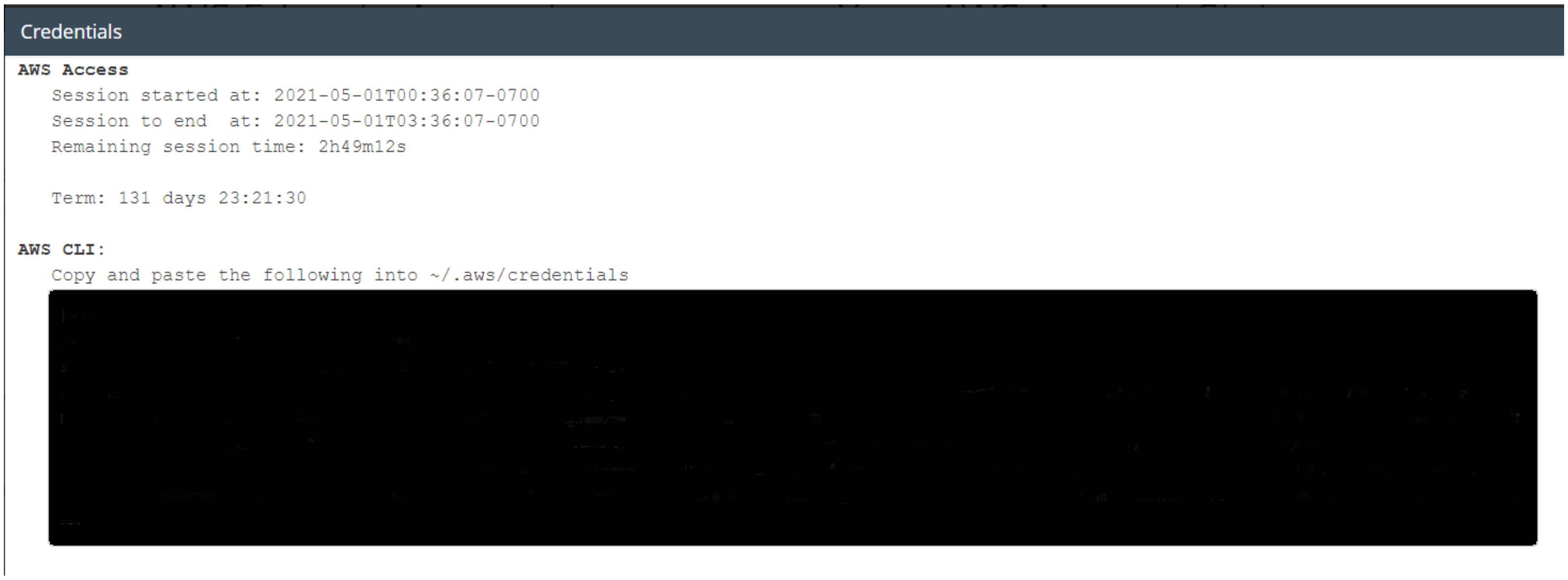
# ASSIGNMENT 1 : SARCASM ANALYSIS

- **Running the program:**

1. You need to have an AWS EC2 account.
2. Create a file “credentials” in the path :
   C:\Users\USERNAME\.aws\credentials on Windows or
   ~/.aws/credentials on Linux, macOS, or Unix.
3. Go to your AWS account details and copy all the text inside the black box into the credentials file.



Credentials

```
AWS Access
   Session started at: 2021-05-01T00:36:07-0700
   Session to end  at: 2021-05-01T03:36:07-0700
   Remaining session time: 2h49m12s

   Term: 131 days 23:21:30

AWS CLI:
   Copy and paste the following into ~/.aws/credentials
```

[redacted]

4. Set the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY environment variables.

   To set these variables on Linux, macOS, or Unix, use **export** :

```
export AWS_ACCESS_KEY_ID=your_access_key_id
export AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

   To set these variables on Windows, use **set** :

```
set AWS_ACCESS_KEY_ID=your_access_key_id
set AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

5. The manager and workers instances fetch their scripts from s3, therefore we need to make sure the jars are in the right buckets in s3 (dsps212-artifacts/jars)

