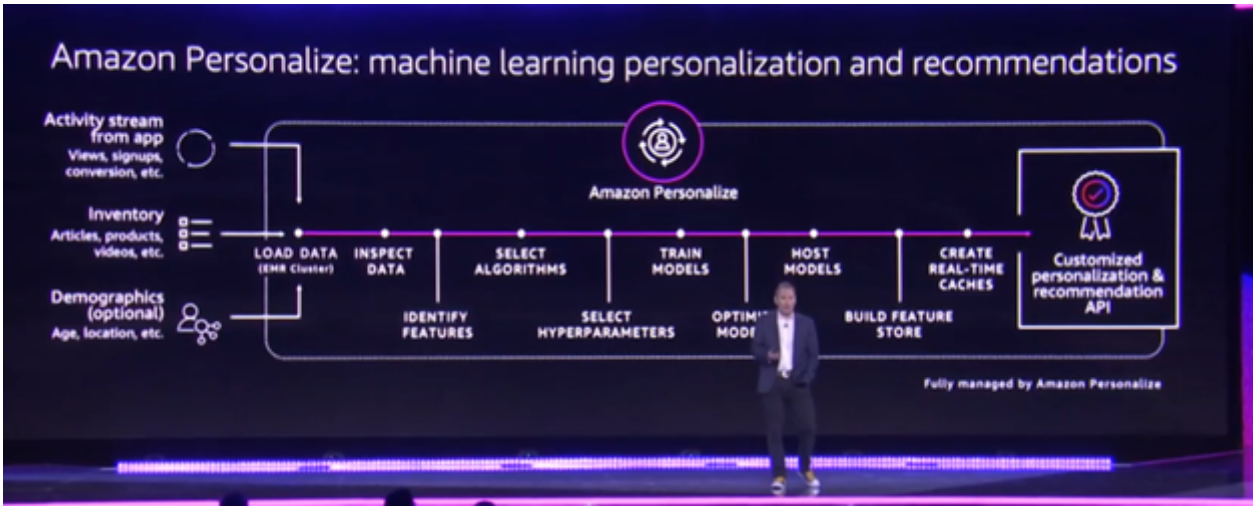
ADD RECOMMENDATIONS API

Use Amazon Personalize and AWS AppSync to create a product recommendation API

In this section, we are going to display a recommended product to our customers. Amazon Personalize is a service that trains a machine learning model based on your customized datasets to make recommendations for your end users and then creates an API to provide real-time recommendations to your end users.

As seen in this photo from re:invent 2018, Personalize handles all of the heavy lifting for creating a recommendation API.

- Loading and inspecting the data
- Selecting and maintaining algorithms
- Configuring and training the machine learning model
- Deploying an efficient, scalable inference endpoint



In another AWS account, there is an already provisioned instance of Amazon Personalize for Andy's pizza shop, which includes all the information needed to make recommendations. This includes:

- The datasets, which includes user, product and interaction data. The dataset include metadata that will allow varied recommendations based on characteristics just as age range, gender and family size. Our sample data provided over 100,000 user interactions!
- The solution, which is the part of Amazon Personlize that performs the model training.
- The campaign, which provides the API endpoint that can be used to retrieve recommendations for a user

> Note
>
> The reason that the Personalize service is already configured is that it takes about an hour for the Personalize service to inspect the data, train the model, and deploy an inference endpoint. This will vary based on the amount of data that is provided.

To add the recommendation API, we are actually going to front it with AWS AppSync. To do this, we will create a new schema type called Recommendations in our **schema.graphql** file. **APPEND** the following to the end of the schema.graphql file and **save schema.graphql**:

```
# Added for recommendations
type ModelRecommendationConnection {
  items: [Recommendation]
  nextToken: String
}

input ModelRecommendationFilterInput {
  userId: ModelStringFilterInput
}

input ModelStringFilterInput {
  eq: String
}

type Query {
  getRecommendations(
    filter: ModelRecommendationFilterInput
    limit: Int
    nextToken: String
  ): ModelRecommendationConnection @function(name:
"getrecommendation-${env}")
```

```
}

type Recommendation {
  userId: String!
  itemId: String!
  priority: Int
}
```

Note

In the above schema there is a new directive called **@function**. This tells amplify to "hook up" this query to a backend lambda function, which is what will actually make the call to the pre-configured Amazon Personalize API.

Next, we need to add the lambda function that will serve as our backend data source for the getRecommendations query. To do this, add a lambda function using **amplify add function**:

```
amplify add function
```

