Building with Amazon DynamoDB Tables

**SPL-TF-200-DBWDDT-1 - Version 1.0.6**

The lab dataset is being provided to you by permission of IMDb and is subject to the terms of the AWS Digital Training Agreement (available at <https://aws.amazon.com/training/digital-training-agreement>). You are expressly prohibited from copying, modifying, selling, exporting, or using this dataset in any way other than for the purpose of completing this lab.

© 2024 Amazon Web Services, Inc. or its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited. All trademarks are the property of their owners.

Note: Do not include any personal, identifying, or confidential information into the lab environment. Information entered may be visible to others.

Corrections, feedback, or other questions? Contact us at [*AWS Training and Certification*](https://support.aws.amazon.com/#/contacts/aws-training).

**Lab overview**

In this lab, you import the previously converted JSON data to Amazon DynamoDB. You run complex queries to meet your business objectives. You also investigate backup and performance monitoring options. Finally, you use an Amazon DynamoDB Accelerator (DAX) cluster to boost the performance of the DynamoDB read/write operations.

OBJECTIVES

After completing this lab, you will be able to:

* Load and verify data in DynamoDB
* Run complex queries to meet your objective
* Improving performance using DAX
* Review logging and monitoring options
* Restore service from backups

TECHNICAL KNOWLEDGE PREREQUISITES

To successfully complete this lab, you should be familiar with:

* Basic navigation of the AWS Management Console.
* Familiarity with SQL commands and basic operations
* Familiarity with the AWS Management Console

DURATION

This lab requires approximately *60* minutes to complete.

ICON KEY

Various icons are used throughout this lab to call attention to different types of instructions and notes. The following list explains the purpose for each icon:

* **Command:** A command that you must run.
* **Expected output:** A sample output that you can use to verify the output of a command or edited file.
* **Note:** A hint, tip, or important guidance.
* **Learn more:** Where to find more information.
* **Task complete:** A conclusion or summary point in the lab.

**Start lab**

1. To launch the lab, at the top of the page, choose **Start lab**.

**Caution:** You must wait for the provisioned AWS services to be ready before you can continue.

1. To open the lab, choose **Open Console**.

You are automatically signed in to the AWS Management Console in a new web browser tab.

**WARNING:** **Do not change the Region unless instructed.**

COMMON SIGN-IN ERRORS

**Error: You must first sign out**



If you see the message, **You must first log out before logging into a different AWS account:**

* Choose the **click here** link.
* Close your **Amazon Web Services Sign In** web browser tab and return to your initial lab page.
* Choose **Open Console** again.

