

In this video I go over a live example of a very simple DynamoDB table designed to store user comments, and then through the detailed steps required to expose update/query operations on that table over a RestFUL API using Amazon API Gateway. DynamoDB is Amazon's cloud hosted NoSQL database.

API Gateway is used here to define and deploy a fulling functional, externally accessible HTTP/rest API to delegate some instructions to the DynamoDB backend. DynamoDB already exposes web services to directly manipulate the data but they are all POST services. There are several advantages to wrapping AWS services (such as Lambda functions as well) in API Gateway services. One of them here is that you can create a GET service to retrieve data from Dynamo which wraps the POST based services available directly against the DynamoDB.

IAM in this example is used to configure the permissions for users/roles/groups to perform operations on AWS resources. In this example we **create policies to update and insert into DynamoDB table** and then inform our API Gateway to run the API invocation with that role.

This is an excellent tutorial but it doesn't include all the details you need on setting up the IAM role in particular the trust relationship and DynamoDB policies. Here is the Trust Relationship I used for the role I created for this example - just copy paste it into the JSON editor for the Trust Relationship

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

Here is the policy I created to be able to add to and query my Comments Dynamo DB table. YOU WILL HAVE TO CHANGE IT FOR YOUR OWN ACCOUNT - these are referring the to the database resources belonging to my account. The wildcard entry ensures that all sub objects of the main table (such as indexes) are also permission

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:PutItem",
        "dynamodb:Query"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:ACCOUNT:table/Comments/*",
        "arn:aws:dynamodb:us-east-1:ACCOUNT:table/Comments"
      ]
    }
  ]
}
```

DynamoDB - AWS Console

DynamoDB - AWS Console

IAM Management Console

Matt Thomas - N. Virginia - Support

Create table

Amazon DynamoDB is a fully managed non-relational database service that provides fast and predictable performance with seamless scalability.

Create table

Recent alerts

No CloudWatch alarms have been triggered.

Total capacity for US East (N. Virginia)

Provisioned read capacity	15	Reserved read capacity	0
Provisioned write capacity	15	Reserved write capacity	0

Service health

Current Status	Details
● Amazon DynamoDB (N. Virginia)	Service is operating normally View complete service health details

What's new

- Enhanced metrics
- Titan graph database integration
- Elasticsearch integration

Related services

- Amazon ElastiCache

Additional resources

- Getting started guide
- Getting started hands-on lab
- FAQ
- Release notes
- Developer guide
- Forums
- Report an issue

DynamoDB - AWS Console

DynamoDB - AWS Console

IAM Management Console

Matt Thomas - N. Virginia - Support

Create table Actions ▾

Filter by table name X

Viewing 2 of 2 Tables

Name	Status	Partition key	Sort key	Indexes	Total read capacity	Total write capacity
Comments	Active	commentId (String)	-	1	10	10
Stories	Active	storyId (String)	-	0	5	5

DynamoDB - AWS Console

DynamoDB - AWS Console

IAM Management Console

Matt Thomas - N. Virginia - Support

Create DynamoDB table

DynamoDB is a schema-less database that only requires a table name and primary key. The table's primary key is made up of one or two attributes that uniquely identify items, partition the data, and sort data within each partition.

Table name* ?

Primary key* Partition key String ?

Add sort key

Table settings

Default settings provide the fastest way to get started with your table. You can modify these default settings now or after your table has been created.

Use default settings

- No secondary indexes.
- Provisioned capacity set to 5 reads and 5 writes.
- Basic alarms with 80% upper threshold using SNS topic "dynamodb".

Additional charges may apply if you exceed the AWS Free Tier levels for CloudWatch or Simple Notification Service. Advanced alarm settings are available in the CloudWatch management console.

Cancel Create

DynamoDB - AWS Console

DynamoDB - AWS Console

IAM Management Console

Matt Thomas - N. Virginia - Support

Create table Actions ▾

Filter by table name X

Name
Comments
Stories

Comments Close

Overview Items Metrics Alarms Capacity Indexes Triggers Access control

Create item Actions ▾

Scan: [Table] Comments: commentId ▲

Viewing 1 to 3 items

commentId	message	pageId	userName
2	Total nonsense	back page	Daisy
1	I really enjoye...	front page	Walter
3	Awesome	middle page	Dennis

The screenshot shows the AWS DynamoDB console with the 'Edit item' dialog open. The dialog displays a JSON object with four items:

```

    {
        "commentId": "String : 2",
        "message": "String : Total nonsense",
        "pageid": "String : back page",
        "userName": "String : Daisy"
    }
  
```

The screenshot shows the AWS DynamoDB console with the 'Access control' tab selected for the 'Comments' table. Under the 'Actions' section, several actions are selected:

- Select all
- BatchGetItem
- BatchWriteItem
- DeleteItem
- .GetItem
- PutItem
- Query
- UpdateItem

You can also specify the things that can be done on this table and generate a JSON permissions representation as below

The screenshot shows the AWS DynamoDB console with the 'Access control' tab selected for the 'Comments' table. Under the 'Actions' section, several actions are selected:

- Select all
- BatchGetItem
- BatchWriteItem
- DeleteItem
- .GetItem
- PutItem
- Query
- UpdateItem

A large JSON document is displayed in the 'Policy document' area:

```

 1- {
 2-   "Version": "2012-10-17",
 3-   "Statement": [
 4-     {
 5-       "Effect": "Allow",
 6-       "Action": [
 7-         "dynamodb:BatchGetItem",
 8-         "dynamodb>DeleteItem",
 9-         "dynamodb:PutItem"
 10-      ],
 11-      "Resource": [
 12-        "arn:aws:dynamodb:us-east-1:515331423217:table/Comments"
 13-      ],
 14-      "Condition": {
 15-        "ForAllValues:StringEquals": {
 16-          "dynamodb:LeadingKeys": [
 17-            "${graph.facebook.com:id}"
 18-          ]
 19-        }
 20-      }
 21-    }
 22-  ]
 23- }
  
```

It also creates this JSON document for you that you can then paste into the policy document for this table. This will be used to assign permissions on this table to certain Roles for the users.

The screenshot shows the AWS IAM Management Console with the 'Welcome to Identity and Access Management' page. It displays the following information:

- IAM users sign-in link: <https://515331423217.signin.aws.amazon.com/console>
- IAM Resources:
 - Users: 2
 - Groups: 1
 - Customer Managed Policies: 2
- Identity Status:
 - Roles: 1
 - Identity Providers: 0
- Security Status:
 - 3 out of 5 complete.
 - Tasks:
 - Delete your root access keys
 - Activate MFA on your root account
 - Create individual IAM users
 - Use groups to assign permissions
 - Apply an IAM password policy

IAM service helps to manage all your resources permissions using Roles, Groups, Policies, etc.

The screenshot shows the IAM Management Console with the 'Policies' section selected. A table lists various policies, including the 'Comments_PutItem' policy which has been selected. The table columns are 'Policy Name', 'Attached Entities', 'Creation Time', and 'Edited Time'. The 'Comments_PutItem' policy was created on May 21, 2016, at 11:38 UTC+0100.

These are all the default policies that AWS has created for use. We have used the **Create Policy** button to create our own **Comments_PutItem** policy above.

The screenshot shows the 'Comments_PutItem' policy details in the IAM Management Console. The policy ARN is arn:aws:iam::515331423217:policy/Comments_PutItem. The policy document is displayed as JSON:

```

1+ {
2+   "Version": "2012-10-17",
3+   "Statement": [
4+     {
5+       "Effect": "Allow",
6+       "Action": [
7+         "dynamodb:PutItem",
8+         "dynamodb:Query"
9+       ],
10+      "Resource": [
11+        "arn:aws:dynamodb:us-east-1:515331423217:table/Comments/*",
12+        "arn:aws:dynamodb:us-east-1:515331423217:table/Comments"
13+      ]
14+    }
15+  ]
16+}

```

This is the policy, it is the JSON syntax that we created much earlier. It describes what people having this policy can do to the Comments table, the **PutItem** and **Query** operations can be performed on the table resources listed.

The screenshot shows the 'Comments' table in the DynamoDB AWS Console. The 'Access control' tab is selected. A JSON policy is attached to the table:

```

1 = [
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": [
7         "dynamodb:BatchGetItem",
8         "dynamodb:DeleteItem",
9         "dynamodb:GetItem",
10        "dynamodb:PutItem",
11        "dynamodb:Query",
12        "dynamodb:UpdateItem"
13     ],
14     "Resource": [
15       "arn:aws:dynamodb:us-east-1:515331423217:table/Comments"
16     ],
17     "Condition": {
18       "ForAllValues:StringEquals": {
19         "dynamodb:LeadingKeys": [
20           "$[graph.facebook.com:id]"
21         ]
22       }
23     }
24   ]
25 ]

```

The screenshot shows the 'Comments' table in the DynamoDB AWS Console. The 'Indexes' tab is selected. A Global Secondary Index (GSI) named 'pageId-index' is listed:

Name	Status	Type	Partition key	Sort key	Attributes	Read capacity	Write capacity	Size	Item count
pageId-index	Active	GSI	pageId (String)	-	commentId, pa	5	5	534	5

There is an index called **pageId-index** created on the Comments table that enables us to query this table, we needed to also assign permission for Query to be able to query the ...Comments/* table.

AWS Services Edit

Dashboard

Create New Role Role Actions

Search IAM

Details Groups Users Roles Policies Identity Providers Account Settings Credential Report Encryption Keys

Showing 1 results

Role Name	Creation Time
Comment_role	2016-05-21 11:42 UTC+0100

A Policy has to be assigned to a user or to a Role,

IAM > Roles > Comment_role

Summary

Role ARN: arn:aws:iam::515331423217:role/Comment_role

Instance Profile ARN(s): /

Path: /

Creation Time: 2016-05-21 11:42 UTC+0100

Permissions Trust Relationships Access Advisor

Managed Policies

The following managed policies are attached to this role. You can attach up to 10 managed policies.

Attach Policy

Policy Name	Actions
Comments_Putitem	Show Policy Detach Policy Simulate Policy

Inline Policies

We have attached the **Comments_Putitem** policy to this **Comment_role** Role,

IAM > Roles > Comment_role

Summary

Role ARN: arn:aws:iam::515331423217:role/Comment_role

Instance Profile ARN(s): /

Path: /

Creation Time: 2016-05-21 11:42 UTC+0100

Permissions Trust Relationships Access Advisor

Managed Policies

The following managed policies are attached to this role.

Attach Policy

Policy Name: Comments_Putitem

Show Policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:PutItem",
        "dynamodb:Query"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:515331423217:table/Comments/*",
        "arn:aws:dynamodb:us-east-1:515331423217:table/Comments"
      ]
    }
  ]
}
```

Cancel

This is the policy used for this role.

