Build Amazon Kinesis Integration with AWS SAM and OpenAPI

You can integrate an API method directly with an Amazon Kinesis Data Stream without the need for a compute layer such as Lambda in the middle. This is achieved through the **AWS service integration** type.

Use AWS SAM and OpenAPI to create an API Gateway REST API with a direct Amazon Kinesis integration

1. Using AWS Cloud9 console, return to the root folder `module-3/kinesis`
2. This code belongs in your SAM template file `template.yaml`

Review the code and then **copy/paste** it into the `template.yaml` file.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: >
  module3-kinesis: Sample SAM Template for module3-kinesis

Globals:

    #Enable Logs
    Api:
        MethodSettings:
            - ResourcePath: "/*"
              HttpMethod: "*"
              DataTraceEnabled: True
              LoggingLevel: INFO
              MetricsEnabled: True
    Function:
        Timeout: 3
        Runtime: nodejs18.x
Parameters:
  KinesisStreamName:
    Type: String
    Default: KinesisStreamAPIGW

Resources:

    #Create the Kinesis Data Stream
    KinesisStreamAPIGW:
        Type: AWS::Kinesis::Stream
        Properties:
          Name: !Ref KinesisStreamName
          StreamModeDetails:
              StreamMode: ON_DEMAND

    #Create the IAM role
    IAMRoleForKinesisIntegration:
        Type: AWS::IAM::Role
        Properties:
            AssumeRolePolicyDocument:
                Version: "2012-10-17"
                Statement:
                  - Effect: Allow
                    Principal:
                      Service:
                          - apigateway.amazonaws.com
                    Action:
                      - 'sts:AssumeRole'
            Path: /
            Policies:
              - PolicyName: PolicyForAPIGWKinesisIntegration
                PolicyDocument:
                  Version: "2012-10-17"
                  Statement:
                    - Effect: Allow
                      Action: 'kinesis:PutRecord'
                      Resource: !GetAtt KinesisStreamAPIGW.Arn

    # Create the API Gateway
    APIWithKinesisIntegration:
        Type: AWS::Serverless::Api
        Properties:
            StageName: dev
            OpenApiVersion: 3.0.3
            DefinitionBody: # an OpenApi definition
              "Fn::Transform":
                  Name: "AWS::Include"
                  Parameters:
                      Location: "openapi.yaml"
            EndpointConfiguration:
                Type: REGIONAL

    # Lambda to Consume from Kinesis
    GetRecordsFunction:
      Type: 'AWS::Serverless::Function'
      Properties:
          CodeUri: ../handlers
          Handler: get-records.handler
          Policies:
            - KinesisCrudPolicy:
                StreamName: !Ref KinesisStreamName
          Events:
            Stream:
              Type: Kinesis
              Properties:
                Stream: !GetAtt KinesisStreamAPIGW.Arn
                BatchSize: 100
                StartingPosition: LATEST
Outputs:
    APIGatewayEndpoint:
        Description: API Gateway Endpoint
        Value:
            Fn::Sub: https://${APIWithKinsesisIntegration}.execute-api.${AWS::Region}.amazonaws.com/dev/streams/${KinesisStreamName}/record
    KinesisARN:
        Description: ARN for Kinesis data stream
        Value:
            !GetAtt KinesisStreamAPIGW.Arn
```

The above template contains a mixture of both **AWS SAM** and **CloudFormation** resources. Within this template, we have created a **Kinesis Data Stream**, an **IAM role** that will give **API Gateway** permission to **put records to the Kinesis** data stream service and an API Gateway. After a successful deployment, the template will output the **API Gateway invoke URL** as well as the **ARN of the Kinesis data stream**.

Lines **(19-33)** are particular important as these are what create the AWS Lambda function responsible for consuming the data that was put in the Amazon Kinesis by the API gateway integration. The handler of the function is the `handlers/get-records.js` As for the event configuration Batch Size, the number of records to send to the function in each batch, is set to **100**. The Starting Position, with events will be consumed, is set to `Latest` meaning the function will process new records that are added to the stream

3. This code belongs in your OpenAPI definition file `openapi.yaml`

Review the code and then **copy/paste** it into the `openapi.yaml` file

```
openapi: 3.0.0
info:
  title: module-3-kinesis-integration
  version: '2016-03-31T18:25:32Z'
paths:
  /streams/{stream-name}/record:
    put:
      parameters:
        - name: stream-name
          in: path
          required: true
          schema:
            type: string
        - in: body
          name: RequestBodyModel
          required: true
          schema:
            $ref: "#/components/schemas/PutRecordMethodRequestPayload"
      requestBody:
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/PutRecordMethodRequestPayload"
        required: true
      responses:
        "200":
          description: 200 response
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/Empty"
      x-amazon-apigateway-request-validator: "basic"
      x-amazon-apigateway-integration:
        type: aws
        credentials:
          Fn::Sub: ${IAMRoleForKinesisIntegration.Arn}
        uri: arn:aws:apigateway:${AWS::Region}:kinesis:action/PutRecord
        responses:
          default:
            statusCode: "200"
        requestParameters:
          integration.request.header.Content-Type: '''application/x-amz-json-1.1'''
        requestTemplates:
          application/json: |
            {
              "StreamName": "$input.params('stream-name')",
              "Data": "$util.base64Encode($input.json('$.Data'))",
              "PartitionKey": "$input.path('$.PartitionKey')"
            }
        passthroughBehavior: when_no_match
        httpMethod: POST
components:
  schemas:
    Empty:
      type: object
    PutRecordMethodRequestPayload:
      title: "PutRecordMethodRequestPayload"
      required:
        - "Data"
        - "PartitionKey"
      type: "object"
      properties:
        Data:
          title: "The Data Schema"
          type: "string"
        PartitionKey:
          title: "The PartitionKey Schema"
          type: "string"
      additionalProperties: false
x-amazon-apigateway-request-validators:
  basic:
    validateRequestParameters: true
    validateRequestBody: true
x-amazon-apigateway-gateway-responses:
  BAD_REQUEST_BODY:
    statusCode: 400
    responseTemplates:
      application/json: "{\"message\": \"$context.error.messages\", \"error\": \"$context.error.validationErrorString\"}"
```

Lines **(33-51)** are particular important as these are what create the direct service integration for Kinesis. The x-amazon-apigateway-integration is an OpenAPI extension that allows us to integrate our APIs to AWS backends.

Lines **(56-69)** defines the model schema of the request to be made using the json schema pattern. This schema not only identifies the name of each attribute, but also specifies its type. The additionalProperties attribute , when set to 'false', signifies that no further attributes can be added to the request. The 'required' attribute denotes the fields that are compulsory in the request. By employing this model, you can ensure that the request conforms to the required format, thereby preventing any unanticipated inputs from reaching your backend.

Lines **(70-79)** define the request validators, which will validate the body of the request. If any issue is detected, a HTTP status code 400 will be returned to the client, indicating an invalid request body, and also provide a message detailing the reason for the error.

Deploy the project

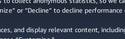
1. To deploy the API Gateway and Kinesis resources to your AWS account, run the following commands from the application root `module-3/kinesis`, where the `template.yaml` file for the sample application is located:

```
sam build && sam deploy --guided
```

The first time that you run the `sam deploy --guided` command, AWS SAM starts an AWS CloudFormation deployment. In this case, you need to say what are the configurations that you want SAM to have in order to get the guided deployment. You can configure it as below.

- **Stack Name**: `module-3-kinesis-integration`
- **AWS Region**: Put the chosen region to run the workshop, **e.g.** `us-east-1`
- **Parameter KinesisStreamName**: leave blank
- **Confirm changes before deploy**: Y
- **Allow SAM CLI IAM role creation**: Y
- **Disable rollback**: N
- **Save arguments to configuration file**: Y
- **SAM configuration file** and **SAM configuration environment** leave blank

2. After configuring the deployment, AWS SAM will display assets that will be created. But first, it will automatically upload the template to a temporary bucket it creates. Then, it will ask you to confirm the changes. Type `y` to confirm.



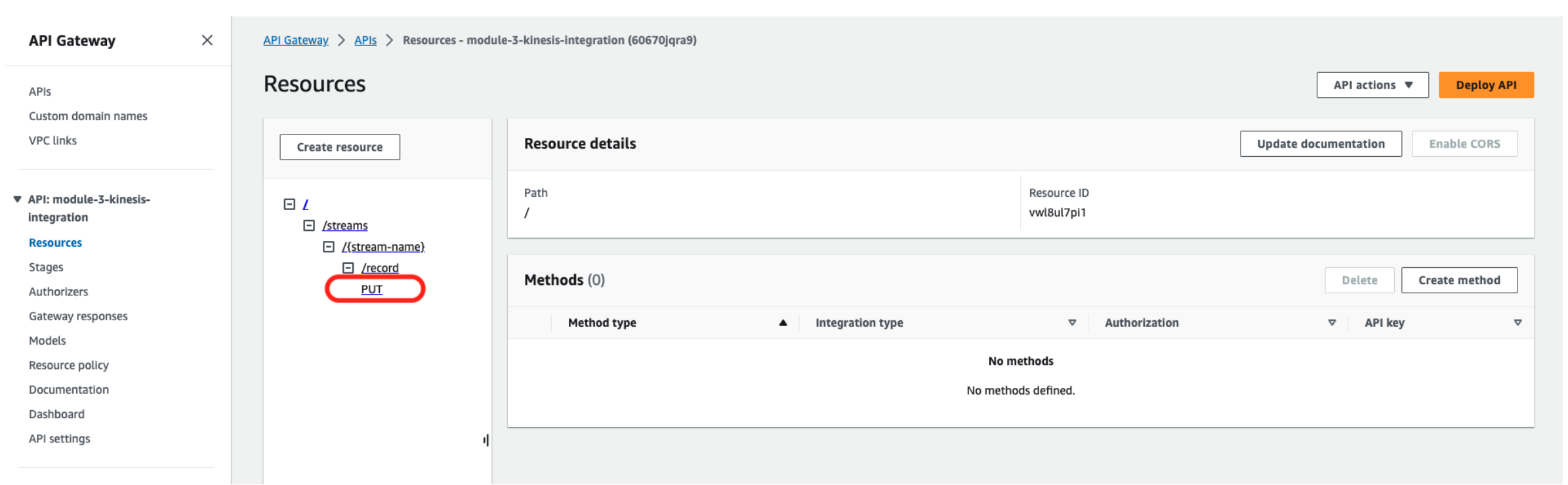
3. Once the deployment has been successful, you will see an '**Outputs'** section that contains the API Gateway invoke URL and the ARN of the Kinesis you just created

