Game Analytics Pipeline

AWS Developer Guide

Kyle Somers

Greg Cheng

Daniel Lee

Timur Tulyaganov

May 2020

Last updated: August 2021 (refer to revisions)

Contents

Customizations for solution deployment 4
Solution API 4
Events 5
Applications 7
Authorizations 12
Common API errors 19
Game event schema 20
JSON schema definition 20
Game event data taxonomy 21
Use the solution API to ingest game events 25
Overview of the integration with Kinesis Data Streams 25
Configure applications to send data to the solution 26
Streaming ingestion details 27
Events processing function 28
Amazon S3 storage configuration 32
AWS Glue ETL workflow 34
Integrate with Amazon Pinpoint 34
Customize the Kinesis Data analytics application 35
Amazon Kinesis Data Analytics for SQL 35
AnalyticsProcessingFunction Lambda Function 36
Customize the streaming SQL queries 36
Example real-time analytics queries 37
Explore the sample athena queries 38
Use Postman with the solution API 40
Step 1. Create an IAM policy 40
Step 2. Create an IAM user 41
Step 3. Create a Postman environment 42
Step 4. Create a Postman collection for the solution API 43

Step 5. Test the solution API 44
Document Revisions 45

About this guide

This developer guide provides information about customizing and extending the Game Analytics Pipeline solution. It includes detailed information about how the solution components work and how they can be customized.

The guide is intended for game developers, IT infrastructure architects, administrators, and DevOps professionals who have practical experience architecting in the AWS Cloud.

Customizations for solution deployment

The Game Analytics Pipeline solution provides an option to ingest data using the solution API, and comes preconfigured with a Game Event Schema that is used to validate and process input events. This developer guide provides technical details about the solution API, processing workflows, and customizations for the real-time streaming analytics application.

Solution API

The Game Analytics Pipeline solution API enables you to send telemetry event data to the solution via REST API. The API is the entry point for applications to send data, and it provides functionality for administrators to programmatically configure registered applications. The solution supports HTTPS only, using a certificate managed by AWS. For information about configuring a custom domain for your REST API, refer to Setting up custom domain names for REST APIs in the *Amazon API Gateway Developer Guide*.

This solution uses a `LambdaAuthorizer` Lambda function to authorize telemetry event requests, and AWS Identity and Access Management (IAM) to authenticate and authorize requests to the configuration endpoints.

The following operations are available in the solution API.

Events

- Send event(s) to the solution API

Applications

- Create and register an application
- List applications
- Get application details
- Delete an application

Authorizations

- Create API Key Authorization for application
- List authorizations
- Get authorization details

- Delete authorization
- Modify authorization status

The solution API also provides a list of common API errors.

Events

Send event(s) to the solution API

Description

Operation: `POST /applications/{applicationId}/events`

This operation enables you to send a batch of up to 500 game events (or 5 MB total) in a single API request to the Game Analytics Pipeline solution. Each event can be up to 1 MB in size. The API validates each event in the request body using the Game Event Schema. This operation proxies the events to Amazon Kinesis Data Streams using the `PutRecords` API. For information about size limitations imposed by the Kinesis Data Streams API, refer to PutRecords in the *Amazon Kinesis Data Streams Service API Reference*.

The `POST /applications/{applicationId}/events` response is modeled to provide similar response behavior to its integrated Kinesis Data Streams `PutRecords` API. This includes an array of response records with each record directly correlated to an event submitted in the request array using natural ordering from the top to the bottom of the request and response. The response `Events` array always includes the same number of records as the request `events` array.

The response `Events` array includes both successful and unsuccessful processed events. The solution API returns a `200 OK` response even if all records in the batch are not processed successfully. The `FailedRecordCount` provides a count of the total number of failed records in the batch. If this field is a non-zero integer, you must inspect the response `Events` array to determine which events in the batch must be retried.

All events in the request must adhere to the Game Event Schema, or the request will be rejected as a `BadRequest`.

The request to send events to the solution API must include a valid API key in the Authorization header, which is authorized to send events for the application.

Path parameters

`applicationId`

The unique application identifier associated with the input events.

Type: String

Required: Yes

Request body

The request accepts the following data in JSON format.

`events`

A batch of Game Event objects generated by a client.

Type: Array of Game Event objects (refer to the Game Event Schema for additional information)

Array members: Minimum number of one item. Maximum number of 500 items.

Required: Yes

Request headers

To send requests to the events endpoint, the request must include a valid API key that is authorized to submit events to the specified `{applicationId}` path parameter.

`Authorization`

The name of the header that contains a valid API key value.

Type: String

Required: Yes

Response

Name	Description
Total	Count of the total records that were processed in the request. This includes successful and failed records.
FailedRecordCount	Count of the total records that were unsuccessfully processed by the API. This field is passed through from the Kinesis Data Streams PutRecords API response.
Events	Array of successful and unsuccessful processed record response objects. A record that was successfully added to the stream returns `{ "Result": "Ok" }`. A record that was unsuccessfully processed returns `{ "Result": "Error", "ErrorCode": "" }` with an `ErrorCode` value returned by Kinesis Data Streams. For additional information on `PutRecords` API Errors, see PutRecords in the *Amazon Kinesis Data Streams Service API Reference*. Records with an error result is retried by the client.

Refer to the AWS CloudFormation stack, **Outputs** tab to retrieve the value of **ApiBasePath** and interact with the solution API.

Example request and response

Example request:

```
POST
https://<REST-API-ID>.execute-api.<aws-
region>.amazonaws.com/live/applications/<Provide-an-{applicationId}-
path-parameter>/events
Authorization: ApiKeyValue

{
   “events”: [Array of up to 500 Game Event Objects]
}
```

Example response:

```
{
 "Total": 3,
  "FailedRecordCount": 0,
  "Events": [
    {
      "Result": "Ok"
    },
    {
      "Result": "Ok"
    },
    {
      "Result": "Ok"
    }
  ]
}
```

Applications

Create and register an application

Description

Operation: `POST /applications`

This operation enables you to register a new application with the solution. This operation returns an `ApplicationId` value that uniquely identifies an application. An application must be registered with the solution before it can submit events to the solution.

This request to generate an application must be signed using the Signature Version 4 request signing process and AWS credentials.

Request body

The request accepts the following data in JSON format.

`Name`

Name of the application to register with the solution.

Type: String

Required: Yes

`Description`

A description for the application.

Type: String

Required: No

> **Note**: These are the only request body parameters recognized in this operation. Additional request body parameters are ignored. Multiple applications can be created with the same `Name`, but each registered application will have a unique `ApplicationId`.

Response

Name	Description
ApplicationId	The unique identifier for the application.
ApplicationName	The name of the application that was registered.
Description	(Optional) The description of the registered application, only returned if a description was provided when the application was created.
UpdatedAt	The date and time the application was last updated.
CreatedAt	The date and time the application was created/registered.

This operation must be performed by an administrator (for example, a backend developer or other technical user who operates the solution). This operation can be done manually using command line tools such as `curl` or an application such as Postman. This can also be integrated into existing game management tools and automated workflows.

To register a new application with the solution, use the solution API to send a `POST` request to the `/applications` endpoint. Refer to the AWS CloudFormation stack, **Outputs** tab and retrieve the value of **ApiBasePath** to interact with the solution API.

Example request and response

Example request:

```
POST
https://<REST-API-ID>.execute-api.<aws-
region>.amazonaws.com/live/applications

{
   “Name”: “string”
   “Description”: “string”
}
```

Example response:

```
{
    "ApplicationId": "d76d064f-ca8b-41ff-839f-4735e9a4b69d",
    "ApplicationName": "SampleGame",
    "Description": "A description for my game",
    "UpdatedAt": "2020-04-26T21:45:50Z",
    "CreatedAt": "2020-04-26T21:45:50Z"
}
```

The API response includes a unique `ApplicationId` value. This information is stored in the solution's Applications DynamoDB Table.