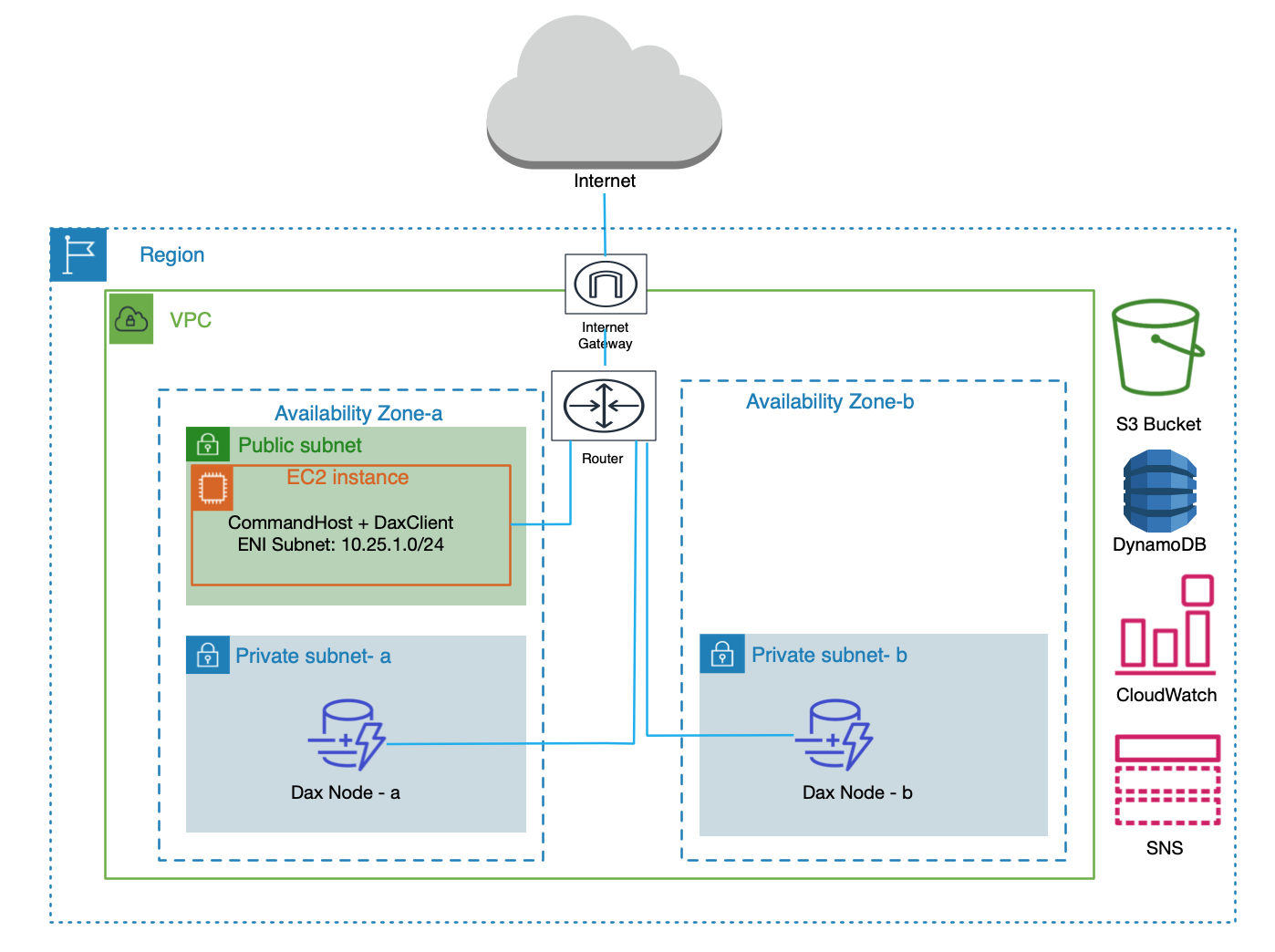
**Error: Choosing Start Lab has no effect**

In some cases, certain pop-up or script blocker web browser extensions might prevent the **Start Lab** button from working as intended. If you experience an issue starting the lab:

* Add the lab domain name to your pop-up or script blocker’s allow list or turn it off.
* Refresh the page and try again.

LAB ENVIRONMENT

As a part of the lab, the DAX cluster is provisioned for you. The accelerator nodes are placed in different Availability Zones inside private subnets. An Amazon EC2 instance, which acts as a CommandHost, is provisioned in a public subnet with access to the internet. To access the DynamoDB database, you must first connect to the CommandHost, then connect to the DynamoDB database. The CommandHost instance is provisioned with all the permissions and access credentials required to communicate with DynamoDB.



*Image description: The preceding diagram shows the setup of this lab, with the EC2 host and the DAX cluster in the Lab VPC, and access to other AWS services outside the VPC.*

SERVICES USED IN THIS LAB

In this lab, you investigate another AWS fully managed nonrelational database service: Amazon DynamoDB.

**Amazon DynamoDB**

DynamoDB is a key-value and document database that delivers single-digit millisecond performance at any scale. It’s a fully managed, multi-Region, multi-master, durable database with built-in security, backup and restore, and in-memory caching for internet-scale applications. DynamoDB can handle more than 10 trillion requests per day and can support peaks of more than 20 million requests per second.

**Amazon DynamoDB Accelerator (DAX):**

DAX is a fully managed, highly available, in-memory cache for DynamoDB that delivers up to a 10x performance improvement—from milliseconds to microseconds—even at millions of requests per second. DAX does all the heavy lifting required to add in-memory acceleration to your DynamoDB tables, without requiring developers to manage cache invalidation, data population, or cluster management.

All backend components, such as Amazon Elastic Compute Cloud (Amazon EC2) instances, AWS Identity and Access Management (IAM) roles, the DAX cluster, and other AWS services, have been built in your lab environment already. The lab contains instructions to review the code, configure the necessary variables, and run the scripts so that you can set up those components when necessary.