You also have to define a trust relationship for this role,

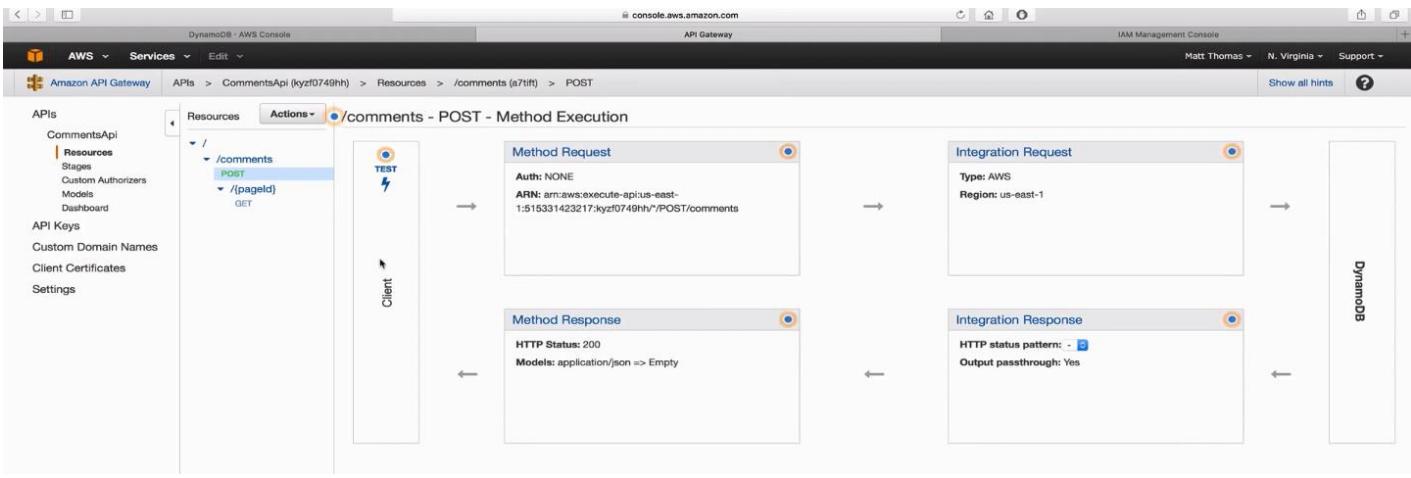
```

1- {
2-   "Version": "2012-10-17",
3-   "Statement": [
4-     {
5-       "Effect": "Allow",
6-       "Principal": {
7-         "Federated": "www.amazon.com"
8-       },
9-       "Action": "sts:AssumeRoleWithWebIdentity",
10-      "Condition": {
11-        "StringEquals": {
12-          "www.amazon.com:app_id": "boom"
13-        }
14-      }
15-    },
16-    {
17-      "Sid": "",
18-      "Effect": "Allow",
19-      "Principal": {
20-        "Service": "apigateway.amazonaws.com"
21-      },
22-      "Action": "sts:AssumeRole"
23-    }
24-  ]
25-}

```

You have to put this entry here that basically says that you are allowing the API Gateway from amazon.com to assume this role and be able to do the things it needs executed

This is the CommentsApi that we created in the API Gateway service.



Screenshot of the AWS API Gateway Integration Request configuration for the POST method of the /comments resource.

Integration type: AWS Service Proxy (selected)

AWS Region: us-east-1

AWS Service: DynamoDB

AWS Subdomain:

HTTP method: POST

Action Pattern:

Execution role: arn:aws:iam::515331423217:role/Comment_role

Credentials cache: Do not add caller credentials to cache key

URL Path Parameters:

URL Query String Parameters:

HTTP Headers:

Body Mapping Templates:



Screenshot of the AWS DynamoDB Tables interface.

Tables: Comments, Stories

Indexes: pageld-index (Active, GSI, pageld(String), commentId, pa 5, 5, 534, 5)

Screenshot of the AWS DynamoDB Access Control interface for the Comments table.

Identity provider: Facebook

Actions: Select all, BatchGetItem, BatchWriteItem, DeleteItem, GetItem, PutItem, Query, UpdateItem

Allowed attributes: All attributes

```

1 - {
2 -   "Version": "2012-10-17",
3 -   "Statement": [
4 -     {
5 -       "Effect": "Allow",
6 -       "Action": [
7 -         "dynamodb:BatchGetItem",
8 -         "dynamodb:DeleteItem",
9 -         "dynamodb:PutItem"
10 -       ],
11 -       "Resource": [
12 -         "arn:aws:dynamodb:us-east-1:515331423217:table/Comments"
13 -       ],
14 -       "Condition": {
15 -         "ForAllValues:StringEquals": [
16 -           "dynamodb:LeadingKeys": [
17 -             "$[graph.facebook.com:id]"
18 -           ]
19 -         }
20 -       }
21 -     }
22 -   ]
23 - }

```

Attach policy instructions:

Screenshot of the AWS IAM Management Console showing the Trust Relationships tab for the Comment_role.

Summary

Role ARN: arn:aws:iam::515331423217:role/Comment_role

Instance Profile ARN(s): /

Path: /

Creation Time: 2016-05-21 11:42 UTC+0100

Permissions **Trust Relationships** **Access Advisor**

You can view the trusted entities that can assume the role and the access conditions for the role. Show policy document

Edit Trust Relationship

Trusted Entities

The following trusted entities can assume this role.

Condition	Key	Value
StringEquals	www.amazon.com:app_id	boom

Screenshot of the AWS IAM Management Console showing the Permissions tab for the Comment_role.

Summary

Role ARN: arn:aws:iam::515331423217:role/Comment_role

Instance Profile ARN(s): /

Path: /

Creation Time: 2016-05-21 11:42 UTC+0100

Permissions **Trust Relationships** **Access Advisor**

Managed Policies

The following managed policies are attached to this role. You can attach up to 10 managed policies.

Attach Policy

Policy Name	Actions
Comments_PutItem	Show Policy Detach Policy Simulate Policy

Inline Policies

Screenshot of the AWS IAM Management Console showing the Show Policy dialog for the Comments_PutItem managed policy.

Summary

Role ARN: arn:aws:iam::515331423217:role/Comment_role

Instance Profile ARN(s): /

Path: /

Creation Time: 2016-05-21 11:42 UTC+0100

Permissions **Trust Relationships** **Access Advisor**

Managed Policies

The following managed policies are attached to this role. You can attach up to 10 managed policies.

Attach Policy

Policy Name	Actions
Comments_PutItem	Show Policy Detach Policy Simulate Policy

Inline Policies

Show Policy

```
{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Action": [ "dynamodb:PutItem", "dynamodb:Query" ], "Resource": [ "arn:aws:dynamodb:us-east-1:515331423217:table/Comments/*", "arn:aws:dynamodb:us-east-1:515331423217:table/Comments" ] } ] }
```

Cancel

The screenshot shows the AWS API Gateway interface. On the left, the navigation bar includes 'AWS Services' and 'Edit'. Under 'APIs', 'CommentsApi' is selected. The main panel shows a 'Method Execution' for the '/comments - POST - Integration Request'. The integration type is set to 'AWS Service Proxy' (selected), pointing to 'DynamoDB'. The AWS Region is 'us-east-1'. The Content-Type is set to 'application/json'. A video overlay in the bottom right corner shows a man speaking.

