Ji Kang

Documentation on program 3

**Architecture and design:**

- A backup program will only be run periodically for the everyday user so this program was written to be command line executable without querying the user for input.

- It utilizes the boto3 and botocore libraries from AWS to GET/PUT objects for comparison as well as handling any exceptions thrown via the botocore.exceptions.<error> library.

- The ‘os’ and ‘time’ imports are used to navigate through the current directory as well as get file’s last modified date for comparison with S3 objects in a given bucket.

- ‘argparse’ is used for the optional flags for this program which is explained in the build and run instructions text document located in the same directory.

- Written in Python3

The program with argparser will set a bucket name and region either through optional flags or by default values in argparse. By default, the program’s default bucket to create/upload to is ‘jikangprog490test’ and the default region is ‘us-west-2’. These can be changed based on the optional flags set. I utilized us-west-2 since it’s just the region that’s closest to us.

The program is primarily exception-driven. It’ll do a head\_bucket call to the specific bucket name to check if it exists. If not, it’ll throw a botocore.exceptions.ClientError (Error = “404”) so then the program knows the bucket doesn’t exist and attempts to create it in the user’s place. If a Error = “403” is thrown, it means the bucket exists but the user’s credentials don’t allow them access to the bucket.

If the program needs to create the bucket first, it’ll call the s3 resource object’s create\_bucket method in a try-except block. Program accomplishes this in **def create\_S3Bucket().** Four things may happen here:

1. The bucket is successfully created then the program can start comparing files for back up.

2. OperationAborted error code is thrown. This is typically thrown when a bucket name is available and valid but a previous bucket with the same name was recently deleted and AWS is in the process of cleaning it up.

3. “InvalidBucketName” error is thrown where the bucket name specified is invalid according to AWS’ guidelines. It’ll print out a error message to the user and direct them to guidelines on naming conventions.

4. “IllegalLocationConstraintError” which essentially means the location region entered is not valid. It’ll print out an error message to the user and direct them to the region table on the AWS website.

Once a existing bucket is found/created and the user has access to them, the program still continues to be exception-driven. Utilizing ‘os’, it’ll get the current working directory AKA the directory the program is being run in and the directory whose files and sub directories are being backed up. It’ll first call **get\_path()** which will get the current working directory and send that and a blank string as arguments to the recursive method **backup\_files(path, subdir)**. The blank string represents the current sub directory structure within this directory. The program will go through each file name, check if its a directory. If so, it’ll concatenate the sub directory name onto the subdir variable and the file name onto the filepath and recursively call itself with these new values. If the file is a normal file, it’ll generate a AWS S3 key value for that file utilizing the subdir variable. This is how the program ensures the sub-directory structure of the local directory is still upheld in the S3 bucket. Utilizing this file key, the program does a s3 resource object load() which does a head\_object call to check if the corresponding object with the same key exists within the same bucket. If so, it’ll get its meta data (mod\_date) which was set on a previous back up. It’ll compare this with the file’s current last modified date. If the S3 bucket’s objects mod date is lower, then we replace it with this updated copy.

If the head\_object call via load() did not find a corresponding object, it’ll throw a 404 error code so we know the object does not exist and we can just upload the files that this error was thrown for. When the file is uploaded, an additional meta data field is added that’s the same as the local file’s last modified date.

The recursive call ends since it iterates only through as many files as there are total in a given directory. By the end, any modified files have been changed in the bucket as well. Currently, the program can only do bulk deletions of files that aren’t coherent. At the end of execution, the program will do a check for any deleted files on the local machine that aren’t coherent with the user’s bucket. It’ll give the user an option to delete them or not.

Program also accounts for the case that the user’s internet may be down or for some very low chance reason, that S3 is down utilizing the UserEndPoint errors. It has implemented the retry logic for such a case like this.

**How I Tested:**

Directions on running the program are located in the build and run instructions document in the same directory.

I would start off small by uploading a single or handful of files. Changing some of them by modifying some text, perhaps one character. Just enough so the last modified date would change. I would then rerun the program on the same bucket with print statements showing me when a file was being updated or uploaded. Since I knew which files I had changed, I knew which print statements should show a upload vs. update. Once I got this working for just a single directory, I started adding subdirectories with the same changes. One update was uploading the original ThreadOS folder then swapping it out for a changed version on my local machine. (Assignments 1 and 2 in CSS430 required changing Shell.java and Scheduler.java). I reran my program on the same directory that the changed ThreadOS sub directory was located to see if it would detect my changes.   
Once the recursive portion of this implementation, I attempted to try on different regions, etc.

I tried creating buckets with invalid names or invalid location constraints to see the exceptions were being handled accordingly.

Overall, different iterations of   
1. New or pre-existing bucket names

2. Small number to large number of files  
3. Single directory to directory with many sub directories

4. Just a directory with changed files, unchanged files, and both

5. Different regions

6. Invalid bucket names or location constraints

7. Internet connection being down

For deleted files, I would upload a folder with files and subdirectories then remove that file on my local machine. I’ll then re-run the program and it’ll detect those files are missing and prompt the message