List applications

Description

Operation: `GET /applications`

This operation enables you to list the applications that are registered with the solution. The response returns a `Count` and an `Applications` array with application response objects.

Response

Name	Description
Count	The count of registered applications.
Applications	Array of registered application response objects. Each application response object includes the `ApplicationId`, `ApplicationName`, `CreatedAt`, `UpdatedAt`, and an optional `Description` if one is set for the application.

Refer to the AWS CloudFormation stack, **Outputs** tab and retrieve the value of **ApiBasePath** to interact with the solution API.

Example request and response

Example request:

```
GET
https://<REST-API-ID>.execute-api.<aws-
region>.amazonaws.com/live/applications
```

Example response:

```
{
    "Applications": [
      {
        "ApplicationId": "d76d064f-ca8b-41ff-839f-4735e9a4b69d",
        "ApplicationName": "SampleGame",
        "Description": "A description for my game",
        "UpdatedAt": "2020-04-26T21:45:50Z",
        "CreatedAt": "2020-04-26T21:45:50Z"
      }
    ],
    "Count": 1
}
```

Get Application details

Description

Operation: `GET /applications/{applicationId}`

This operation enables you to describe the details of a registered application.

Path parameters

`applicationId`

The unique application identifier.

Type: String

Required: Yes

Response

Name	Description
ApplicationId	The unique identifier for the application.
ApplicationName	The name of the registered application.
Description	(Optional) The description of the registered application, only returned if a description was provided when the application was created.
UpdatedAt	The date and time the application was last updated.

Name	Description
CreatedAt	The date and time the application was created/registered.

Refer to the AWS CloudFormation stack, **Outputs** and retrieve the value of **ApiBasePath** to interact with the solution API.

Example request and response

Example request:

```
GET
https://<REST-API-ID>.execute-api.<aws-
region>.amazonaws.com/live/applications/<provide-an-{applicationId}-
path-parameter>
```

Example response:

```
{
  "ApplicationId": "d76d064f-ca8b-41ff-839f-4735e9a4b69d",
  "ApplicationName": "SampleGame",
  "Description": "A description for my game",
  "UpdatedAt": "2020-04-26T21:45:50Z",
  "CreatedAt": "2020-04-26T21:45:50Z"
}
```

Delete an application

Description

Operation: `DELETE /applications/{applicationId}`

This operation enables you to delete a registered application.

> **Important:** Data that was ingested by deleted applications remains in Amazon Simple Storage Service (Amazon S3) after deletion, but new data cannot be submitted to the solution API after an application is deleted. When an application is deleted, all associated API key authorizations are also deleted.

Path parameters

`applicationId`

The unique application identifier.

Type: String

Required: Yes

Response

`Delete successful`

Refer to the AWS CloudFormation stack, **Outputs** tab and retrieve the value of **ApiBasePath** to interact with the solution API.

Example request and response

Example request:

```
DELETE
https://<REST-API-ID>.execute-api.<aws-region>.amazonaws.com/live/applications/<provide-an-{applicationId}-path-parameter>
```

Example response:

```
"Delete successful"
```

Authorizations

Create API key authorization for the application

Description

Operation: `POST /applications/{applicationId}/authorizations`

This operation enables you to generate a new API key that is authorized to submit events to a specified `{applicationId}` path parameter. This operation returns an `ApiKeyValue` field that should be added in the **Authorization** header when sending events to the solution API events endpoint.

Path parameters

`applicationId`

The unique application identifier.

Type: String

Required: Yes

Request body

The request accepts the following data in JSON format.

`Name`

A name for the API key that is being created.

Type: String

Required: Yes

`Description`

A description for the API key that is being created.

Type: String

Required: No

> **Note**: These are the only request body parameters recognized in this operation. Additional request body parameters are ignored. Multiple API keys can be created with the same `Name`, but each API key will have a unique `ApiKeyId`.

Response

Name	Description
ApiKeyId	The unique identifier for the API key authorization.
ApiKeyName	The name of the API key authorization that was created.
ApiKeyValue	The value of the API key. This value should be included in the Authorization header of requests from applications to send events to the events endpoint. **Note**: Manage this value as a shared secret and set applications to handle this value discreetly.
ApiKeyDescription	(Optional) The description of the API key authorization, only returned if a description was provided.
ApplicationId	The identifier of the application that the API key authorization is associated.
UpdatedAt	The date and time the API key authorization was last updated.
CreatedAt	The date and time the API key authorization was created/registered.
Enabled	Supported values are `true` and `false`. By default, all authorizations generated in the solution are enabled (`true`). If you would like to disable a key without deleting it, you can manually set this value to `false` in the Authorizations DynamoDB table.

Refer to the AWS CloudFormation stack, **Outputs** tab and retrieve the value of **ApiBasePath** to interact with the solution API.

Example request and response

Example request:

```
POST
```

```
https://<REST-API-ID>.execute-api.<aws-
region>.amazonaws.com/live/applications/<provide-an-{applicationId}-
path-parameter>/authorizations

{
   "Name": "MySampleKey"
   "Description": "A description describing my new API Key"
}
```

Example response:

```
{
    "ApiKeyId": "01af2cb3-8b1f-4bc0-801a-884a30fcb8cd",
    "ApiKeyValue": "2c9c0aece7a78cc35210dda2c4d43897",
    "ApiKeyName": "MySampleKey",
    "ApplicationId": "d76d064f-ca8b-41ff-839f-4735e9a4b69d",
    "ApiKeyDescription": "A description describing my new API Key",
    "CreatedAt": "2020-04-26T21:46:25Z",
    "UpdatedAt": "2020-04-26T21:46:25Z",
    "Enabled": true
}
```

List authorizations

Description

Operation: `GET /applications/{applicationId}/authorizations`

This operation enables you to list the API key authorizations associated with the application.

Path parameters

`applicationId`

The unique application identifier.

Type: String

Required: Yes

Response

Name	Description
Count	The count of API key authorizations.
Authorizations	Array of API key authorization response objects. Each authorization response object includes the `ApiKeyId`, `ApiKeyName`, `ApiKeyDescription` (optional, if set), `ApplicationId`, `CreatedAt`, `UpdatedAt`, and `Enabled`.

Example request and response

Example request:

```
GET
https://<REST-API-ID>.execute-api.<aws-
region>.amazonaws.com/live/applications/<provide-an-{applicationId}-
path-parameter>/authorizations
```

Example response:

```
{
    "Authorizations": [
      {
        "ApiKeyId": "01af2cb3-8b1f-4bc0-801a-884a30fcb8cd",
        "ApiKeyValue": "2c9c0aece7a78cc35210dda2c4d43897",
        "ApiKeyName": "MySampleKey",
        "ApplicationId": "d76d064f-ca8b-41ff-839f-4735e9a4b69d",
        "ApiKeyDescription": "A description describing my new API
Key",
        "CreatedAt": "2020-04-26T21:46:25Z",
        "UpdatedAt": "2020-04-26T21:46:25Z",
        "Enabled": true
      }
    ],
    "Count": 1
}
```

Get authorization details

Description

Operation: `GET /applications/{applicationId}/authorizations/{apiKeyId}`

This operation enables you to describe the details of an API key authorization.

Path parameters

`applicationId`

The unique application identifier.

Type: String

Required: Yes

`apiKeyId`

The unique identifier of the API key authorization.

Type: String

Required: Yes

Response

Name	Description
ApiKeyId	The unique identifier for the API key authorization.
ApiKeyName	The name of the API key authorization.
ApiKeyValue	The value of the API key. This value should be included in the Authorization header of requests from applications to send events to the events endpoint. **Note:** Manage this value as a shared secret and set applications to handle this value discreetly.
ApiKeyDescription	(optional) The description of the API key authorization, only returned if a description is set.
ApplicationId	The identifier of the application that the API key authorization is associated with.
UpdatedAt	The date and time the API key authorization was last updated.
CreatedAt	The date and time the API key authorization was created/registered.
Enabled	Supported values are `true` and `false`. By default, all authorizations generated in the solution are enabled (true). If you would like to disable a key without deleting it, you can manually set this value to false in the Authorizations DynamoDB table.