We also MUST specify the Role that can invoke this particular endpoint as the Comments Role above

The screenshot shows the AWS API Gateway interface, identical to the previous one but with a different video overlay. The integration type is set to 'AWS Service Proxy' (selected), pointing to 'DynamoDB'. The AWS Region is 'us-east-1'. The Content-Type is set to 'application/json'. A video overlay in the bottom right corner shows a man speaking.

We are going to intercept the POST request from our API Gateway and then apply a mapping template to the body of the request, the Content-Type we are expecting is JSON

The screenshot shows the AWS API Gateway console with the following details:

- APIs**: CommentsApi
- Resources**: /comments
- Actions**: Edit, Test, Delete
- AWS Subdomain**: `CommentsApi.kyzf0749hh.`
- HTTP method**: POST
- Action**: PutItem
- Execution role**: `arn:aws:iam::515331423217:role/Comment_role`
- Credentials cache**: Do not add caller credentials to cache key
- URL Path Parameters**
- URL Query String Parameters**
- HTTP Headers**
- Body Mapping Templates** (selected):
 - Request body passthrough: When no template matches the request Content-Type header
 - When there are no templates defined (recommended)
 - Never
- Content-Type**: application/json
- Generate template**: application/json
- Mapping Template (JSON code):**

```

1- {
2-   "TableName": "Comments",
3-   "Item": {
4-     "commentId": {
5-       "$": "$input.path('$')"
6-     },
7-     "pageId": {
8-       "$": "$input.path('$')"
9-     },
10-    "userName": {
11-      "$": "$input.path('$')"
12-    },
13-    "message": {
14-      "$": "$input.path('$')"
15-    }
16-  }
17- }
```
- Integration Request**: Type: AWS, Region: us-east-1
- Integration Response**: HTTP status pattern: -, Output passthrough: Yes

We will then take the expected JSON and transform it into a form that DynamoDB service's PutItem endpoint understands using the mapping template shown above. Note how we are hardcoding the TableName to be Comments so that this is the only table that this endpoint can interact with.

The screenshot shows the AWS API Gateway console with the following details:

- APIs**: CommentsApi
- Resources**: /comments
- Actions**: Edit, Test, Delete
- Method Request** (Client side):
 - Auth: NONE
 - ARN: `arn:aws:execute-api:us-east-1:515331423217:kyzf0749hh/POST/comments`
- Integration Request** (DynamoDB side):
 - Type: AWS
 - Region: us-east-1
- Method Response** (Client side):
 - HTTP Status: 200
 - Models: application/json => Empty
- Integration Response** (DynamoDB side):
 - HTTP status pattern: -
 - Output passthrough: Yes

We can test this

Screenshot of the AWS API Gateway Method Execution interface for the /comments - POST method. The request body is:

```
1: {
2:   "commentId": "1124",
3:   "pageId": "breaking-news-story-01-18-2016",
4:   "message": "Just Saying Thank You",
5:   "message": "I really enjoyed this youtube video!"
```

The response shows the execution log and a video thumbnail of a man speaking.

Screenshot of the AWS DynamoDB Items page for the Comments table. The table contains four items:

commentId	message	pageId	userName
2	Total nonsense	back page	Daisy
1	I really enjoye...	front page	Walter
3	Awesome	middle page	Dennis
1124	I really enjoy...	breaking-new...	Just Saying T...

Screenshot of the AWS IAM Policies page. It shows a list of policies:

Policy Name	Attached Entities	Creation Time	Edited Time
Comments_PutItem	2	2016-05-21 11:38 UTC+0100	2016-05-22 15:24 UTC+0100
AdministratorAccess	1	2015-02-06 18:39 UTC+0100	2015-02-06 18:39 UTC+0100
AmazonAPIGatewayAdministrator	0	2015-07-09 18:34 UTC+0100	2015-07-09 18:34 UTC-0100
AmazonAPIGatewayInvokeFullAccess	0	2015-07-09 18:36 UTC+0100	2015-07-09 18:36 UTC+0100
AmazonAPIGatewayPushToCloudWatchLogs	0	2015-11-11 23:41 UTC+0100	2015-11-11 23:41 UTC+0100
AmazonAppStreamFullAccess	0	2015-02-06 18:40 UTC+0100	2015-02-06 18:40 UTC+0100
AmazonAppStreamReadOnlyAccess	0	2015-02-06 18:40 UTC+0100	2015-02-06 18:40 UTC+0100
AmazonCognitoDeveloperAuthenticatedIdentities	0	2015-03-24 17:22 UTC+0100	2015-03-24 17:22 UTC+0100
AmazonCognitoPowerUser	0	2015-03-24 17:14 UTC+0100	2015-03-24 17:14 UTC+0100
AmazonCognitoReadOnly	0	2015-03-24 17:06 UTC+0100	2015-03-24 17:06 UTC+0100

```

1 * {
2     "Version": "2012-10-17",
3     "Statement": [
4         {
5             "Effect": "Allow",
6             "Action": [
7                 "dynamodb:PutItem",
8                 "dynamodb:Query"
9             ],
10            "Resource": [
11                "arn:aws:dynamodb:us-east-1:515331423217:table/Comments/*",
12                "arn:aws:dynamodb:us-east-1:515331423217:table/Comments"
13            ]
14        }
15    ]
16 }

```

Let us change the above

```

1 * {
2     "Version": "2012-10-17",
3     "Statement": [
4         {
5             "Effect": "Allow",
6             "Action": [
7                 "dynamodb:PutItem",
8                 "dynamodb:Query"
9             ],
10            "Resource": [
11                "arn:aws:dynamodb:us-east-1:515331423217:table/Comments/*",
12                "arn:aws:dynamodb:us-east-1:515331423217:table/Comments"
13            ]
14        }
15    ]
16 }

```

This policy is invalid.

```

1 * {
2     "Version": "2012-10-17",
3     "Statement": [
4         {
5             "Effect": "Allow",
6             "Action": [
7                 "dynamodb:PutItem",
8                 "dynamodb:Query"
9             ],
10            "Resource": [
11                "arn:aws:dynamodb:us-east-1:515331423217:table/Comments/*",
12                "arn:aws:dynamodb:us-east-1:515331423217:table/Comments"
13            ]
14        }
15    ]
16 }

```

Comments_PutItem was updated successfully.