AWS SERVICES NOT USED IN THIS LAB

AWS services that are not used in this lab are disabled in the lab environment. In addition, the capabilities of the services used in this lab are limited to what the lab requires. Expect errors when accessing other services or performing actions beyond those provided in this lab guide.

**Task 1: Create a table and upload data**

In this task, you review the JSON object designed for you. You then create a new table and upload the data.

TASK 1.1: REVIEWING THE JSON DATA

Below is an example snippet of the JSON document that is used as the source for all data used in this lab. You import the contents of the JSON document to an DynamoDB table.

{

"genres": "Drama",

"title": "Auge um Auge",

"runtimeMinutes": 90,

"titleId": "tt0103735",

"cast": [

{

"category": "actress",

"birthPlace": "Ankara, Turkey",

"name": "Renan Demirkan",

"nicknames": "",

"gender": "female",

"birthDate": "12/6/1955",

"characters": "[\"Mona\"]",

"deathDate": "",

"akas": ""

},

{

"category": "actor",

"birthPlace": "Istanbul, Turkey",

"name": "Demir Gakgal",

"nicknames": "",

"gender": "male",

"birthDate": "15/7/1937",

"characters": "",

"deathDate": "22/3/2012",

"akas": "Demir Goekoel"

},

{

"category": "actor",

"birthPlace": "",

"name": "Andreas Pauls",

"nicknames": "",

"gender": "male",

"birthDate": "0/0/0",

"characters": "",

"deathDate": "",

"akas": ""

}

],

"year": 1992

}

TASK 1.2: CREATING THE DYNAMODB TABLE

1. Copy the **CommandHostSessionManagementUrl** value from the left side of the lab page, and paste it in a new browser tab. The terminal for the CommandHost instance opens.
2. **Command:** To access the **home** directory, run the following command:

cd ~

1. **Command:** To add the instance name to the command prompt to help you identify which instance you are connected to, run the following command:

source ~/.bashrc

Several files have already been loaded on the Amazon EC2 instance for this lab. The following command shows you a list of these files.

1. **Command:** Run the following command:

ls -ltrh

In the output, note the *cast\_full.json* file. This is where the IMDb data is stored.

This data must be imported to the database. In this lab, you use a DynamoDB database.

**Note:** The following script, *createUpdate.py*, imports data into the DynamoDB database. The script creates a table named **Cast** and assigns a partition key on the year column and the sort key on the title column.

from \_\_future\_\_ import print\_function # Python 2/3 compatibility

import boto3

import json

import decimal

import time

import os

count = 0

session = boto3.session.Session()

region = session.region\_name

dynamodb = boto3.resource('dynamodb', region\_name=region)

table\_name = 'Cast' # table name

pk = 'year' # primary key

sk = 'title' # sort key

file\_name = 'cast\_full.json'

def create\_table():

try:

table = dynamodb.create\_table(

TableName=table\_name,

KeySchema=[

{

'AttributeName': pk,

'KeyType': 'HASH' #Partition key

},

{

'AttributeName': sk,

'KeyType': 'RANGE' #Sort key

}

],

AttributeDefinitions=[

{

'AttributeName': 'year',

'AttributeType': 'N'

},

{

'AttributeName': 'title',

'AttributeType': 'S'

},

],

BillingMode='PAY\_PER\_REQUEST'

#ProvisionedThroughput={

# 'ReadCapacityUnits': 125,

# 'WriteCapacityUnits': 125

# }

)

print("Table status:", table.table\_status)

except:

print("Table exist:Uploading data")

table = dynamodb.Table('Cast')

def add\_table():

table = dynamodb.Table(table\_name)

count = 0

with open(file\_name) as json\_file:

movies = json.load(json\_file, parse\_float = decimal.Decimal)

with table.batch\_writer(overwrite\_by\_pkeys=[pk, sk]) as batch:

for movie in movies:

titleId = movie['titleId']

title = movie['title']

year = int(movie['year'])

genres = movie['genres']

runtimeMinutes = int(movie['runtimeMinutes'])

cast = movie['cast']

count = count + 1

print("Adding record count:", count)

batch.put\_item(

Item={

'titleId': titleId,

'year': year,

'title': title,

'genres': genres,

'runtimeMinutes': runtimeMinutes,

'cast': cast,

}

)

def main():

create\_table()

add\_table()

if \_\_name\_\_ == "\_\_main\_\_":

main()

Now you run the script to create the table and import the data.