Example request and response:

Example request:

```
GET
https://<REST-API-ID>.execute-api.<aws-region>.amazonaws.com/live/applications/<provide-an-{applicationId}-path-parameter>/authorizations/<provide-an-{apiKeyId}-path-parameter>
```

Example response:

```
{
    "ApiKeyId": "01af2cb3-8b1f-4bc0-801a-884a30fcb8cd",
    "ApiKeyValue": "2c9c0aece7a78cc35210dda2c4d43897",
    "ApiKeyName": "MySampleKey",
    "ApplicationId": "d76d064f-ca8b-41ff-839f-4735e9a4b69d",
    "ApiKeyDescription": "A description describing my new API Key",
    "CreatedAt": "2020-04-26T21:46:25Z",
    "UpdatedAt": "2020-04-26T21:46:25Z",
    "Enabled": true
}
```

Delete authorization

Description

Operation: `DELETE /applications/{applicationId}/authorizations/{apiKeyId}`

The operation enables you to delete an API key associated with an application.

Path parameters

`applicationId`

The unique application identifier.

Type: String

Required: Yes

`apiKeyId`

The unique identifier of the API key authorization.

Type: String

Required: Yes

Response

`Delete successful`

Example request and response

Example request:

```
DELETE
https://<REST-API-ID>.execute-api.<aws-region>.amazonaws.com/live/
applications/<Provide an {applicationId} path
parameter>/authorizations/<Provide an {apiKeyId} path parameter>
```

Example response:

```
"Delete successful"
```

Modify authorization status

Description

Operation: `PUT /applications/{applicationId}/authorizations/{apiKeyId}`

This operation enables you to update the configuration of an API Key authorization. This operation currently supports updating the `Enabled` status to disable an API key without deleting it from the database.

> **Important**: API Gateway authorization caching is enabled in the solution API. It may take up to 300 seconds (5 minutes) before a change to the `Enabled` status of an API key is detected by the `LambdaAuthorizer` Lambda function. To reduce this time, you can modify or disable the Authorization Cache. Reducing or removing this cache TTL (time-to-live) results in additional queries to the `Authorizations` DynamoDB table and increases costs.

Path parameters

`applicationId`

The unique application identifier.

Type: String

Required: Yes

`apiKeyId`

The unique identifier of the API key authorization.

Type: String

Required: Yes

Body parameters

The request accepts the following data in JSON format.

`Enabled`

The status of the API key authorization.

Type: Boolean

Required: Yes

`Description`

A description for the API key.

Type: String

Required: No

> **Note**: These are the only body parameters recognized in this operation. Additional request body parameters are ignored.

Example request and response

Example request:

```
PUT
https://<REST-API-ID>.execute-api.<aws-region>.amazonaws.com/live/
applications/<provide-an-{applicationId}-path-
parameter>/authorizations/<provide-an-{apiKeyId}-path-parameter>

{
   "Enabled": Boolean
}
```

Example response:

```
{
    "Enabled": Boolean
}
```

Common API errors

This section lists potential API errors. Errors returned by the API include an HTTP Status Code as well as a response body that includes `error` and `error_detail`. These error response body fields provide additional details about the error.

`BadRequest`

The server could not process the request.

HTTP Status Code: 400

`AccessDenied`

The request was denied because access to the specified resource is forbidden or because invalid or missing credentials were provided in the request.

HTTP Status Code: 403

`NotFoundException`

The request processing has failed because of an unknown error, exception, or failure.

HTTP Status Code: 404

`InternalFailure`

The request processing has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

Game event schema

The Game Analytics Pipeline solution uses a JSON schema to validate and process telemetry data sent to the solution. For additional information on JSON schema, refer to json-schema.org.

JSON schema definition

The solution configures the following JSON schema to validate telemetry data sent to the solution API. It is also used by the `EventsProcessing` Lambda function to validate ingested data before it is transformed and loaded into Amazon Simple Storage Service (Amazon S3).

> **Note** The `application_id` field is not required when sending events using the solution API events endpoint because it is automatically set in the data record by the API using the `{applicationId}` input path parameter after the request has been authorized. Applications that integrate directly with Amazon Kinesis Data Streams must provide an `application_id` for each event that is submitted.

```
{
    "$schema": "http://json-schema.org/draft/2020-12/schema",
    "title": "Game Analytics JSON Event Schema",
    "description": "Format of events that are ingested to the AWS Game
Analytics Pipeline",
    "type": "object",
    "additionalProperties": false,
    "properties": {
      "event": {
        "$ref": "#/definitions/event"
      },
      "application_id": {
        "type": "string",
        "pattern":        "^[0-9a-f]{8}-[0-9a-f]{4}-[1-5][0-9a-f]{3}-
[89ab][0-9a-f]{3}-[0-9a-f]{12}$",
        "description": "The application identifier (UUID) this event
is associated with"
      }
    },
    "required": ["event", "application_id"],
    "definitions": {
     "event": {
```

```
        "type": "object",
        "additionalProperties": false,
        "properties": {
         "event_id": {
             "type": "string",
             "pattern":    "^[0-9a-f]{8}-[0-9a-f]{4}-[1-5][0-9a-f]{3}-
[89ab][0-9a-f]{3}-[0-9a-f]{12}$",
             "description":  "The  unique  identifier  for  the  event,
formatted as UUID v4 string."
         },
         "event_type": {
             "type": "string",
             "pattern": "^[A-Za-z0-9-_.]+$",
             "description": "Identifies the type of event"
         },
         "event_name": {
             "type": "string",
             "pattern": "^[A-Za-z0-9-_.]+$",
             "description": "Name of the event that occurred"
         },
         "event_timestamp": {
             "type": "number",
             "description": "The time in seconds since the Unix epoch
at which this event occurred (set by producer of event)."
         },
         "event_version": {
             "type": "string",
             "pattern": "^[A-Za-z0-9-_.]+$",
             "description": "An API version for this event format."
         },
         "event_data": {
             "type": "object"
         }
     },
     "required":    ["event_id",    "event_type",    "event_timestamp",
"event_name"]
     }
   }
}
```

Important: The solution's Game Event Schema uses the `additionalProperties` property in the JSON schema definition to enforce the top-level JSON fields that can be ingested. This schema is implemented to prevent new fields from being ingested into the solution's `raw_events` AWS Glue table before they have been defined in the AWS Glue table definition.

Game event data taxonomy

In order to derive value from the data generated within your game, we recommend developing a data taxonomy so that users can refer to a common definition of fields and their

purpose for analytics. A sample game event data taxonomy that defines event types and data fields is generated by the solution's sample demo script.

