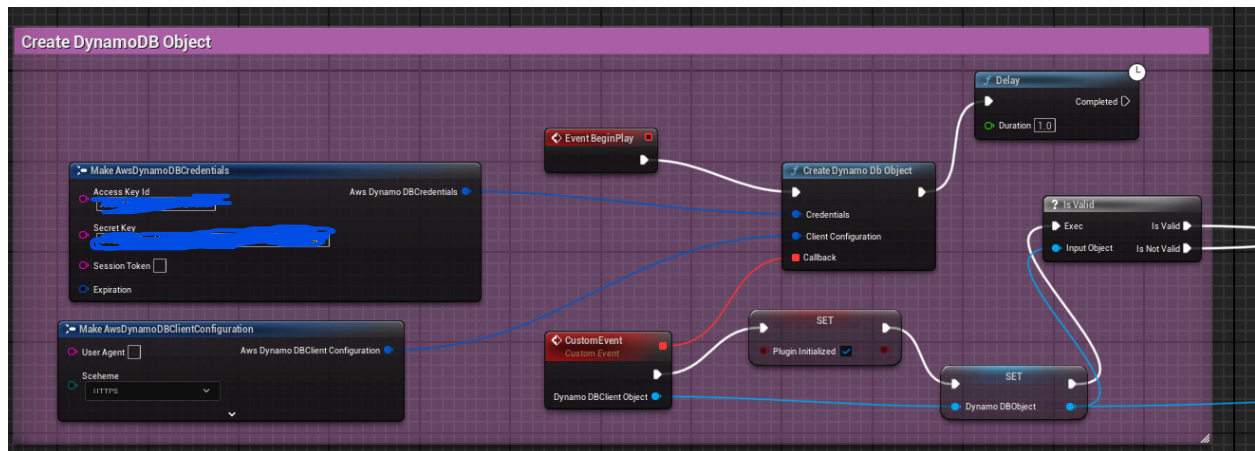


1. Download AWS DynamoDB Database Plugin by eelDev from the unreal engine marketplace and install it to your unreal engine version in the library.
2. Open your project in the unreal engine editor and go to Edit -> Plugins -> Search for "AWS" and look for "AwsCore::DynamoDB" and click the check to activate this plugin and then restart your editor.
3. Create a blueprint actor in your unreal engine because you will be attaching your DynamoDB blueprints (like queries and such) to an actor and placing it in your level.

Creating DynamoDB object:



- Once you have the plugin ready to go, follow these instructions on how to create the DynamoDB object so that you can successfully perform a Query or Scan to the database table.
- You have to get the access key ID and secret key for your access to the aws dynamodb table.
- Then, you want to include the "Make AwsDynamoDBClientConfiguration" node where you do not have to change much, except for using the drop-down menu to change your region to match your region in the AWS console.
- -Nodes: Make AwsDynamoDBCredentials, Make AwsDynamoDBClientConfiguration, Event BeginPlay, Custom event with DynamoDB Client Object object reference, Set for the value of a DynamoDBObject variable you need to create, and a check if it's valid.

The screenshot displays the AWS Step Functions console interface. On the left, the 'Make QueryRequest' state is configured with the following parameters:

- Table Name:** DigitalTwin
- Index Name:** (empty)
- Select:** ALL ATTRIBUTES
- Attributes To Get:** Limit (0), Consistent Read (checked)
- Key Conditions:** (empty)
- Query Filter:** Conditional Operator (NOT SET)
- Scan Index Forward:** (unchecked)
- Exclusive Start Key:** (empty)
- Return Consumed Capacity:** (NOT SET)
- Projection Expression:** (empty)
- Filter Expression:** (empty)
- Key Condition Expression:** (empty)
- Expression Attribute Names:** (empty)
- Expression Attribute Values:** (empty)

The main workflow diagram shows the following steps:

- Query:** A Lambda function call state that triggers the 'Query' function. It has a 'Request' input and outputs 'Success', 'Result', 'Error Type', and 'ErrorMessage'.
- Branch:** A decision state that checks the 'Request' status. It branches into 'True' and 'False' paths.
- Break QueryResult:** A state that processes the 'QueryResult' and outputs 'Items' and 'Count'.
- For Each Loop:** A loop state that iterates over the 'Array' of items. The loop body contains:
 - Print String:** A state that prints the 'In String' value, which is 'Query Unsuccessful'.
 - Convert Struct To Json String:** A state that converts the 'Struct' to a JSON string, outputting 'Out Json String'.
 - Print String:** A state that prints the 'In String' value, which is 'Loop Completed'.
- Print String:** A final state that prints the 'In String' value, which is 'Loop Completed'.



The workflow is marked as 'Development Only' in several places, indicating it is not yet ready for production.

- Firstly, note that this query is simply printing table items to the output log, but you should adjust the blueprints (starting at the for loop body) to perform whatever you are trying to do.
- Secondly, here, I am using the table's partition key as the "key condition expression" in the query request node. For our table, it's ObjectID so I made the key condition expression "ObjectID = :objID."
- Then, you would simply use this setup and then set the DynamoDB object in the "Query" node to the dynamoDB object you created earlier.
- Nodes: Make QueryRequest, Map of strings to attribute value structures variable for the expression attribute values map, string variable for key condition, Query, Branch, Break Query result, For each loop, and then the rest of the nodes depend on the functionality you intend to implement.

