Distributed System Programming - Assignment 1

- Hagai Levi 302988175
- Tom Leibovich 200456267

DSP Modes

For security and quality-of-code reasons our project can run with 3 modes:

- DEV_LOCAL - In this mode the project runs with a local mock of aws, which simulates ec2, s3 and SQS. Mock specification is detailed below.
- DEV - In this mode the program runs with an .aws credentials file. This mode should be used to run the local application.
- Default - In this mode, the program uses its predefined security groups. This mode is used on the EC2 instances and ensures that each instance can use only its granted privileges. This way is more secure than uploading your credential file to the cloud.

Local credentials

- In production, machines that are started on EC2 are using `InstanceProfileCredentials`. If you are running the manager/workers locally, you will need to use regular credentials. To do so, add the environment variable

`DSP_MODE=DEV`. Note that this collides with the Mocks
- In order to allow the manager to start workers, we need to give it permissions in EC2. To do so, define the `MANAGER_I_AM_PROFILE_NAME` in `Utils.java`, and add AWS IAM role with sufficient permissions. This allows us to use InstanceProfileCredentialsProvider as described in Providing AWS Credentials in the AWS SDK for Java.
- In order to run the local application, first of all define your credentials as explained in AWS documentation

Compiling

- We used Maven as our build tool for this project. In order to compile the project use `scripts/build.sh`.
- We use maven profiles, the `dev` profile is active by default and includes aws jars. The `production` profile is activated when we build the production jars, this way we keep our production jars small, and add the aws jar (over 45 MB) in runtime When creating the jars with the `production` maven profile, all the jars that are contained in the aws sdk are expected to be found on the computer that runs the jar, under `aws-java-sdk-1.10.64` directory. For more detailes see `getManagerUserDataScript()` or `getWorkerUserDataScript()` in `EC2Utils.java`.

Building

After building the jars using `scripts/build.sh`, you need to deploy the jars to s3, for that we are using AWS CLI

Running

- In order to run the local machine:

  ```
  export DSP_MODE=DEV

  java -jar yourjar.jar inputFileName outputFileName n
  ```

Mocks

- We use fake_sqs to mock AWS SQS, follow the instruction in the repo to install (Basically, if you have ruby, `sudo gem install fake_sqs`)
- We use fakes3 to mock AWS S3, follow the instruction in the repo to install (Basically, if you have ruby, `sudo gem install fakes3`)
- We use aws-mock to mock AWS EC2, follow the instruction in the repo to install. *We currently use a fork because the original repo doesn't have the* `InstanceType` *s that are using.*
- To use the mocks instead of using AWS, add an environment variable `DSP_MODE` with the value `DEV-LOCAL`. If you want to enable only specific mock, use `DSP_MODE_<service-name>=DEV-LOCAL`. For example `DSP_MODE_EC2=DEV-LOCAL`. **Note** that the mocks support a subset of the actual actions that are possible in AWS

AMIs

- We used ami-a78499cd which is a custom-made AMI which we created in order to install java 8.
- For the worker (which required a lot of memory for NLP processing) we used t2.small machine.
- For the manager we used t2.micro.

Time of tweetLinks.txt processing

- It took --------- seconds for the program to run.

Rate of tasks to workers (N)

- Because of the EC2 20 instance limit, we used tweetNumber/WorkerNumber.
- WorkerNumber = 17.

Scalability

- No task in our project relies solely on RAM.
- Each task is processed “one-at-a-time” with the support of S3 and SQS.

Persistence

Crash handling is performed throughout the project in all modules:

- Manager crash - If a manager crashes the local module has a heartbit thread that checks if the manager is up, and if it crashed the heartbit will request a new manager.

- Worker crash - A similar solution is implemented in the manager. If a worker crashes the WorkerMonitor will request a new worker instead.
- Visibility timeout - SQS queues have visibility timeout, which means that when a message is fetched from the queue it is invisible for all other machines. This is true only for a limited time, after this timeout the message is visible again and two workers could possibly process the same message. We prevent this by extending the timeout every few seconds.

Logging

- We used Log4J for logging.