Now we have removed this users' access to the Comments table for PutItem and Query. Let us now see what we will get when we make the POST call again as below

Screenshot of the AWS API Gateway Method Execution interface for a POST request to /comments. The response body shows an AccessDeniedException.

```
{
  "type": "com.amazon.coral.service#AccessDeniedException",
  "message": "User: arn:aws:sts::515331423217:assumed-role/Comment_role/BackplaneAssumeRoleSession is not authorized to perform: dynamodb:PutItem on resource: arn:aws:dynamodb:us-east-1:515331423217:table/Comments"
}
```

We now see a message saying the Role we are using does not have access to work on the Comments DynamoDB table

Screenshot of the AWS API Gateway Method Execution interface for a POST request to /comments. The response body shows a successful response.

```
{"commentId": "1125", "pageId": "breaking-news-story-01-18-2016", "userName": "Just Saying Thank You", "message": "I really enjoyed this youtube video!!"}
```

We can reverse the policy and test again

We can now see the new record showing up in DynamoDB.

Let us now test the GET endpoint with the **pageid** as a route parameter.

Note that if you are calling DynamoDB directly, you have to make a POST call even if you are actually doing a GET call from the browser. The difference is that when making a GET call, you will have to make a POST call but with the Query action instead of the **PutItem** action. We are using the same execution Role that was used for the POST request earlier.

Screenshot of the AWS IAM Management Console showing a policy named "Comments_PutItem". The policy document grants "Allow" access to the "dynamodb:PutItem" and "dynamodb:Query" actions on the "Comments" table across all regions.

```

1- {
2-     "Version": "2012-10-17",
3-     "Statement": [
4-         {
5-             "Effect": "Allow",
6-             "Action": [
7-                 "dynamodb:PutItem",
8-                 "dynamodb:Query"
9-             ],
10-            "Resource": [
11-                "arn:aws:dynamodb:us-east-1:515331423217:table/Comments/*",
12-                "arn:aws:dynamodb:us-east-1:515331423217:table/Comments"
13-            ]
14-        }
15-    ]
16- }

```

Screenshot of the AWS API Gateway Resource configuration for the "/comments/{pageId}" endpoint. The "Body Mapping Templates" section shows a mapping template for "application/json" content-type that extracts the "pageId" from the URL path parameter and sets it as a key in the "params" object.

```

Content-Type: application/json
application/json
Add mapping template
Request body passthrough: When there are no templates defined (recommended)
Generate template:
1- {
2-     "TableName": "Comments",
3-     "IndexName": "pageId-index",
4-     "KeyConditionExpression": "pageId = :v1",
5-     "ExpressionAttributeValues": {
6-         ":v1": {
7-             "S": "${input.params('pageId')}"
8-         }
9-     }
10- }

```

We are going to be pulling the page-id parameter from the URL request parameters and then put it in the **params** object that we will send in the request to DynamoDB

Screenshot of the AWS API Gateway Method Execution configuration for the "/comments/{pageId}" endpoint. It shows the "Integration Response" section where the "HTTP status regex" is set to "-" and the "Method response status" is set to "200".

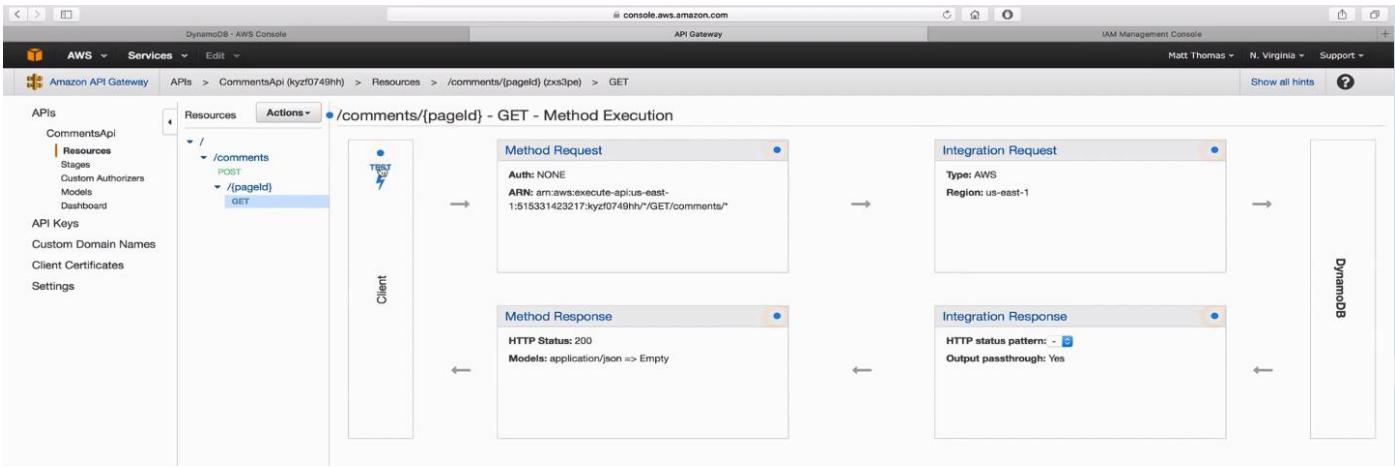
HTTP status regex	Method response status	Output model	Default mapping
-	200		Yes

Screenshot of the AWS API Gateway console showing the configuration for a GET method on the '/comments/{pageId}' resource. The 'Method Response' section is set up for a 200 status code, with the 'Content-Type' set to 'application/json'. A video player in the bottom right corner shows a man speaking.

For a 200 response, we can then map back the DynamoDB response before sending it to the browser

Screenshot of the AWS API Gateway console showing the configuration for a GET method on the '/comments/{pageId}' resource. The 'Method Response' section is set up for a 200 status code, with the 'Content-Type' set to 'application/json'. A video player in the bottom right corner shows a man speaking.

