Home Assignment 1

Overview

For this assignment, you will build a basic application using Amazon's cloud services. You will become familiar with how to launch and access virtual machines using EC2, and learn how to execute code on these machines. You will learn how to use Amazon's Simple Queue Service (SQS), as well as Amazon's Simple Storage Service (S3).

There are several ways of doing this assignment, including a variety of languages you can use, and a variety of ways of interacting with AWS. For example, in order to interact with AWS, you can use the AWS management console, command-line tools, SOAP APIs, REST APIs, as well as libraries in other languages. However, I will only provide nice detailed instructions on how to do this assignment in one particular way (you are welcome to do it another way, but I will be unable to help you along the way if you run into issues). Specifically, I will give guidance on how to use the AWS management console to create AWS resources, and how to use the AWS SDK for Java to interact with these resources (both from within AWS, and outside of AWS).

In order to do the assignment in the way I will outline, which uses the AWS management console for various purposes, you will have to set up an AWS account. To set up an account, you will have to provide a credit card number, but this credit card will not be charged because everything we will do in this assignment falls under the "Free Usage Tier" of AWS. **However, if you wish to do some of the tutorials provided by AWS, or test some of their example code, you might end up being charged.**

For a great overview of some best practices for using the Amazon Web Services I recommend this white paper.

Note! Do not make any of your AWS keys, credentials, or even keys to your S3-buckets publicly available (e.g. GitHub), or you might end up with a juicy bill.

Setting up an AWS Account

1. To sign up for a Free-tier AWS Account just follow the instructions on https://aws.amazon.com/free/.
2. Become accustomed with the AWS Management console. They provide a useful 'getting started tutorial'.
3. Play around and create a EC2Instance, an SQS-queue, and an S3-bucket.
    - It would be a good idea to `ssh` into your EC2Instance, modify it a little bit and then save it as an AMI-image.

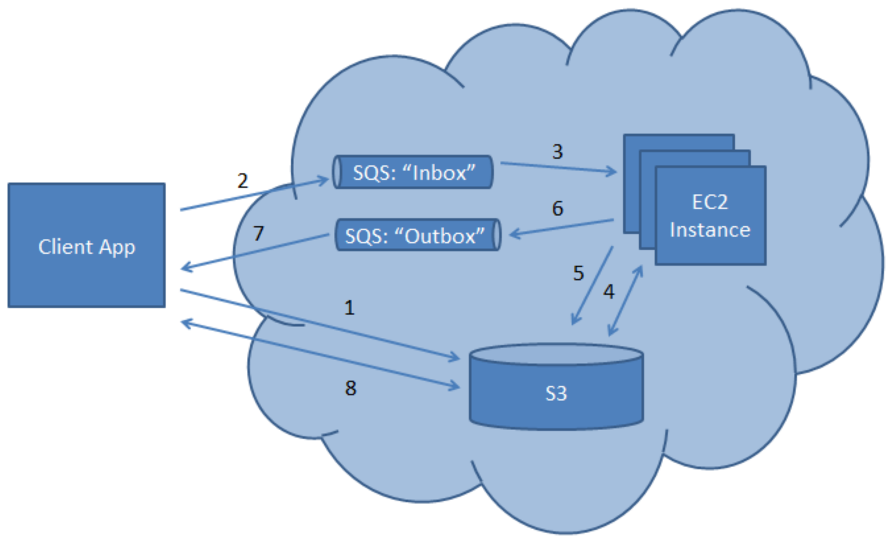
Setting up Eclipse

If you don't like Eclipse I'm sure you can find another way..

- If you don't have Eclipse installed you can find it here. The "EE"-version of Eclipse is recommended.
- Setup the "AWS Toolkit for Eclipse"
  - Instructions are given here.
- Try out some of the "AWS java sample code"-examples.
- Look at this API on how to create and run a regular EC2Instance.
- The "AWS Java sample code" only show you how to setup a EC2SpotInstance, which we will not be using

Create an application using SQS, EC2, and S3

Now you are ready to create your application! The structure of the application should look like below:



For a more thorough description of the application you can read this nice article by Amazon. The different parts of the application are defined as:

Client App

Runs locally on your machine. Should do the following:

- User inputs a comma-separated list of numbers.
- Puts this list in the S3-bucket (and remembers the key/pointer to this object).
- Puts a message in the SQS-inbox with a key/pointer to the object in the S3-bucket, along with a process to be executed on these numbers.
- Waits until a response is generated in the SQS-outbox (should contain a pointer to a new, and processed, object in the S3-bucket along with the process executed).
- Reads the result from the S3-bucket
- Prints the results along with the original numbers and the the process that was done.
- Deletes the result-message from the SQS-outbox

The available processes should be:

- Min
- Max

- Product
- Sum
- Sum-of-squares

Note: You need some clever way to make sure that the result in the SQS-outbox is indeed the result to your request and not the request made by a different client app (meaning it should be possible to have several client apps).

SQS Queues

You will need one "inbox" where the requests will be put and one "outbox" for the responses.

Some notes on the SQS-queue:

- The SQS-queue is not a FIFO queue.
- To make sure that only one instance can read each message at a time you need to set up a "VisibilityTimeout" for your queue.
- To make sure that your thread waits for a new message to arrive (if the queue is empty) you need to enable "Long Polling".
- These can both be set when creating the queue:

```
// Queue Attributes
Map queueAttributes = new HashMap();
queueAttributes.put("ReceiveMessageWaitTimeSeconds", "20");
queueAttributes.put("VisibilityTimeout", "10");

/*
* Create an Inbox and an Outbox Queue
*         - Enable SQS Long Polling. Forces the component that tries to receive a massage to wait.
*   - Set the Visibility Timeout. Ensures that only the intended can receive the message during
*         the timeout.
*/
System.out.println("Creating a SQS-inbox.\n");
CreateQueueRequest createInboxQueueRequest = new CreateQueueRequest("Inbox");
String inboxUrl =
sqs.createQueue(createInboxQueueRequest.withAttributes(queueAttributes)).getQueueUrl();
```

2+ EC2 Instances

These instances read from the "inbox" queue, process the requests, put the processed object in S3 bucket, send a response message to the "outbox"-queue, and delete the message from the "inbox"-queue once they have finished processing it.

To create an EC2 Instance from a certain AMI you can do as below:

```
AWSCredentials credentials = null;
try {
   credentials = new ProfileCredentialsProvider("YourCredentialsName").getCredentials();
} catch (Exception e) {
   throw new AmazonClientException(
         "Cannot load the credentials from the credential profiles file. " +
         "Please make sure that your credentials file is at the correct " +
         "location (/Users/Username/.aws/credentials), and is in valid format.",
         e);
}

// Create an EC2Client
AmazonEC2 ec2 = new AmazonEC2Client(credentials);
ec2.setEndpoint("ec2.eu-central-1.amazonaws.com"); // It is better to use 'setEndpoint()' than
'setRegion'

RunInstancesRequest runInstancesRequest = new RunInstancesRequest();

// AMI for normal Linux EC2Instance (setRegion Frankfurt): ami-b43503a9
runInstancesRequest.withImageId("ami-b43503a9")
                          .withInstanceType("t2.micro") // Select t2.micro to be eligible for free tier
                  .withMinCount(1)
                  .withMaxCount(1)
                  .withKeyName("YourKeyName") // Which you will use to ssh into your instance
                  .withSecurityGroups("YourSecurityGroup");

RunInstancesResult runInstancesResult = ec2.runInstances(runInstancesRequest);
```

S3 Bucket

The S3 bucket will contain all of the requests/responses which the system has processed. This information should at least include: MessageID, timestamp, the three numbers, process, result (if it's a result)

Some good info on the S3-bucket can be found here.

Note that all S3-buckets need unique names!

To create a S3-bucket the following code can be used:

```
String bucketName = "your-special-bucket-" + UUID.randomUUID();

/*
 * Create a new S3 bucket - Amazon S3 bucket names are globally unique,
 * so once a bucket name has been taken by any user, you can't create
 * another bucket with that same name.
 *
 * You can optionally specify a location for your bucket if you want to
 * keep your data closer to your applications or users.
 */
System.out.println("Creating bucket " + bucketName + "\n");
s3.createBucket(bucketName);
```

Some useful links

- Article on creating a similar application (By Amazon)
- AWS Java API
- Web Application Hosting in the AWS Cloud (Amazon White Paper)
- General AWS Documentation

- Some more articles and tutorials
- AWS Java Developer Center