October 9, 2014

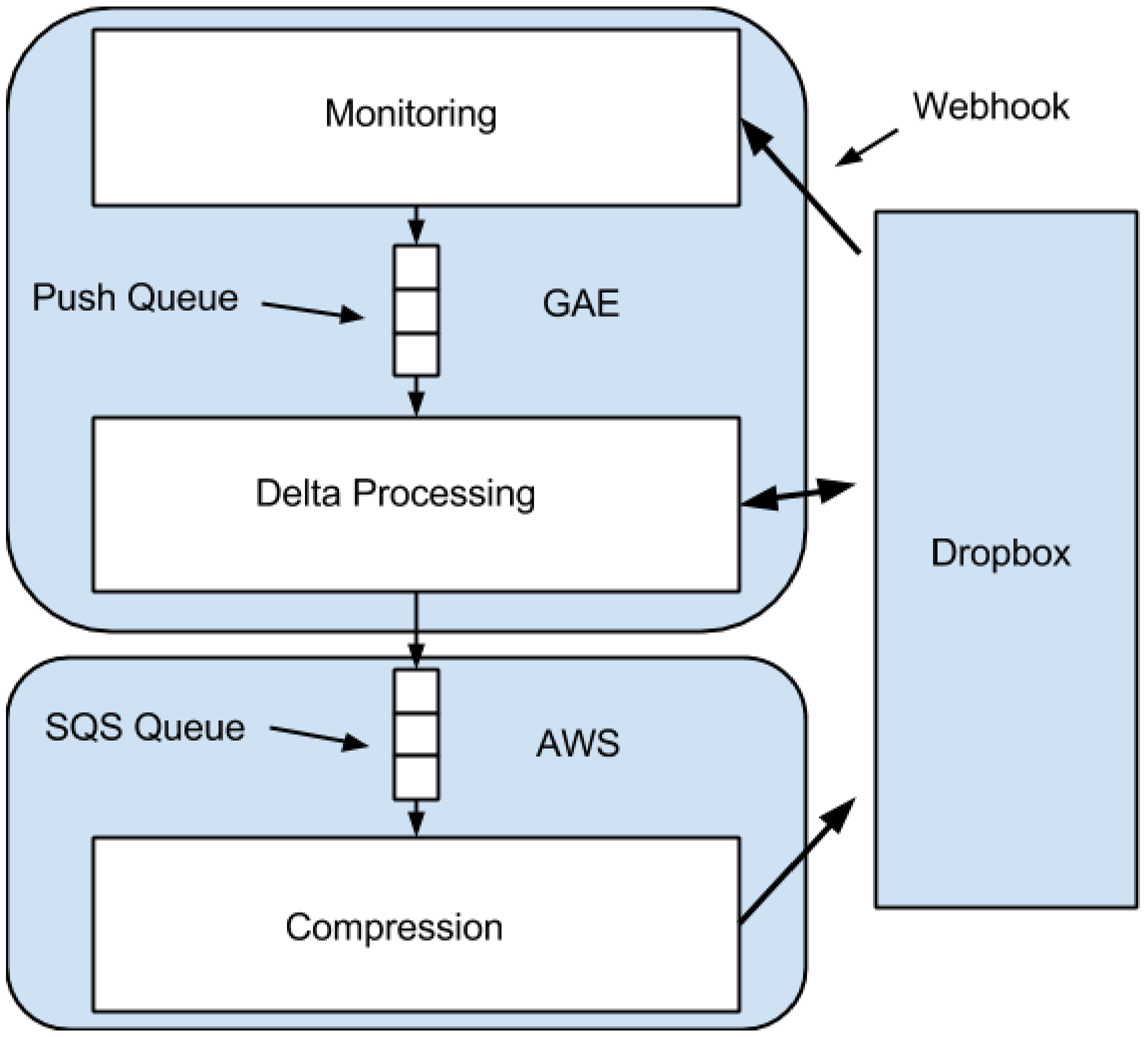
CSC 346 : Assignment 4

File Compression Service

Milestone Due: Monday October 20 at 11:59pm
Final Version Due: Thursday October 23 at 11:59pm