Screenshot of the AWS DynamoDB console showing the 'Edit item' dialog for a comment. The item contains four fields: commentId, message, pageId, and userName. The 'pageId' field is highlighted in yellow. A video player in the bottom right corner shows a man speaking.



Let us test this endpoint now

The screenshot shows the AWS API Gateway Method Test interface. The path is /comments/{pageId} - GET. The 'Path' field contains 'pageId' with the value 'breaking-news-story-01-18-2016'. The 'Request Body' section notes that it is not supported for GET methods. The 'Response Body' section displays a JSON array of comments:

```
{
  "comments": [
    {
      "commentID": "1125",
      "userName": "Just Saying Thank You",
      "message": "I really enjoyed this youtube video!!"
    },
    {
      "commentID": "1124",
      "userName": "Just Saying Thank You",
      "message": "I really enjoyed this youtube video!!"
    }
  ]
}
```

The 'Logs' section shows the execution log for the request:

```
Execution log for request test-request
Sun May 22 22:01:59 UTC 2016 : Starting execution for request: test-invoke-request
Sun May 22 22:01:59 UTC 2016 : HTTP Method: GET, Resource Path: /comments/{pageId}
Sun May 22 22:01:59 UTC 2016 : Method invocation initiated
Sun May 22 22:01:59 UTC 2016 : Method invocation completed
Sun May 22 22:01:59 UTC 2016 : Endpoint request body before transformations: { "pageId": "breaking-news-story-01-18-2016" }
Sun May 22 22:01:59 UTC 2016 : Endpoint request body after transformations: { "pageId": "breaking-news-story-01-18-2016" }
*****
```

We then get the correct response with the objects as above

The screenshot shows the AWS IAM Policy Details interface. The policy is named 'Comments_PutItem'. The 'Policy Document' tab is selected, displaying the following JSON code:

```
1. {
2.   "Version": "2012-10-17",
3.   "Statement": [
4.     {
5.       "Effect": "Allow",
6.       "Action": [
7.         "dynamodb:PutItem",
8.         "dynamodb:Query"
8.       ],
9.       "Resource": [
10.        "arn:aws:dynamodb:us-east-1:515331423217:table/Comments/*",
11.        "arn:aws:dynamodb:us-east-1:515331423217:table/Comments"
12.      ]
13.    }
14.  ]
15. }
```

Screenshot of the AWS IAM Management Console showing the policy editor for a global IAM policy named "Comments_PutItem". The policy document is displayed in JSON format:

```
1 - {  
2 -     "Version": "2012-10-17",  
3 -     "Statement": [  
4 -         {  
5 -             "Effect": "Allow",  
6 -             "Action": [  
7 -                 "dynamodb:PutItem",  
8 -                 "dynamodb:Query"  
9 -             ],  
10 -            "Resource": [  
11 -                "arn:aws:dynamodb:us-east-1:515331423217:table/Comments/*",  
12 -                "arn:aws:dynamodb:us-east-1:515331423217:table/Comments"  
13 -            ]  
14 -        }  
15 -    ]  
16 -}
```

The "Resource" section contains two entries: "arn:aws:dynamodb:us-east-1:515331423217:table/Comments/*" and "arn:aws:dynamodb:us-east-1:515331423217:table/Comments". A yellow highlight is present over these two lines.

At the bottom right, there are buttons for "Save as default version", "Cancel", "Validate Policy", and "Save".

Let's edit again

Screenshot of the AWS IAM Management Console showing the policy editor for the same global IAM policy "Comments_PutItem". The policy document has been modified:

```
1 - {  
2 -     "Version": "2012-10-17",  
3 -     "Statement": [  
4 -         {  
5 -             "Effect": "Allow",  
6 -             "Action": [  
7 -                 "dynamodb:PutItem",  
8 -                 "dynamodb:Query"  
9 -             ],  
10 -            "Resource": [  
11 -                "arn:aws:dynamodb:us-east-1:515331423217:table/Comments/bob",  
12 -                "arn:aws:dynamodb:us-east-1:515331423217:table/Comments"  
13 -            ]  
14 -        }  
15 -    ]  
16 -}
```

The "Resource" section now contains two entries: "arn:aws:dynamodb:us-east-1:515331423217:table/Comments/bob" and "arn:aws:dynamodb:us-east-1:515331423217:table/Comments". A yellow highlight is present over these two lines.

At the bottom right, there are buttons for "Save as default version", "Cancel", "Validate Policy", and "Save".

Screenshot of the AWS IAM Management Console showing the policy editor for the global IAM policy "Comments_PutItem". The policy document has been updated successfully:

```
Comments_PutItem was updated successfully.
```

The policy document remains the same as the previous screenshot:

```
1 - {  
2 -     "Version": "2012-10-17",  
3 -     "Statement": [  
4 -         {  
5 -             "Effect": "Allow",  
6 -             "Action": [  
7 -                 "dynamodb:PutItem",  
8 -                 "dynamodb:Query"  
9 -             ],  
10 -            "Resource": [  
11 -                "arn:aws:dynamodb:us-east-1:515331423217:table/Comments/bob",  
12 -                "arn:aws:dynamodb:us-east-1:515331423217:table/Comments"  
13 -            ]  
14 -        }  
15 -    ]  
16 -}
```

At the bottom right, there are buttons for "Edit", "Save as default version", "Cancel", "Validate Policy", and "Save".

Screenshot of the AWS API Gateway console showing a successful test execution for a GET method on the '/comments/{pageId}' endpoint.