   By running the following maven builds:
   From the maven projects:

- DSP_Ass1_Manager -> Plugins -> assembly -> (2 clicks on) assembly: assembly
- DSP_Ass1_Manager -> Plugins -> s3-upload -> (2 clicks on) s3-upload:s3-upload
- DSP_Ass1_Worker -> Plugins -> assembly -> (2 clicks on) assembly: assembly
- DSP_Ass1_Worker -> Plugins -> s3-upload -> (2 clicks on) s3-upload:s3-upload

6. Run the program by executing:
   >java -jar DSPS212localApp.jar <input_file_1.txt [input_file_2.txt ...]> <output_file_1.html [output_file_2.html ...]> <num_reviews_per_worker> [<terminate>]
   Or
   Run LocalMain from your IDE.

## **<u>Specs and benchmarks:</u>**

- Worker and Manager types and AMI's: t2.medium with java and git installed.
- We used t2.medium due to the large size of the worker handler's instances.
- We are using an IAM role with permissions for EC2, S3, SQS, and role transfer.
- We used n = 20. Due to account restrictions we limited the maximum number of Worker instances to be 15.
  n = 20 attempts to lunch the maximum number of instances (15).
- Due to instance type restriction, only 8 t2.medium instances can run in parallel while the rest are terminated automatically by the console.
- We run 2 benchmark tests – single localApp with 2 files (each with ~500 reviews),
  single localApp with 5 files (same size).
- The first test took 6 minutes and the second one 10 minutes.
- We expected such results due to overhead complexity of instances initialization.

- **<u>The program flow:</u>**

1. LocalMain creates the localApps to run.
2. The Local Application:
    - Creates a unique message queue from the manager to it.
    - Check if a manager instance was already created and if not, create it and the queue from all localApps to it.
    - Uploads the input files with the list of reviews to S3.
    - For each file, send a message to the Manager which contains:
        - The queue name to send reply through.
        - The bucket which holds the input file.
        - The input file name.
        - The output file name.
        - The desired number of reviews per Worker.
    - Waits on its queue to get the result details from manager.
    - For each result (output_file) creates an html file.
    - After finishing all files, sends a termination message to manager.

3. On startup, the Manager instance downloads the jar file from S3 and runs the ManagerMain.
4. The ManagerMain creates 2 queues (incoming and outgoing) to communicate with the Workers.
5. The Manager instance is waiting on its queue for messages from localApps.
6. The manager has a threadpool so each of the following tasks won't have to wait to be executed:
    - Get an input file from localApp and send its reviews to the workers' queue.
    - Get an analysis message from a worker, add it to an output file and send it to the localApp in case this is the last review of the output file.
7. The Manager has 3 maps:
    - localQName_to_Qurl
    - outputFileName_to_localQName
    - fileReviews (outputFileName_to_reviewsIdsSet)

8. For each task message the Manager:
    - Updates its maps.
    - downloads an input file from S3.

- Reads each line in the file and sends all the reviews in it together (in a batch) to the workers' shared queue.
- Each message is given a key and logged for the file (to detect duplicated results).
- Checks the number of reviews sent to workers and the number of workers exists, and add more workers if needed.