Event type dictionary

The following table contains a list of example `event_types` that are used in the solution's prebuilt sample streaming and interactive SQL queries.

event_type	Category	Producer	Description	Included fields
login	Core Platform	Client	User logs into their account in the client.	platform, last_login_time
logout	Core Platform	Client	User logs out of their account in the client.	
user_registration	Core Platform	Client	User registers for an account in the client.	country_id, platform
user_report	Core Platform	Client	User has reported another user's bad behavior.	report_id, reported_user, report_reason
user_sentiment	Core Platform	Client	User has taken a survey about their sentiment towards the game.	user _rating
tutorial_progression	Game Progression	Client	User makes it through sections of the tutorial.	tutorial_screen_id, tutorial_screen_version
user_rank_up	Game Progression	Platform Server	User ranked up due to performance.	user_rank_reached
level_started	Game Progression	Client	User has started a level in the game.	level_id, level_version
level_completed	Game Progression	Client	User has successfully completed a level in the game.	level_id, level_version
level_failed	Game Progression	Client	User has failed to complete a level in the game.	level_id, level_version
matchmaking_start	Match Progression	Platform Server	Matchmaking process starts.	match_id, match_type
matchmaking_complete	Match Progression	Platform Server	Matchmaking has successfully completed.	match_id, match_type, matched_slots

event_type	Category	Producer	Description	Included fields
matchmaking_failed	Match Progression	Platform Server	Matchmaking has failed.	match_id, match_type, matched_slots, matching_failed_msg
match_start	Match Progression	Platform Server	Game session has started - reported for each user.	match_id, map_id, client_latency
match_end	Match Progression	Platform Server	Match has ended - reported for each user.	match_id, map_id, game_result_type, exp_gained
user_knockout	Match Progression	Platform Server	User has knocked out another user in the match.	match_id, map_id, spell_id, exp_gained
item_viewed	Monetization	Client	User has viewed an item in the store.	item_id, item_version
iap_transaction	Monetization	Platform Server	User has purchase an item in the in game store.	item_id, item_version, item_amount, currency_type, currency_amount, transaction_id
lootbox_opened	Monetization	Client	User has opened a lootbox.	lootbox_id, lootbox_cost, item_rarity, item_id, item_version, item_cost

Data field dictionary

The following table contains a list of example data fields with their descriptions and data types that are used in the solution's queries and reports. This list represents a superset of the fields described in the Game Event Schema. Unlike the Game Event Schema, which represents a schema definition for the format of input events, a data field dictionary can be used to help you plan your overall data management strategy and also contains output fields generated during processing.

Name	Type	Definition
event_timestamp	number	The time in seconds since the Unix epoch at which this event occurred, set by the producer of event.
application_name	string	The name of the data application, set by the event's processing function.
application_id	string	The unique identifier of the registered application that generated the event. This is a random UUID.
event_id	string	A random UUID that uniquely identifies this event.
event_type	string	The type of event being transmitted, which allows categorization of common events within a type. The solution's sample data producer script sets this value equal to the `event_name`.

Name	Type	Definition
event_name	string	The name of the event. The sample script uses the same value for `event_name` and `event_type` fields.
event_version	string	The API version for this event format.
client_latency	number	The latency of the client to the game server in milliseconds.
last_login_time	number	The timestamp of the last time the user logged in, represented as seconds since Unix epoch.
tutorial_screen_id	string	The name of the screen in the tutorial that the user has completed.
tutorial_screen_version	number	The version of the screen in the tutorial.
country_id	string	The country code of the user registering to play the game (USA, RUS, CHN, etc.).
platform	string	The game platform of the user registering to play the game (PC, IOS, PS4, etc.).
user_rank_reached	string	The level or rank the user has achieved.
match_id	string	A random UUID that uniquely identifies this match.
match_type	string	The type of match being played such as battle royal, team deathmatch, and 1v1.
matched_slots	number	The number of slots matched at the end of the matchmaking session.
matching_failed_msg	string	For failed matchmaking sessions, this is the reason - matchmaking timed out, users quit, etc.
map_id	string	The name of the map being played on in the game session.
clan_id	string	A UUID that uniquely defines a collection of users as part of a clan.
match_result_type	string	The match result (win, lose, placement (top 5 or top 10)) - determined by match type.
exp_gained	number	The amount of experience the user gained whether it be at the end of the match or from a single knockout.
most_used_spell	string	The spell that was used for the most amount of time during the session by the user.
spell_id	string	The name of the spell used.
item_id	string	A UUID that identifies an item in the in-game shop.
item_version	number	The version of the item.
item_amount	number	The amount of a product being purchased, consumed, or retained.
transaction_id	string	A random UUID that uniquely identifies the in-game transaction.
currency_type	string	The type of currency being used in the transaction (USA, CAD, EUR, etc.).
currency_amount	number	The amount of currency used in the transaction.
level_id	string	The ID of the level that is being played.

Name	Type	Definition
level_version	number	The version of the level being played.
lootbox_id	string	A UUID that identifies a lootbox owned by a user.
lootbox_cost	number	The monetary value of the lootbox.
item_rarity	string	The rarity of the item opened in the lootbox (common, uncommon, rare, legendary, etc.).
item_cost	number	The cost of an item opened in a lootbox.
report_id	string	A UUID that identifies a bad user behavior report.
reported_user	string	A UUID that uniquely identifies the user who reportedly exhibited bad behavior.
report_reason	string	The bad behavior that is being reported (griefing, cheating, AFK, racism/harassment, etc.).
user_ rating	number	The rating provided by the user when asked their satisfaction with the game (such as, how are you enjoying the game, rating: 1 - 5).

Use the solution API to ingest game events

Clients that require custom RESTful integration, or are unable to integrate directly with the Kinesis data stream, can use the solution API events endpoint. The events endpoint is configured with an API Gateway Lambda authorizer to authorize requests to the endpoint using API keys. The following sections describe technical implementation and use details for the solution API.

Overview of the integration with Kinesis Data Streams

The events endpoint is configured with an AWS Service backend integration to proxy events to the Kinesis data stream. The API transforms data into the correct format for Amazon Kinesis Data Streams using a request mapping template. The `{application_id}` path parameter is extracted during request transformation and automatically set in the record before it is sent to the Kinesis data stream.

> **Important**: The solution API requires than an `event_id` field is included for all events sent to the solution as specified in the Game Event Schema. The API request mapping template uses this field as the Partition Key when sending data records to the Kinesis data stream to provide uniform data distribution across the stream shards by event. The `event_id` field is set by the producer application using any value that uniquely identifies the generated event.

Configure applications to send data to the solution

The solution API enables administrators to programmatically register new applications with the solution and generate API keys that can be distributed to developers for use in their game applications to publish events to the events endpoint. The solution uses the `ApplicationAdminServiceFunction` AWS Lambda function to store and retrieve application configuration data from Amazon DynamoDB tables. The solution API is used interactively by users to either configure the solution or integrate into automated tools or scripts.

AWS Identity and Access Management (IAM) is used to authenticate and authorize requests to these solution administrative endpoints. This enables users to rely on the same authentication solution for managing the API and AWS resources deployed by the solution. Backend applications can securely interact with the solution API using temporary AWS credentials from IAM roles. Use the following steps to create an application, generate API keys, and submit game events to the solution API.

Step 1. Create an application

Use the solution API to register your game with the solution (refer to Create and Register an Application). This operation generates a unique application identifier that is included with each event that is sent to the solution.

Step 2. Generate API keys

Note: This step can be skipped if you do not intend to ingest data into the solution using the solution API events endpoint. For example, if you plan to integrate your game directly with Amazon Kinesis Data Streams.

If you plan to send events to the solution API events endpoint, first generate one or more API keys that are required by the API to authorize requests from clients. To generate a key, send a POST request to the authorizations endpoint. This request generates an API key value that sends events to the specified application events endpoint. Repeat this process if multiple keys are required. Requests to generate API keys must be authenticated with AWS IAM credentials.

Note: The solution does not impose restrictions on the number of keys that can be created per application. While API keys are suitable for access control when integrated with an HTTPS endpoint, do not use them to identify users or devices, or as a form of user authentication. The Lambda authorizer provided with the solution can be integrated with identity providers such as Amazon Cognito to perform bearer token authentication schemes (such as JWT) if authentication is required.

Step 3. Submit game events

After you generate an API key for use with an application, you can send events to the solution API. Events are submitted to the events endpoint using a POST Request to the Solution Events Endpoint with an `{applicationId}` path parameter. The `Authorization` header must include a valid `apiKeyValue` (which is authorized to send telemetry data to the specified `{applicationId}`) or the request will be denied.

Streaming ingestion details

The Game Analytics Pipeline solution implements a serverless streaming data ingestion architecture for processing event telemetry data from games. This section includes technical implementation details describing how events are processed after they are ingested into the solution. Figure 1 describes the streaming data ingestion architecture.