Request:

```
Path: /comments/{pageId}
Method: GET
Query Strings: breaking-news-story-01-18-2016
```

Response Headers:

```
Content-Type: application/json
```

Logs:

```
Execution log for request test-request
Sun May 22 22:02:52 UTC 2016 : Starting execution for request: test-invoke-request
Sun May 22 22:02:52 UTC 2016 : HTTP Method: GET, Resource Path: /comments/{pageId}
Sun May 22 22:02:52 UTC 2016 : Method request path: {pageId=breaking-news-story-01-18-2016}
Sun May 22 22:02:52 UTC 2016 : Method request query string: {}
Sun May 22 22:02:52 UTC 2016 : Method request headers: {}
Sun May 22 22:02:52 UTC 2016 : Method request body before transformations: null
Sun May 22 22:02:52 UTC 2016 : Endpoint request URL: https://dynamodb.us-east-1.amazonaws.com/?Action=Query
Sun May 22 22:02:52 UTC 2016 : Endpoint request headers: {Authorization=*****}
*****
*****
X-amzn-apigateway-api-id=1dnyzj0749hh...
9nh, X-Amz-Security-Token=AgpG0332Z21...
3r/fdkFyij1f8gqMKHEH71W0BjK2Q+M4Vn2...
YDorMsNo/L1kCF4R6E8ln0aT1+0B82EE0f2...
vNxCVD16Pw0oU/Z8713PG/seM+tp8sxz8C...
JsnBV0IgywvAqXaKwvAqXaKwvAqXaKwvAqX...
VzuobmPq0pEtUgPrgtS9HwAtU19m0r61...
Sun May 22 22:02:52 UTC 2016 : Endpoint response body before transformations: {"__type":"com.amazon.coral.service#AccessDeniedException","Message":"User: arn:aws:sts::515331423217:assumed-role/C3ketrEBbGHMdvBZnDATE0"}
```

Screenshot of the AWS API Gateway console showing a failed test execution for a GET method on the '/comments/{pageId}' endpoint due to unauthorized access.

Request:

```
Path: /comments/{pageId}
Method: GET
```

Response Headers:

```
Content-Type: application/json
```

Logs:

```
Execution log for request test-request
Sun May 22 22:02:52 UTC 2016 : Starting execution for request: test-invoke-request
Sun May 22 22:02:52 UTC 2016 : HTTP Method: GET, Resource Path: /comments/{pageId}
Sun May 22 22:02:52 UTC 2016 : Method request path: {pageId=breaking-news-story-01-18-2016}
Sun May 22 22:02:52 UTC 2016 : Method request query string: {}
Sun May 22 22:02:52 UTC 2016 : Method request headers: {}
Sun May 22 22:02:52 UTC 2016 : Method request body before transformations: null
Sun May 22 22:02:52 UTC 2016 : Endpoint request URL: https://dynamodb.us-east-1.amazonaws.com/?Action=Query
Sun May 22 22:02:52 UTC 2016 : Endpoint request headers: {Authorization=*****}
*****
*****
X-amzn-apigateway-api-id=1dnyzj0749hh...
9nh, X-Amz-Security-Token=AgpG0332Z21...
3r/fdkFyij1f8gqMKHEH71W0BjK2Q+M4Vn2...
YDorMsNo/L1kCF4R6E8ln0aT1+0B82EE0f2...
vNxCVD16Pw0oU/Z8713PG/seM+tp8sxz8C...
JsnBV0IgywvAqXaKwvAqXaKwvAqXaKwvAqX...
VzuobmPq0pEtUgPrgtS9HwAtU19m0r61...
Sun May 22 22:02:52 UTC 2016 : Endpoint response body after transformations: {"__type":"com.amazon.coral.service#AccessDeniedException","Message":"User: arn:aws:sts::515331423217:assumed-role/C3ketrEBbGHMdvBZnDATE0"}
```

We again get an empty array returned along with a message that says that the role being used is not authorized to perform the Query on the Comments table. This shows how important it is to grant the correct permissions on our database resources.

```

1- {
2-     "Version": "2012-10-17",
3-     "Statement": [
4-         {
5-             "Effect": "Allow",
6-             "Action": [
7-                 "dynamodb:PutItem",
8-                 "dynamodb:Query"
9-             ],
10-            "Resource": [
11-                "arn:aws:dynamodb:us-east-1:515331423217:table/Comments/*",
12-                "arn:aws:dynamodb:us-east-1:515331423217:table/Comments"
13-            ]
14-        }
15-    ]
16- }

```

Correct the edit again

```

{
  "comments": [
    {
      "commentId": "1125",
      "userName": "Just Saying Thank You",
      "message": "I really enjoyed this youtube video!!!"
    },
    {
      "commentId": "1124",
      "userName": "Just Saying Thank You",
      "message": "I really enjoyed this youtube video!!!"
    }
  ]
}

```

We are now able to query again with the correct permission.

```

{
  "comments": [
    {
      "commentId": "1125",
      "userName": "Just Saying Thank You",
      "message": "I really enjoyed this youtube video!!!"
    },
    {
      "commentId": "1124",
      "userName": "Just Saying Thank You",
      "message": "I really enjoyed this youtube video!!!"
    }
  ]
}

```

Let us now present our API to the world in a proper manner, we can use Stages as below

Stages allow us to create different versions of our APIs for things like dev, prod, staging, test, etc.

The youtube2 stage now has an actual copy of the former stage we worked in but with the **youtube2** stage name. we also have the public URL for this API.

```

{
  "comments": [
    {
      "commentId": "1125",
      "userName": "Just Saying Thank You",
      "message": "I really enjoyed this youtube video!!"
    },
    {
      "commentId": "1124",
      "userName": "Just Saying Thank You",
      "message": "I really enjoyed this youtube video!!"
    }
  ]
}
  
```

We can query this GET call from our browser and get response as above

Securing Your Public API Gateway Endpoints with API Key

The screenshot shows the AWS API Gateway Stage Editor for the 'prod' stage of the 'CommentsApi'. The 'Settings' tab is selected, displaying options for metering and caching. Under 'Cache Settings', there is a checkbox for 'Enable API cache'. Under 'CloudWatch Settings', there are checkboxes for 'Enable CloudWatch Logs' and 'Enable CloudWatch Metrics'. Below these, the 'Default Method Throttling' section allows setting a rate limit of 500 requests per second with a burst limit of 1000 rps. A video overlay of a man speaking is visible in the top right corner.

There are 2 ways to protect your public API Gateway endpoints by restricting who can call them, **use a secret API Key** or **use a dedicated IAM Role**.

The screenshot shows the AWS API Gateway Resources page for the '/comments/{pageid}' endpoint. The 'Actions' dropdown is open, showing the 'Methods' option. A video overlay of a man speaking is visible in the top right corner.

