Cache - Cloud Storage App

A Project Report

Course: CS 403/633/733

Term: Summer 2020

Group Members:
Jordan Amison, Andrew Balfour, Phillip Brasfield, Baylee Byers, Andrew McCormick

Abstract:

Cache is a cloud based file storage application. As a group we used multiple AWS modules to create a scalable and reliable service. The result is a functional user-friendly app that allows local files to be stored securely in the AWS cloud.

Contents:

Introduction:

The objective of this group was to create a cloud based application using multiple aspects of the cloud. We decided to create a cloud based storage app similar to Dropbox or OneDrive. Leveraging the power of AWS, our app, Cache, uses S3 for secure file storage and organization and Cognito for secure authentication, user management, and load balancing. Connected together with a light Python 3 app, Cache offers a simple user interface that allows for easy uploading, downloading, and syncing of files between the cloud and a local machine. All the while scaling storage and user access seamlessly.

Background:

Our goal from the very beginning with this project was to create a cloud storage app with a gui and to develop it in an organized and collaborative way.

Every group project starts with a need for centralized organization and great communication tools. Gitlab was the obvious choice for our repository service. We used it to manage different builds and test code as the project took shape this semester. We used Discord as our communication platform of choice. The chat channel was great for quick discussions throughout the week and the voice channel and screen sharing feature were perfect for our weekly progress meetings. Canvas groups were a requirement that we embraced by using the calendar feature to mark out meeting dates and the collaboration tab to share live google docs, including this report!

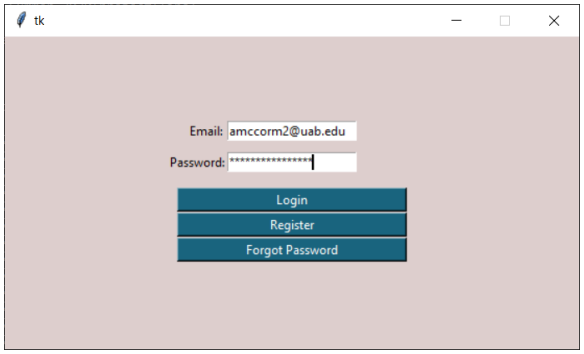
For our cloud storage app, Cache, we knew from the beginning that S3 would be a perfect choice. After research, trial, and error, we settled on Cognito for authentication, load balancing, and user management. For our GUI we used Python 3 and tkinter for its ease of use and compatibility with AWS using the boto3 library.

First Idea:

The original idea was to type all the code and then place it in the Elastic Beanstalk environment since it handles the load balancing, works with platforms like Python and Java, could connect to the s3 more easily and allow the user more control with it. Researching further, we learned that passing credentials would be more challenging than we anticipated. Cognito was ultimately the better choice with it being able to manage credentials more effectively. Now we only needed to connect everything together.

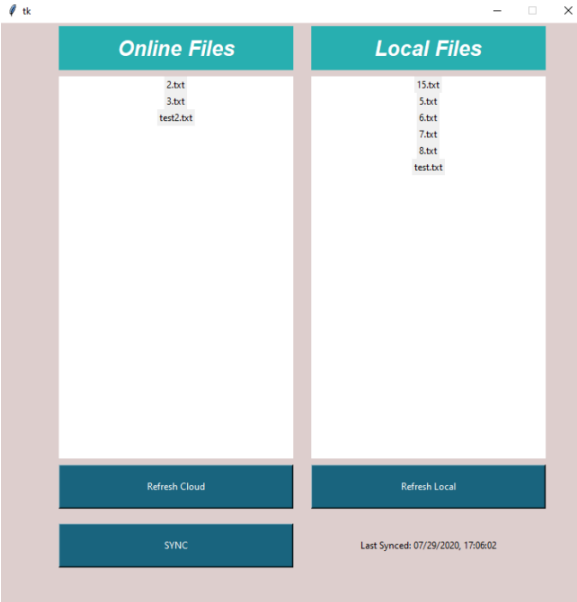
Design and Implementation:

The Cache app experience starts with the user authenticating through a simple Python tkinter GUI. The user also has the option to register if it's their first time using the app and an option to reset their password if it's forgotten..
(Login Screen)



Once the user has successfully authenticated with their email, password, and 2FA key(sent to email) they proceed to the main app page.