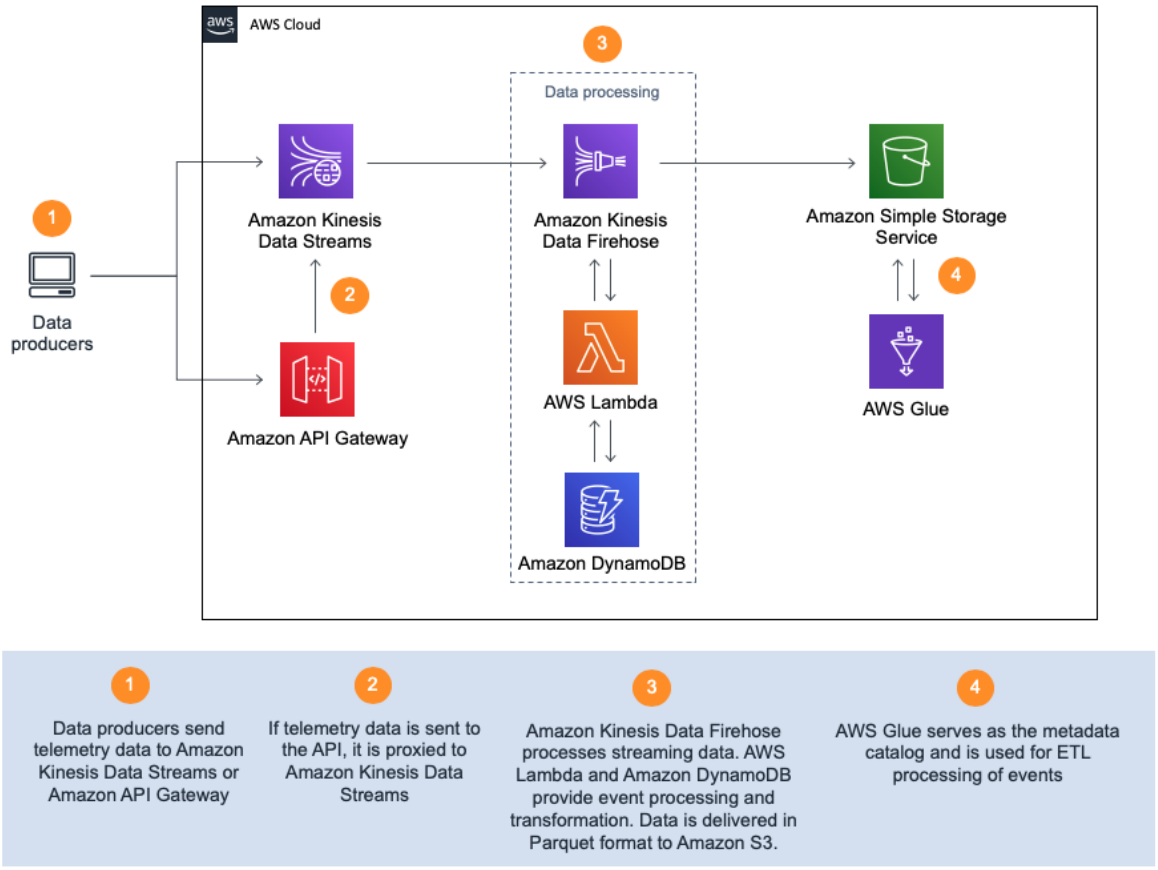

Figure 1: Streaming data ingestion architecture

Telemetry events are either sent to the solution's Amazon Kinesis Data Streams (events stream) or ingested through the solution API, which automatically forwards events to the stream. Amazon Kinesis Data Firehose reads and buffers the records (events) into preconfigured batch sizes (either every 3 MB of ingest data or every 60 seconds with the solution's default settings) and submits them to an AWS Lambda Data Transformation function (`EventsProcessingFunction`) for preprocessing before the events are delivered to Amazon Simple Storage Service (Amazon S3) for storage. AWS Glue is used for data cataloging and ETL processing workflows, as described in AWS Glue ETL Workflow.

Events processing function

The Events processing function validates each input event against a preconfigured Game Event Schema (deployed with the function code) to help enforce data hygiene and sanitize raw events. This process also enables faster insights by loading preprocessed Apache Parquet formatted data into Amazon S3 for immediate querying.

Event input format

Each data record sent to Kinesis Data Streams must be valid JSON and is expected to adhere to the following format, which is also the format that events are transformed into by Amazon API Gateway.

```
{
  "event": {},
  "application_id": "string"
}
```

Each data record (event) should contain an object including the following fields:

- The `event` field maps to the event field of the solution's Game Event Schema.
- The `application_id` field is used to identify the registered application that generated the event. Each record sent to the pipeline is validated for an `application_id` field that matches a registered entry in the `Applications` Amazon DynamoDB table.

Event enrichment and metadata

The function enriches events with processing metadata to provide additional context related to the ingestion and processing of the event before they are loaded into Amazon S3. This metadata object includes the following properties, and is loaded as a JSON-encoded string when Kinesis Data Firehose performs record format conversion to Apache Parquet.

`ingestion_id`

A unique identifier to associate the event with the AWS Lambda invocation that processed the event. This field is set using the AWS Lambda `awsRequestId` field provided in the AWS Lambda context object during runtime. All events processed by the same Lambda invocation have the same value. This value can assist with troubleshooting and help with correlation of events processed in the same batch.

Type: String

JSON Path: `$.metadata.ingestion_id`

`processing_timestamp`

An Epoch-formatted timestamp (Unix timestamp in seconds) generated by the `EventsProcessingFunction` Lambda function when the event is processed.

Type: String

JSON Path: `$.metadata.processing_timestamp`

`status`

The processing result generated by the `EventsProcessingFunction` Lambda function. Valid values include: `ok`, `schema_mismatch`, and `unregistered`.

Type: String

JSON Path: `$.metadata.processing_result.status`

`request_id`

The identifier that API Gateway assigns to the API request. This value is only set if the event was ingested using the solution REST API events endpoint. The API Gateway value—`$context.requestId`—is used.

Type: String

JSON Path: `$.metadata.api.request_id`

`request_time_epoch`

The Epoch-formatted timestamp (Unix timestamp in seconds) generated by API Gateway. This value is only set if the event was ingested using the solution API. The API Gateway value—`$context. request_time_epoch`—is used.

Type: Number

JSON Path: `$.metadata.api.request_time_epoch`

`application_name`

The application name defined in the `Applications` DynamoDB table. If an event is registered within the `Applications` DynamoDB table, then the function inserts the application name into the event, otherwise this field is empty.

Type: String

JSON Path: `$.application_name`

Handling invalid events

The `EventsProcessingFunction` Lambda function validates each event to ensure that the event is associated with a registered application, and that it conforms to the game events schema format in the following ways:

- **Unregistered events:** Events that do not contain a valid `application_id` as defined in the applications DynamoDB table are treated as unregistered events and are set with a status of `unregistered` in the processing metadata of the event.
- **Schema mismatch:** Events that are registered but do not contain a valid event schema are set with a status of `schema_mismatch` in the processing metadata. If an event has a malformed schema, these errors are included in the `validation_errors` field of the processing metadata. Only the fields defined in the event schema are extracted during event processing; any additional fields are ignored and are not loaded into S3 storage.

Due to the highly cacheable nature of the `Applications` DynamoDB query results, the `EventsProcessingFunction` Lambda function stores a built-in cache of `Applications` query results from the `Applications` DynamoDB table and stores it locally into the Lambda function. This cache is refreshed automatically every 60 seconds (which is configurable) by the Lambda function and persists between Lambda invocations to reduce the number of queries to DynamoDB.