1. **Command:** Run the following command:

python3 createUpdate.py

**Note:** Loading data to the database takes about 5 minutes. You can proceed to the next step once the upload starts, signified by multiple “Adding record count” messages.

The output from this script shows each record being added.

**Learn more:** DynamoDB SDKs are available in multiple programming languages. Refer to *Getting started with DynamoDB and the AWS SDKs* in the **Additional resources** section for more information.

TASK 1.3: VIEW THE NEW TABLE AND DATA IN THE DYNAMODB CONSOLE

1. From the AWS management console, at the top of the page, in the unified search bar, search for and choose

DynamoDB

.

1. In the left navigation pane, choose **Tables**.
2. Choose the **Cast** table link.
3. Choose **Explore table items** .

There are several items that appear.

1. Wait for the upload taking place on the command host to complete. The total count of records is 85,437.

**Task complete:** You have successfully created a table and uploaded data.

**Task 2: Run queries using the DynamoDB console, AWS CLI, and AWS SDK**

In this task, you use both scan and query operations to search for items in the table using the console.

TASK 2.1: CREATE A TABLE SCAN IN THE CONSOLE

A **scan** operation examines every item in a table to find items that match the filters you have set.

1. Return to the AWS management console where the table items from the **Cast** table are listed.
2. In the **Scan or query items** panel at the top, ensure that **Scan** is selected, and then expand the **Filters** section.
3. Enter or select the following values:

* **Attribute name** :

year

* **Type** : Number
* **Condition** : Equal to
* **Value** :

1994

1. Choose **Run** .

This search scans all items in the table but filters the results to titles that were released in 1994.

**Note** A “This table has more items to retrieve…” message appears with an option to choose a button to retrieve more items. This is because DynamoDB paginates the results of scan operations. Just ignore this message and button for now.

You can add additional filters to narrow the list of visible items. Add two more filters to limit the results to comedies that were 75 minutes in length.

**Note** Filter values are case sensitive.

1. Choose **Add filter** and enter or select the following values:

genres

 | **String** | **Contains** |

Comedy

1. Choose **Add filter** again, and enter or select the following values:

runtimeMinutes

 | **Number** | **Equal to** |

75

1. Choose **Run** .

When the search completes, the list is limited to titles from 1994 in the Comedy genre that are 75 minutes long.

**Note** This operation may take several minutes to complete, and it is likely to encounter a performance wall. If this happens, choose the **Continue operation** button to continue.

1. When the scan has completed, note the time required to finish the scan and record the value of the **Read capacity units consumed**.

**Learn more:** Read capacity units represent a measurement of how much work is required to complete a DynamoDB read operation. Refer to *Read/write capacity mode* in the **Additional resources** section for more information.

TASK 2.2: CREATE A TABLE QUERY IN THE CONSOLE

A **query** operation works differently from a scan. Instead of examining every item in a table, queries use indexes to find the exact items that match the requirements. The table’s partition key defines the base table index, and if a sort key is defined, it defines the sort order of items within each partition key. Indexes allow queries to run much faster than table scans.

In the following steps, you change the scan into a query. This allows you to take advantage of the index that has been created. You replace the filter you added on the year attribute.

1. In the **Scan or query items** panel at the top, select **Query**.
2. For **year (Partition key)**, enter

1994

, and leave the **title(Sort key)** value empty.

1. In the filters section, choose **Remove** next to the **year** filter.
2. Choose **Run** .
3. Compare the performance of the query to that of the scan, specifically noting the duration and the number of Read capacity units consumed.

TASK 2.3: CREATE A SCAN AND QUERY IN THE AWS CLI

You can use the AWS Command Line Interface (AWS CLI) to perform table scans and queries on the DynamoDB data.

1. Return to the browser tab with the **CommandHost** terminal window.

* **Note:** If you closed the previous tab or lost the connection, select [here](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBWDDT-1%3A1.0.6-a91c8925/en-US#reconnect) for directions on how to reconnect.