The screenshot displays a Step Functions workflow titled "Scan Blueprints". The workflow is composed of several nodes connected by arrows:

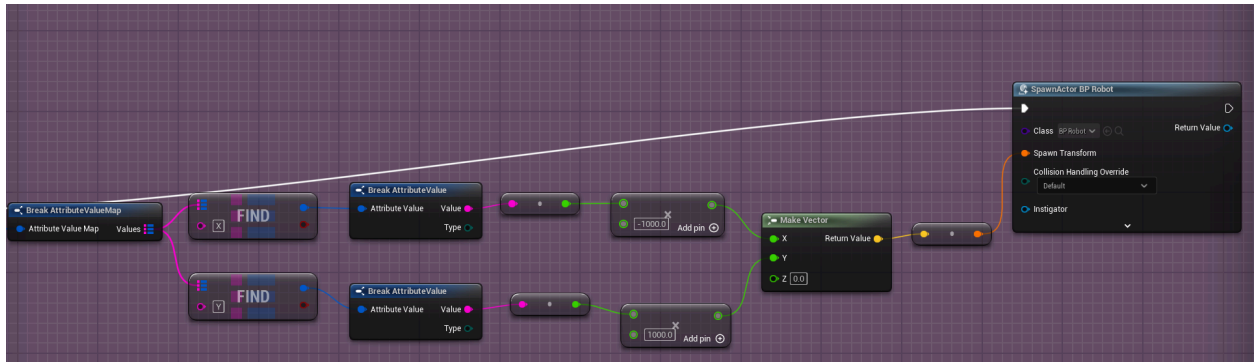
- Make ScanRequest**: A node with a "Scan Request" output. It has a "Table Name" field set to "DigitalTwin".
- Scan**: A node that receives input from "Make ScanRequest". It has a "DynamoDBObject" and "Request" input. It has a "Break ScanResult" node as a child.
- Branch**: A node that receives input from "Scan". It has a "Condition" field set to "True". It splits the flow into two paths: "True" and "False".
- Print String** (True path): A node that receives input from the "True" branch. It has a "In String" field set to "DynamoDBObject is not valid".
- Print String** (False path): A node that receives input from the "False" branch. It has a "In String" field set to "Query Unsuccessful".
- For Each Loop**: Two nodes, one for each path from the "Branch" node. They both have a "Loop Body" field set to "Array".
- Break AttributeValueMap**: A node that receives input from both "For Each Loop" nodes. It has a "Attribute Value Map" field set to "Values".
- Print String** (Loop Completed): A node that receives input from "Break AttributeValueMap". It has a "In String" field set to "Loop Completed".
- Find**: A node that receives input from "Print String (Loop Completed)". It has a "Find" field set to "Values".

The "Make ScanRequest" node has a "Scan Request" output. The "Scan" node has a "DynamoDBObject" and "Request" input. The "Break ScanResult" node has a "Scan Result" input. The "Branch" node has a "Condition" field set to "True". The "Print String" nodes have a "In String" field set to "DynamoDBObject is not valid" and "Query Unsuccessful". The "For Each Loop" nodes have a "Loop Body" field set to "Array". The "Break AttributeValueMap" node has a "Attribute Value Map" field set to "Values". The "Print String (Loop Completed)" node has a "In String" field set to "Loop Completed". The "Find" node has a "Find" field set to "Values".

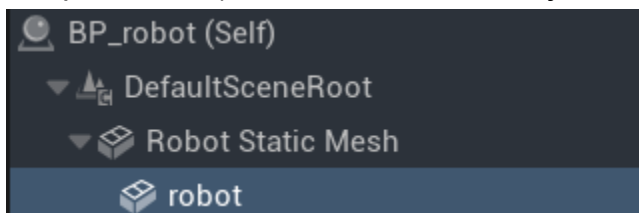
- | | | |
|----------------------------------|----------------|--|
| ▼ Expression Attribute Values... | 1 Map elements |   |
| ▼ :objType | 2 members | ▼ |
| Value | Robot | |
| Type | STRING | ▼ |

- Then from there, it's the same thing where you can set the DynamoDB object you created to the DynamoDBObject pin in the "Scan" node and then implement your

functionality from there.



- Here is the rest of our Scan blueprints that get the objects of type robot and spawn them into the level at X and Y coordinates provided to us by the Database table. I get these X and Y coordinates by using these “FIND” nodes after I break the attribute value map from the array element pin the for each loop node. Then I get the value which converts the X and Y numbers from string to floats for the vectors that the robots will spawn at.
- After that, when you hit the play button in the level, the robots will spawn in at the X and Y coordinates from the database.
- Nodes: Make ScanRequest, Map of strings to attribute value structures variable for the expression attribute values map, Scan, Branch, Break ScanResult, For each loop, Break attribute Value map, and then the rest of nodes depend on the functionality you are attempting to implement, but for these blueprints we do include the nodes: Find (for X and Y in the table), then Break AttributeValue, convert from string to float, multiply to scale up to the map level size, make those X and Y values the vectors (for the robot 3D objects), convert the vector to transform and we used SpawnActor BP_Robot(custom blueprint class we made for the robot 3D objects) node to spawn in the robots at those specific transforms.
- Bp_robot (actor blueprint class): Just do this in the components section of the blueprint actor (robot here is the 3D object:



Simple Character & Camera Blueprints:

