**Program 4 Report**

**Location URL**

The location of the service is located at:

http://tutflask-env.eba-zp74vgtn.us-east-2.elasticbeanstalk.com

Note that the current state of the service holds no values until data is loaded. The data is loaded from the S3 link provided (i.e., https://s3-us-west-2.amazonaws.com/css490/input.txt)

**Location of Object**

The location of the object is located at:

<https://testbucket436.s3.us-east-2.amazonaws.com/testfile.txt>

Note that the current state of the service does not have test file generated until data is loaded.

**Implementation**

The program uses Flask, a micro web framework for Python, to deploy a python service to AWS elastic beanstalk environment. Part of this service includes html files, CSS files, and a credential file. The html and CSS files work hand in hand to display the service and accept inputs for Flask to compute. The computation file is used by Flask to generate credentials required for Flask computation (i.e., access key, secret access key, region name) to AWS. Flask’s computation includes the following:

* Load: Loads data from a hardcoded link of a provided test file to DynamoDB. For data to be accepted, it must be the following:
  + lastName firstName [attribute1=value1 attribute2=value222.]
  + lastName firstName

Please note the following:

* + The service accepts data regardless of how many tabs spaces are placed between each attribute.
  + The service does not accept data that does not include a first name and a last name.
  + For data that includes an empty equal sign (i.e., attribute1=), the service accepts this as an attribute with no value.
  + If the equal sign does not exist, then this attribute will be ignored (i.e., attribute1).
  + For existing data, they will be overwritten. The service acknowledges the data loaded by displaying the following message in the output box:
    - “Data Loaded”

In addition to loading to DynamoDB, the service also loads the data as an object to S3. Note that the S3 object is overwritten every time data is loaded.

* Query: Scans a page from the DynamoDB using input as a filter. The scan presents the following pattern:
  + Accepts firstname and lastname to return data
  + Accepts firstname or lastname to return data
  + Accepts empty input to return the message:
    - “Please Enter First Name and Last Name to Get Data!”
  + Accepts bad input to return the message:
    - “No Data Found”
* Clear
  + The service clears all data in DynamoDB and deletes the S3 object. Note that the service assumes that data given to DynamoDB will not exceed a page, which is also another assumption used for scanning. Otherwise, data might not be fully deleted or queried.

**Deployment**

The service is deployed using the following AWS services: S3, DynamoDB, and elastic beanstalk. The service uses a public S3 to store loaded objects. The service uses DynamoDB to store the loaded data according to their keys so that they can be queried through a scan. The service uses Elastic beanstalk to deploy flask services by handling any required infrastructures and management complexity. Thus, when flask is deployed, the requirement is to only specify what you are deploying and a zip file of the flask service. Although not clearly revealed during deployment, elastic beanstalk uses the CloudWatch, S3, Auto Scaling, Load Balancing, and VPC. These AWS services serve following purpose:

* CloudWatch – Monitors AWS resources to examine the health of the deployed services
* S3 – Archives the different versions of the Flask service
* Auto Scaling – Defines a collection of Flask service instances that are treated as a group for scaling and management.
* Elastic Load Balancer – Distributes incoming traffic to Flask service(s) according to the auto scaling group.
* VPC – Creates a virtual networking environment for the Flask service

Figure 1 describes how the listed services work together to create and interact with the Flask service.

Figure 1: Deployment Diagram of Flask in AWS



**Scaling**

To scale the Flask service, configurations must be placed to scale S3 (test file bucket), DynamoDB, and elastic beanstalk. To scale S3, Amazon has stated that S3 automatically scales to high request rates. This fact can be seen from the following developer guide:

* https://docs.aws.amazon.com/AmazonS3/latest/dev/optimizing-performance.html

However, the guide does not guarantee that responses during high request rates will be fast. Thus, Amazon recommends the use of Elasticache to cache pages in memory. This recommendation is placed to handle the speed of responding during high request rates. For the Flask service, this recommendation is not implemented as large data will not be sent to the service. If large amount of data is sent, with a notice in advance, then this feature will be activated.

To scale DynamoDB, auto scaling feature can be implemented to increase or decrease read and write capacity through set alarms, in CloudWatch. This is done by defining the target utilization and a range of provisioned capacity. The target utilization specifies the buffer between consumed capacity units and provisioned units. Thus, the smaller the value of target utilization, the closer the consumed capacity will be to the provisioned units. This specifies how much capacity should be provisioned according to the capacity units consumed. The range of provisioned capacity specifies simply sets a minimum and maximum provisioned capacity regardless of how much load DynamoDB is handling. This puts a limit on how the DynamoDB can scale. For the Flask service, auto scaling features are turned off for the following reasons:

* Avoid unnecessary charges
* It is guaranteed that large amount of data will not be used.

If large amount of data is sent, with a notice in advance, then this feature will be activated.

To scale elastic beanstalk, auto scaling feature can be implemented to increase the number of instances running according to network behavior from CloudWatch. This is done by defining the number of instances created or deleted when under or above a specified threshold. The feature can also specify the types of instances that are created from the feature. Currently, this feature is used with default values which are to increment and decrement instances by 1 with the upper threshold of 6000000 and lower threshold of 2000000.

**Monitoring**

Monitoring of the service can be accomplished by using CloudWatch. CloudWatch monitors and provides data for all of the AWS services used by the Flask service, which includes elastic beanstalk, S3, and DynamoDB. The current configuration of CloudWatch displays the following metric: alerts, and metrics regarding NetworkIO for Ec2, DiskIO for EC2, CPUUsage for EC2, S3 size, DynamoDB write capacity, and DynamoDB read capacity. The use and visualization of these metrics can be seen from the figures below:

Figure 1: CloudWatch Dashboard

**Graphical user interface, application, Word

Description automatically generated**

Figure 2: CloudWatch Alert

**Graphical user interface, application

Description automatically generated**

Figure 3: CloudWatch Metrics

**A picture containing chart

Description automatically generated**

**Service-Level Agreement**

To calculate the service level agreement of the Flask service, the availability of the AWS services used for deployment and AWS services used for writing, reading, and storing objects must be examined. Notice that AWS services that are not directly used by the flask service are not considered (i.e., CloudWatch). Thus, availability of AWS services includes:

* EC2 (from Elastic Beanstalk): 99.99%
  + Source
    - https://aws.amazon.com/compute/sla/
* S3: 99.9%
  + Source
    - https://aws.amazon.com/s3/sla/
* DynamoDB: 99.99%
  + This is assuming the use of standard tables, not global tables
  + Source
    - <https://aws.amazon.com/dynamodb/sla/>

With the availability of each relevant AWS services identified, the Flask service must have the following SLA: 0.9999 \* 0.999 \* 0.9999 = 0.9988, **which is availability of 99.88%**