One of the tricks that makes navigating the code simpler is to use variables. For this task, the variable definitions are stored in two different JSON files, named *expression-attribute-names.json* and *expression-attribute-values.json* :

*expression-attribute-names.json*

{

"#YR": "year",

"#TT":"title"

}

This file defines two variables and maps them to attributes in the DynamoDB table.

*expression-attribute-values.json*

{

":a": {"S": "Sport"}

}

Attribute values are substitutes for the actual values that you want to compare because they are unknown at runtime. This file defines that **:a** is the variable name, and the value of the variable is **Sport**.

These two files are used in the following table scan statement. The statement using the AWS CLI scans the cast table, filters the items to the Sport genre, and projects the year and title attributes in the results.

1. **Command:** Run the following command:

aws dynamodb scan --table-name Cast \

--filter-expression "genres = :a" \

--projection-expression "#YR, #TT" \

--expression-attribute-names file://expression-attribute-names.json \

--expression-attribute-values file://expression-attribute-values.json --no-cli-pager

**Expected output:** The results of the scan appear in JSON format.

**Learn more:** The AWS CLI Command Reference contains instructions for managing all parameters related to scans. Refer to *AWS CLI Command Reference* in the **Additional resources** section for more information.

To read an item from a DynamoDB table, use the **GetItem** operation. You must provide the name of the table along with the partition key. You can include a sort key if one is defined.

The following command retrieves the movie titled **18 Shades of Dust**. The command projects the titleId, title, runtimeMinutes, genre, and a list of cast members for each result:

1. **Command:** Run the following command:

aws dynamodb get-item --table-name Cast \

--key '{"year":{"N": "1999"},"title":{"S":"18 Shades of Dust"}}' \

--expression-attribute-name '{"#c": "cast"}' \

--projection-expression "titleId, title, runtimeMinutes, genres, #c" --no-cli-pager

Note the line beginning with **–key** in the command. We could have used the **expression-attribute-names** and **expression-attribute-values** JSON files here as we did in the table scan.

**Learn more:** Expressions can be used in a varity of ways with DynamoDB. Refer to *Using expressions in DynamoDB* in the **Additional resources** section for more information.

TASK 2.4 CREATE A SCAN AND QUERY WITH THE AWS SDK

In this example, you use a Python script that has already been created for you. This script performs a table scan to return all movies released between 1990 and 1991 with a title that begins between A and D. The script then projects the list of actors for each movie.

1. Review the *scan.py* script:

from \_\_future\_\_ import print\_function # Python 2/3 compatibility

import boto3

import json

import decimal

from boto3.dynamodb.conditions import Key, Attr

table\_name = 'Cast'

pk = 'year'

sk = 'title'

session = boto3.session.Session()

region = session.region\_name

class DecimalEncoder(json.JSONEncoder):

def default(self, o):

if isinstance(o, decimal.Decimal):

if o % 1 > 0:

return float(o)

else:

return int(o)

return super(DecimalEncoder, self).default(o)

dynamodb = boto3.resource('dynamodb', region\_name=region)

table = dynamodb.Table(table\_name)

fe = Key(pk).between(1990, 1991) & Key(sk).between('A', 'D')

pe = "#yr, title, #ca"

ean = { "#yr": "year", "#ca": "cast",}

esk = None

response = table.scan(

FilterExpression=fe,

ProjectionExpression=pe,

ExpressionAttributeNames=ean

)

for i in response['Items']:

print(json.dumps(i, cls=DecimalEncoder))

while 'LastEvaluatedKey' in response:

response = table.scan(

ProjectionExpression=pe,

FilterExpression=fe,

ExpressionAttributeNames= ean,

ExclusiveStartKey=response['LastEvaluatedKey']

)

#parsing and printing the JSON response

for i in response['Items']:

print(i['year'], ":", i['title'] + " and the actors are:")

for j in i['cast']:

print(j['name'])

print('\n')

1. **Command:** Run the following command:

python3 scan.py

Notice that the output from the script is easier to read than the raw output from the **GetItem** operation.

In the next example, you use another Python script that has already been created for you. This script runs a table query to return all movies released in 2005 with a title that begins between A and L. The script then projects the list of actors for each movie.