Let us now restrict access to who can make the GET method calls to our endpoint, we will use a secret API Key

The screenshot shows the AWS API Gateway Method Execution interface for the GET method of the '/comments/{pageid}' endpoint. It displays the 'Method Request' (Auth: NONE, ARN: arn:aws:execute-api:us-east-1:b15331423217:kyzf0749hh/GET/comments/{pageid})' and 'Method Response' (HTTP Status: 200, Models: application/json => Empty). To the right, the 'Integration' section shows it is connected to an AWS Lambda function with Type: AWS Lambda and Region: us-east-1. A video overlay of a man speaking is visible in the top right corner.

APIs

CommentsApi

Resources

Stages

Custom Authorizers

Models

Dashboard

API Keys

Custom Domain Names

Client Certificates

Settings

Feedback English

MacBook-Pro:sbin mattua\$ curl https://kyzf0749hh.execute-api.us-east-1.amazonaws.com/prod/comments/7

```
{ "comments": [ ]}
```

MacBook-Pro:sbin mattua\$

There are 2 Authorization Settings that we can tune, the **Authorization header setting** and the **API Key Required setting**.

APIs

CommentsApi

Resources

Stages

Custom Authorizers

Models

Dashboard

API Keys

Custom Domain Names

Client Certificates

Settings

Feedback English

MacBook-Pro:sbin mattua\$ curl https://kyzf0749hh.execute-api.us-east-1.amazonaws.com/prod/comments/7

```
{ "comments": [ ]}
```

MacBook-Pro:sbin mattua\$

The Authorization header setting can be used with the above options

APIs

CommentsApi

Resources

Stages

Custom Authorizers

Models

Dashboard

API Keys

Custom Domain Names

Client Certificates

Settings

Actions

Method Execution /comments/{pageId} - GET - Method Request

Authorization Settings

Authorization: NONE

API Key Required: false

Request Paths

URL Query String Parameters

HTTP Request Headers

Request Models Create a Model

Feedback English

MacBook-Pro:sbin mattua\$ curl https://kyzf0749hh.execute-api.us-east-1.amazonaws.com/prod/comments/7

{

 "comments": [

]

MacBook-Pro:sbin mattua\$ curl https://kyzf0749hh.execute-api.us-east-1.amazonaws.com/prod/comments/7

{

 "comments": [

]

}You have new mail in /var/mail/mattua

MacBook-Pro:sbin mattua\$

We can now enable the API Key Required setting so that this endpoint should only work for requests with the correct API Key.

APIs

CommentsApi

Resources

Stages

Custom Authorizers

Models

Dashboard

API Keys

Custom Domain Names

Client Certificates

Settings

Actions

Method Execution /comments/{pageId} - GET - Method Request

Authorization Settings

Authorization: NONE

API Key Required: true

Request Paths

URL Query String Parameters

HTTP Request Headers

Request Models Create a Model

Actions

METHOD ACTIONS

Delete Method

RESOURCE ACTIONS

Create Method

Create Resource

Enable CORS

Delete Resource

API ACTIONS

Deploy API

Import API

Delete API

Method Execution /comments/{pageId} - GET - Method Request

information about this method's authorization settings and the parameters it can receive.

Authorization Settings

Authorization: NONE

API Key Required: true

Request Paths

URL Query String Parameters

HTTP Request Headers

Request Models Create a Model

Feedback English

You then need to deploy the new version of your API to make this change effective

MacBook-Pro:sbin mattua\$ curl https://kyzf0749hh.execute-api.us-east-1.amazonaws.com/prod/comments/7

```
{
  "comments": [
  ]
}
```

MacBook-Pro:sbin mattua\$ curl https://kyzf0749hh.execute-api.us-east-1.amazonaws.com/prod/comments/7

```
{
  "comments": [
  ]
}
```

You have new mail in /var/mail/mattua

MacBook-Pro:sbin mattua\$

MacBook-Pro:sbin mattua\$ curl https://kyzf0749hh.execute-api.us-east-1.amazonaws.com/prod/comments/7

```
{"message": "Forbidden"}
```

MacBook-Pro:sbin mattua\$

When we now try to access this API, we get a Forbidden message as above because we did not pass on an API Key.

MacBook-Pro:sbin mattua\$ curl https://kyzf0749hh.execute-api.us-east-1.amazonaws.com/prod/comments/7

```
{"message": "Forbidden"}
```

MacBook-Pro:sbin mattua\$

You can create an API Key using the API Keys tab on the left, an API Key is just any long string as shown below

The screenshot shows the AWS API Gateway API Keys page. A new API key named "MY_API_KEY" has been created. It has an API key value of "43AcGnfH9F8h0zoscqgZxRa70dueuJmaDj3ig7gi" and a description "desc". The "Enabled" checkbox is checked. Under "API Stage Association", the "Select API" dropdown is set to "CommentsApi" and the "Select stage" dropdown is set to "prod". A table titled "Stages Enabled" shows "CommentsApi" is associated with the "prod" stage. A "Save" button is visible.

```

MacBook-Pro:sbin mattua$ curl https://kyzf0749hh.execute-api.us-east-1.amazonaws.com/prod/comments/7
{"message": "Forbidden"}MacBook-Pro:sbin mattua$
You have new mail in /var/mail/mattua
MacBook-Pro:sbin mattua$
MacBook-Pro:sbin mattua$
MacBook-Pro:sbin mattua$ curl -H "X-API-KEY: 43AcGnfH9F8h0zoscqgZxRa70dueuJmaDj3ig7gi" https://kyzf0749hh.execute-api.us-east-1.amazonaws.com/prod/comments/7
{
  "comments": [
  ]
}MacBook-Pro:sbin mattua$ 

```

The API Key is generated for us by the API Gateway automatically when we click **Create** button, we then need to tell it what API this is for and what stage to use this key for. This API Key will have to be specified as a header parameter in the HTTP request. This means that we can not test this from the browser just like a hacker would but we will use the **curl** tool and provide the XAPIKEY_HEADER as above.