Test the Integration

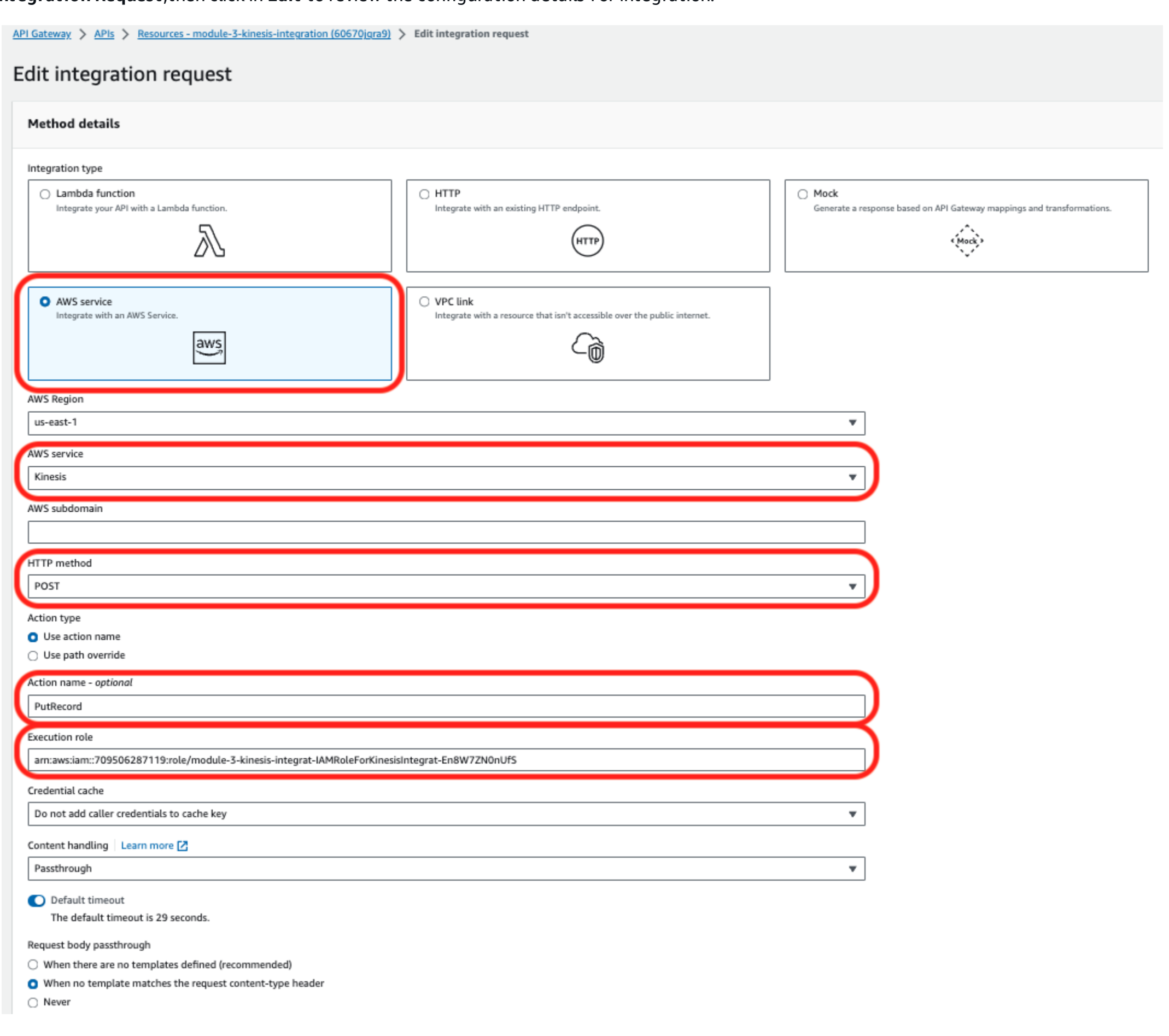
Now the API has been deployed, the validation of the request body is in place and a lambda will consume the stream, its time to test the integration.

Test Kinesis Integration using API Gateway console

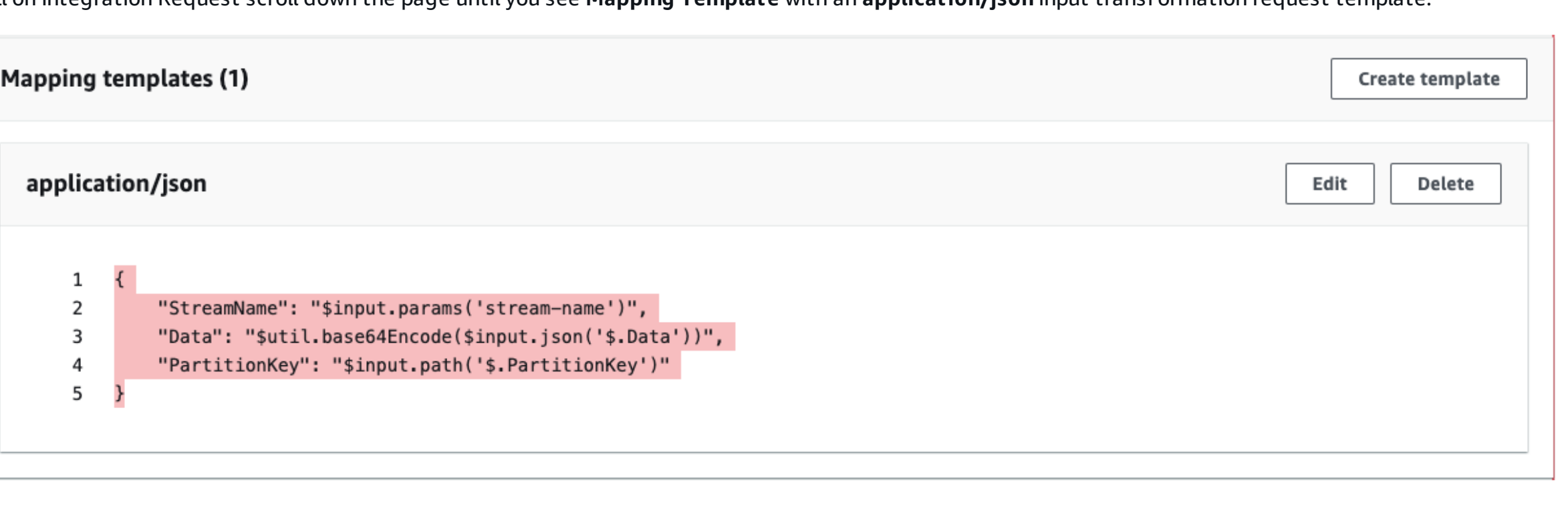
1. Navigate to the API Gateway console and select the API that has been created. It will be named **module-3-kinesis-integration**.
2. Select the PUT method from the resources section to the left of the screen to bring up the API request flow window.



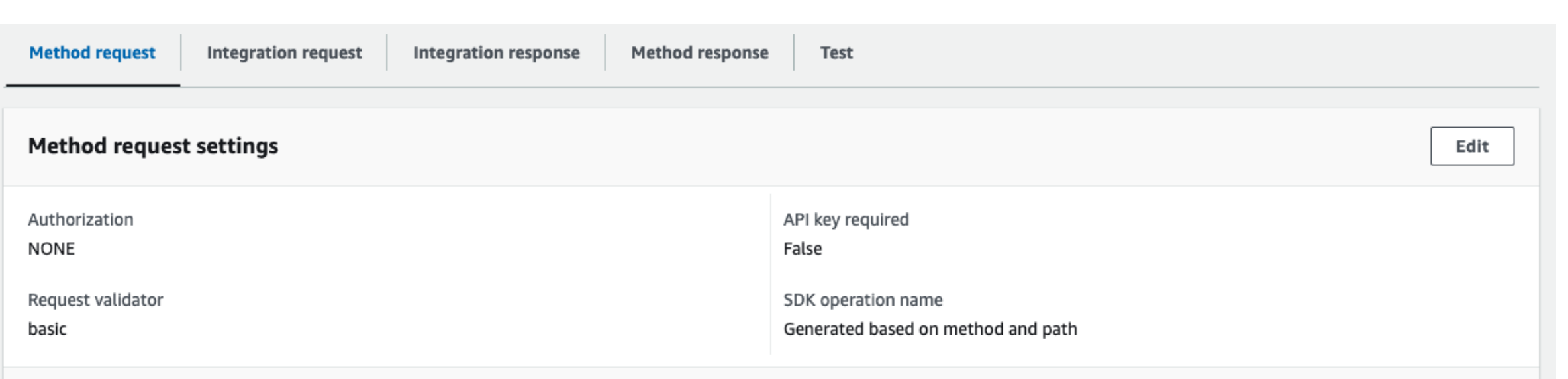
3. Click in **Integration Request**,then click in **Edit** to review the configuration details for integration.



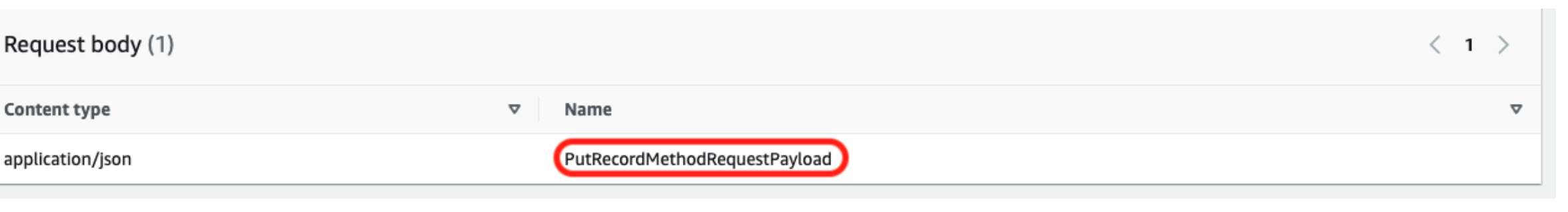
4. When you finish reviewing. Click in **Save** to return to the previous screen.
5. Still on Integration Request scroll down the page until you see **Mapping Template** with an **application/json** input transformation request template:



6. Go back to Method request, select the METHOD REQUEST button, review the configuration details for integration.

