README

211810650_325114569

- Credentials security: Our credentials are saved in ***~/.aws/credentials*** using the credential provider while creating an instance. In addition we run chmod 400 to make the file secure.
- Scalability: Yes, our code should work properly with a large number of clients. We used thread pool for the tasks that take a lot of time, for example downloading the links files, distributing all the links to the workers, and uploading the result file.
  In addition, after performing OCR, we ensured that the workers will delete the image file.
- Persistence:
  - If a worker dies:
    the count of the active workers will decrease by one and the manager will correspond accordingly. We managed to do so, by checking the state of the available worker(by their tag name). This is documented properly in the src code.
  - If a worker fails to do ocr:
    we used visibility timeout on the messages in the SQS queues. After receiving a message, if a worker failed to do OCR, he won't delete the message and will allow for other workers to receive the same message.
  - If a woker stalls:
    The visibility timeout is set high enough so that two workers won't process the same message.
  - If the manager dies:
    Our idea is to create a special instance that his whole purpose is to create a new manager if the original crashed.
    This instance will check in a timed loop if the manager instance status is *terminated.* After he detects that the original manager crashed, he will:
    1. Terminate all running workers
    2. Create a new manager and queues for this manager
    3. Create and resend all the local request messages based on their buckets on s3

    In addition, the manager will check if this instance got crashed and create a new one if so.

  - We have probably taken care of most of the possible outcomes, because we checked every possible exception that we may encounter from the aws interface. Therefore we have reduced dramatically the number of failures in the system.
  - In case of an interruption in the connection or other problem that does not depend on us, we catched the error and printed out the problem and the cause.
  - We have assumed that the amazon services won't fail in some critical section (we think that it is ok to assume so, because amazon is a big and reliable company).

- Threads:

  We tried to use more threads for the tasks that take a longer time. For example download and upload file, as mentioned above.

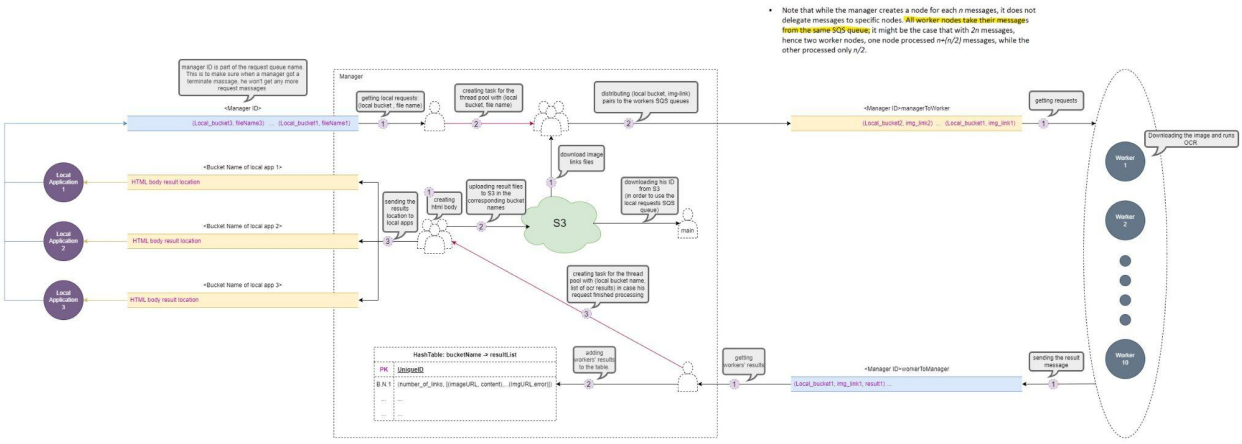
  - Microservices /thread pool tasks
    - DownloadAndDestributeTask:
      Responsible for downloading the links file from s3, creating all the messages (one for each link), and distributing them to the workers via SQS.
    - UploadAndSendTask:
      Responsible for uploading the result file to s3 and sending a message with the location back to the local via SQS.

  - ActOnMessageThreads

    ActOnMessageThreads is an abstract class which is generalizing the idea of a thread which performs an action on a specific SQS message. In out the design we had two different ActOnMessage roles:
    - GetLocalRequestsThread:
      Responsible for getting the clients request (image links).
      WorkersResuktThread:
      Responsible for getting the OCR results from the workers.

  In addition, in order to not be ‘busy waiting’ we used the wait and notify design pattern.

- High quality of this this picture is attached to the submission folder



- Termination process. Simply the termination works. We made sure to terminate all workers, queue, buckets and the manager if a local application requests it.

    - We also handled some edge cases:
      The termination message will always be the last message the manager will receive. Even before the local sends the manager the termination message, he deletes the *'managerExistsBucket'* and by that notifies that the manager is terminated. This makes sure that the next local will create a new manager.

- Our workers are working hard because we assume that each OCR takes enough time thus we don't have racing conditions. Even though we used only one queue to distribute the tasks as mentioned in the assignment. If we had used more queues to the workers we could have managed the tasks better.
- Our manager is doing exactly what he should and no more. Our code is separated into microservices with high cohesion.
- Distributing: We are distributing all the tasks to many different workers therefore we are taking advantage of many processes to perform the heavy OCR task.
  Almost every part in the system awaits for others. The local waits for the manager's result, the manager waits for the worker's results, the workers wait for the manager to send them links to download. In addition the threads in the manager are using wait and notify as described above.