Review the *query.py* code:

from \_\_future\_\_ import print\_function # Python 2/3 compatibility

import boto3

import json

import decimal

from boto3.dynamodb.conditions import Key, Attr

session = boto3.session.Session()

region = session.region\_name

table\_name = 'Cast'

pk = 'year'

sk = 'title'

class DecimalEncoder(json.JSONEncoder):

def default(self, o):

if isinstance(o, decimal.Decimal):

return str(o)

return super(DecimalEncoder, self).default(o)

dynamodb = boto3.resource('dynamodb', region\_name=region)

table = dynamodb.Table(table\_name)

print("Movies in 2005 - titles A-L, and list of actors")

response = table.query(

ProjectionExpression="#yr, title, #ca",

ExpressionAttributeNames={ "#yr": "year", "#ca": "cast" },

KeyConditionExpression=Key(pk).eq(2005) & Key(sk).between('A', 'L')

)

for i in response['Items']:

print(i['year'], ":", i['title'] + " and the actors are:")

for j in i['cast']:

print(j['name'])

print('\n')

1. **Command:** Run the following command:

python3 query.py

**Task complete:** You have successfully run queries using the DynamoDB console, AWS CLI, and AWS SDK.

**Task 3: Improve performance with DAX**

In this task, you run the same queries with and without Amazon DynamoDB Accelerator (DAX) to see the performance improvements that DAX can provide.

**Note:** The DAX client has already been installed on the CommandHost instance.

TASK 3.1: CREATE THE DYNAMODB TEST TABLE AND LOAD DATA

The following Python script creates a new table in the DynamoDB database.

1. **Command:** Run the following command:

python3 01-create-table.py

The following Python script loads data into the table.

1. **Command:** Run the following command:

python3 02-write-data.py

TASK 3.2: QUERY THE TABLE WITHOUT THE DAX CLIENT

The next set of commands are run using the DynamoDB endpoint. Later, you run the same commands using the DAX cluster to see the performance improvement.

1. **Command:** Run each of the following Python scripts one at a time. Each one returns the total and average script runtime. Make note of these times to use later.

python3 03-getitem-test.py

python3 04-query-test.py

python3 05-scan-test.py

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Total time: 33.068488121032715 sec - Avg time: 0.6613697624206543 sec

TASK 3.3: QUERY THE TABLE WITH THE DAX CLIENT

To use the DAX cluster, you must first identify the DAX cluster address and port. You can do that in the console or here in the AWS CLI.

1. **Command:** To return the DAX cluster endpoint, run the following command:

aws dax describe-clusters --query "Clusters[\*].ClusterDiscoveryEndpoint"

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[

{

"Address": "daxcluster.ocr9pt.dax-clusters.us-west-2.amazonaws.com",

"Port": 8111,

"URL": "dax://daxcluster.ocr9pt.dax-clusters.us-west-2.amazonaws.com"

}

]

1. Copy the **Address** value into a text editor to save the information for later.
2. Copy and paste the following commands to a text editor:

python3 03-getitem-test.py (DAX cluster address):8111

python3 04-query-test.py (DAX cluster address):8111

python3 05-scan-test.py (DAX cluster address):8111

1. In the text editor, replace the **(DAX cluster address)** placeholder in each command with the cluster address you saved.
2. Copy and paste each updated command, one at a time, from the text editor into the terminal window. Press ENTER to run each command.

Each command returns the total and average time the script took to run. Make note of these times.

**Note:** You may ignore the warning message *ANTLR runtime and generated code versions disagree: 4.9.3!=4.7.2* , after you ran the python command with the DAX Cluster id mentioned in it.

Compare the timing with and without DAX. Notice the significant reduction in run time for the get item tests run with DAX.

DAX provides access to eventually consistent data from DynamoDB tables. You may see performance gain if the tests are performed multiple times against DAX.

**Learn more:** This lab only used one of many DAX operations. Refer to *In-memory acceleration with DynamoDB Accelerator (DAX)* in the **Additional resources** section for more information.