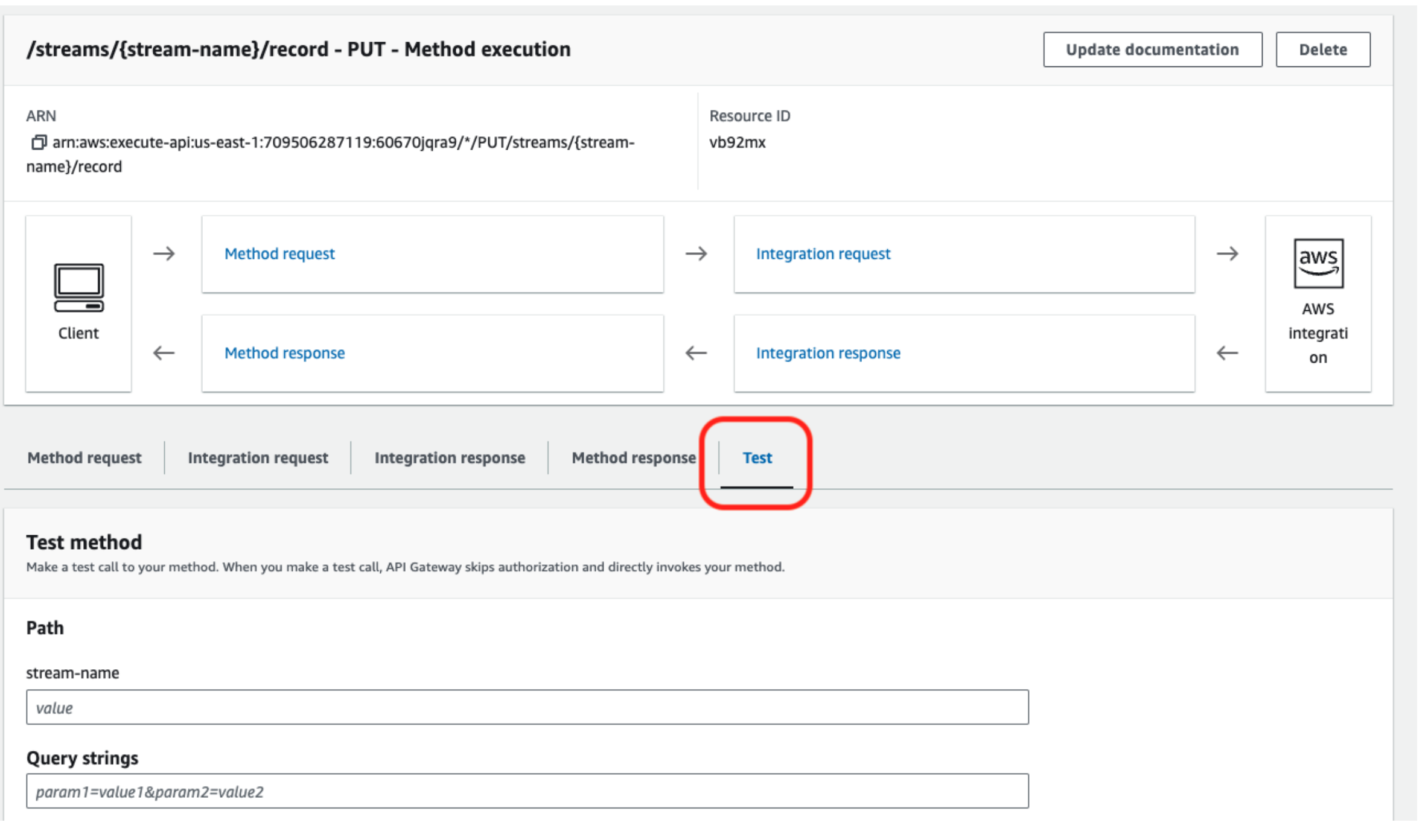


7. Scroll down to the Request Body to validate the model name.



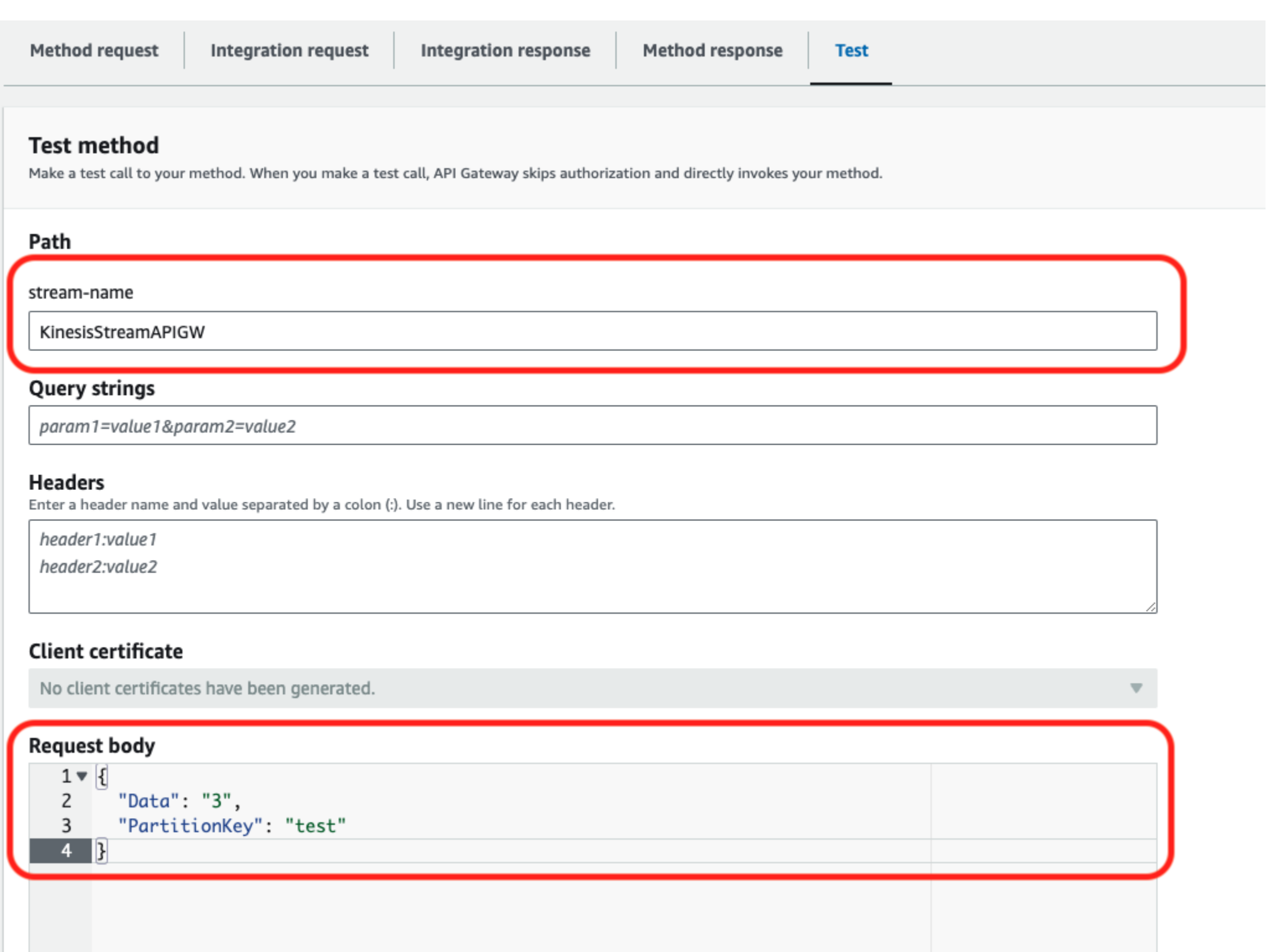
8. Scroll up and select the TEST button.

Warning! Testing methods with the API Gateway console may result in changes to resources that cannot be undone. Testing a method with the API Gateway console is the same as calling the method outside of the API Gateway console. For example, if you use the API Gateway console to call a method that deletes an API's resources, if the method call is successful, the API's resources will be deleted.



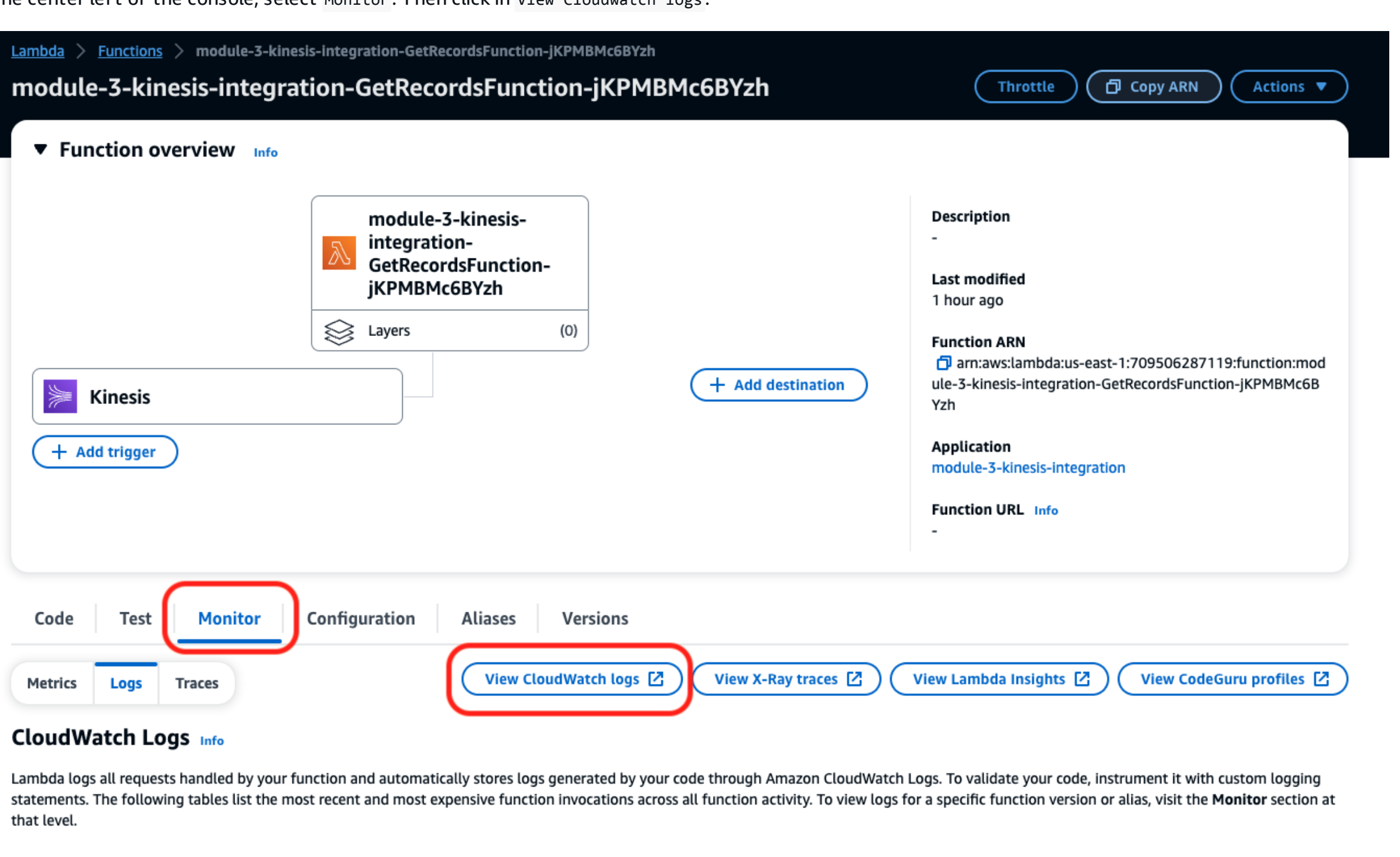
9. This will open the window that allows you to test invocations to your backend. In our case, we want to send a message to the Kinesis Data Stream witch will be consumed by the lambda. Fill the {stream-name} field the `KinesisStreamAPIGW`, the name of the kinesis stream created with the SAM. You can verify the name in the console by accessing the kinesis data stream console.
10. Fill the Request Body field with the following json