Event output format

Events are validated and transformed into the following format before they are returned to Kinesis Data Firehose for delivery to Amazon S3:

```
{
  "event_id": "string",
  "event_type": "string",
  "event_name": "string",
  "event_version": "string",
  "event_timestamp”: number,
  "event_data": {},
  "application_id": "string",
```

```
    "application_name": "string",
    "metadata": {
      "ingestion_id": "string",
      "processing_timestamp": number
      "processing_result": {
        "status": "ok|schema_mismatch|unregistered",
        "validation_errors": array (optional if errors)
      },
      "api": {
        "request_id": "string",
        "request_time_epoch": number
      }
    }
  }
```

Ingesting unstructured event data

The Game Event Schema supports ingesting unstructured JSON data that may be unique to an event, which provides flexibility to ingest data from a variety of applications. The Game Event Schema supports sending unstructured data in the `event_data` field, which is not validated during events processing (except to validate that the input is an object). The `event_data` field can be used to include custom JSON data with your game event that is loaded as a JSON-encoded string by Kinesis Data Firehose during the Apache Parquet record format transformation that occurs before data is loaded into Amazon S3 for storage.

> **Note**: The Game Event Schema uses the `additionalProperties` property in the JSON schema definition to restrict the top-level JSON fields that can be ingested into the solution. This approach is implemented to prevent the ingestion of new fields into the `raw_events` AWS Glue table before they have been defined in the AWS Glue table definition. This approach provides developers the control over the structure of the raw events before they are loaded into Amazon S3, and enables Kinesis Data Firehose to automatically transform data into Apache Parquet format with reduced post-processing.

Amazon Athena provides support for accessing and querying JSON-encoded data in Amazon S3. For more information about best practices for querying and analyzing JSON data, refer to Querying JSON in the *Amazon Athena User Guide*. The sample Athena queries provide examples that demonstrate how to extract custom fields from Game event schema `event_data` property.

Error handling and troubleshooting

The streaming data ingestion provides automatic error handling and retry of failed records, and key solution metrics can be monitored using the Amazon CloudWatch Operational Health Dashboard. If AWS Lambda is unable to transform data or Kinesis Data Firehose encounters errors (invalid JSON), the records are retried before they are delivered to the

`firehose-errors/processing-failed/` prefix in the `analytics` S3 bucket in their raw format.

Amazon CloudWatch metrics and logs provide tracking for Amazon S3 delivery failures and AWS Lambda processing errors. The solution template can also be customized to configure Kinesis Data Firehose to send a backup of raw data to Amazon S3 by setting the **S3BackupMode** mapping to `Enabled` in the AWS CloudFormation template.

For additional information on failure handling, refer to Amazon Kinesis Data Firehose Data Delivery in the *Amazon Kinesis Data Firehose Developer Guide*.

Amazon S3 storage configuration

The solution is designed to optimize storage for cost savings and performance. AWS Glue Data Catalog stores the metadata and schema information. Amazon S3 stores the game events. The solution deploys the following S3 buckets:

- The `Analytics` bucket stores solution data and uses the following prefixes:
  - **raw_events:** This prefix contains all of the raw events delivered by Kinesis Data Firehose.
  - **firehose-errors:** This prefix contains all error records that could not be processed and transformed by Kinesis Data Firehose.
  - **glue-scripts:** This prefix contains the source code referenced and used by the AWS Glue ETL job.
  - **glueetl-tmp:** This prefix contains temporary artifacts generated by the AWS Glue ETL job.
  - **athena_query_results:** This prefix is used to store query outputs for the Athena workgroup deployed with the solution.
  - **processed_events**: This prefix is generated by the AWS Glue ETL job for storing processed events which are partitioned by the `ApplicationId` property and the ingestion time.
- The `solutionlogsbucket` S3 bucket contains the Amazon S3 access logs generated by the `Analytics` S3 bucket and can be used to analyze bucket usage and to perform troubleshooting. This S3 bucket is configured with a lifecycle policy to transition data to Amazon S3 Standard-Infrequent Access (S3 Standard-IA) 30 days after it is initially stored in the bucket.

Data delivery format and partitioning

Amazon Kinesis Data Firehose is configured to deliver data to Amazon S3 in Apache Parquet format with SNAPPY compression, a popular columnar storage format. The solution deploys a `raw_events` table in the AWS Glue Data Catalog that Kinesis Data Firehose uses to provide automatic record format conversion to Apache Parquet when loading data to Amazon S3.

The solution also automatically partitions data in Amazon S3 using Apache Hive-compatible date-based Amazon S3 prefixes with server-side timestamps generated by Kinesis Data Firehose using daily partitions. This partition structure improves query performance for queries that include dates in `WHERE` clauses, which is common in time-series data analysis. When these types of queries are issued against the data lake using compatible tools, only the Amazon S3 folder prefixes containing the relevant data ranges are accessed, which reduces query costs and improves performance.

Data is delivered to Amazon S3 in the following format:

```
/raw_events/year=<YYYY>/month=<MM>/day=<DD>/<filename>.parquet
```

When you deploy the AWS CloudFormation template, the `GluePartitionCreator` Lambda function automatically creates new daily partitions in the `raw_events` AWS Glue table. This Lambda function automatically runs hourly and creates a new partition in the AWS Glue table for the current UTC date if one does not already exist.

Automated S3 object lifecycle management

Amazon S3 configures object lifecycle management for data. The following Amazon S3 Lifecycle policies are used by the solution, and can be customized within the AWS CloudFormation template:

- The `Analytics` S3 bucket is configured to transition data in the `raw_events` and `processed_events` prefixes to Amazon S3 Intelligent-Tiering (S3 Intelligent-Tiering) after seven days in order to provide cost savings for datasets with unknown or changing access patterns, such as data lakes.
- The `solutionlogsbucket` S3 bucket is configured with a lifecycle policy to transition data to S3 Standard-IA after 30 days.

> **Important**: S3 Intelligent-Tiering charges a small tiering fee and has a minimum eligible object size of 128 KB for auto-tiering. Smaller objects can be stored but will always be charged at the Amazon S3 Standard tier rates.

The default Kinesis Data Firehose buffer settings are optimized to generate objects in Amazon S3 that are large enough to realize the cost savings from S3 Intelligent-Tiering.

The solution does not implement any data expiration actions in its lifecycle policies. To setup data expiration or other custom Amazon S3 Lifecycle rules, modify the Amazon S3 Lifecycle configuration for the S3 buckets.

AWS Glue ETL workflow

The solution deploys an AWS Glue workflow to orchestrate batch processing of event data in Amazon S3. By default, the workflow uses an AWS Glue ETL job to re-organize raw data into Amazon S3 prefixes based on the game's `ApplicationId`. This workflow enables data from separate applications to be repartitioned into separate physical locations in Amazon S3 for easier data management and access control. The ETL workflow can also be modified to address additional ETL use cases as needed.

The AWS Glue workflow is made up of the following components:

- **AWS Glue Trigger:** An on-demand trigger to start an AWS Glue ETL job. This on-demand trigger can be converted to a recurring schedule if desired.
- **AWS Glue ETL Job:** Configures an Apache Spark Python ETL job to read data from the `raw_events` AWS Glue table in the AWS Glue Data Catalog and repartitions event data into the following partitioned format in the Amazon S3 bucket: `/processed_events/application_id=<application_id>/year=<YYYY>/month=<MM>/day=<DD>/<filename>.parquet`
- **AWS Glue Crawler**: Triggers when the ETL job completes and detects new partitions or table definition changes generated as a result of the ETL job. The crawler creates (or updates) a table named `partitioned_events` in the `gameeventsdatabase` AWS Glue database with the updated table definition and partition format.

The AWS Glue ETL job enables AWS Glue job bookmarks to prevent the reprocessing of S3 objects that have already been processed.

Amazon CloudWatch Events (Amazon EventBridge) is configured to detect job and crawler status change events and to generate notifications. These notifications are delivered to the **Notifications** Amazon Simple Notification Service (Amazon SNS) topic.

Integrate with Amazon Pinpoint

Amazon Pinpoint provides a multi-channel user engagement and analytics solution that includes client SDK integration and automatic instrumentation to capture common types of events, including those commonly found in games. If you require multi-channel user

engagement features, or need a prepackaged client SDK and instrumentation for your game, you can integrate with Amazon Pinpoint, which can forward events to Amazon Kinesis Data Stream for processing and storage.