Overview

For this assignment you will create Dropbox App-based service that automatically compresses files stored in Dropbox. Gzip compression is used so that the file can be uncompressed using gunzip. This service will be implemented using GAE, GAE Push Queues, AWS EC2, AWS SQS, and Dropbox App as illustrated in the following diagram:



Monitoring Tier

The Monitoring Tier (MT) is a GAE application that is responsible for monitoring the Dropbox contents and causing new or modified files to be compressed. It does this by receiving webhook notifications from Dropbox. It must implement both the GET and POST methods as shown in the Dropbox webhook tutorial (https://www.dropbox.com/developers/webhooks/tutorial), and it must also validate the webhook invocation as shown in the tutorial. When a webhook notification is received the MT enqueues a message to the Delta Processing tier using a GAE push queue. This will cause the Delta Processing Tier to fetch the deltas from Dropbox and process them.

Delta Processing Tier

The Delta Processing Tier (DPT) is invoked when the MT enqueues a message in the GAE push queue. It is implemented in the same GAE application as the Monitoring tier (as shown in the GAE Push Queue demo in lecture). The DPT must validate that the invocation is from the GAE Push Queue (by checking for the

X-AppEngine-QueueName header also as shown in the demo). The DPT performs the following operations:

- Use the current cursor to receive new deltas from Dropbox. Unlike the previous assignment, the cursor cannot be stored in a global variable because GAE may start multiple instances of the DPT. Instead, the cursor must be stored in a location that is accessible by all the instances. We will solve this problem by storing the cursor in a Dropbox file named '/.cursor'. As a result, the DPT must first read the cursor from the cursor file (if it exists), and write the current cursor back to the file after processing all deltas. Note that there is a feedback loop here -- processing deltas will cause the cursor file to change which will cause a new delta to be created which will cause the cursor file to change, etc. Break the loop by not updating the cursor file if the only delta is for the cursor file itself. Also note that the same deltas may be processed multiple times (how might this happen?) but also note that it is ok to compress the same file more than once.

- If the deltas indicate that a file has been created or modified and the file is not already compressed (it does not end with .gz), then enqueue a message to the Compression Tier via the AWS SQS queue. The message contains the name of the file to compress.

The DPT should create the SQS queue if it does not already exist.

Compression Tier

The Compression Tier (CT) is a Python program that receives messages from an AWS SQS queue (see http://boto.readthedocs.org/en/latest/ref/sqs.html for Boto SQS documentation). It runs in an EC2 instance. Each message contains the name of a file to be compressed. In response to receiving a message the CT reads (using get_file()) the indicated file (if it exists (why might it not exist?)) and produces a compressed version of the file (using put_file()). The compressed version is in gzip format and is produced using the Python gzip module. The name of the compressed file is the name of the uncompressed file with the suffix ".gz" (e.g. foo.txt becomes foo.txt.gz).

The CT should create the SQS queue if it does not already exist.

Configuration

The tiers are configured via a settings.cfg file with the following contents:

```
[Dropbox]
token: <token>
secret: <secret>

[AWS]
region: <AWS region to use>
```

queue: <name of SQS queue>

Development and Testing

Your service must run its MT and DPT tiers in GAE and its CT tier in EC2. You must test with at least two EC2 instances running the CT.

Be sure to test your tiers in isolation before composing them. For example, you could first implement the CT as a stand-alone program that takes the file name as a command-line argument. Then you could integrate it with SQS and write a small testing program to put the messages in the queue. You can then verify that multiple CT instances receive the messages from the queue correctly compress the Dropbox files.

You may use Python and/or Java in your implementation.

Milestone

The milestone consists of three simplified versions of the three tiers that do not use Dropbox. The tiers perform the following actions:

- MT - when the webhook is invoked sends a message consisting of the current time to the DPT via the push queue.
- DPT - forwards messages from the MT to the CT via the SQS queue.
- CT - receives messages from the SQS queue and prints them.

Turnin

- Turn in only the files you wrote or modified for this assignment to the appropriate D2L dropbox. Make sure every file contains your name and is well-documented to facilitate partial credit. Include a README.txt file that explains how to install your solution in GAE and EC2. Do not turn in all files, and especially do not turn in your settings.cfg file.