Project Milestone #4: Complete back-end by connecting to persistence layer

Re-submit Assignment

Due Apr 15 by 11:59pm **Points** 60 **Submitting** a file upload

Complete all of the functionality described in the **Course Project** overview (including the requirements for authentication tokens and the handling of passwords), using DynamoDB to persist data rather than using hard-coded dummy data. Your model classes are a good place to start when identifying the DynamoDB tables you will need to define.

In this milestone you must architect your system to meet the following performance requirements:

1. The perceived latency of the create new status operation (from the perspective of the author) is to be less than 1000 milliseconds.
2. When a new status is created, that status is visible in the feeds of all of the followers of the author within 120 seconds, for authors with up to 10K followers.
3. Each page of a user's feed is returned in less than 1000 milliseconds, from the perspective of the user.

To meet these performance requirements you are to do the following:

1. When a new status is posted, the feed of each follower is updated. (That is feeds are updated at write-time rather than assembled at read-time.)
2. Use two AWS SQS queues for asynchronously processing feed updates.
3. Your feed table is to be configured with no more than 100 WCUs.

We will discuss these techniques further in class.

Design

The back-end for your application will run in AWS Lambda. Some of this code will be triggered from the AWS API Gateway, other code will be triggered from AWS SQS. Your design should include the following layers:

- The Lambda handler
- A services layer to which the handler delegates each request
- A DAO layer which now interacts with your DynamoDB tables

In all of these layers, look for ways to avoid duplicating code. The Template Method or Strategy pattern may help.

Your implementation is to meet the "user and session management" requirements in the **project overview.**

Testing

The tests required for milestone 3 are still required for milestone 4 and should continue to pass.

DynamoDB Notes

DynamoDB Provisioned Capacity

- No matter what architecture you develop, your performance will be capped by the capacity you provision for writing to the feed table in DynamoDB.
- To learn about provisioned capacity for reads (RCUs), read the following: **https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ProvisionedThroughput.html#ItemSizeCalculations.Reads (https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ProvisionedThroughput.html#ItemSizeCalculations.Reads)**
- To learn about provisioned capacity for writes (WCUs), read the following: **https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ProvisionedThroughput.html#ItemSizeCalculations.Writes (https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ProvisionedThroughput.html#ItemSizeCalculations.Writes)**
- Batch writes will be more efficient than individual put-items, because you will have fewer network round trips. A batch-write operation is limited to 25 items. If you include 25 items, you will consume 25 write capacity units, as long as each item is no more than 1 KB in size.
- Updating the feeds when a status is posted by an author that has 10,000 followers will require writing 10,000 items, possibly in 400 batches (10000/25). To write that many items in 120 seconds will require WCU setting of around 10,000/120.

Populating Your Database With Test Data

For testing and passing-off, you will need to create ~10,000 test users and add data to your follows table such that you have at least one user with 10,000 or more followers. To do this, consider writing a script using the AWS CLI or a simple program using the AWS SDK API to create users and add followers to your follows table. The generated users need not have profile pictures, or they could all have the same profile picture (do whatever makes sense for your implementation).

When you run your script/program to populate your database, you will probably need to temporarily increase the WCU settings on your users and follows tables (e.g., to 100). Once the test users and followers have been created, decrease the WCU settings on these tables back to their original levels (e.g., to 1).

Minimizing AWS charges

There may be a small charge associated with this lab, but to minimize this consider the following:

- Turn up the DynamoDB WCUs for your feed table while testing and passing-off, but turn it down otherwise to avoid getting charged for capacity you are not using. Also, be sure to turn down the WCU settings on your users and follows tables after your test data has been generated (as previously mentioned).
- When a lambda trigger is associated with an SQS queue, AWS regularly polls the queue for new messages so it can call your lambda when new messages arrive. All of this polling incurs AWS charges. Therefore, when you are not actively testing or passing-off, disconnect your lambdas from your SQS queues (i.e., remove the lambda triggers in the SQS configuration). This will avoid incurring unnecessary charges

Pass off

When you join the queue to pass off, make sure to pull up your database on the AWS Console. Additionally, start your emulator and test making a post while you are in the queue. This is to make sure that your lambdas are up, loaded, and running.

Submission

- Pass off your project with a TA by the due date at 5pm
- Submit to Canvas a zip file containing your project in its current form
- Submit to Canvas a PDF file containing your milestone report which includes the following:
  - A description of each of your DynamoDB tables, that is to include: (1) the name and description of each table, (2) list your partition and sort keys, and (3) list any other fields .
  - A UML sequence diagram demonstrating what happens when a user sends a status. This sequence diagram should include front-end and back-end objects.

Rubric

[20 pts] For meeting the three performance requirements listed above, while using no more than 100 WCUs on the feed table. For full points, you must update feeds asynchronously when a status is created and must use SQS.

[20 pts] For DAOs that correctly write to DynamoDB, with profile pictures stored in S3.

[10 pts] For authentication and session management that meet requirements.

[5 pts] For database schema (based on submitted table descriptions.

[5 pts] For sequence diagram.

Related Content

Some gotchas for AWS and $$$$ and tips to avoid getting charged (middle of the page)

DynamoDB example code (literally copy and pasteable)

Maven Dependencies that are needed: `com.amazonaws:aws-java-sdk-core:1.11.547` and `com.amazonaws:aws-java-sdk-dynamodb:1.11.547`

M4 Passoff

Criteria	Ratings		Pts
Feed table updated in 120 seconds	**10.0 pts** **Full Marks**	**0.0 pts** **No Marks**	10.0 pts
WCU's <= 100	**5.0 pts** **Full Marks**	**0.0 pts** **No Marks**	5.0 pts
SQS used properly	**4.0 pts** **Full Marks**	**0.0 pts** **No Marks**	4.0 pts
User latency for paging and sending status <= 1K milliseconds	**1.0 pts** **Full Marks**	**0.0 pts** **No Marks**	1.0 pts
DAOs for expected tables There should be a one to one mapping from DAO to table, and vice versa.	**16.0 pts** **Full Marks**	**0.0 pts** **No Marks**	16.0 pts
Single S3 access point	**4.0 pts** **Full Marks**	**0.0 pts** **No Marks**	4.0 pts
No plaintext passwords in table / passwords are hashed	**4.0 pts** **Full Marks**	**0.0 pts** **No Marks**	4.0 pts
Handle token timeout	**2.0 pts** **Full Marks**	**0.0 pts** **No Marks**	2.0 pts
Username and Password work as expected Incorrect credentials are denied access, but correct credentials log in as expected.	**4.0 pts** **Full Marks**	**0.0 pts** **No Marks**	4.0 pts
Expected tables present in schema	**3.0 pts** **Full Marks**	**0.0 pts** **No Marks**	3.0 pts
Tables have right keys	**2.0 pts** **Full Marks**	**0.0 pts** **No Marks**	2.0 pts
Sequence diagram is clear, matches code structure, and is syntactically correct	**5.0 pts** **Full Marks**	**0.0 pts** **No Marks**	5.0 pts
			Total Points: 60.0