify this to suit your use cases. You can learn more about [Personalize schemas](#).

```
{
  "type": "record",
  "name": "Interactions",
  "namespace": "com.amazonaws.personalize.schema",
  "fields": [
    {
      "name": "USER_ID",
      "type": "string"
    },
    {
      "name": "ITEM_ID",
      "type": "string"
    },
    {
      "name": "EVENT_TYPE",
      "type": "string"
    },
    {
      "name": "TIMESTAMP",
      "type": "long"
    }
  ],
  "version": "1.0"
}
```

The examples show how multiple Segment track events are mapped into this schema and used to train a Personalize solution.

From Redshift

If you already use Redshift, training your model on the data in your warehouse is the simplest way to get up and running.

Unload Data from Redshift to S3

Assuming you have a Personalize schema like that described [below](#), you can use the following query to pull out all the data from your Order Completed, Product Added, and Product Viewed events.

```
unload ('
  select
    user_id as USER_ID,
    products_sku as ITEM_ID,
    event as EVENT_TYPE,
    date_part(epoch,"timestamp") as TIMESTAMP
  from prod.order_completed
  UNION
  select
    user_id as USER_ID,
    products_sku as ITEM_ID,
    event as EVENT_TYPE,
    date_part(epoch,"timestamp") as TIMESTAMP
  from prod.product_added
  UNION
  select
    user_id as USER_ID,
    products_sku as ITEM_ID,
    event as EVENT_TYPE,
    date_part(epoch,"timestamp") as TIMESTAMP
  from prod.product_viewed
')
to
's3://mybucket/my_folder'
credentials 'aws_access_key_id=AWS_ACCESS_KEY_ID;aws_secret_access_key=AWS_SECRET_ACCESS_KEY;token=AWS_SESSION_TOKEN'
HEADER
REGION AS '<your-region>'
DELIMITER AS ','
PARALLEL OFF;
```

Note: Use `date_part(epoch,"timestamp") as TIMESTAMP` because Personalize requires timestamps to be specified in UNIX/epoch time.

Verify the Output file Browse to the S3 service page in the AWS console and navigate to the bucket path specified in the `unload` command. You should see the output file.

From Snowflake

There are a few ways to load a CSV into S3 from your [Snowflake](#) warehouse. This example shows loading the data directly into an S3 bucket.

Assuming you have a Personalize schema like that described [below](#), you can use the following query to pull out all the data from your Order Completed, Product Added, and Product Viewed events.

Unload Data from Snowflake to S3

```
copy into
s3://mybucket/my_folder/my_file.csv
from
(
  select
    user_id as USER_ID,
    products_sku as ITEM_ID,
    event as EVENT_TYPE,
    date_part(epoch,"timestamp") as TIMESTAMP
  from prod.order_completed
  UNION
  select
    user_id as USER_ID,
    products_sku as ITEM_ID,
    event as EVENT_TYPE,
    date_part(epoch,"timestamp") as TIMESTAMP
  from prod.product_added
  UNION
  select
    user_id as USER_ID,
    products_sku as ITEM_ID,
    event as EVENT_TYPE,
    date_part(epoch,"timestamp") as TIMESTAMP
  from prod.product_viewed
)
file_format=(type=csv)
single = true -- Personalize requires a single CSV file
credentials = (aws_key_id='xxxx' aws_secret_key='xxxxx' aws_token='xxxxxx');
```

This example uses temporary S3 credentials, which are generated by AWS STS and expire after a specific period of time. Temporary credentials are [recommended](#) to protect access to the bucket.

Verify the Output file Go to the S3 service page in the AWS console and navigate to the bucket path specified in the `unload` command. You should see the output file.

From S3

Historical Data Preparation

Segment's S3 destination contains a copy of all of the source data you configured to go to S3. In your S3 bucket there's a folder called `/segment-logs`. Under this folder is another folder for each source of data you connected to your Segment S3 destination.

Note that this step is not required unless you plan to do batch data extraction from S3.

Your Glue ETL job will need to crawl each source folder to extract the backup data that forms your training set. Analysis of this data set is beyond the scope of this document. It is strongly recommended you familiarize yourself with the types of events that can be sent through Segment. Segment's event structure is described in detail on Segment's [HTTP source](#) documentation.

The following examples show how to configure an AWS Glue job to convert Segment historical data into the Apache Avro format that Personalize wants to consume for training data sets.

Create AWS Glue ETL Job

To create an AWS Glue ETL Job:

- 1. Create a new AWS service IAM role using the following execution policies. These policies give your Glue job the ability to write to your S3 bucket:

Policy 1:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:*",
        "s3:GetBucketLocation",
        "s3:ListBucket",
        "s3:ListAllMyBuckets",
        "s3:GetBucketAcl",
        "ec2:DescribeVpcEndpoints",
        "ec2:DescribeRouteTables",
        "ec2:CreateNetworkInterface",
        "ec2:DeleteNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcAttribute",
        "iam:ListRolePolicies",
        "iam:GetRole",
        "iam:GetRolePolicy",
        "cloudwatch:PutMetricData"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:CreateBucket"
      ],
      "Resource": [
        "arn:aws:s3:::aws-glue-*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::aws-glue-*/**",
        "arn:aws:s3:::*/*aws-glue-*/**"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::crawler-public*",
        "arn:aws:s3:::aws-glue-*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:*:*/aws-glue/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateTags",
        "ec2:DeleteTags"
      ],
      "Condition": {
        "ForAllValues:StringEquals": {
          "aws:TagKeys": [

```

```

        "aws:tagkeys": [
            "aws-glue-service-resource"
        ]
    },
    "Resource": [
        "arn:aws:ec2:*:*:network-interface/*",
        "arn:aws:ec2:*:*:security-group/*",
        "arn:aws:ec2:*:*:instance/*"
    ]
}
]
}

```

Policy 2:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::{ your bucket arn }/segment-logs/*",
        "arn:aws:s3:::{ your bucket arn }/transformed/*"
      ]
    }
  ]
}

```

2. Navigate to the Glue service in your AWS console.

3. Click **Get started** and then select **Jobs** from the left navigation on the Glue console page.

4. Select **Spark script editor** and click **Create**.

5. The following code sample is the source code for a generic Glue job. Copy the code example to your clipboard and paste it into the Glue editor window, modifying as necessary to reflect the names of the events you wish to extract from the Segment logs (see line #25).

```

import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from awsglue.context import GlueContext
from awsglue.dynamicframe import DynamicFrame
from awsglue.job import Job
from pyspark.context import SparkContext
from pyspark.sql.functions import unix_timestamp

## @params: [JOB_NAME, S3_JSON_INPUT_PATH, S3_CSV_OUTPUT_PATH]
args = getResolvedOptions(sys.argv, ['JOB_NAME', 'S3_JSON_INPUT_PATH', 'S3_CSV_OUTPUT_PATH'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

# Load JSON into dynamic frame
datasource0 = glueContext.create_dynamic_frame.from_options('s3', {'paths': [args['S3_JSON_INPUT_PATH']], 'recursive': True}, 'json')
print("Input file: ", args['S3_JSON_INPUT_PATH'])
print("Input file total record count: ", datasource0.count())

# Filters the JSON documents that we want included in the output CSV
supported_events = ['Product Added', 'Order Completed', 'Product Clicked']
def filter_function(dynamicRecord):
    if ('anonymousId' in dynamicRecord and
        'userId' in dynamicRecord and
        'properties' in dynamicRecord and
        'sku' in dynamicRecord["properties"] and
        'event' in dynamicRecord and
        dynamicRecord['event'] in supported_events):
        return True
    else:
        return False

# Apply filter function to dynamic frame
interactions = Filter.apply(frame = datasource0, f = filter_function, transformation_ctx = "interactions")
print("Filtered record count: ", interactions.count())

# Map only the fields we want in the output CSV, changing names to match target schema.
applymapping1 = ApplyMapping.apply(frame = interactions, mappings = [ \
    ("anonymousId", "string", "ANONYMOUS_ID", "string"), \
    ("userId", "string", "USER_ID", "string"), \
    ("properties.sku", "string", "ITEM_ID", "string"), \
    ("event", "string", "EVENT_TYPE", "string"), \
    ("timestamp", "string", "TIMESTAMP_ISO", "string")], \
    transformation_ctx = "applymapping1")

# Repartition to a single file since that is what is required by Personalize
onepartitionDF = applymapping1.toDF().repartition(1)
# Coalesce timestamp into unix timestamp
onepartitionDF = onepartitionDF.withColumn("TIMESTAMP", \
    unix_timestamp(onepartitionDF['TIMESTAMP_ISO'], "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'"))
# Convert back to dynamic frame
onepartition = DynamicFrame.fromDF(onepartitionDF, glueContext, "onepartition_df")

# Write output back to S3 as a CSV
glueContext.write_dynamic_frame.from_options(frame = onepartition, connection_type = "s3", \
    connection_options = {"path": args['S3_CSV_OUTPUT_PATH']}, \
    format = "csv", transformation_ctx = "datasink2")

job.commit()