```
{
  "Data": "3",
  "PartitionKey": "test"
}
```

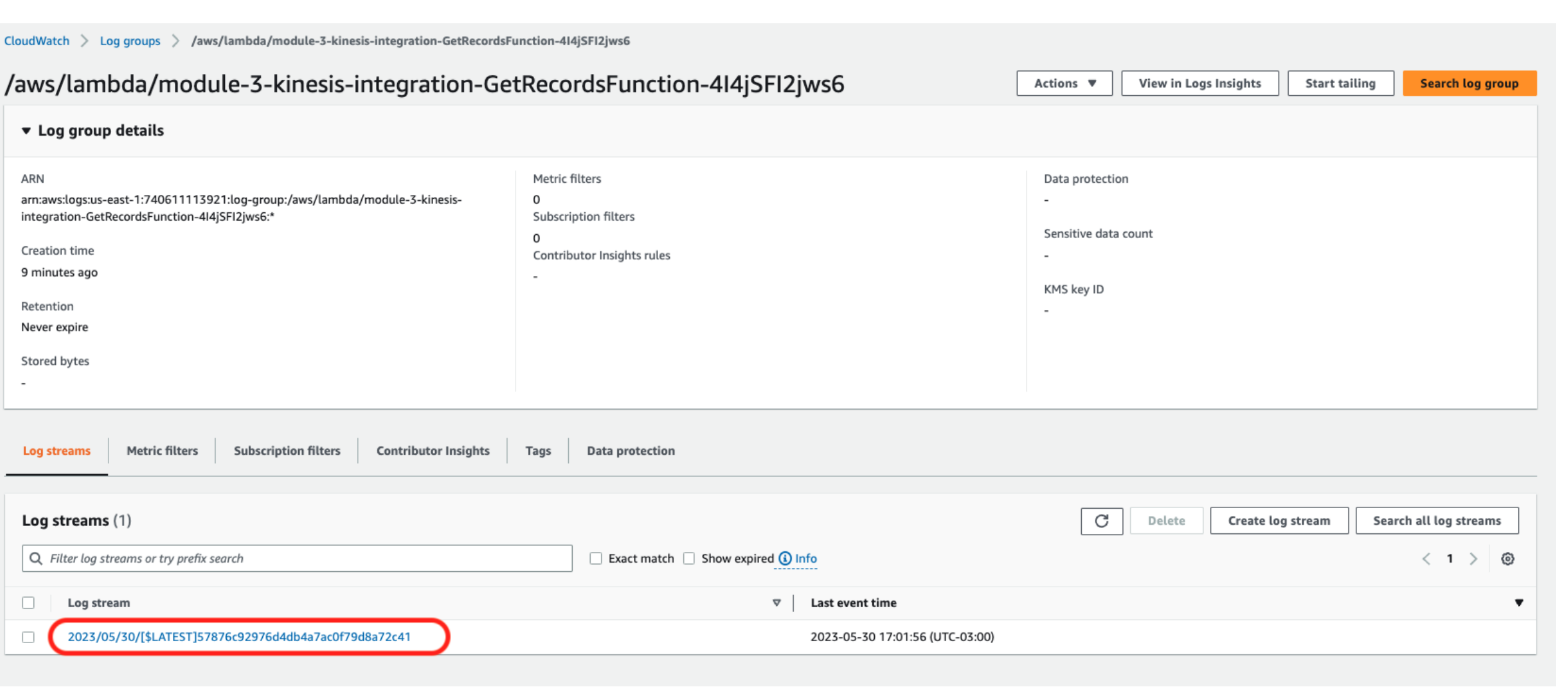


11. Scroll to the bottom of the page and hit the Test button. From this, you should recieve a Status code of 200 meaning the message was sucessfully delivered to the Kinesis Data Stream.

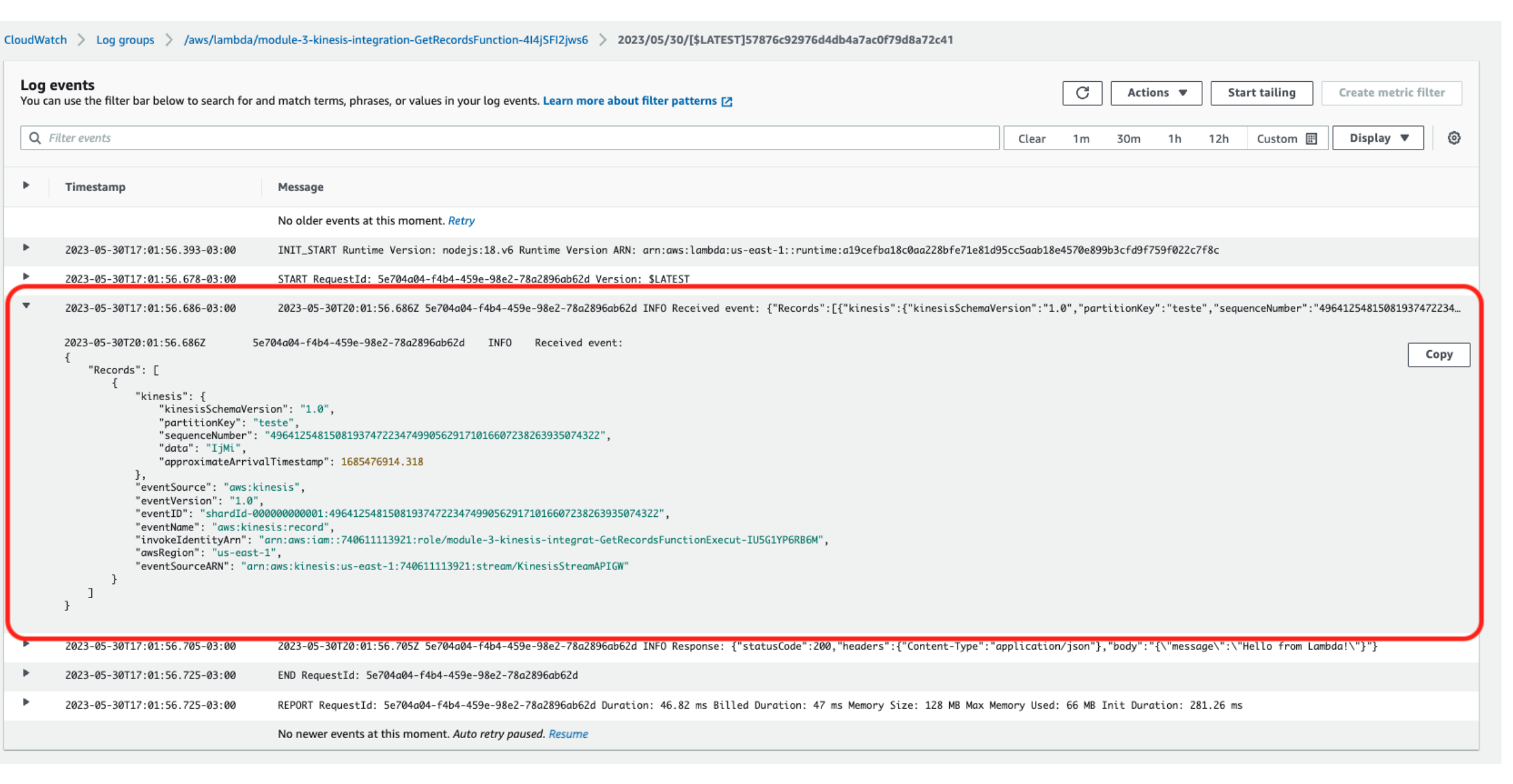
12. Go to lambda console, find and click the lambda with name starting with `module-3-kinesis-integration-GetRecordsFunction`.
13. In the center left of the console, select `Monitor`. Then click in `View Cloudwatch logs`.



14. Click in the fist log stream available.

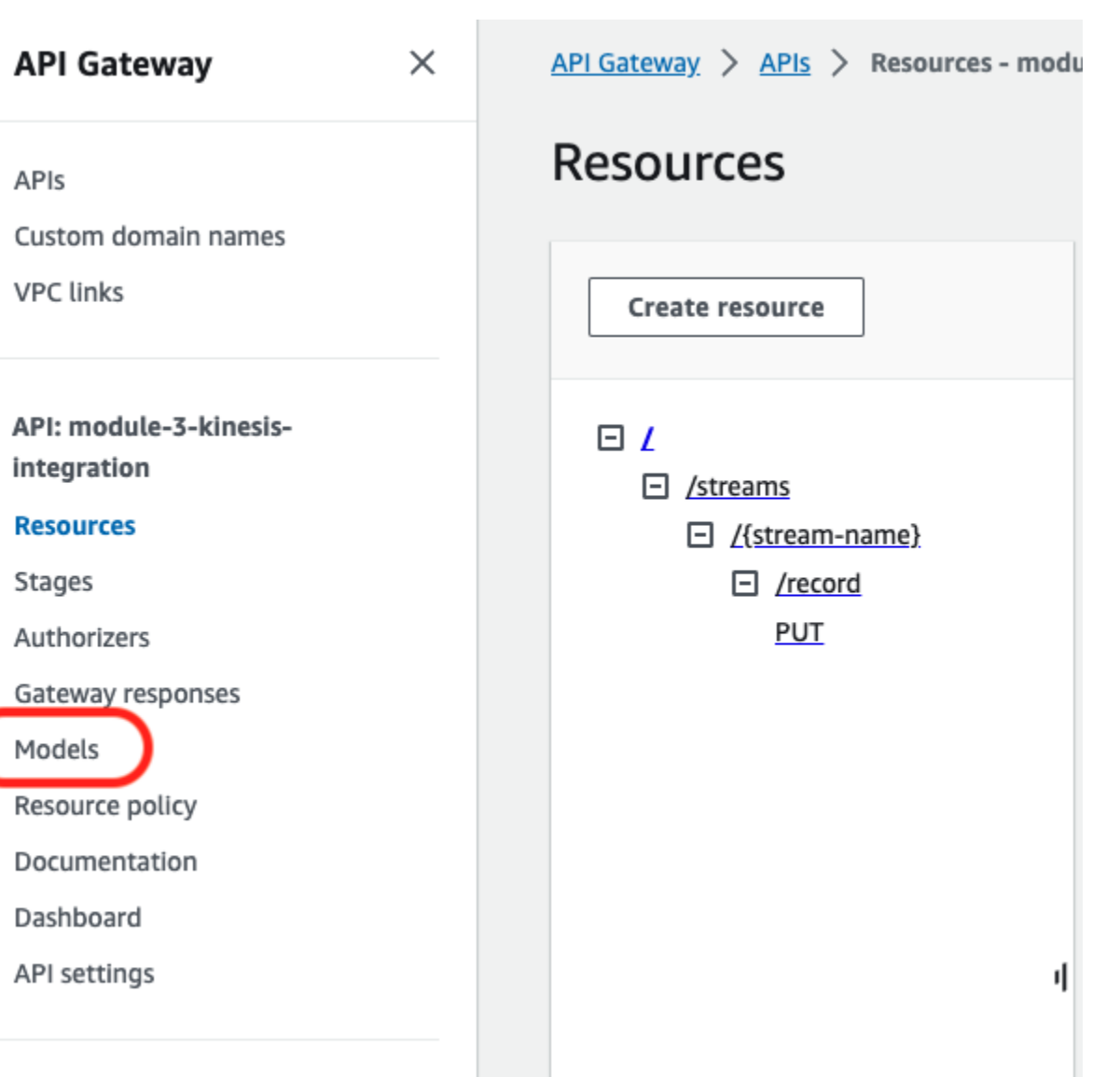


15. Expand any line with the beginning `Received event:` to validate the record sent throught API Gateway to Kinesis and consued by the lambda

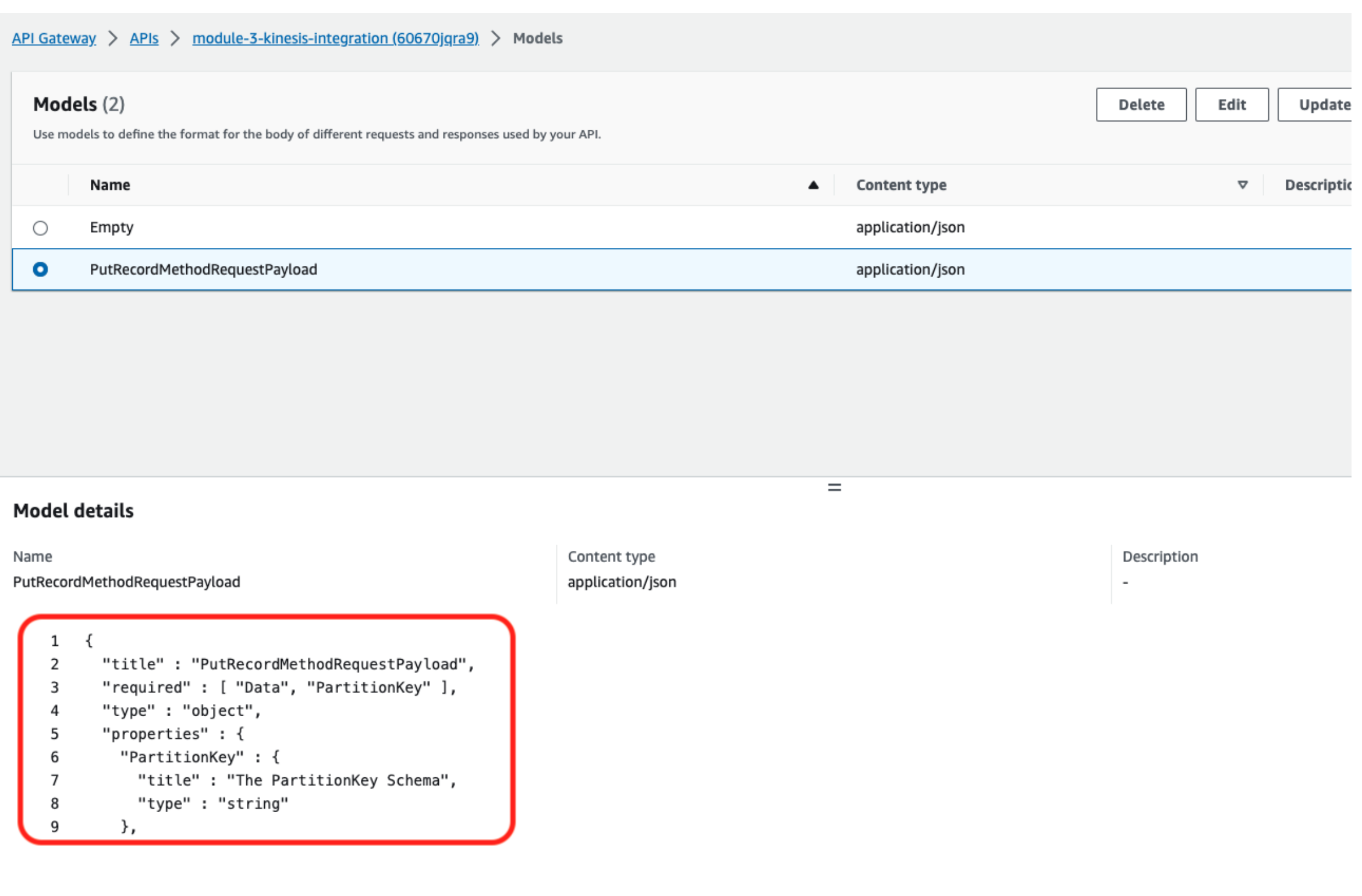


Test the deployed API with invalid parameters in the request

1. Navigate to the API Gateway console and select the API that has been created. It will be named **module-3-kinesis-integration**.