(Main App Page)

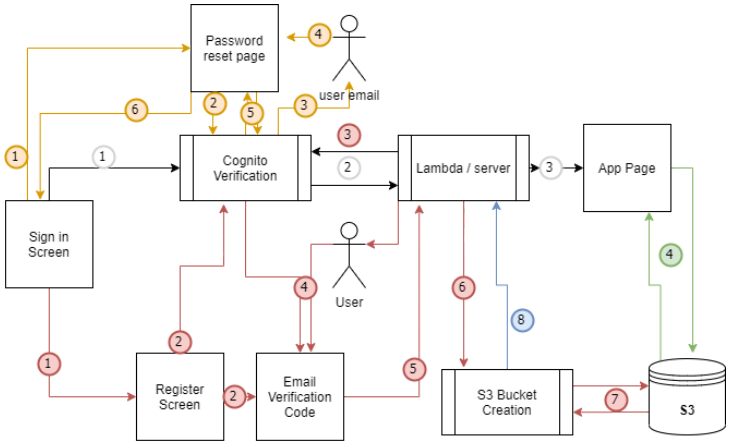


From here the user may view the contents of their sync folder(~Desktop\test). This will be where files downloaded from the cloud end up. The user may also select local files to upload manually from this page in a future build. This is the intended design, but time constraints changed how this page actually functions, details in the Conclusion.

Ideally, syncing with Cache uploads files from the local folder to the cloud and downloads files from the cloud that don't yet exist locally, but the current build only mirrors the local folder to S3.

Cache uses Python 3 and tkinter as a frontend and GUI. The python app is designed to authenticate using Cognito then to connect the user to their personal bucket in S3. Buckets are named using a common prefix followed by the username to ensure uniqueness across AWS. For example *cache51-username* for a user and *cache51-admin* for the admin bucket that we could store private userdata used for debugging and testing.

<Block Diagram>

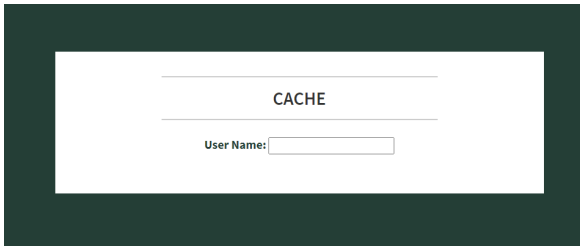


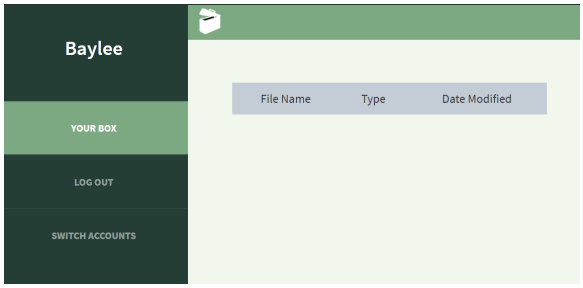
Here is the block diagram of our application. Squares being a local Python\tkinter page, rectangles with two additional vertical lines being Amazon Web Services, and the cylinder specifically being storage on S3. The red path is for creating an account, with blue being a return after success. Yellow\orange path is for resetting a password.The black path is typical login and use. Green is standard operation after login in, syncing up and down between the local app and pc and S3..

Discussion:

Our group formed and started prototyping app designs the first week of class. Knowing the layout of the app was a good way to plan out the functions we wanted to eventually implement.

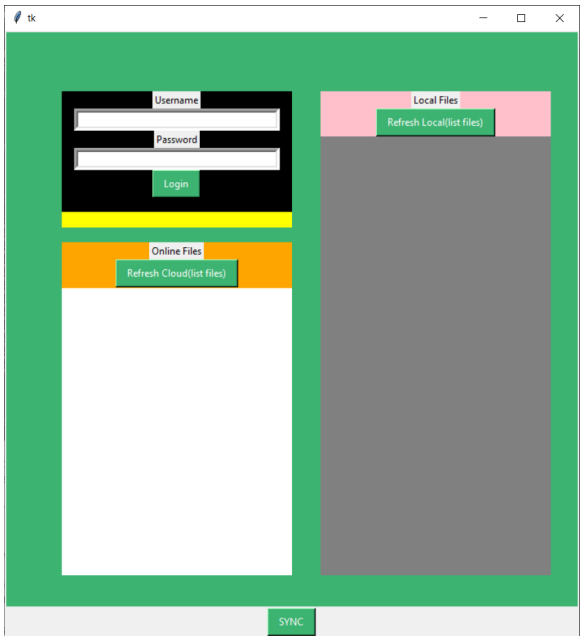
Our first gui prototype was made in html.