```

6 Select the **Job details** tab.

7 Enter a name for your Glue job.

8 Leave Type as **Spark**.

9 Make any optional changes on the Job details page, and click **Save** to save the job script.

To review key parts of the Python script in more detail:

The script is initialized with a few job parameters. You'll see how to specify these parameter values when the job below runs. For now, see that Segment is passing in the location of the raw JSON files using `S3_JSON_INPUT_PATH` and the location where the output CSV should be written through `S3_CSV_OUTPUT_PATH`.

```
args = getResolvedOptions(sys.argv, ['JOB_NAME', 'S3_JSON_INPUT_PATH', 'S3_CSV_OUTPUT_PATH'])
```

2 The Spark and Glue contexts are created and associated. A Glue Job is also created and initialized.

```
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)
```

3 The first step in Segment's Job is to load the raw JSON file as a Glue DynamicFrame.

```
datasource0 = glueContext.create_dynamic_frame.from_options('s3', {'paths': [args['S3_JSON_INPUT_PATH']], 'rec
urse': True}, 'json')
```

4 Since not all events that are written to S3 by Segment are relevant to training a Personalize model, Segment uses Glue's Filter transformation to keep the records needed.

5 The datasource0 DynamicFrame created above is passed to Filter.apply(...) function along with the filter_function function. It's in filter_function where Segment keeps events that have a product SKU and userId specified. The resulting DynamicFrame is captured as interactions.

```
def filter_function(dynamicRecord):
    if dynamicRecord["properties"]["sku"] and dynamicRecord["userId"]:
        return True
    else:
        return False

interactions = Filter.apply(frame = datasource0, f = filter_function, transformation_ctx = "interactions")
```

6 Segment calls Glue's ApplyMapping transformation, passing the interactions DynamicFrame from above and field mapping specification that indicates the fields Segment wants to retain and their new names. These mapped field names become the column names in Segment's output CSV. You'll notice that Segment is using the product SKU as the ITEM_ID and event as the EVENT_TYPE. Segment also renames the timestamp field to TIMESTAMP_ISO since the format of this field value in the JSON file is an ISO 8601 date and Personalize requires timestamps to be specified in UNIX time (number seconds since Epoch).

```
apmapping1 = ApplyMapping.apply(frame = interactions, mappings = [ \
    ("anonymousId", "string", "ANONYMOUS_ID", "string"), \
    ("userId", "string", "USER_ID", "string"), \
    ("properties.sku", "string", "ITEM_ID", "string"), \
    ("event", "string", "EVENT_TYPE", "string"), \
    ("timestamp", "string", "TIMESTAMP_ISO", "string")], \
    transformation_ctx = "apmapping1")
```

7 To convert the ISO 8601 date format to UNIX time for each record, Segment uses Spark's withColumn(...) to create a new column called TIMESTAMP that is the converted value of the TIMESTAMP_ISO field. Before Segment can call withColumn, Segment needs to convert the Glue DynamicFrame into a Spark DataFrame. That is accomplished by calling toDF() on the output of ApplyMapping transformation above. Since Personalize requires Segment's uploaded CSV to be a single file, Segment calls repartition(1) on the DataFrame to force all data to be written in a single partition. After creating the TIMESTAMP in the expected format, DyanmicFrame.fromDF() is called to convert the DataFrame back into a DyanmicFrame.

```
# Repartition to a single file
onepartitionDF = apmapping1.toDF().repartition(1)
# Coalesce timestamp into unix timestamp
onepartitionDF = onepartitionDF.withColumn("TIMESTAMP", \
    unix_timestamp(onepartitionDF['TIMESTAMP_ISO'], "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'"))
# Convert back to dynamic frame
onepartition = DynamicFrame.fromDF(onepartitionDF, glueContext, "onepartition_df")
```


Segment's CSV is written back to S3 at the path specified by the `S3_CSV_OUTPUT_PATH` job property and commits the job.

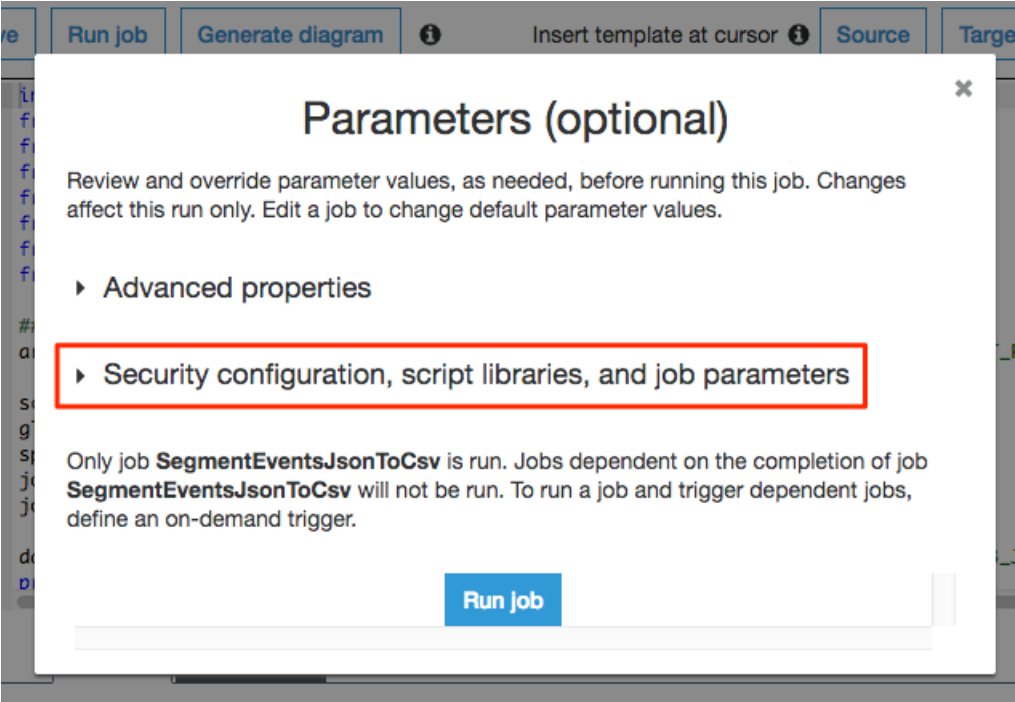
```
glueContext.write_dynamic_frame.from_options(frame = onepartition, connection_type = "s3", \
      connection_options = {"path": args['S3_CSV_OUTPUT_PATH']}, \
      format = "csv", transformation_ctx = "datasink2")

job.commit()
```

Run Your AWS Glue ETL Job

With Segment's ETL Job script created and saved, it's time to run the job to create the CSV needed to train a Personalize Solution. To do this:

- 1. Open another AWS console browser tab/window by right-clicking on the AWS logo in the upper left corner of the page and select **Open Link in New Tab** (or Window).
- 2. While still in the Glue service console and the job listed, click **Run job**. This will cause the Parameters panel to display.
- 3. Click the **Security configuration, script libraries, and job parameters** section header to cause the job parameters fields to be displayed.



- 4. Scroll down to the **Job parameters** section. This is where Segment will specify the job parameters that Segment's script expects for the path to the input data and the path to the output file.
- 5. Create two job parameters with the following key and value.

Be sure to prefix each key with `--` as shown. Substitute your account ID for `[ACCOUNT_ID]` in the values below. You copy the bucket name to your clipboard from the S3 service page in the tab/window you opened above. The order they are specified does not matter.

KEY	VALUE
--S3_JSON_INPUT_PATH	s3://personalize-data-[ACCOUNT_ID]/raw-events/
--S3_CSV_OUTPUT_PATH	s3://personalize-data-[ACCOUNT_ID]/transformed

Run job

Generate diagram

Insert template at cursor

Source

Target

Parameters (optional)

10

Job timeout (minutes)

Delay notification threshold (minutes)

Job parameters

Key	Value
--S3_CSV_OUTPUT_PATH	s3://personalize-data-22412
--S3_JSON_INPUT_PATH	s3://personalize-data-22412
Type key...	Type value...