The `EventsProcessingFunction` Lambda function must be customized to process events that arrive in formats that do not match the Game Event Schema, including events that originate from Amazon Pinpoint.

Note: Additional costs apply when using Amazon Pinpoint. Refer to the Amazon Pinpoint pricing page for more information.

Customize the Kinesis Data analytics application

The Game Analytics Pipeline solution provides a real-time streaming analytics application that allows developers to generate custom metrics and key performance indicators (KPIs) from raw application event data, and filter events using custom SQL. This application tracks usage behavior and aggregates metrics to power live dashboards. This application consists of an Amazon Kinesis Data Analytics for SQL Applications to process Kinesis stream data and an `AnalyticsProcessingFunction` AWS Lambda function to process analytics application outputs. The `AnalyticsProcessingFunction` Lambda function publishes metrics to Amazon CloudWatch for metrics storage and monitoring, which is integrated with Amazon Simple Notification Service (Amazon SNS) for notifications and alerts.

To use this feature, the `EnableStreamingAnalytics` AWS CloudFormation parameter must be set to **Yes** when deploying the solution, which is the default setting.

Amazon Kinesis Data Analytics for SQL

Amazon Kinesis Data Analytics for SQL uses SQL queries to generate custom metrics over a preconfigured interval, which can be customized. The streaming SQL results are continuously inserted into an application destination table—`DESTINATION_STREAM`—for further processing by the `AnalyticsProcessingFunction` Lambda function. If the solution is deployed into **Dev** mode, the application processes metrics over one minute windows, otherwise **Prod** mode is configured to process metrics over five minute windows.

Note: The KinesisAnalyticsApp `Interval` parameter can be modified in AWS CloudFormation to specify a custom interval.

To customize or add additional SQL queries, refer to the instructions in the Customize the streaming SQL queries section.

AnalyticsProcessingFunction Lambda Function

The `AnalyticsProcessingFunction` Lambda function processes metric outputs generated by the Amazon Kinesis Data Analytics application and publishes to Amazon CloudWatch. This Lambda function is configured to process data from the `DESTINATION_STREAM` Kinesis Data Analytics output table. Each time window in the application generates a batch of output metrics that are sent to AWS Lambda as an array of JSON objects, with each object representing a generated metric to be processed.

Output fields in the metric that are prefixed with `DIMENSION_` are treated by this Lambda function as a CloudWatch metric dimension when publishing the metrics to Amazon CloudWatch. This enables the Lambda function to process metrics from multiple games and applications and dynamically generate dimensions based on the SQL query outputs. During processing, the Lambda function strips the dimension prefix from the dimension name before it publishes the metric. Refer to the GitHub repository to modify this Lambda function as needed. By default, metrics are published to CloudWatch with five minute granularity.

Metrics processed by this Lambda function are published to the following Amazon CloudWatch metrics namespace:

```
<cloudformation-stack-name>/ AWSGameAnalytics
```

Error handling

This Lambda function processes destination outputs from the Kinesis Data Analytics application `error_stream`, which contains errors generated by the Kinesis Data Analytics application. If error records are received by this Lambda function from Kinesis Data Analytics, they are logged to the CloudWatch Logs group associated with the `AnalyticsProcessingFunction` Lambda function, and can be further processed within CloudWatch. AWS Lambda marks these records as `ok` in the response and logs a count of `KinesisAnalyticsErrors`. A CloudWatch metric filter tracks the value of `KinesisAnalyticsErrors` and generates a CloudWatch alarm when any errors are detected within a five minute period. For additional information, refer to Using a Lambda Function as Output in the *Amazon Kinesis Data Analytics for SQL Applications Developer Guide*.

Customize the streaming SQL queries

By default, the solution configures the Kinesis Data Analytics application to generate custom metrics from the input streaming data. The outputs are recorded into a `DESTINATION_STREAM` table, and processed by the `AnalyticsProcessingFunction` Lambda function. The following is an example `DESTINATION_STREAM` table.

```
CREATE STREAM "DESTINATION_STREAM"(
METRIC_NAME VARCHAR(1024),
METRIC_TIMESTAMP BIGINT,
METRIC_UNIT_VALUE_INT BIGINT,
METRIC_UNIT VARCHAR(1024),
DIMENSION_APPLICATION_ID VARCHAR(1024),
DIMENSION_APP_VERSION VARCHAR(1024),
DIMENSION_COUNTRY_ID VARCHAR(1024),
DIMENSION_CURRENCY_TYPE VARCHAR (1024),
DIMENSION_SPELL_ID VARCHAR (1024),
DIMENSION_MISSION_ID VARCHAR (1024),
DIMENSION_ITEM_ID VARCHAR (1024),
OUTPUT_TYPE VARCHAR(1024));
```

Modifying the analytics application schema

You can customize the real-time streaming analytics application to meet your requirements. You can either update the queries and schema in the AWS CloudFormation template or modify the SQL queries in the AWS Management Console and manually redeploy changes.

Use this procedure to modify your analytics application schema from the AWS Management Console to capture different data fields that can be ingested in to the solution.

1. Navigate to the Amazon Kinesis Data Analytics console.
2. Select your application to expand the details, then choose **Application Details**.
3. Under **Source**, choose the edit icon to edit the streaming data source connection.
4. Choose **Edit Schema**.
5. Choose **Add column** and then insert the column name, column type, length (if applicable), and row path for the data field you want to add to input.
   - To extract a top-level JSON event field, your row path will look like `$.event.<field-name>`
   - To extract a field nested within `event_data`, your row path will look like `$.event.event_data.<field-name>`
6. Choose **Save schema and update stream samples**.

The new stream sample includes the data fields that you have added and queries can now access these new fields.

Example real-time analytics queries

The following are metrics generated by the Kinesis Data Analytics application. In order to use these queries, your application must send events that contain the appropriate fields.

Real-time metrics

`TotalEvents`

This query retrieves a total count of events made per application and application version within the time window. It achieves this by performing a count function on all unique `event_id` values. It is used for checking overall application functionality and activity rates.

Event Types Used: All

Fields Required: not applicable

`TotalLogins`

This query retrieves a total count of users who logged into the application and application version in the time window. It achieves this by performing a count function of all events of the login event type. It is used for checking trends for logins, or potential user connectivity errors.

Event Types Used: `login`

Fields Required: not applicable

`KnockoutsBySpell`

This query retrieves a total number of knockouts by each spell that is used during the game, within the time window. It is used for tracking popularity or efficiency of spells for potential balancing.

Event Types Used: `user_knockout`

Fields Required: `spell_id`

`Purchases`

This query retrieves the total count of purchases for each currency type, for the time window. It achieves this by performing a count function on events of the `iap_transaction` event type, and group by the `currency_type` field. This is to track purchasing demographics.

Event Types Used: `iap_transaction`

Fields Required: `currency_type`

Explore the sample athena queries

When the solution is deployed in **Dev** mode, Amazon Athena queries are created to measure typical games industry KPIs. These queries utilize the event types and data fields found in the data taxonomy and are deployed with the AWS CloudFormation template. Use the Amazon Athena console to explore these queries.

The following queries, including the event types they are associated with and the fields from the AWS Glue Data Catalog that they require, are deployed with the solution.

`TotalIapTransactionsLastMonth`

Event Types Used: `iap_transaction`

Data Fields Used: `$.event_data.transaction_id`

`NewUsersLastMonth`

Event Types Used: `user_registration`

Data Fields Used: not applicable

`TotalPlaysByLevel`

Event Types Used: `level_started`

Data Fields Used: `$.event_data.level_id`

`TotalFailuresByLevel`

Event Types Used: `level_failed`

Data Fields Used: `$.event_data.level_id`

`TotalCompletionsByLevel`

Event Types Used: `level_completed`

Data Fields Used: `$.event_data.level_id`

`LevelCompletionRate`

Event Types Used: `level_started, level_completed`

Data Fields Used: `$.event_data.level_id`

`AverageUserSentimentPerDay`

Event Types Used: `user_sentiment`

Data Fields Used: `$.event_data.user_rating`

`UserReportedReasonsCount`

Event Types Used: user_report

Data Fields Used: `$.event_data.report_reason`

Use Postman with the solution API

Postman is a client application that makes it easy to manage and interact with REST APIs by providing a user interface. After you install Postman, you can configure the solution API resources and methods through the Postman user interface.

Use the following procedure to create a solution API AWS Identity and Access Management (IAM) policy and user and set up Postman. For detailed instructions, follow the links for each step.

Step 1. Create an IAM policy

Step 2. Create an IAM user

Step 3. Create a Postman environment

Step 4. Create a Postman collection for the solution API

Step 5. Test the solution API

Step 1. Create an IAM policy

To use the solution API application and authorization endpoints, you must create an IAM user that has permissions to invoke the API endpoints. In this section, you create a policy that provides users with the ability to interact with the endpoints, and then create a user account and attach the policy directly to the user account.

1. Sign in to the IAM console.
2. In the navigation pane, choose **Policies**, and then choose **Create policy**.
3. On the **JSON** tab, use the following code.

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

```
            "Sid": "SolutionAdminExecuteApi",
            "Effect": "Allow",
            "Action": [
                "execute-api:Invoke"
            ],
            "Resource": [
                "arn:aws:execute-api:<aws-region>:<account-id>:<api-
id>/live/*"
            ]
        }
    ]
}
```

Replace *<aws-region>*, *<account-id>*, and *<api-id>* with the values from the API that was deployed by AWS CloudFormation. This gives the user permissions to invoke all of the resources in the solution API. You can scope these permissions further to manage specific applications and authorizations.

4. Choose **Review policy**.
5. In the **Name** field, enter a name for the policy (for example, `GameAnalyticsPostmanAdminPolicy`).
6. Choose **Create policy**.

Step 2. Create an IAM user

After you create the policy, create an IAM user and attach the policy to it. When you create the user, IAM provides you with a set of credentials that you can use to allow Postman to execute operations against the solution API.

1. Sign in to the IAM console.
2. In the navigation pane, choose **Users**, and then choose **Add user**.
7. Under **Set user details**, **User name**, enter a name that identifies the user account (for example, `GameAnalyticsPostmanAdminUser`).
8. Under **Select AWS access type**, **Access type**, choose **Programmatic access**. Then choose **Next: Permissions**.
9. Under **Set permissions**, choose **Attach existing policies directly**. In the list of policies, choose the policy that you created earlier. Then choose **Next: Tags**.
10. On the **Add tags** page, optionally add tags that help you identify the user. For more information about using tags, see Tagging IAM Users and Roles in the *IAM User Guide*. Then choose **Next: Review**.

11. On the **Review** page, review and confirm the settings for the user.
12. Choose **Create user**.
13. On the **Success** page, record the credentials that are shown in the **Access key ID** and **Secret access key** columns.

> **IMPORTANT**: Record both the access key ID and the secret access key and save them in a secure location for use later in this tutorial. This is the only time that you are able to view the secret access key.

Step 3. Create a Postman environment

You can create one or more environments in Postman. After the environment is created, you can import a collection that contains a request template for each of the operations in the solution API.

In Postman, an environment is a set of variables that are stored as key-value pairs. You can use environments to quickly change the configuration of the requests that you make through Postman without having to change the API requests.

Use this procedure to create an environment for your deployment of the Game Analytics Pipeline solution. You can create an environment for each deployment of the solution that you manage.

1. In Postman, on the **File** menu, choose **New**.
2. In the **Create New** window, choose **Environment**.
3. On the **MANAGE ENVIRONMENTS** window, for **Environment Name,** enter the name of the CloudFormation Stack that you deployed or another value (ex. Game Analytics Stack). Enter the following variable values:

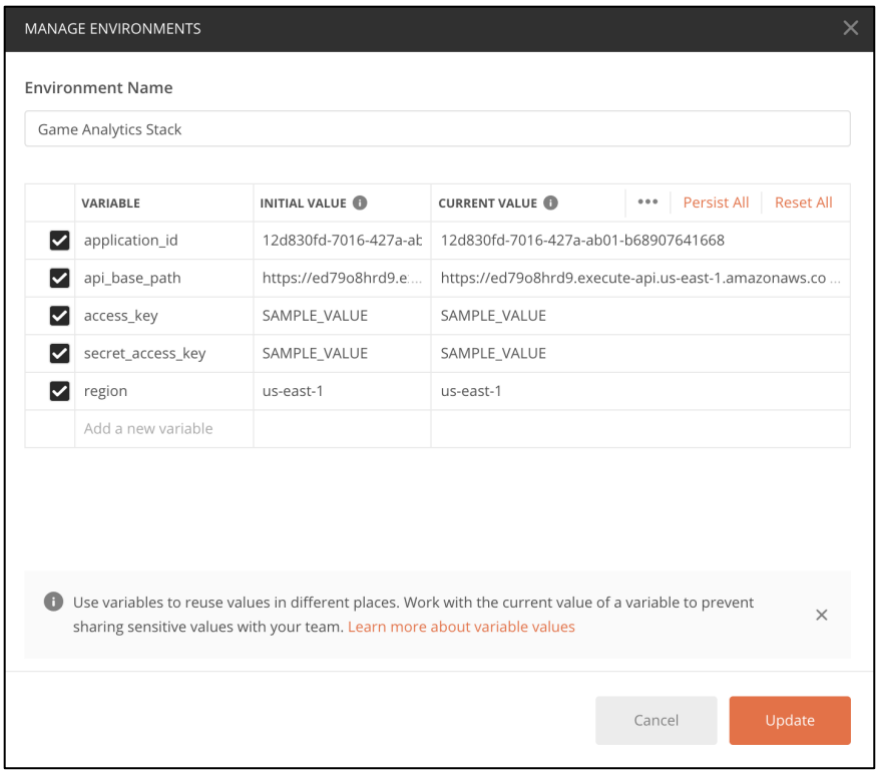

Figure 2: Configure Postman environment

4. Choose **Update** (or **Add**).

> **Important**: Postman includes features that enable you to share and export environments. If you use these features, be careful to **not** share your access key ID and secret access key with others that do not require access to these credentials.
>
> For more information, refer to IAM Best Practices in the *IAM User Guide*.

Step 4. Create a Postman collection for the solution API

In Postman, a collection is a group of API requests. Use this procedure to create a new collection that contains a request template for each operation in the Game Analytics Pipeline solution API.

1. In Postman, on the **File** menu, choose **Import**.
2. On the **Import** window, choose **Import From Link**, and then enter the following URL: https://raw.githubusercontent.com/awslabs/game-analytics-pipeline/master/source/resources/game-analytics-pipeline-postman-collection.json.

3. Choose **Import**. Postman imports the Game Analytics Pipeline solution collection, which includes folders for **Applications**, **Authorizations**, and **Events** endpoints.

Step 5. Test the solution API

Use this procedure to test the solution API.

1. In the navigation pane, expand the `game-analytics-pipeline` collection, and then expand the **Applications** folder.
2. In the list of requests, choose **Create Application**.
3. Use the **Environment** selector to choose the environment that you created earlier in Step 2.1.
4. Choose **Send**. If the request is successful, the response pane shows a status of `200 OK`.

To interact with other resources included in the solution API using Postman, refer to Solution API for information on request and response format requirements.

Document Revisions

Date	Change
May 2020	Initial release
September 2020	This update modifies the solution to remove the tracking of player-identifiable and client-identifiable fields from the JSON event telemetry that is ingested and processed by the solution. For additional information about version 1.1.0, refer to the CHANGELOG.md file in the GitHub repository.
August 2021	Release version 1.1.1: Bug fixes and updates to node.js support in Lambda functions. For more information, refer to the CHANGELOG.md file in the GitHub repository.