TASK 3.4: DELETE THE TABLE

1. **Command:** To delete the TryDaxTable, run the following command:

python3 06-delete-table.py

**Task complete:** You have successfully improved performance with DAX.

**Task 4: Monitor logs and server metrics**

Monitoring is an important part of maintaining the reliability, availability, and performance of a DynamoDB database and all of the AWS services that DynamoDB interacts with. Collecting monitoring data from all services in your AWS solution helps you to more easily debug failures if they occur. AWS provides several tools for monitoring your DynamoDB resources and responding to potential incidents. These tools also include metrics for DAX.

**Tools for monitoring DynamoDB automatically**

* Amazon CloudWatch Alarms
* Amazon CloudWatch Logs
* Amazon CloudWatch Events
* AWS CloudTrail log monitoring

**Tools for monitoring DynamoDB manually**

* DynamoDB dashboard
* CloudWatch home page

In this task, you use several of these tools to determine the health of DynamoDB and DAX.

1. Return to the browser tab with the DynamoDB console view.
2. In the left navigation pane, choose **Dashboard**.

TASK 4.1 CHANGE CAPACITY SETTINGS

When your DynamoDB database is configured in on-demand capacity mode, you do not have to worry about capacity. However, if your database is stable, you may save money by switching to provisioned capacity mode. This section shows you how to switch to provisioned capacity mode and how to change the write capacity.

1. In the left navigation pane, choose **Tables**.
2. Choose the name of the **Cast** table, then choose the **Additional settings** tab.
3. In the **Read/write capacity** area, choose **Edit** .
4. Select **Provisioned**.
5. In the **Table capacity** section, under **Read capacity**, for **Auto scaling**, select **Off**.
6. For **Provisioned capacity units**, enter

10

.

1. In the **Table capacity** section, under **Write capacity**, for **Auto scaling**, select **Off**.
2. For **Provisioned capacity units**, enter

10

.

1. Choose **Save changes** .

TASK 4.2 CREATE A DATABASE ALARM

Next you create an alarm to notify you via email if the write capacity consumed reaches a specified value.

1. Choose the **Monitor** tab, then expand the **Alarms** section and choose **Manage in CloudWatch**
2. In the CloudWatch console tab that just opened, in the left navigation pane under **Alarms**, choose **All alarms**.
3. Choose **Create alarm** .
4. Choose **Select metric** .
5. In the **Metrics** section, choose **DynamoDB**.
6. Choose **Table Metrics**.
7. Select the checkbox for the row where **TableName** is **Cast** and **Metric Name** is **ConsumedWriteCapacityUnits**.
8. Choose **Select metric** .
9. Find the **Period** drop-down option and select **1 minute** from it.
10. In the **Conditions** section:

* For **Threshold type**, select **Static**
* For **Whenever ConsumedWriteCapacityUnits is…**, select **Greater/Equal**
* For **than…**, enter

5

1. Choose **Next** .
2. In the **Notification** section, uder **Send a notification to the following SNS topic**, select **Create new topic**.
3. For **Create a new topic…**, enter

dynamodb\_alert

.

1. For **Email endpoints that will receive the notification…**, enter an email address to which you have access.
2. Choose **Create topic** .
3. Go to your email service to find an email to confirm subscription to the alarm.
4. Open the email and choose **Confirm subscription**.
5. Return to the **CloudWatch** console.
6. Choose **Next** .
7. On the **Add name and description** page, enter the following:

* **Alarm name** : Enter

dynamodb\_alarm

* **Alarm description - optional** : Enter

Alarm created for catching write capacity consumption alerts for DynamoDB table.

1. Choose **Next** .
2. Choose **Create alarm** .

TASK 4.3: TRIGGER THE ALARM

1. Return to the tab with the **CommandHost** terminal window.

* **Note:** If you closed the previous tab or lost the connection, look [here](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBWDDT-1%3A1.0.6-a91c8925/en-US#reconnect) for directions on how to reconnect.

To trigger the alarm, you run a Python script to upload data to the **Cast** table. This creates enough traffic to trigger the alarm you just created.

1. **Command:** Run the following command. You do not need to wait for the entire upload to complete:

cd ~

python3 createUpdate.py

1. Return to the tab with the DynamoDB console.
2. In the left navigation pane, choose **Dashboard**.

You see the alarm you created. The **State** changes from *Ok* to *In alarm* when it is triggered. It may also start as *Insufficient data*. You may need to refresh the screen after a minute to see the alarm state change.