Only job **SegmentEventsJsonToCsv** is run. Jobs dependent on the completion of job **SegmentEventsJsonToCsv** will not be run. To run a job and trigger dependent jobs, define an on-demand trigger.

Run job

6. Click **Run job** to start the job. Note that this dialog scrolls.

7. Once the job has started running, you'll see log output in the **Logs** tab at the bottom of the page. It may take a few minutes to complete.

8. When the job completes, click the **X** in the upper right corner of the page to exit the job script editor.

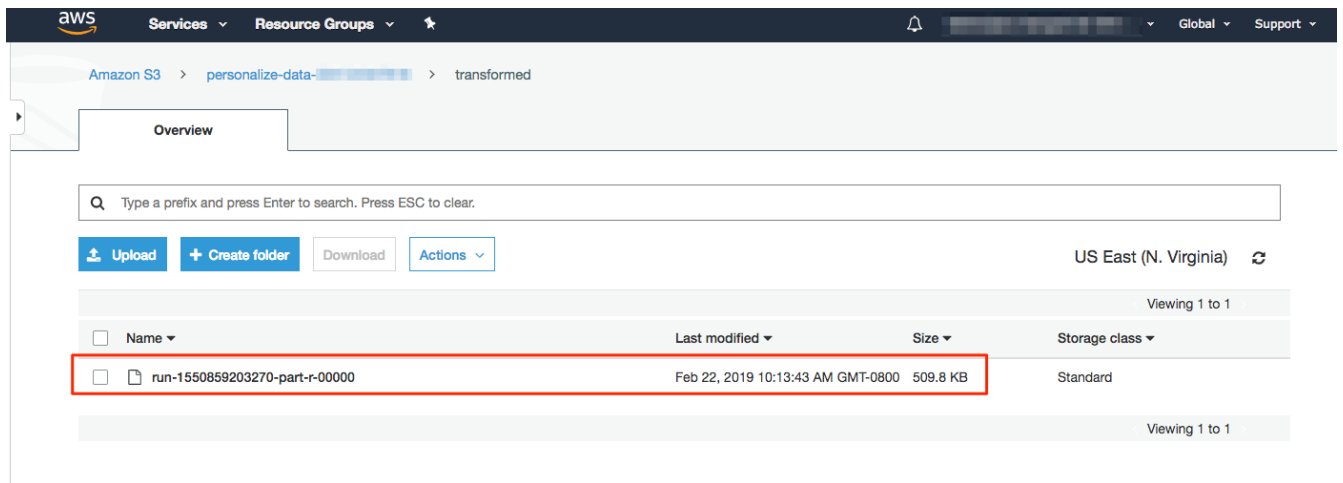
Verify Output File

To verify the output file:

1. Go to the S3 service page in the AWS console and find the bucket with a name starting with **personalize-data-...**

2. Click on the bucket name. If the job completed successfully, you'll see a folder named **transformed**.

3. Click on **transformed** and you'll see the output file created by the ETL job.



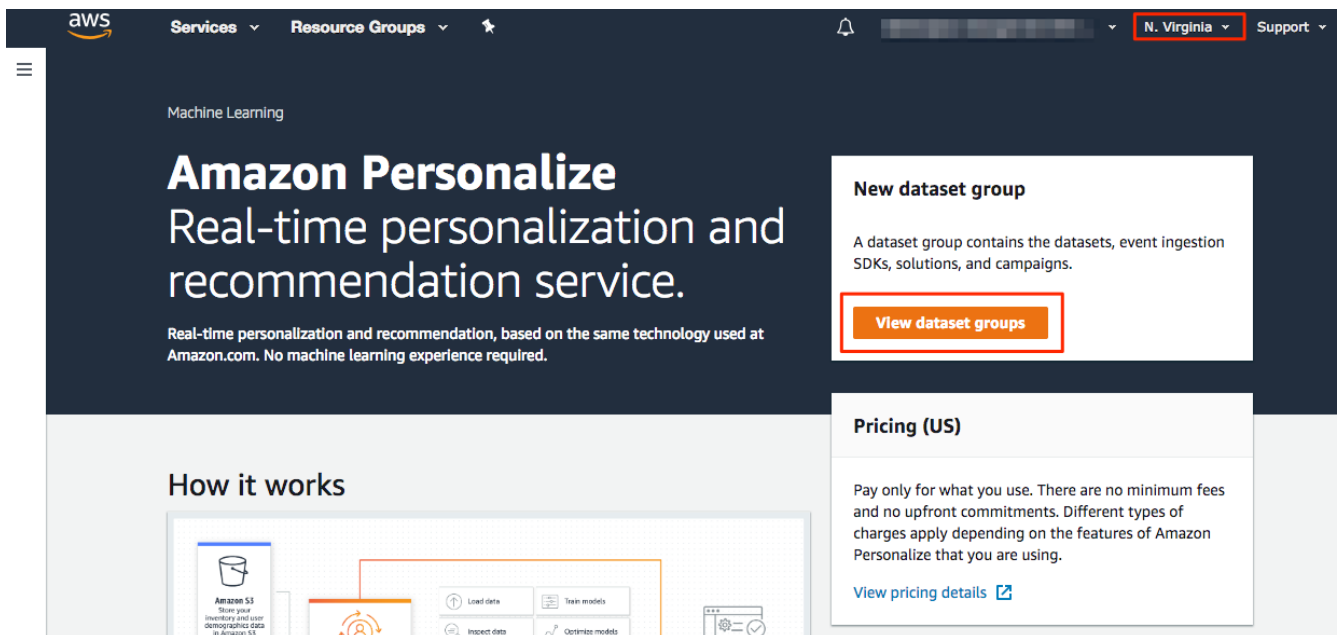
Create Personalize Dataset Group, Solution and Campaign

Create Personalize Dataset Group

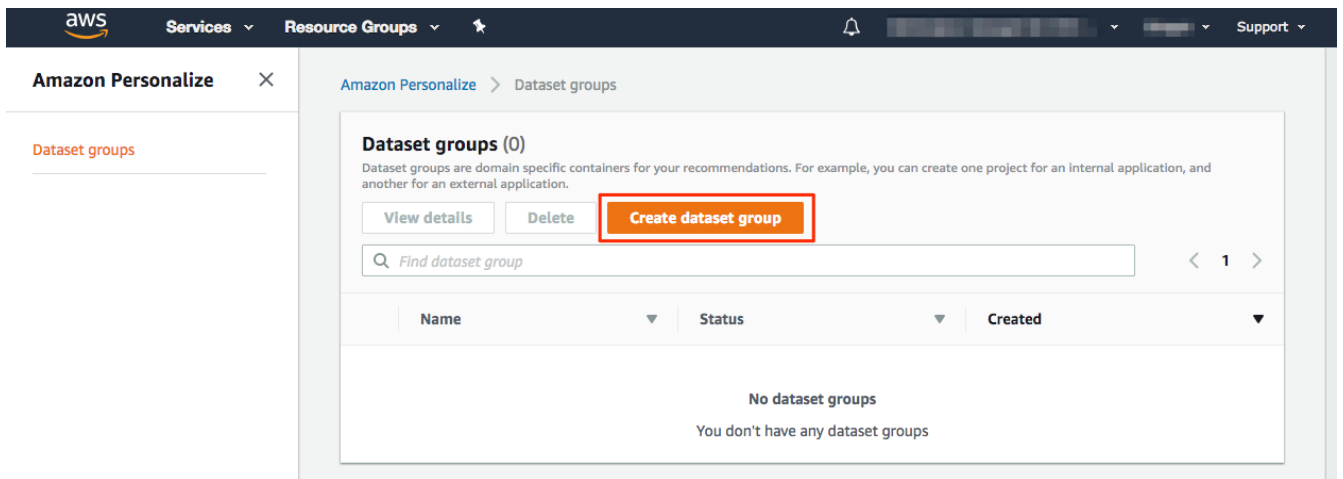
To create a personalize dataset group:

1. Browse to the Amazon Personalize service landing page in the AWS console.

2. Click **View dataset groups** to get started.



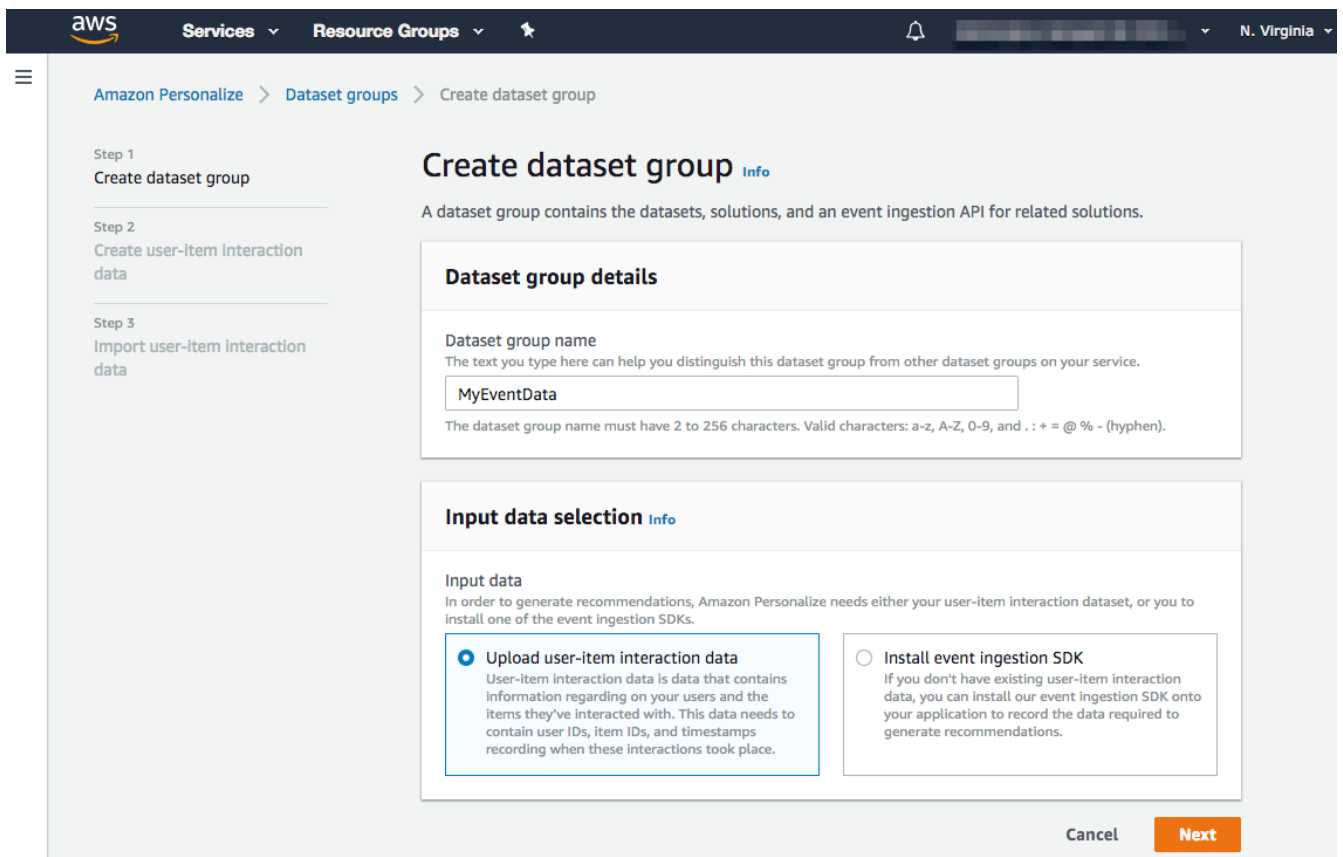
3. On the Dataset Groups page, click **Create dataset group**.



4 On the **Create dataset group** page, give your dataset group a name.

5 Select **Upload user-item interaction data** since Segment will be uploading the CSV prepared in the previous steps.

6 Click **Next** to continue.



7 On the **Create user-item interaction data** page, select **Create new schema** and give your schema a name.

8 scroll down to the **Schema definition** editor. Dataset schemas in Personalize are represented in [Avro](#). Learn more about [For detailed Personalize schema definitions](#).

Avro is a remote procedure call and data serialization framework developed within Apache's Hadoop project. It uses JSON for defining data types and protocols, and serializes data in a compact binary format.

This example uses the following example schema:

```
{
  "type": "record",
  "name": "Interactions",
  "namespace": "com.amazonaws.personalize.schema",
  "fields": [
    {
      "name": "USER_ID",
      "type": "string"
    },
    {
      "name": "ITEM_ID",
      "type": "string"
    },
    {
      "name": "EVENT_TYPE",
      "type": "string"
    },
    {
      "name": "TIMESTAMP",
      "type": "long"
    }
  ],
  "version": "1.0"
}
```

The required fields for the user-item interaction dataset schema are `USER_ID`, `ITEM_ID`, and `TIMESTAMP`. There's also an optional field `EVENT_TYPE`.

9 copy the contents of Avro schema to your clipboard and paste it into the "Schema definition" editor (replacing the proposed schema).

10 **Click Next** to save the schema and move to the next step.

The **Import user-item interaction data** step is displayed next. To complete this form Segment needs to get two pieces of information from IAM and S3. Give your import job a name and set the automatic import to **Off**.

12 For the **IAM service role**, select **Create a new role** from the dropdown.

13 In the next pop-up, Segment recommends listing your bucket name in the **Specific S3 buckets** option, but you're free to choose the option that best suits your needs.

14 Find the location of the CSV file you generated in the earlier steps. This needs to be configured in the **Data Location** field on this screen.

The screenshot shows the AWS IAM console interface for creating a dataset group. The breadcrumb navigation at the top reads: Amazon Personalize > Dataset groups > Create dataset group. On the left, a sidebar shows the progress: Step 1 'Create dataset group' is completed, Step 2 'Create user-item interaction data' is the current step, and Step 3 'Import user-item interaction data' is next. The main heading is 'Import user-item interaction data' with an 'Info' link. Below the heading, a note states: 'In this step, you create a dataset import job which imports your data from S3.'

The 'Dataset import job details' section contains the following fields and options:

- Dataset import job name:** A text input field containing 'MyEventDataImportJob'. A note below states: 'The dataset import job name must have 2 to 256 characters. Valid characters: a-z, A-Z, 0-9, and . : + = @ % - (hyphen).'
- Automatic dataset import jobs:** A section with an 'Info' link. A note states: 'This option automatically runs this dataset import job on a regular schedule so that you won't have to manually do so.' There are two radio buttons: 'On' (unselected) and 'Off' (selected). A note for 'Off' states: 'You'll have to manually import your dataset.'
- IAM service role:** A section with a note: 'Amazon Personalize requires permissions to access your S3 bucket. Choose an existing role with access or create a role in the IAM console with the [AmazonPersonalizeFullAccess](#) IAM policy attached.' Below this is a dropdown menu labeled 'Enter a custom IAM role ARN'.
- Custom IAM role ARN:** A text input field containing 'arn:aws:iam::[redacted]:role/service-role/module-personalize-Person'.

A blue information box with an 'i' icon contains the following text: **Additional s3 bucket policy required**. 'In addition to the IAM service role defined above, Amazon Personalize also requires you to add a bucket policy to the S3 bucket containing your data file(s) so that it can process them. Follow the instructions [described here](#) to add the required bucket policy to your S3 bucket.'

The 'Data location' section has an 'Info' link and a note: 'Choose the S3 location of your data.' Below this is a text input field containing 's3://personalize-data-[redacted]/transformed/run-1551108667029-part-r-00000'.

15 After clicking the **Finish** button at the bottom of the page, you'll return to the Personalize Dashboard where you can monitor the progress of your interaction dataset as it is being created.

Be patient as this process can take a long time to complete.

aws Services Resource Groups

Amazon Personalize

Dataset groups

MyEventData

Dashboard

Datasets

Event trackers Info

Solutions and recipes Info


Campaigns Info

Your user-Item Interaction dataset is being uploaded.

Amazon Personalize > Dataset groups > MyEventData > Dashboard

Dashboard

Overview



Upload datasets


Datasets are required to create solutions, which are then used to generate recommendations.

User-item interaction data

Create in progress view

User data Import

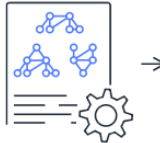
Item data Import



Install event ingestion SDK

The event ingestion SDK allows you to track user events in your application and feed them to your solutions.


SDK Installation Start



Create solutions

Solutions help you generate recommendations. They consist of custom models trained on your datasets along with the underlying infrastructure required to generate recommendations.