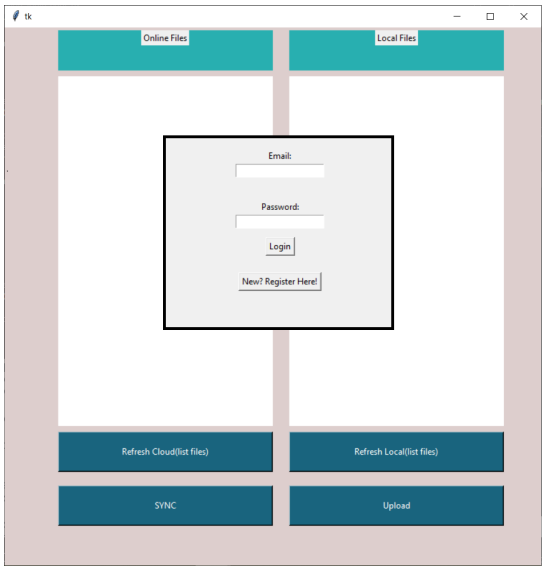
As we learned more about the functions available in AWS and decided which we would use, boto3 was identified as an invaluable tool that was, unfortunately, incompatible with our initial design.

So for the GUI and backend we moved from html and javascript to python 3 and tkinter. This is the first build of our new app in python.



Frames of the window were color coded to help understand tkinter's layout for the first time. At this stage, everything on the local side was more or less functional and buttons were directed to functions waiting to be attached to the cloud layer.

The GUI evolved with the project scope. Colors changed and the login frame was broken out into a new window.



(Notice the "Upload" button we wanted as a feature)

At the same time our GUI and local app were evolving, our cloud design evolved with them.

From the very beginning we knew that S3 would be the best solution for our Cloud Storage goal. With the limitation of buckets requiring unique names, we decided to use the prefix "*cache51-*". We would have a *cache51-admin* for any universal data we needed and users would each have their own bucked named *cache51-username*.

For user authentication we pursued many ideas, like uploading a plain text document with user and password to compare against a table stored in S3 or RDS, maybe with hashing. We looked at different AWS modules(ex. Elastic Beanstalk) and tested a few, ending up with Cognito. A good choice for how it handled load balancing and offered two-factor authentication for login.

We created a very nice functional app with AWS, but it's not without its limitations. Files uploaded with spaces in their name *may* cause issues with the bucket object naming scheme of S3. As far as we're aware objects can't have spaces because they each have a unique http address and need http compatible names.

Also, Python 3's tkinter module was chosen for its ease of development, but it is not compatible with mobile devices and wouldn't have been our number one choice if given more time.

We also ran into limitations using free AWS Educate accounts. We were never able to authenticate in-app without a current aws_session_token, which expired after a few hours. In the final build we use Andrew Balfour's free-tier AWS account for Cognito and S3.

While creating our Dropbox clone we learn some interesting history about Dropbox's relationship with AWS.In the year 2016 the business Dropbox mostly severed its ties with Amazon's AWS platform and built its own data centers. The thought process as to why Dropbox would want to build its own data centers was based around the enormous amount of control around some aspects of what Dropbox does like storing files. Amazon's AWS would not be able to provide Dropbox the level of control over aspects of their hardware. Dropbox felt that by managing their own private data centers they would be able to better control their destiny. In 2016 when Dropbox started to make the transition they had a whopping 500 petabytes of data to move. The dozen member infrastructure team at Dropbox had a large task at hand. The team would be building at a large scale of three US data centers and building the network backbone. The customer expected that when they opened up Dropbox on their personal computer and requested a file that their documents would instantaneously sync, it was up to the small to make sure those expectations are kept throughout the move. Ultimately though Dropbox was able to make this giant leap because it felt the need for control over their customers data. Not only was Dropbox able to save the business $74.6m in operational expenses two years after the move, they were also able to solidify the customers' trust in Dropbox storing their data.

Task distribution:

Starting the first week of class, we met once a week, sometimes more, on Discord to discuss project objectives and to assign tasks to each group member. Every group member contributed valuable research and resources to read and study that were relevant to AWS and the tools we used.

Phillip Brasfield researched Lambda and how it might be implemented in the project, along with contributing a section in the report about Dropbox's history, and the PowerPoint used on presentation day.

Andrew McCormick built the Python 3/tkinter prototype around boto3 and compiled the report.

Andrew Balfour researched and tested authentication options and implemented Cognito.

Jordan Amison researched load balancing and ways for Beanstalk to be implemented. Also presented for the group.

Baylee Byers implemented the filesystem and file management system and merged the group's local and cloud work, while further developing the GUI.

Conclusion:

Future iterations of this project would have us fix a few limitations we were unable to address given the time constraints.

One limitation is how the app syncs. The current build uses the folder %user%\Desktop\test as a master to upload to and delete from the cloud. A future iteration would have 1:1 parity between the online bucket and the local folder and an in-app delete button. Going further there would be right-click support in app to easily select files to delete. And we would give the user the option to choose where the local synced folder would be.

One other potential limitation of the app is the algorithm we use to compare our local and online file lists in *syncCloud*. The small number of files we'd use for a given test were never an issue, but larger file numbers could incur a worst-case runtime of O(n^2). There are python 3 functions that we could implement to improve performance for a large number of files if given more time.

Cost Analysis:

The cost averages to about 3 cents per users if they are each given 100GB. This figure does not include API(read/write) costs. I couldn't find a reliable source for the average calls a user might make in a month. If we were going to commercialize this app more cost analysis would be needed.

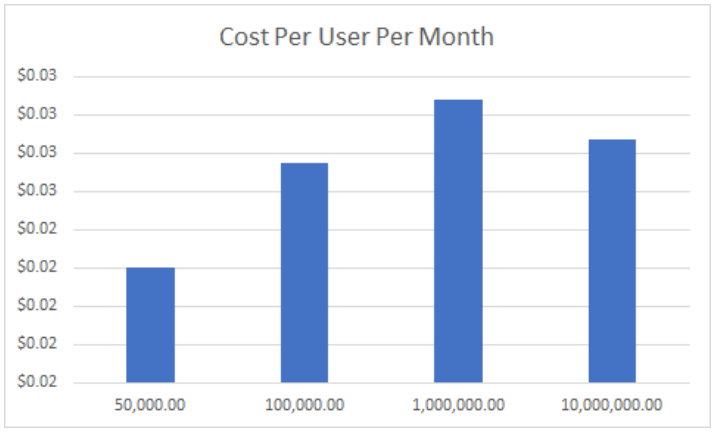
Amazon Cognito Pricing

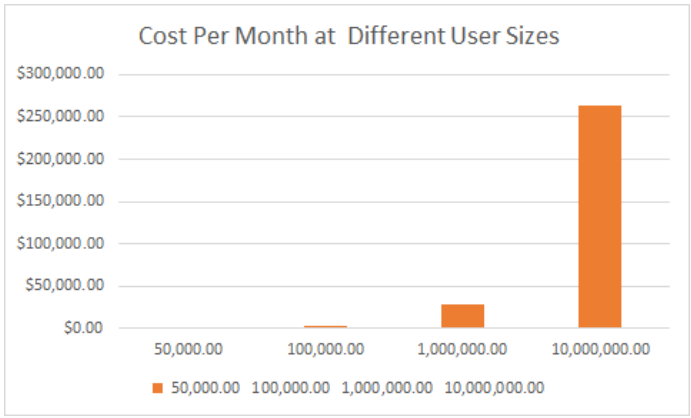
Users	Cost per user
≤ 50,000	Free
50,001-100,000	$0.0055
Next 900,000	$0.0046
Next 9,000,000	$0.00325
> 10,000,000	$0.0025

Amazon S3 Pricing(Standard Tier)

Storage per Month	Cost per GB
First 50 TB	$0.023
Next 450 TB	$0.022
> 500 TB	$0.021

Assuming the average user uses 100GB. Doesn't include API (PUT, COPY, POST, LIST,ect) costs.





Users	Total Cost Per Month	Cost Per User Per Month
50,000	$1,150.00	$0.02
100,000	$2,572.00	$0.03
1,000,000	$27,415.00	$0.03
10,000,000	$263,665.00	$0.03

API costs for S3:
$0.005/10,000 read requests.
$0.005/1,000 write requests.
$0.02/GB to transfer to different AWS region.

References:

boto3.amazonaws.com

https://aws.amazon.com/elasticbeanstalk/faqs/

https://aws.amazon.com/cognito/pricing/

https://aws.amazon.com/s3/pricing/

https://techcrunch.com/2017/09/15/why-dropbox-decided-to-drop-aws-and-build-its-own-infrastructure-and-network/

https://www.datacenterknowledge.com/cloud/here-s-how-much-money-dropbox-saved-moving-out-cloud

https://techcrunch.com/2019/06/21/three-years-after-moving-off-aws-dropbox-infrastructure-continues-to-evolve/

https://www.wired.com/2016/03/epic-story-dropboxs-exodus-amazon-cloud-empire/

https://www.sumologic.com/insight/s3-cost-optimization/#:~:text=~%20%240.03%20%2F%20GB%20%2F%20month%2C,0.06%20%2F%20GB%20to%20the%20internet