You should have also received an email from AWS Notifications about the alarm.

TASK 4.4: MONITOR DAX

1. In the left navigation pane, under **DAX**, choose **Clusters**.
2. Choose the name of the **daxcluster**, and then choose the **Monitor** tab.

Here you can monitor DAX metrics from the last hour up to the last two weeks.

**Learn more:** This lab only covers a couple of basic monitoring options. Refer to *Monitoring and logging* in the **Additional resources** section for more information.

**Task complete:** You have successfully monitored logs and server metrics.

**Task 5: Backup and restore a DynamoDB table**

DynamoDB provides on-demand backup capabilities. The service allows you to create full backups of your tables for long-term retention and archival for regulatory compliance needs. You can back up and restore your DynamoDB table anytime with a single action in the AWS Management Console or with a single API call. Backup and restore actions run with zero impact on table performance or availability.

In this task, you back up your current table. Once the backup is complete, you then delete the current table and restore service from the backup.

TASK 5.1: CREATE A BACKUP

1. In the DynamoDB console, in the left navigation pane, choose **Backups**.
2. Choose **Create backup** , then select **Create on-demand backup**.
3. Configure the following:

For **Source table:**, select **Cast**

For **Backup settings**, select **Customize settings** and:

* For **Backup management**, select **Backup with DynamoDB**
* For **Backup name**, enter

backup\_cast

1. Choose **Create backup** .

TASK 5.2: DELETE THE CAST TABLE AND RESTORE IT FROM BACKUP

1. In the left navigation pane, choose **Tables**.
2. Select the checkbox for the **Cast** table, then choose **Delete** .
3. In the pop-up message box, unselect *Delete all CloudWatch alarms for Cast*
4. Enter

confirm

 in the confirmation text box.

1. Choose **Delete** .

This process takes a minute or two to complete.

1. Wait until the table no longer appears in the **Tables** pane.
2. In the left navigation pane, choose **Backups**.
3. Select the checkbox for the **backup\_cast** backup, and choose **Restore** .
4. For **Name of restored table**, enter

Cast

1. At the bottom of the page, choose **Restore** .

You see the table being created from the backup.

**Learn more:** You can use the DynamoDB on-demand backup capability to create full backups of your tables for long-term retention, and archiving for regulatory compliance needs. Refer to *Using On-Demand backup and restore for DynamoDB* in the **Additional resources** section for more information.

**Task complete:** You have successfully backed up data and restored a DynamoDB table.

**Conclusion**

You have successfully done the following:

* Loaded and verified data in DynamoDB
* Run complex queries to meet your objective
* Reviewed logging and monitoring options
* Restored service from backups

**End lab**

Follow these steps to close the console and end your lab.

1. Return to the **AWS Management Console**.
2. At the upper-right corner of the page, choose **AWSLabsUser**, and then choose **Sign out**.
3. Choose **End lab** and then confirm that you want to end your lab.

**Additional resources**

* [Getting started with DynamoDB and the AWS SDKs](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStarted.html)
* [Read/write capacity mode](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ReadWriteCapacityMode.html)
* [AWS CLI Command Reference](https://docs.aws.amazon.com/cli/latest/reference/dynamodb/scan.html)
* [Using expressions in DynamoDB](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Expressions.html)
* [In-memory acceleration with DynamoDB Accelerator (DAX)](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DAX.html)
* [Monitoring and logging](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/monitoring.html)
* [Using On-Demand backup and restore for DynamoDB](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/BackupRestore.html)

**Appendix**

RECONNECTING TO THE COMMANDHOST

To reconnect to either CommandHost follow these steps:

* STEP 1: Copy the correct URL from the left of this lab and paste it in a new browser tab.
* **Command:** STEP 2: Copy and paste the following command to the terminal

cd ~

source ~/.bashrc

[Return to the instructions](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-DBWDDT-1%3A1.0.6-a91c8925/en-US#topinstructions)

For more information about AWS Training and Certification, see [*https://aws.amazon.com/training/*](https://aws.amazon.com/training/).

*Your feedback is welcome and appreciated.*  
If you would like to share any feedback, suggestions, or corrections, please provide the details in our [*AWS Training and Certification Contact Form*](https://support.aws.amazon.com/#/contacts/aws-training).