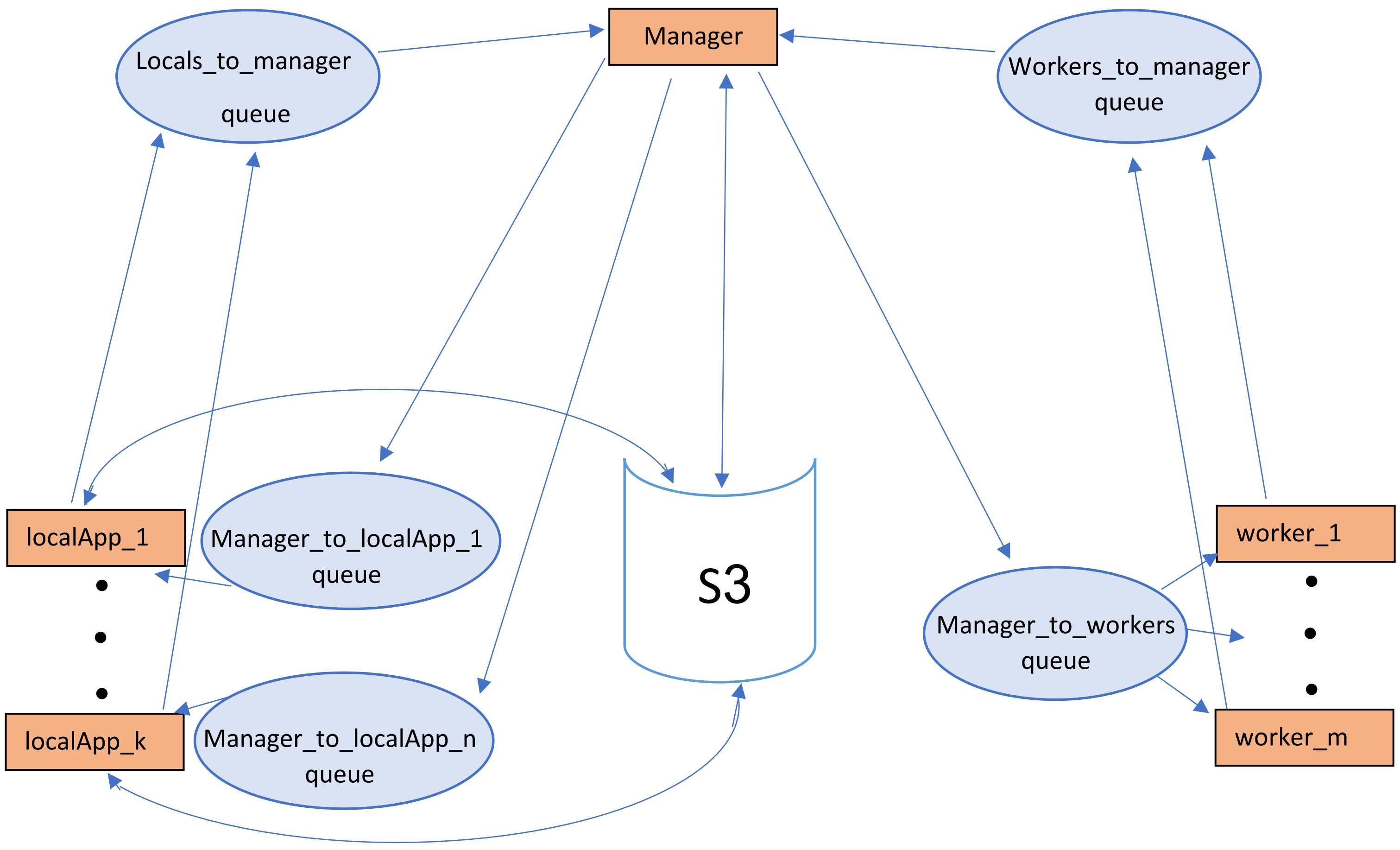
9. In case of a termination message from a localApp the Manager:
    - Blocks messages from localApps.
    - Handle the remaining files.
    - Once all files are done, terminates all Workers, deletes worker-manager queues and locals->manager queue, and finally terminates its own instance.
10. The Manager also waits for Worker results messages.

    For each analysis (worker) message the Manager:
    - Checks if this review's analysis was already delivered by another worker and if so, dumps it.
    - Writes this analysis to the matching output_file.
    - If it's the last analysis for the file uploads it to S3 and sends a message about it to the corresponding LocalApp.

11. On startup, the Worker instance downloads the jar file from S3 and runs the WorkerMain.
12. The Workers waits for messages on the Workers shared queue.

    For each manager message, the Worker:
    - Performs the requested analysis on the review.
    - Sends the results to the manager.

- **Diagram of the flow:**


Locals_to_manager queue
Manager
Workers_to_manager queue
localApp_1
Manager_to_localApp_1 queue
S3
Manager_to_workers queue
worker_1
localApp_k
Manager_to_localApp_n queue
worker_m