**1.**

Sampath Valluri – [svalluri1@uco.edu](mailto:svalluri1@uco.edu)

**2.**

My project implementation involves creating a file synchronization prototype using AWS services. The key steps were:

**AWS Infrastructure Setup:**

I created an EC2 instance (using a free-tier eligible instance type) and an S3 bucket, ensuring both are in the same region.

I configured IAM roles and policies so that the EC2 instance can securely access the S3 bucket with precise permissions (list, add, retrieve, and delete objects).

**Create an IAM Role with S3 Permissions:**

In the AWS IAM console, create a role designated for EC2.

Attach a policy such as AmazonS3FullAccess (or a custom policy for more restricted access) to grant the necessary permissions to interact with S3.

Attach the IAM Role to the EC2 Instance:

In the EC2 dashboard, select your instance and navigate to Actions → Security → Modify IAM Role.

Choose the newly created IAM role so that your instance can automatically assume the role and access S3 resources without storing credentials locally.

**Java Program Development:**

A Java program was built to automate file operations in a dedicated "sync" folder on the EC2 instance.

This program creates and updates three files: date\_time.dat for timestamps, mem\_names.dat for recording group member names, and log.dat for tracking all file operations.

Two threads run concurrently—one updates the timestamp file at random intervals, and the other appends member names, ensuring independent timelines for each file.

**Synchronization via Bash Script and Cron:**

A bash script using the AWS CLI was written to synchronize the "sync" folder with the S3 bucket. It logs each sync event in a file (cron\_log.dat).

A cron job was scheduled (every 2 minutes for demonstration) to run this script automatically, ensuring that all file changes are periodically uploaded to S3.

I also configured the cron job so that it can be canceled manually after a 10-minute demonstration period.

**Testing and Documentation:**

Throughout the process, logs from both the Java program (log.dat) and the cron sync (cron\_log.dat) were generated to provide a detailed trace of file operations and synchronization events.

Screenshots and log outputs were captured to verify that each step (EC2/S3 setup, IAM configurations, Java file operations, and synchronization) was executed correctly.

**3.**

**Screenshots**

**A computer screen with a black screen

AI-generated content may be incorrect.A computer screen with a white screen

AI-generated content may be incorrect.A computer screen with text on it

AI-generated content may be incorrect.A computer screen shot of a black screen

AI-generated content may be incorrect.A computer screen shot of a black screen

AI-generated content may be incorrect.A computer screen with a black background

AI-generated content may be incorrect.A computer screen with a black background

AI-generated content may be incorrect.A computer screen with a black background

AI-generated content may be incorrect.A computer screen shot of a black screen

AI-generated content may be incorrect.A computer screen with a black background

AI-generated content may be incorrect.A computer screen with a black rectangle

AI-generated content may be incorrect.A computer screen shot of a black screen

AI-generated content may be incorrect.A computer screen with a black rectangle

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A computer screen with a computer screen and a computer screen

AI-generated content may be incorrect.A computer screen with a computer screen

AI-generated content may be incorrect.A computer screen with a computer screen

AI-generated content may be incorrect.A computer screen with a computer screen

AI-generated content may be incorrect.A computer screen with text on it

AI-generated content may be incorrect.A computer screen with a black and white website

AI-generated content may be incorrect.A computer screen with a computer screen and a computer screen

AI-generated content may be incorrect.**

**4.**

**A)** AWS provides 750 hours per month of free-tier usage for t2.micro instances for 12 months after account creation.

IAM (Identity and Access Management) is a free AWS service, meaning creating and managing IAM roles and policies does not incur costs.

AWS CLI usage itself is free.

The aws s3 sync command primarily incurs data transfer costs, but:

Data transfers within AWS (EC2 to S3 in the same region) are free.

My S3 bucket and EC2 instance are in the same AWS region, ensuring zero data transfer costs.

By carefully selecting free-tier eligible services (t2.micro, 5GB S3 storage, IAM), ensuring data transfer remains within AWS, and monitoring usage via the AWS Free Tier Dashboard, I can confidently confirm that this project remains within the AWS Free Tier and will not incur charges.

**B)**

**Terminate Active Resources:**

Once the project demonstration is complete, shut down or terminate the EC2 instance and any other compute resources. This stops ongoing compute charges.

**Delete the S3 Bucket and Its Contents:**

Remove the S3 bucket (or delete all objects inside it) so that you’re not billed for storage once your project is over.

**Remove Custom IAM Roles/Policies:**

Delete any custom IAM roles or policies created specifically for this project. This avoids accidental usage or permissions that might trigger other services.

**Monitor Billing and Set Up Alerts:**

Enable AWS Budgets and billing alerts. These tools can help you monitor your usage and get notified if you’re approaching any free-tier limits or if unexpected charges occur.

**Double-Check Free Tier Limits:**

Ensure that all services used (like EC2, S3, and IAM) remain within the free-tier limits during your project. Regularly check your AWS Free Tier usage reports to confirm compliance.

**C)**

**update your cron job's schedule:**

0 0 \*/2 \* \* /home/ubuntu/project/scripts/sync\_script.sh>>/home/ubuntu/sync/cron\_log.dat 2>&1

Explanation:

0 0: This sets the time to midnight (00:00).

\*/2: In the day-of-month field, this means "every 2 days" (i.e., every other day).

\* \*: The month and day-of-week fields remain as wildcards, meaning the job will run regardless of the month or day of the week.

With this updated control, the sync script will be executed at midnight every other day, ensuring that your EC2 folder is synchronized to your S3 bucket on an every-other-day schedule.

**D)**

**log.dat**

**05:29:35 – Start-Up:**

The program started at 05:29:35. At this moment, it created the necessary files:

date\_time.dat and mem\_names.dat were created.

Immediately afterward, it appends its first entries:

The current timestamp ("02-17-2025 05:29:35") was written into date\_time.dat.

"Sridatta" was appended to mem\_names.dat.

Right after these actions, the program logs how long it will wait before performing the next update:

A 5-minute wait is set for the next update of date\_time.dat.

A 3-minute wait is set for the next update of mem\_names.dat.

05:29:41 – Immediate Second Update:

Shortly after the initial setup (just 6 seconds later), the program performs another round of appends:

Again, "Sridatta" is added to mem\_names.dat.

The timestamp "02-17-2025 05:29:41" is appended to date\_time.dat.

It then reaffirms the wait periods for the next actions (5 minutes for date\_time.dat, 3 minutes for mem\_names.dat).

05:32:35 – First Scheduled Update:

At 05:32:35, which is roughly 3 minutes after the last mem\_names.dat update:

The program appends "Sampath" to mem\_names.dat.

It then schedules another 3-minute wait for mem\_names.dat.

05:34:35 and 05:35:35 – Additional Updates:

The program continues:

At 05:34:35, it appends a new timestamp ("02-17-2025 05:34:35") to date\_time.dat and changes the waiting period to 1 minute.

At 05:35:35, it appends:

A new timestamp ("02-17-2025 05:35:35") to date\_time.dat.

"Sridatta" again to mem\_names.dat.

It then updates wait times accordingly (1 minute for date\_time.dat and 5 minutes for mem\_names.dat).

**05:36:35 – Further Update**:

Finally, at 05:36:35, another timestamp ("02-17-2025 05:36:35") is appended to date\_time.dat, with a wait of 5 minutes scheduled afterward.

**cron\_log.dat**

**05:30:01 – First Synchronization:**

The sync begins at 05:30:01.

During this process, the AWS CLI uploads all the files present in the sync folder—this includes the Java-generated files (date\_time.dat, mem\_names.dat, log.dat) and cron\_log.dat itself.

By 05:30:02, the sync is completed.

**05:32:01 – Second Synchronization**:

At 05:32:01, another sync is triggered.

Since the Java program had an update at 05:32:35 for mem\_names.dat shortly afterward, the sync captures the state of the files up to that moment.

Sync completes at 05:32:02.

**05:34:01 – Third Synchronization**:

At 05:34:01, the sync process starts again.

By now, the update at 05:34:35 for date\_time.dat is about to happen. The sync at 05:34:01 captures the files before that update.

The sync completes at 05:34:02.

**05:36:01 – Fourth Synchronization**:

The process starts at 05:36:01, capturing the state of the files as updated by the Java program until this moment.

The sync completes at 05:36:02.

Subsequent Syncs (05:38:01, 05:40:01, etc.):

The cron job continues to trigger the sync every 2 minutes.

Each sync logs which files are updated and how much data was transferred, ensuring that any changes made by the Java program are pushed to the S3 bucket.