2. In the left corner click in `Models`

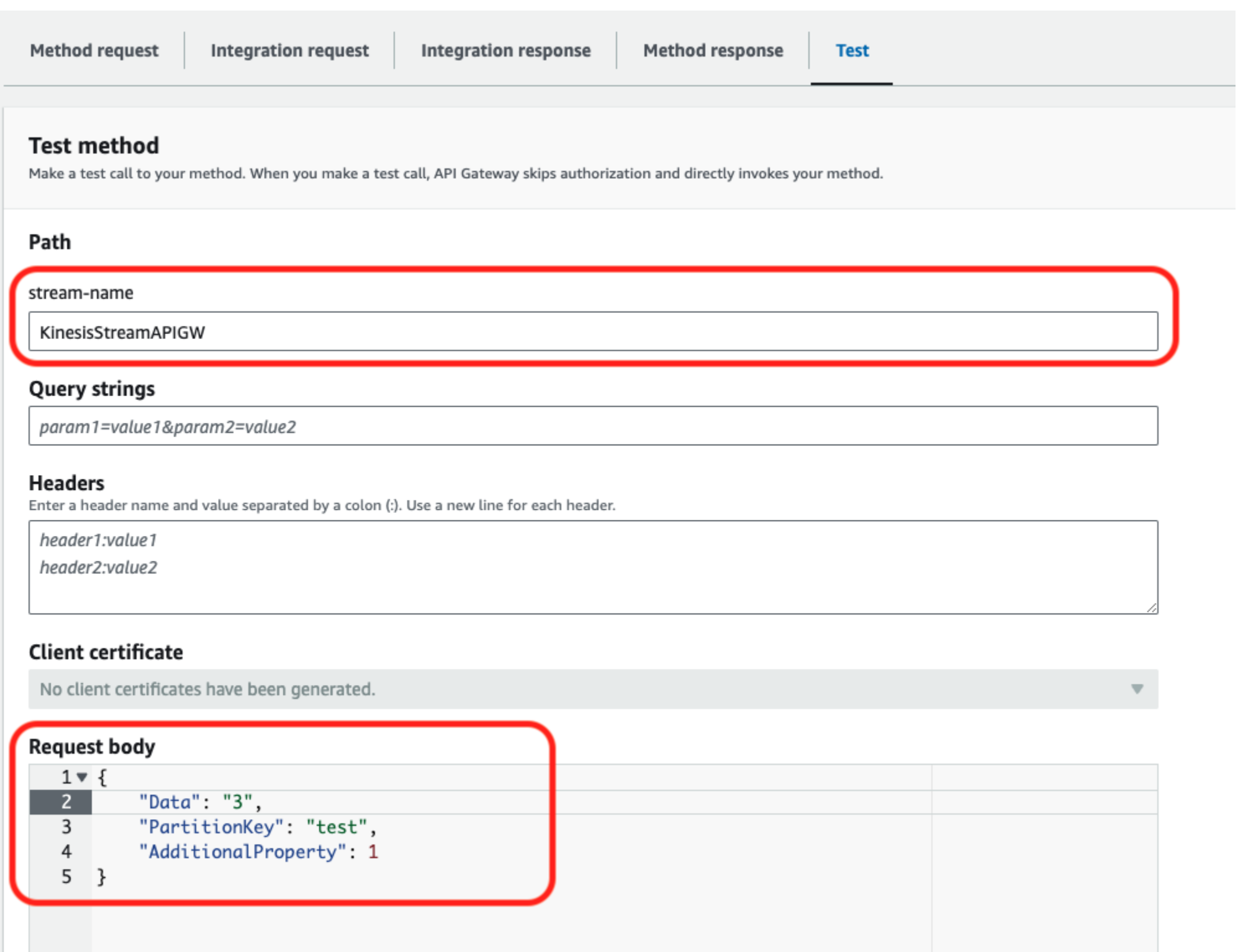


3. Select the model `PutRecordMethodRequestPayload`.
4. This JSON has been created as part of the SAM template to validate requests for the PUT method of the `/record` resource on the API. It is presented in the Json Schema format. Essentially, it declares two fields: `Data` and `PartitionKey`, both of which are required and must be of the string type. The `additionalProperties=false` setting implies that only the declared properties can be included in the request. Any property that deviates from this pattern will be rejected during the validation process.



5. Let's test the validation. Click in `Resources` in the left corner.
6. Select the PUT method from the resources section to the left of the screen to bring up the API request flow window.
7. Select the TEST tab.
8. Fill the {stream-name} field the `KinesisStreamAPIGW`.
9. First let's validate a request with extra properties. Copy the following json and paste in the `Request Body`