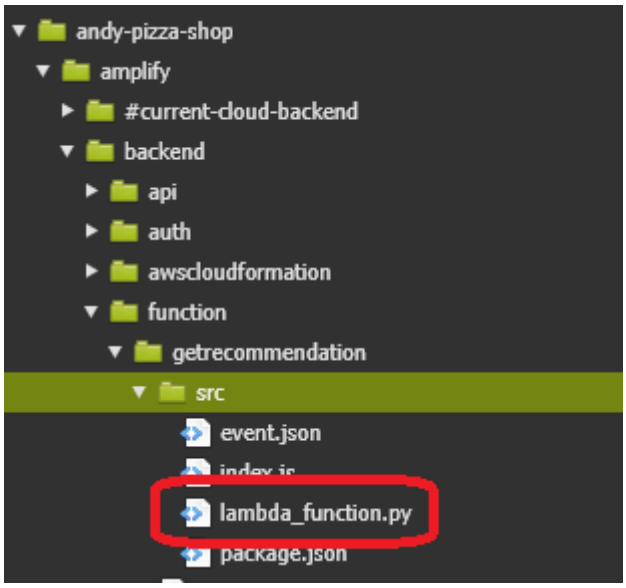
Enter the following information for your function:

- Friendly name for your resource: `getrecommendation`
- AWS Lambda Function name: `getrecommendation`
- Function template: Hello World function
- Access other resources: No
- Edit lambda function now: Yes

```
hoog:~/environment/andy-pizza-shop (master) $ amplify add function
Using service: Lambda, provided by: awscloudformation
? Provide a friendly name for your resource to be used as a label for this category in the project: getrecommendation
? Provide the AWS Lambda function name: getrecommendation
? Choose the function template that you want to use: Hello world function
? Do you want to access other resources created in this project from your Lambda function? No
? Do you want to edit the local lambda function now? Yes
Successfully added resource getrecommendation locally.
```

Next, we need to make some changes to the amplify directory that was just added. Under **amplify/backend/function/getrecommendation/src**, add a new file called

lambda_function.py



Add the following code in **lambda_function.py**.

```
import json
import boto3
import os

regionName = 'us-east-1'
PERSONALIZE_ACCOUNT_ID = "396459200938"
PERSONALIZE_CAMPAIGN_NAME = "camp-second-gen-pizza"
PERSONALIZE_ACCOUNT_ROLE = "PersonalizePizzaRole"
CAMPAIGN_ARN = "arn:aws:personalize:us-east-1:" + PERSONALIZE_ACCOUNT_ID
+ ":campaign/" + PERSONALIZE_CAMPAIGN_NAME

def getSession(accountId, orgRoleName):
    client = boto3.client('sts')
    response = client.assume_role(
        RoleArn='arn:aws:iam::' + accountId + ':role/' + orgRoleName,
        RoleSessionName='tempAccountEnableCloudTrail',
        DurationSeconds=900)
    accessKey = response['Credentials']['AccessKeyId']
    secretKey = response['Credentials']['SecretAccessKey']
    sessionToken = response['Credentials']['SessionToken']
    # Now create new session in account
    session = boto3.session.Session(
        aws_access_key_id=accessKey, aws_secret_access_key=secretKey,
aws_session_token=sessionToken, region_name=regionName)
    return session
```

```
def lambda_handler(event, context):
    # TODO implement
    print(event)
    userToGet = event["arguments"]["filter"]["userId"]["eq"]
    session = getSession(PERSONALIZE_ACCOUNT_ID,
PERSONALIZE_ACCOUNT_ROLE)
    personalizeRt = session.client('personalize-runtime')
    response = personalizeRt.get_recommendations(campaignArn =
CAMPAIGN_ARN, userId = userToGet)
    print("Recommended items")
    results = []
    priority = 0
    for item in response['itemList']:
        newItem = {
            "itemId": item['itemId'],
            "priority": priority,
            "userId": userToGet
        }
        results.append(newItem)
        priority = priority + 1
    print(results)
    fullResults = {
        "data": {
            "getRecommendations": {
                "items": results
            }
        }
    }
    return {
        "items": results
    }
```

Save the lambda_function.py file. Next, open the **amplify\backend\function\getrecommendation\getrecommendation-cloudformation-template.json** file and make these very specific changes:

1. Modify the **Handler** property to `lambda_function.lambda_handler` (~ line 28)

```
                                "Handler":
"lambda_function.lambda_handler",
```

Change the **Runtime** property to `python3.7` (~ line 60)

```
                "Runtime": "python3.7",
```

Under the **PolicyDocument** property, add another entry under **Statement** (~ line 119 - don't forget the leading **comma**!):

```
,
{
        "Effect": "Allow",
        "Action": "sts:AssumeRole",
        "Resource": "arn:aws:iam::396459200938:role/PersonalizePizzaRole"
}
```

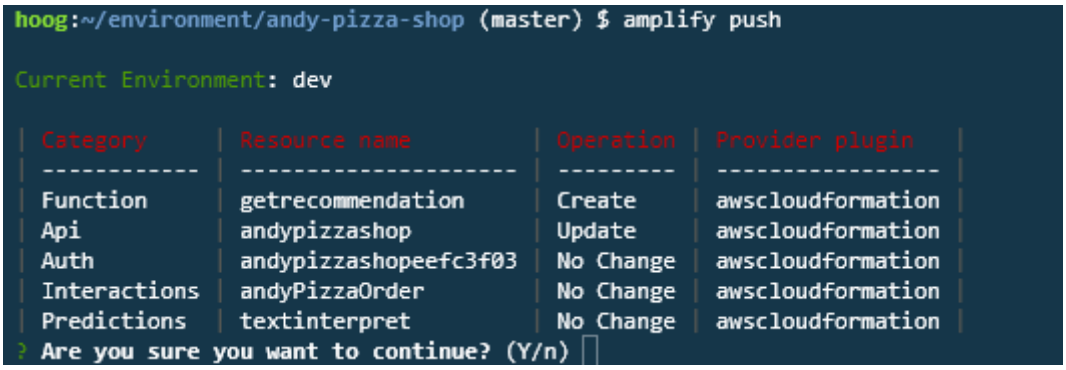
Save the **getrecommendation-cloudformation-template.json** file

Note

What are we doing in the above steps? Amplify creates a node.js function by default, but the lambda function we are providing is written in python. Therefore, we need to replace the default node.js function with a python 3.7 function.