Solution training Start



Launch campaigns

Campaigns create an endpoint that your application calls to get recommendations from a solution. They also provide you with analytics on the solution's usage.

Campaign creation Start

Create Personalize Solution

Once Segment's event CSV is finished importing into a user-item interaction dataset, Segment can create a Personalize Solution. To do this:

From the Dashboard page for the dataset group created above, click **Start** in the **Create solutions** column.

aws Services Resource Groups

Admin/jjory-Isengard @ 2241-... N. Virginia Support

Amazon Personalize

Dataset groups

MyEventData

Dashboard

Datasets

Event trackers Info


Solutions and recipes Info

Campaigns Info

Amazon Personalize > Dataset groups > MyEventData > Dashboard

Dashboard

Overview



Upload datasets


Datasets are required to create solutions, which are then used to generate recommendations.

User-item interaction data

Active view

User data Import


Item data Import



Install event ingestion SDK

The event ingestion SDK allows you to track user events in your application and feed them to your solutions.


SDK Installation Start



Create solutions

Solutions help you generate recommendations. They consist of custom models trained on your datasets along with the underlying infrastructure required to generate recommendations.

Solution creation Start



Launch campaigns

Campaigns create an endpoint that your application calls to get recommendations from a solution. They also provide you with analytics on the solution's usage.

Campaign creation Start

On the **Create solution** page, enter a **Solution name**.

For a discussion on the different recipes you can use with Personalize, see Amazon's [Choosing a recipe](#) documentation.

The screenshot shows the AWS Personalize console's 'Create solution' page. The breadcrumb trail at the top reads: Amazon Personalize > Dataset groups > MyEventData > Solutions > Create solution. The main heading is 'Create solution' with an 'Info' link. Below this is a descriptive sentence: 'You create a solution using a recipe that is tailored to a specific use case.'

The 'Solution configuration' section contains the following fields:

- Solution name:** A text input field containing 'MyEventSolution'. Below it, a note states: 'The solution name must have 2 to 256 characters. Valid characters: a-z, A-Z, 0-9, and . : + = @ % - (hyphen).'
- Recipe selection:** A section with an 'Info' link. It explains that a recipe consists of recommendation algorithms and data processing steps. Two radio buttons are present: 'Manual' (selected) with the description 'Choose the recipe manually.', and 'Automatic (AutoML)' with the description 'Amazon Personalize will find the best recipe for your dataset.'
- Recipe:** A dropdown menu showing 'aws-hrnn' with a description: 'Predicts items a user will interact with. A Hierarchical Recurrent Neural Netw...'
- Minimum provisioned transactions per second:** A section with an 'Info' link. It explains that this is the minimum amount of throughput in transactions per second (TPS) provisioned for the solution. A text input field contains the value '1'. Below it, a note states: 'Type in a value between 1 and 500.'

At the bottom right of the configuration box, there are two buttons: 'Cancel' and 'Finish' (which is highlighted in orange).

Click **Finish** to create your Solution. This process can take several minutes to several hours to complete.

aws Services Resource Groups Admin/jjory-Isengard @ 2241-... N. Virginia Support

Amazon Personalize X

Dataset groups

▼ MyEventData

Dashboard

Datasets

Event trackers

Solutions and recipes

Campaigns

Solution creation in progress...
Creating solution MyEventSolution. The creation time depends on the size of your dataset.

Amazon Personalize > Dataset groups > MyEventData > Dashboard

Dashboard

Overview

Upload datasets

Datasets are required to create solutions, which are then used to generate recommendations.

User-item interaction data Active view

User data

Item data

Install event ingestion SDK

The event ingestion SDK allows you to track user events in your application and feed them to your solutions.

SDK Installation

Create solutions

Solutions help you generate recommendations. They consist of custom models trained on your datasets along with the underlying infrastructure required to generate recommendations.

Solution creation Create in progress view

Launch campaigns

Campaigns create an endpoint that your application calls to get recommendations from a solution. They also provide you with analytics on the solution's usage.

Campaign creation

Create Personalize Campaign

A deployed solution is known as a campaign, and is able to make recommendations for your users. To deploy a solution, you create a campaign in the console or by calling the CreateCampaign API. You can choose which version of the solution to use. By default, a campaign uses the latest version of a solution.

To create a Personalize campaign:

From the Dataset Group Dashboard, click **Create new campaign**.

aws Services Resource Groups N. Virginia Support

Amazon Personalize X

Dataset groups

▼ MyEventData

Dashboard

Datasets

Event trackers

Solutions and recipes

Campaigns

Amazon Personalize > Dataset groups > MyEventData > Dashboard

Dashboard

Overview

Upload datasets

Datasets are required to create solutions, which are then used to generate recommendations.

Install event ingestion SDK

The event ingestion SDK allows you to track user events in your application and feed them to your solutions.

Create solutions

Solutions help you generate recommendations. They consist of custom models trained on your datasets along with the underlying infrastructure required to generate recommendations.

Launch campaigns

Campaigns create an endpoint that your application calls to get recommendations from a solution. They also provide you with analytics on the solution's usage.

2. Enter the name for your campaign.

3. Select the solution you created above and click **Create campaign**.

The screenshot shows the AWS Personalize console's 'Create new campaign' page. The breadcrumb trail is 'Amazon Personalize > Dataset groups > MyEventData > Campaigns > Create new campaign'. The main heading is 'Create new campaign'. Below it is a 'Campaign details' section. The 'Campaign name' field contains 'segment-workshop-campaign'. A note states: 'The text you type here appears in the Campaign dashboard and detail page. It can help you distinguish this campaign from others.' Below the field, a validation message says: 'The campaign name must have 2 to 256 characters. Valid characters: a-z, A-Z, 0-9, and . : + = @ % - (hyphen)'. The 'Solution' dropdown menu is set to 'MyEventSolution'. At the bottom right are 'Cancel' and 'Create campaign' buttons.

4. Personalize will start creating your new campaign. This process can take several minutes.

The screenshot shows the AWS Personalize console after a campaign has been created. A blue banner at the top says 'Campaign creation in progress... Creating campaign segment-workshop-campaign. This may take a few minutes to hours...'. The breadcrumb trail is 'Amazon Personalize > Dataset groups > MyEventData > Campaigns > segment-workshop-campaign'. The main heading is 'segment-workshop-campaign'. There are 'Delete' and 'Update' buttons. Below the heading are tabs for 'Personalization API' and 'Details'. The 'Details' tab is active, showing 'Campaign inference' information. It includes a note: 'To get recommendations for this campaign in your application, you need to use the getRecommendations API call. You can learn more about the usage and requirements for this API call in the documentation and the other links listed below.' followed by a link to the 'Amazon Personalize GetRecommendations Developer Guide'. At the bottom, it shows the 'Campaign arn Info' as 'arn:aws:personalize:us-east-1:224124347618:campaign/segment-workshop-campaign'.

In the next section, Segment will build a real-time clickstream ingestion pipeline that accepts events from Segment and can query the solution you just deployed.

Getting Recommendations and Live Event Updates

Once you deploy your Personalize solution and enable a Campaign, your Lambda instance consumes event notifications from Segment and uses the Solution and Campaign to react to events which drive your business cases.

The example code Segment provides below shows how to forward events to the Personalize Solution you deployed to keep your model updated. It then forwards an `identify` event back to Segment with the recommendations from your Solution.

Set up Segment IAM policy & role for invoking your Lambda

Segment will need to be able to call (“invoke”) your Lambda in order to process events. This requires you to configure an IAM role for your Lambda which allows the Segment account to invoke your function.

Create an IAM policy

To create an IAM policy:

1. Sign in to the [Identity and Access Management \(IAM\) console](#) and follow these instructions to [Create an IAM policy](#) to allow Segment permission to invoke your Lambda function.

2. Select **Create Policy from JSON** and use the following template policy in the Policy Document field. Be sure to change the {region}, {account-id} and {function-names} with the applicable values. Here’s example of a Lambda ARN `arn:aws:lambda:us-west-2:355207333203:function:my-example-function`.



NOTE: You can put in a placeholder ARN for now, as you will need to come back to this step to update with the ARN of your Lambda once that’s been created.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction"
      ],
      "Resource": [
        "lambda ARN 1",
        "lambda ARN 2",
        "...",
        "lambda ARN n"
      ]
    }
  ]
}
```

Create an IAM role

To create an IAM role:

1. Sign in to the [Identity and Access Management \(IAM\) console](#) and follow these instructions to [Create an IAM role](#) to allow Segment permission to invoke your Lambda function.

2. While setting up the new role, add the policy you created in the [previous step](#).

3. Finish with any other set up items you may want (like tags).

4. Search for and click on your new roles from the [IAM home](#).

5. Select the **Trust Relationships** tab, then click **Edit trust relationship**.

Summary

Role ARN

Role description

Instance Profile ARNs

Path

Creation time

Maximum CLI/API session duration

Give this link to users who can switch roles in the console

Permissions

Trust relationships

Tags

A

You can view the trusted entities that can assume the r

Edit trust relationship

Trusted entities

The following trusted entities can assume this role.

Trusted entities

The account 595280932656

6. Copy and paste the following into your trust relationship. You should replace `<your-source-id>` with either the Source ID of the attached Segment source (the default) or the custom external ID you set in your Amazon Lambda destination settings.



NOTE: Your Source ID can be found by navigating to **Settings > API Keys** from your Segment source homepage.

For security purposes, Segment will set your Workspace ID as your External ID. If you are currently using an External ID different from your Workspace ID, reach out to Segment support so they can change it and make your account more secure.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::595280932656:role/customer-personalize-prod-destination-access"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "YOUR_SEGMENT_SOURCE_ID"
        }
      }
    }
  ]
}
```

If you have multiple Sources using this Role, or require the use of multiple externalIds, replace the `sts:ExternalId` setting above with:

```
"sts:ExternalId": ["YOUR_SEGMENT_SOURCE_ID", "ANOTHER_SOURCE_ID", "AN_EXTERNAL_ID", "ANOTHER_EXTERNAL_ID"]
```

Build a Lambda Function to Process Segment Events

In order to process events from Segment, you will need to provide a Lambda function that can handle your event flow. This function can be used to forward events to a Tracker for your Personalize solution, or to post process events, get recommendations from your Solution, or to push these results back to Segment for later use in the destinations you have configured.

Segment allows you to send each call type (`track`, `identify`, etc) to a different Lambda function. The example below shows one generic function that could be used to handle any call.

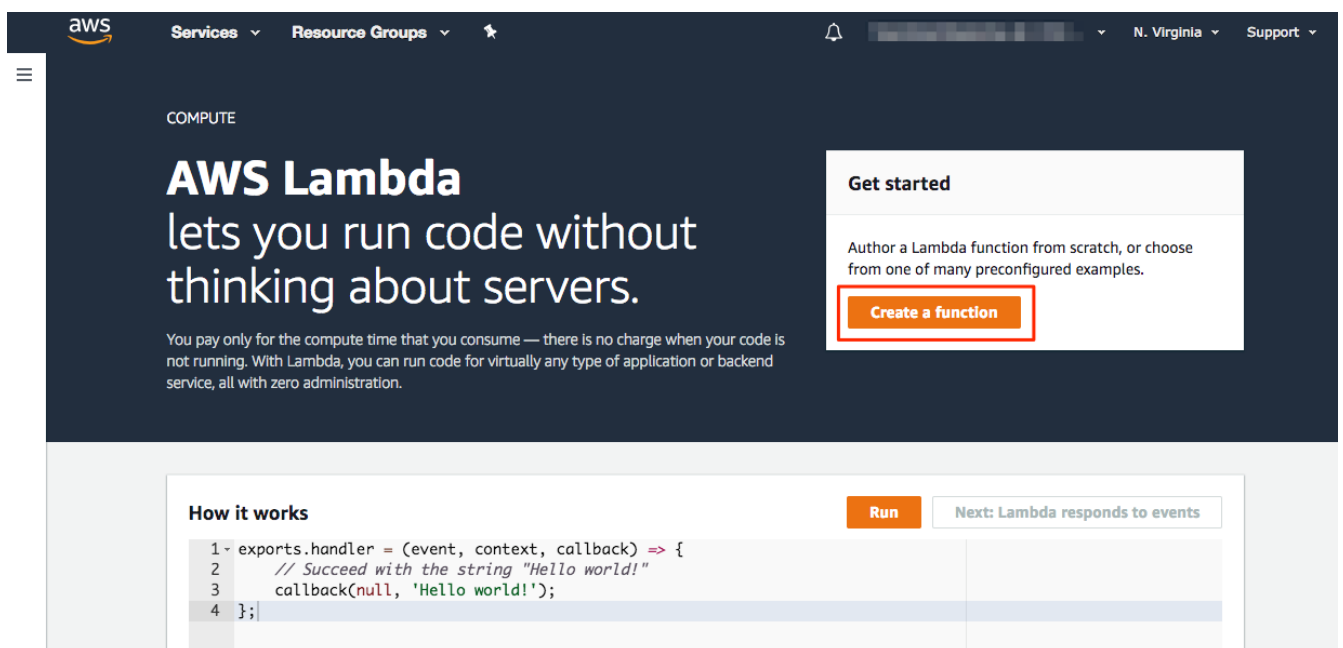
Segment provides an example Lambda function, written in Python, for you to get up and running.

To build a Lambda function to process Segment events:

1. Go to the Lambda service page in your AWS account.

2. Ensure that you are in AWS Region 'us-west-2'. You must be in us-west-2 so that Segment's Lambdas can communicate with your resources.

3. Click **Create a function** to create a new function.



4. Select **Author from scratch** since Segment will be providing the source code for the function.

5. Enter a name for your function and select **Python 3.7** for the runtime.

6. For the **Role** field, select **Create a new role from AWS policy templates** from the dropdown.

7. Create a **Role name** that makes sense for you, and leave **Policy templates** empty. You will come back to modify this role shortly.

8. Click **Create function**.

Function name
Enter a name that describes the purpose of your function.

personalizeDemo

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)
Choose the language to use to write your function.

Python 3.7

Permissions [Info](#)
Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.

▼ Choose or create an execution role

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

Create a new role from AWS policy templates

① Role creation might take a few minutes. The new role will be scoped to the current function. To use it with other functions, you can modify it in the IAM console.

Role name
Enter a name for your new role.

personalize-demo-role

Use only letters, numbers, hyphens, or underscores with no spaces.

Policy templates [Info](#)
Choose one or more policy templates.

Cancel Create function

Lambda Function Source Code

1. Download the .zip file at https://github.com/segmentio/segment-lambda-recipes/blob/master/segment-personalize/support/segment_personalize_boilerplate.zip.

2. Within the Lambda function screen, scroll down to the **Function code** panel.

3. For **Code entry type** choose **Upload a .zip file**, and click **Upload**, then load the .zip you downloaded in the first step.

You should now be able to see the code (and associate folders) in the code editor.

Code entry type: Edit code inline Runtime: Python 3.7 Handler: app.lambda_handler

```

1  import json
2  import boto3
3  import os
4  import requests # Needed for Segment Events REST APIs
5  import dateutil.parser as dp
6  import init_personalize_api as api_helper # This is only needed for the Personalize beta
7
8  from botocore.exceptions import ClientError
9
10 connections_endpoint_url = "https://api.segment.io/v1"
11 connections_source_api_key = os.environ['connections_source_write_key']
12
13 def api_post(url, key, payload):
14     myResponse = requests.post(url, auth=(key, ''), json=payload)
15     if(myResponse.ok):
16         jData = json.loads(myResponse.content)
17         return jData
18     else:
19         myResponse.raise_for_status()
20
21 def set_user_traits(user_id, traits):
22     # Sends an identify call to Personas to update a user's traits
23     formatted_url = "{s}/identify".format(connections_endpoint_url)
24     message = { "traits": traits, "userId": user_id, "type": "identify" }
25     try:
26         response = api_post(formatted_url, connections_source_api_key, message)
27     except HTTPError as error:
28         status = error.response.status_code
29         if status >= 400 and status < 500:

```

Segment will call your lambda once per event. The provided code maps Segment event fields from the Segment event it gets, and sends them to your Personalize Tracker. It then calls Personalize to get a recommendation for the userId in the event, and pushes that recommendation back as a user trait into Segment, using the `identify` call.

Make sure you are clicking **Save** frequently during the next steps!

