Amazon's DynamoDB

160

Evaluate Amazon's DynamoDB as a back-end for a web application.

Using DynamoDB

In this part of the Project 3, you will get hands-on experience using AWS DynamoDB, which is a NoSQL distributed database, similar to HBase or Cassandra. As a NoSQL Database, DynamoDB's functionality is limited compared to SQL databases- this is to ensure consistently high performance at very large scales. As an application developer using DynamoDB, your only parameters are how much read and write throughput your application needs- you do not have to worry about scaling, tuning, replication, etc.

In this part of the project, we will build a web application that retrieves images which have been categorized into different types of objects. This is useful for image processing and computer vision algorithms. We take a commonly used approach - store the **image location** into the database, not the image itself. We will store the images on S3, because it is easy to link directly to those files and it scales well.

The image dataset we will be using is called Caltech-256. The website for it is here (there is also a paper that describes it in more detail here.) It has 30,607 images separated into 256 categories (plus one "clutter" category). Each category has at least 80 images. Please note that you do not have to download or upload the dataset- we have already placed all the images into our S3 course bucket and made them public.

Note: For this checkpoint, assign the tag with Key: `Project` and Value: `3.4` for all resources

Task 1 Your first task is to create a DynamoDB database that contains a list of all the images and their locations on S3. You can create the initial Table either through the AWS web console, or through the AWS API. Think about costs and throughput as you are writing your code (see hints below). Populate the table using a program written in any language that supports the DynamoDB API that will upload all of the items from the CSV file here. The DynamoDB table should have the following fields:

- **Category**
  - Use this field as your **hash key**.
  - Type= `STRING`
  - There are 275 unique categories, please find the full list here.
- **Picture**
  - Use this field as your **range key**.
  - Type=`Number`
  - This is the index of a picture within a category, it starts at 0 and ends anywhere between 80 and 827, depending on the category.
- **S3URL**

- Type=`STRING`
- Web-Accessible S3 URL of the picture

Task 2 Your next task is to connect a web application to DynamoDB. The AMI (`ami-acf963c5`) has a functional Apache, PHP and AWS SDK for PHP installed, along with a pre-written web application located in (`/var/www`). Use the `t1.micro` instance type to launch an instance using this AMI.

Once you have finished testing the webpage, complete the checkpoint Quiz below.

NOTE: Unlike the previous projects, you will be required to leave the instance with the webapp and your DynamoDB database **RUNNING** so that we may grade them. After we complete grading, we shall send you instructions to shut down your instance and dynamodb. Please set the DynamoDB read throughput to 2 and write throughput to 1 after you have finished coding and testing, to minimize costs.

Bonus Part (For extra credit)

For bonus credit, we encourage you to improve the web interface for this application. The Caltech-256 dataset has a nice "taxonomy" image here. Your task is to improve the given HTML/PHP image display page, so that the users do not have to know the exact names of all the categories. You can create multiple dynamically generated drop-down menus, depending on what they are looking for, or dynamically generated links, or something else (it just has to be dynamically generated, so that changes in your DynamoDB Table would show up immediately).

For example, on the web page, a user would first only see the options of **Animate** or **Inanimate**. If they choose Animate, then they see another drop-down menu of **{animal, extinct, human, insects, mythological, plant}**, then they choose **plant**, and see **{trees, plants, flowers, fruits & vegetables}**, then they choose **plants** and finally see **{bonsai, cactus, fern}**.

One approach could be to create a new Table in DynamoDB, where each Item is keyed on "super"-category name has an Attribute which is a String-Set of all of its subcategories. You can use a small portion of the “super categories” if that's too time consuming to type them all in.

You can extend the PHP code on the AMI, or you can create a JSP page using the AWS Java SDK, or if you have experience with Ruby-on-Rails, there is an AWS SDK for Ruby as well. JSP and Ruby will require you to install/configure additional software on your Instance. If you want to use .NET, please post a private request to Piazza.

Since the webapp for the non-bonus part of the project is located in `/index.html`, create a different HTML file or subdirectory other than `/index.html` for the bonus part of the project.

Notes and Hints

1. **Items** are similar to rows in a relational database like MySQL.
2. **Attributes** are like columns in MySQL
3. DynamoDB offers two types of keys for tables - one that has a single Hash Key which must be unique, and one that uses a combined Key + Range Key, where the combination is unique.
    a. Range keys are useful for time-series databases. Also for our application because of several images in the same category.
4. DynamoDB charges you for throughput values as well as the size of your data
    a. DynamoDB pricing info
    b. Units are kB reads/second and kB writes/second.
    c. Writes cost 5x more than reads.
    d. You are also charged storage for 100 bytes of Indexing per Item.
    e. You can increase throughput values an unlimited number of times, but each request can be a maximum of 2x the current value. This take a few minutes to complete

f. Decreasing DynamoDB throughput can be done only 2 times a day (from Midnight to Midnight UTC). These requests take about 30 seconds to complete.

5. Like Cassandra, DynamoDB offers read-level consistency options. Eventually consistent reads cost 0.5 units while strongly consistent reads cost 1 unit. The default consistency model for reads is eventual. With strongly consistent reads, any pending writes will have to finish first before the read operation is serviced.
6. DynamoDB limits the size of an item to 64 kB. This includes attribute names and values.
7. The DynamoDB **scan** function is expensive (both in time and cost), because it has to read all the data in the range of the scan. There is a limit of 1MB per return value, so you might have to call the function many times to find what you are looking for. Scans support filtering for both items and attributes, but you will still be charged for the entire data read in the scan.
8. Please refer to the DynamoDB FAQ, client Java API Reference, Python API Reference and PHP API Reference for additional information.
9. You have two options to allow the Instance to get access to your DynamoDB Table(s).
   - The first option (recommended) - create an IAM role for access, then choose that Role while launching the instance. Here is an example policy that grants all DynamoDB function calls to any of your Tables:

```
{
    "Statement": [
        {
            "Action": [
            "dynamodb:*"
            ],
            "Effect": "Allow",
            "Resource": [
            "*"
            ]
        }
    ]
}
```

     A more complex example can be found here.
   - Otherwise, SSH to your Instance, and edit the file `/var/www/config.php`, and fill in your account values for `key` and `secret`.

checkpoint

Amazon DynamoDB