Next, run **amplify push** to push our recommendation logic to AWS.

```
amplify push
```



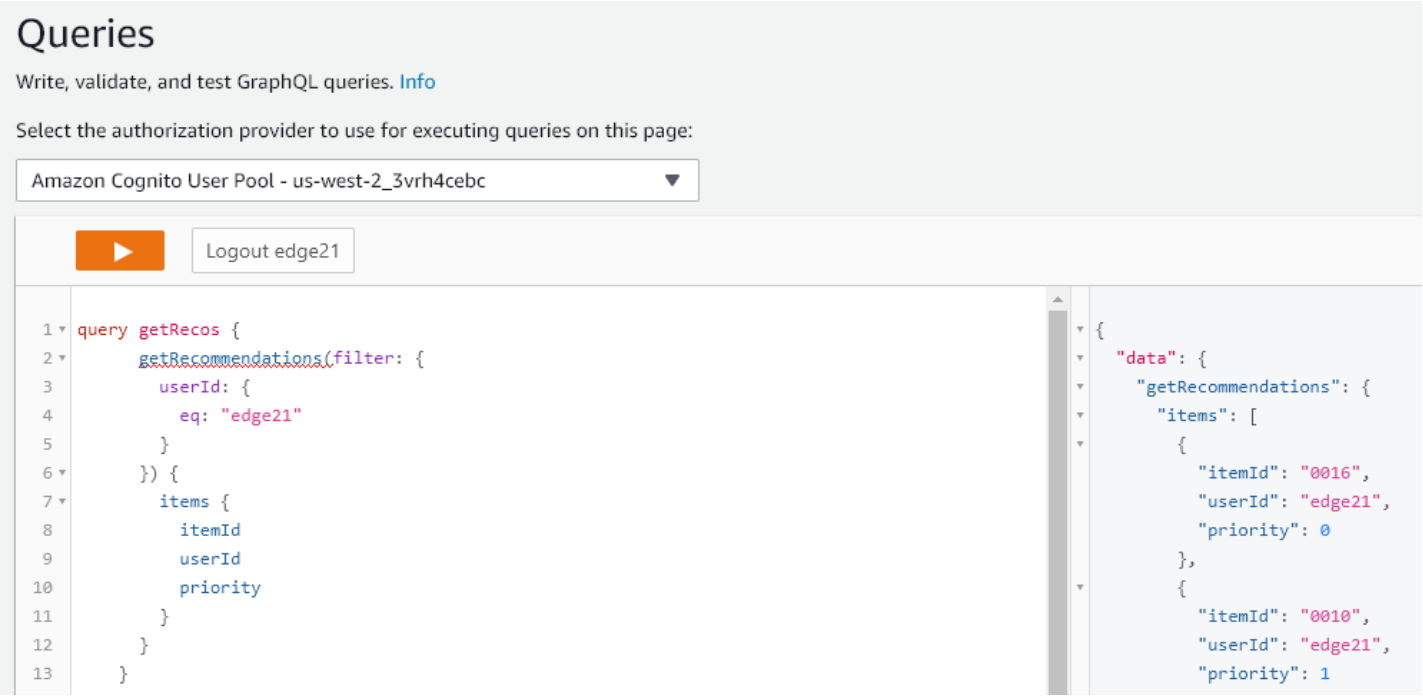
Use the default settings for generating code. You should receive a completion message when it is done deploying.

Test the getRecommendation query

After the deployment completes, head over to the AWS AppSync console and go to our Queries section for our pizza API. Enter the following query:

```
query getRecos {
      getRecommendations(filter: {
        userId: {
          eq: "edge21"
        }
      }) {
        items {
          itemId
          userId
          priority
        }
      }
    }
```

Now, you should receive back a list of recommended product ids. We are going to use this to display in our application.



If you try a few other usernames in the above query, such as `hair54` , `everybody63` and `subject91` you will receive differently ranked recommendations.

Note

There were over 3000 usernames used to generate recommendations on over 100,000 product interactions. These 4 usernames have different characteristics:

Username	Age	Gender	State	Family Size	Delivery Orders	Carryout Orders
edge21	19	M	Tennessee	1	20	0
everybody63	24	F	Nevada	1	8	13
subject91	56	F	Nebraska	5	6	15
hair54	26	M	Illinois	1	13	5

At this point, we have tested our API and are now ready to integrate this into our application.