```
{
  "Data": "3",
  "PartitionKey": "test",
  "AdditionalProperty": 1
}
```

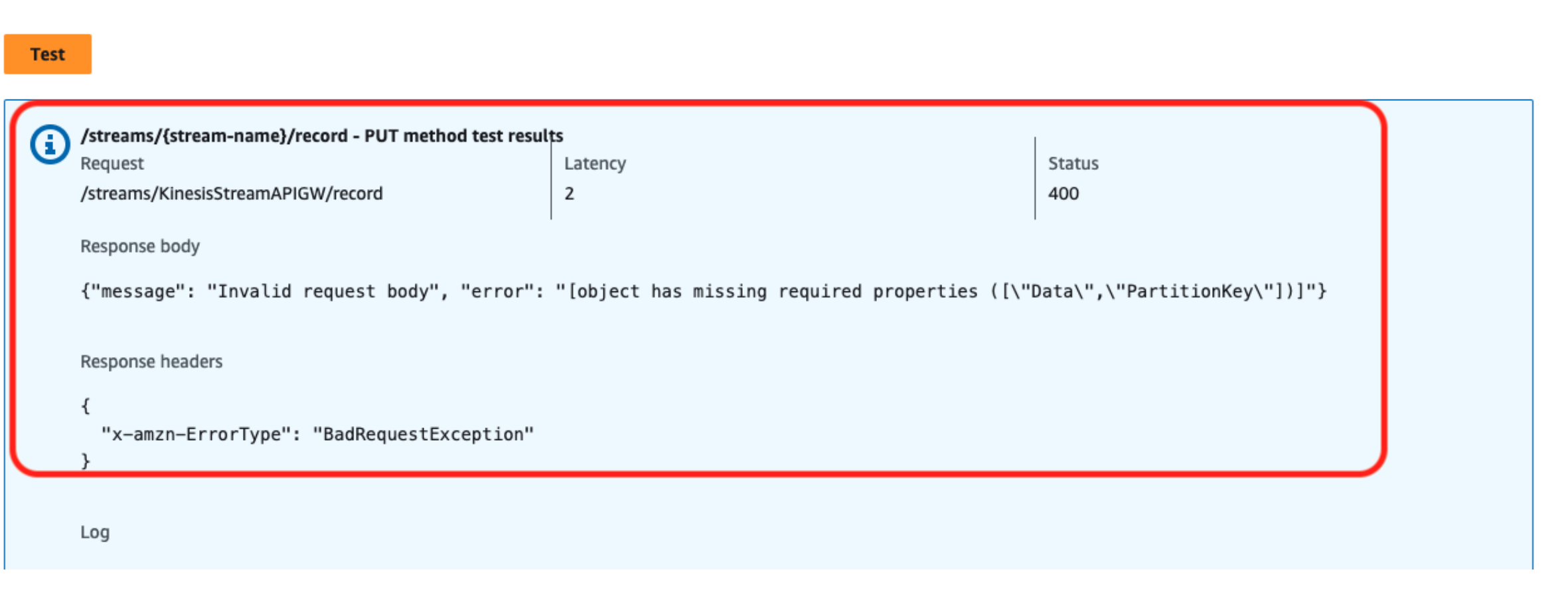


10. Scroll to the bottom of the page and hit the Test button. From this, you should recieve a Status code of 400 meaning the Invalid request body and with the message: `"[object instance has properties which are not allowed by the schema: [\"AdditionalProperty\"]]"`. Note that this request didn't go to kinesis and was validated in the API Gateway in the Method Request phase of the integration.

11. Now insert a empty json in the Request Body Field and click in Test

```
{
}
```

12. The response this time is still a Status Code 400 with Invalid request body but with the message: `"[object has missing required properties ([\"Data\",\"PartitionKey\"])]"` meaning the validation did not found the required properties and failed.



13. The final test involves an incorrectly named field and an incorrect data type. The `Data` property will not start with a capital letter, and the `PartitionKey` value will be an integer rather than a string. After pasting the JSON in the Request Body, click on `TEST`.

```
{
  "data": "3",
  "PartitionKey": 1
}
```

14. In this final test you got the message: `[object has missing required properties ([\"Data\"]), object instance has properties which are not allowed by the schema: [\"data\"]]"` meaning it valitaded the first property and being already wrong it returned the error signal.

Test the deployed API using cURL

1. Open a new terminal window in your AWS Cloud9 environment.
2. Copy the below cURL command and paste it into the terminal window, replacing `<api-id>` with your API ID and `<region>` with the region where your API is deployed. You can also get the full URL from the Outputs section from your SAM deployment. You can also find the full invoke URL in the API Gateway console by navigating to **Stages > dev**.

```
curl -X PUT \
 'https://<api-id>.execute-api.<region>.amazonaws.com/dev/streams/KinesisStreamAPIGW/record' \
  -H 'Content-Type: application/json' \
  -d '{"Data": "eyJhIjogMX0=","PartitionKey": "test"}'
```

3. The **Response Body** output should be:

```
 {
   "SequenceNumber":"49642425449682996849330992744638041493183653490899025922",
    "ShardId":"shardId-000000000000"
 }
```

Challenge
Create new **resource** and **method** in the API Gateway to send **multiple messages** to the Kinesis Data Stream in the **same request**.