Wire up Personalize API using Lambda Layer (Preview only)

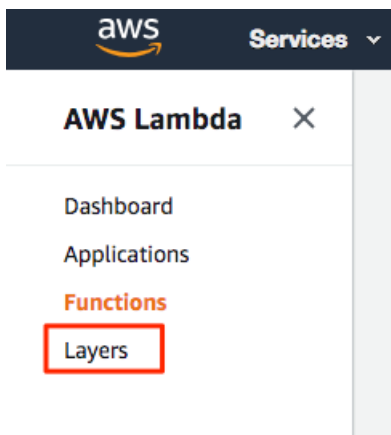
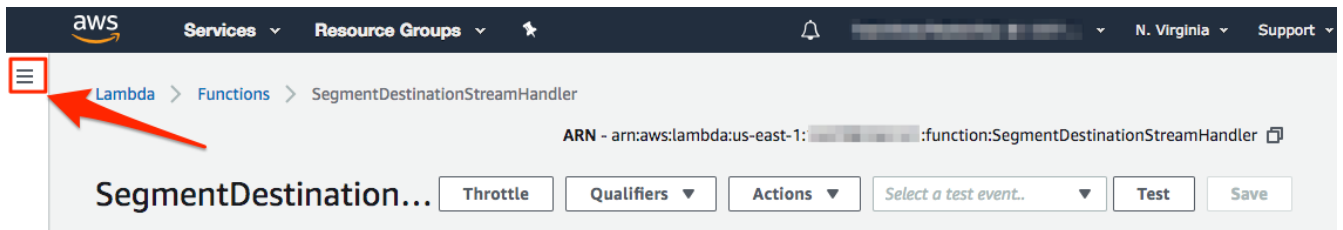
You will notice in the function source the following `import` and function call.

```
import of import init_personalize_api as api_helper
...
api_helper.init()
```

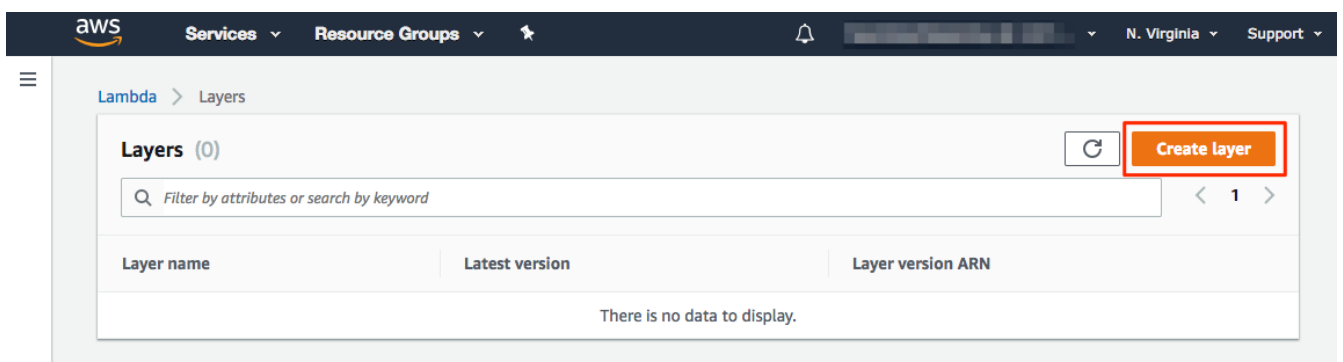
This `import` and function call uses some boilerplate code, packaged as a [Lambda Layer](#) needed to configure the Personalize API with the AWS Python SDK. This is only necessary while Personalize is in Preview. Once Personalize is GA and the API is bundled with the Python SDK, as well as other language SDKs, this supporting Layer will no longer be needed.

To install Segment's Layer:

1. Open the Lambda navigation panel and click **Layers**.



2. From the Lambda Layers view, click **Create layer**.



3. Create the layer by specifying a name such as "PersonalizeApiInstaller", browsing to the pre-made zip in https://github.com/segmentio/segment-lambda-recipes/blob/master/segment-personalize/support/python_personalize_init.zip, and select **Python 3.7** as the compatible runtime.

4. Click **Create** to upload the zip file and create the layer.

aws Services Resource Groups

Lambda > Layers > Create layer

Create layer

Layer configuration

Name
PersonalizeApiInstaller

Description
Installs Personalize API with Python SDK (required only while in Preview)

Code entry type
Upload a .zip file

Upload python_personalize_init.zip (8.5 kB)
For files larger than 10 MB, consider uploading using Amazon S3.

Compatible runtimes Info
Select all compatible runtimes for your layer.
Select runtimes
Python 3.6 X Python 3.7 X

License Info

Cancel Create

5. Add the layer just created to Segment's function.

6. Return to the Lambda function by opening the Lambda navigation panel and clicking **Functions**.

aws Services Resource Groups

Lambda > Functions > SegmentDestinationStreamHandler

ARN - arn:aws:lambda:us-east-1: :function:SegmentDestinationStreamHandler

SegmentDestination... Throttle Qualifiers Actions Select a test event.. Test Save

aws Services

AWS Lambda X

- Dashboard
- Applications
- Functions**
- Layers

7. Click on your function name to access the configuration page again.

8. In the Lambda Designer, click the **Layers** panel below the function name and then **Add layer** in the **Referenced layers** panel at the bottom of the page.

Configuration | **Monitoring**

▼ **Designer**

Add triggers

Choose a trigger from the list below to add it to your function.

- API Gateway
- AWS IoT
- Alexa Skills Kit
- Alexa Smart Home
- Application Load Balancer
- CloudFront
- CloudWatch Events
- CloudWatch Logs

SegmentDestinationStreamHandler

Layers (0)

Add triggers from the list on the left

Amazon CloudWatch Logs

Amazon Kinesis

personalize

Resources that the function's role has access to appear here

Referenced layers [Info](#)

Add a layer

▲ Merge earlier

▼ Merge later

Remove

Merge order	Layer name	Layer version	Layer version ARN
There is no data to display.			

9. Select the layer you just added and the latest version.

10. Click **Add** to add the layer to the function.

aws Services ▾ Resource Groups ▾ ⭐

TeamRole/MasterKey @ 1742-... ▾ N. Virginia ▾ Support

☰

Lambda > Layers > Add layer to function

Add layer to function

Layer selection

Select an existing AWS-validated layer or layer in your account, or provide a layer that has been shared with you. You can connect a maximum of 5 layers to a function.

☒ Select from list of runtime compatible layers
 ☐ Provide a layer version ARN

Select from list of runtime compatible layers

Layer

PersonalizeApiInstaller ▾

Version

1 ▾

Cancel **Add**

Update your IAM role for your Lambda to call Personalize

You need to modify the IAM Role & Policy originally created with this Lambda to allow it to send and receive data from Personalize. To do this:

1. From the **Execution role** section of your Lambda function, click the ****View the**** link.

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

Use an existing role ▼

Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

service-role/personalize-demo-role ▼



[View the personalize-demo-role role](#) on the IAM console.

2. Click the arrow next to your policy in this role, then **Edit Policy**.

Permissions

Trust relationships

Tags

Access Advisor

Revoke s

▼ Permissions policies (1 policy applied)

Attach policies

Policy name ▼

▼ sara-personalize-policy

Policy summary

{ } JSON

Edit policy

7
8

"Resource": "arn:aws:logs:us-west-2:",
},

3. Add the code below to the existing permissions from within the JSON editor.

4. Click **Review Policy** and **Save Changes**.

```
{
  "Effect": "Allow",
  "Action": [
    "personalize:GetRecommendations",
    "personalize:Record",
    "personalize:PutEvents"
  ],
  "Resource": [
    "*"
  ]
}
```

Wire-up Personalize Event Tracker

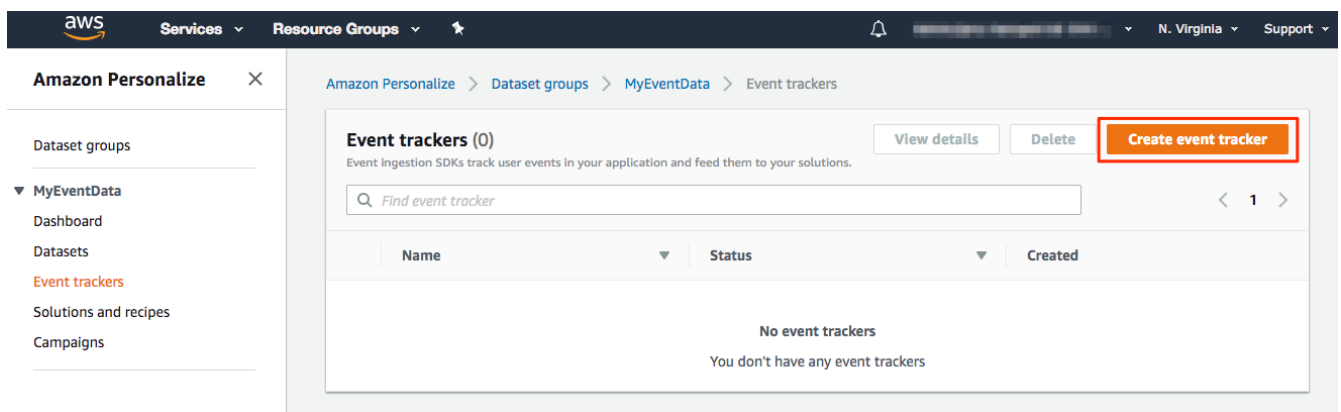
Another dependency in the function is the ability to call the Personalize [PutEvents API](#) endpoint as shown in the following excerpt.

```
personalize_events.put_events(  
    trackingId = os.environ['personalize_tracking_id'],  
    userId = event['userId'],  
    sessionId = event['anonymousId'],  
    eventList = [  
        {  
            "eventId": event['messageId'],  
            "sentAt": int(dp.parse(event['timestamp']).strftime('%s')),  
            "eventType": event['event'],  
            "properties": json.dumps(properties)  
        }  
    ]  
)
```

The `trackingId` function argument identifies the Personalize Event Tracker which handles the events Segment submits. This value is passed to Segment's Lambda function as an Environment variable.

You need to create a Personalize Event Tracker for the Dataset Group you created earlier. To do this:

- 1 In another browser tab/window, go to the Personalize service landing page in the AWS console.
- 2 Click on your Dataset Group and then **Event trackers** in the left navigation.
- 3 Click **Create event tracker** button.



- 4 Enter a name for your Event Tracker.

- 5 You need to configure a role for Personalize to that allows it to execute the tracker. This is the same as the execution role you defined earlier for Personalize. Often this is automatically included as a policy labelled "AmazonPersonalizeFullAccess"

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "personalize:*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:*Personalize*",
        "arn:aws:s3:*personalize*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "personalize.amazonaws.com"
        }
      }
    }
  ]
}

```

This may be automatically included as policy "AmazonPersonalize-ExecutionPolicy-"

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:ListBucket"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::{ your s3 bucket }"
      ]
    },
    {
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::{ your s3 bucket }/*"
      ]
    }
  ]
}

```

The Event Tracker's tracking ID is displayed on the following page and is also available on the Event Tracker's detail page. Copy this value to your clipboard.

Returning to the Lambda function, paste the Event Tracker's tracking ID into an Environment variable for the function with the key `personalize_tracking_id`.

Add environment variables for Segment and for the function to tell it the Personalize Campaign to call for retrieving recommendations.

To obtain the Personalize Campaign ARN, go to the Personalize service landing page in the AWS console.

Select the Dataset Group you created earlier and then Campaigns in the left navigation.

Click on the campaign you created earlier and copy the **Campaign ARN** to your clipboard.

The screenshot shows the Amazon Personalize console interface. On the left, the 'Campaigns' link is highlighted in the navigation menu. The main content area displays the details for the 'segment-workshop-campaign'. A blue banner at the top indicates 'Campaign creation in progress...'. Below this, the breadcrumb trail shows the path: Amazon Personalize > Dataset groups > MyEventData > Campaigns > segment-workshop-campaign. The campaign name 'segment-workshop-campaign' is prominently displayed with 'Delete' and 'Update' buttons. Under the 'Personalization API' tab, the 'Campaign inference' section provides instructions on using the 'getRecommendations' API call and includes a link to the 'Amazon Personalize GetRecommendations Developer Guide'. At the bottom, the 'Campaign arn' is shown as 'arn:aws:personalize:us-east-1:123456789012:campaign/segment-workshop-campaign'.

Return to your Lambda function and scroll down to the **Environment variables** panel.

Add an environment variable with the key `personalize_campaign_arn` and value of the Campaign ARN in your clipboard.

Scroll to the top of the page and click **Save** to save your changes.

The screenshot shows the 'Environment variables' panel in the AWS Lambda console. It includes a descriptive text about environment variables and a table of existing variables. Each variable has a 'Remove' button next to it.

Key	Value	Action
connections_source_write_key	[Redacted]	Remove
endpoint_url	https://personalize-runtime.us-west-2.amazonaws.com	Remove
personalize_campaign_arn	arn:aws:personalize:us-west-2:123456789012:campaign/segment-workshop-campaign	Remove
personalize_tracking_id	[Redacted]	Remove
region_name	us-west-2	Remove
Key	Value	Remove

You need a key for the Segment source that will get Segment's update events. Go back to your Segment workspace tab or window, and click on the source which will receive events from your Lambda, and copy the write key from the **Overview** tab.

Segment allows you to send each call type to a different Lambda. If you leave the Lambda field blank for a given call type, Segment won't attempt to send any of those calls.

Track

There are two settings relevant for track calls:

1. **Lambda for track calls** - the Lambda where the Segment app should route track calls.
2. **Events** - a list of specific events to send. You may send *all* track events (see setting details for instructions on how), but use caution with this option, as it may significantly increase your Lambda costs.

FAQ

What is the Log Type Setting?

This setting controls the **Log Type** for your Lambda function using Cloud Watch. Select option **Tail** if you would like to see [detailed logs](#) in Cloud Watch.

My Lambda <> Segment connection is timing out, what do I do?

Due to how Segment's event delivery system, [Centrifuge](#), works, your Lambda can't take more than five seconds to run per message. If you're consistently running into timeout issues, you should consult the [AWS Lambda docs](#), as well as docs for your language of choice, for tips on optimizing performance.

Engage

You can send computed traits and audiences generated using [Engage](#) to this destination as a **user property**. To learn more about Engage, schedule a [demo](#).

For user-property destinations, an [identify](#) call is sent to the destination for each user being added and removed. The property name is the snake_cased version of the audience name, with a true/false value to indicate membership. For example, when a user first completes an order in the last 30 days, Engage sends an Identify call with the property `order_completed_last_30days: true`. When the user no longer satisfies this condition (for example, it's been more than 30 days since their last order), Engage sets that value to `false`.

When you first create an audience, Engage sends an Identify call for every user in that audience. Later audience syncs only send updates for users whose membership has changed since the last sync.



Real-time to batch destination sync frequency

Real-time audience syncs to Amazon Personalize may take six or more hours for the initial sync to complete. Upon completion, a sync frequency of two to three hours is expected.

Settings

Segment lets you change these destination settings from the Segment app without having to touch any code.

SETTING	DESCRIPTION
Client Context	<code>map</code> , defaults to <code>{}</code> . An optional map to pass to the Lambda function. See AWS Lambda documentation for more information.
External ID (Read-Only)	<code>string</code> , defaults to <code>#SEGMENT_WORKSPACE_ID</code> . This is an optional string Segment will use to assume the role provided to invoke the Lambda function. If this setting is not defined, we'll use the Source ID. This value is read-only. Reach out to support if you wish to change it. For more information about external IDs while assuming AWS roles, check here .

SETTING	DESCRIPTION
Lambda (required)	<code>string</code> . The name of the Lambda function to invoke. These are the supported name formats: * Function name (<code>my-function</code>) or with alias (<code>my-function:v1</code>). * Function ARN (<code>arn:aws:lambda:us-west-2:123456789012:function:my-function</code>). * Partial ARN (<code>123456789012:function:my-function</code>). You can append a version number or alias to any of the formats.
Log Type	<code>select</code> . Lambda <code>log type</code> . By default <code>None</code> . Select <code>Tail</code> if you would like to see detailed logs in Cloud Watch.
Region	<code>string</code> . AWS Region where the lambda lives. If it is not defined, we'll use <code>us-west-2</code> by default.
Role Address (required)	<code>string</code> . The address of the AWS role that will be invoking Lambda (ex: <code>arn:aws:iam::874699288871:role/example-role</code>).

This page was last modified: 09 Aug 2024

Need support?

Questions? Problems? Need more info? Contact Segment Support for assistance!

Visit our Support page

Help improve these docs!

Edit this page

+ Request docs change

Was this page helpful?

Yes

No

Get started with Segment

Segment is the easiest way to integrate your websites & mobile apps data to over 300 analytics and growth tools.

Your work e-mail

Request Demo

or

Create free account

© 2025 Segment.io, Inc.

Privacy

Terms

Website Data Collection Preferences

