**Deployment requirements and steps**

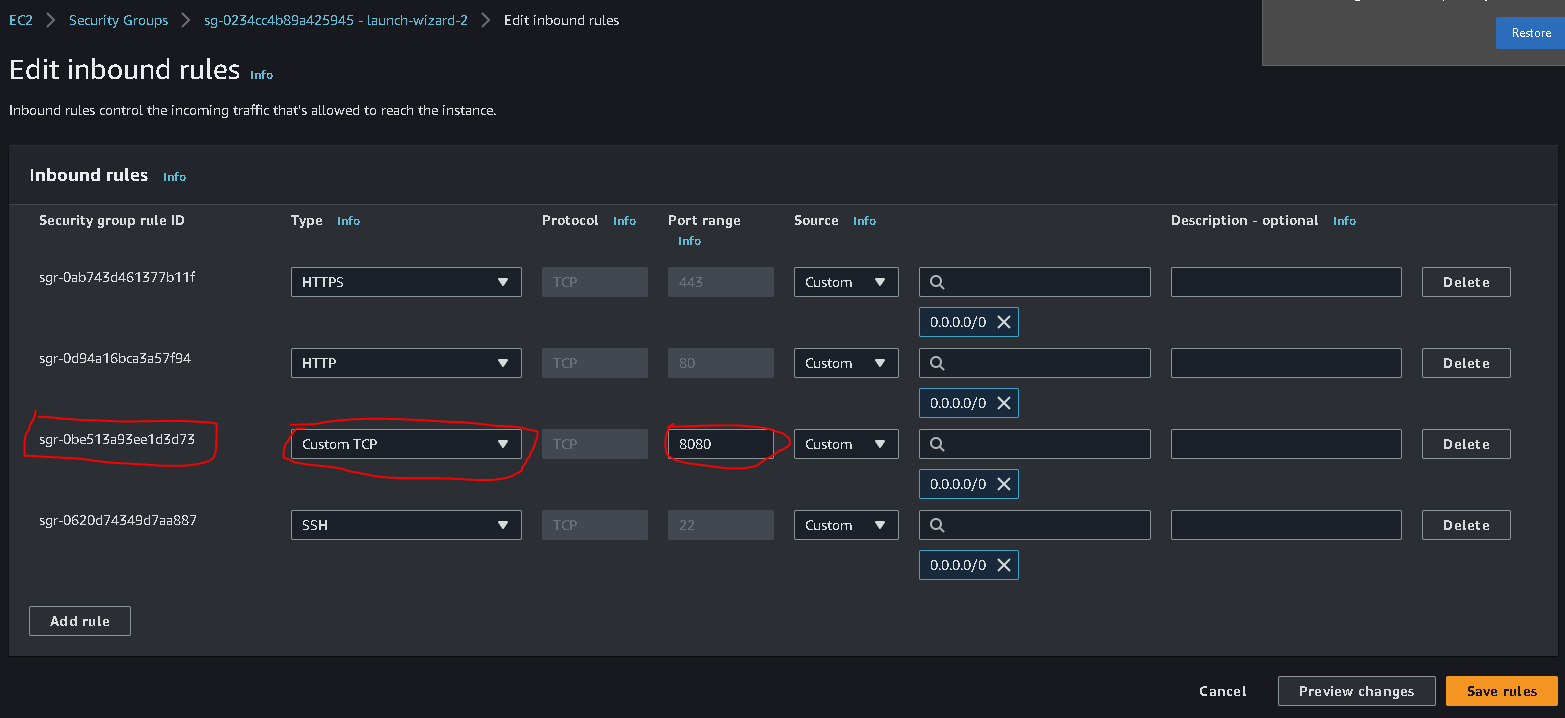
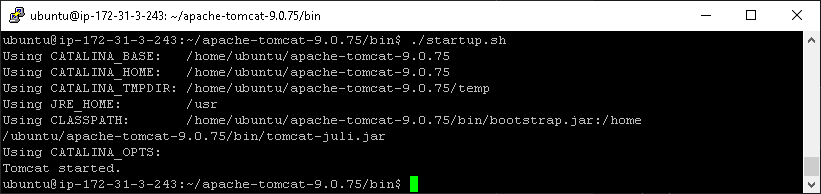
* JDK 1.8.0\_361

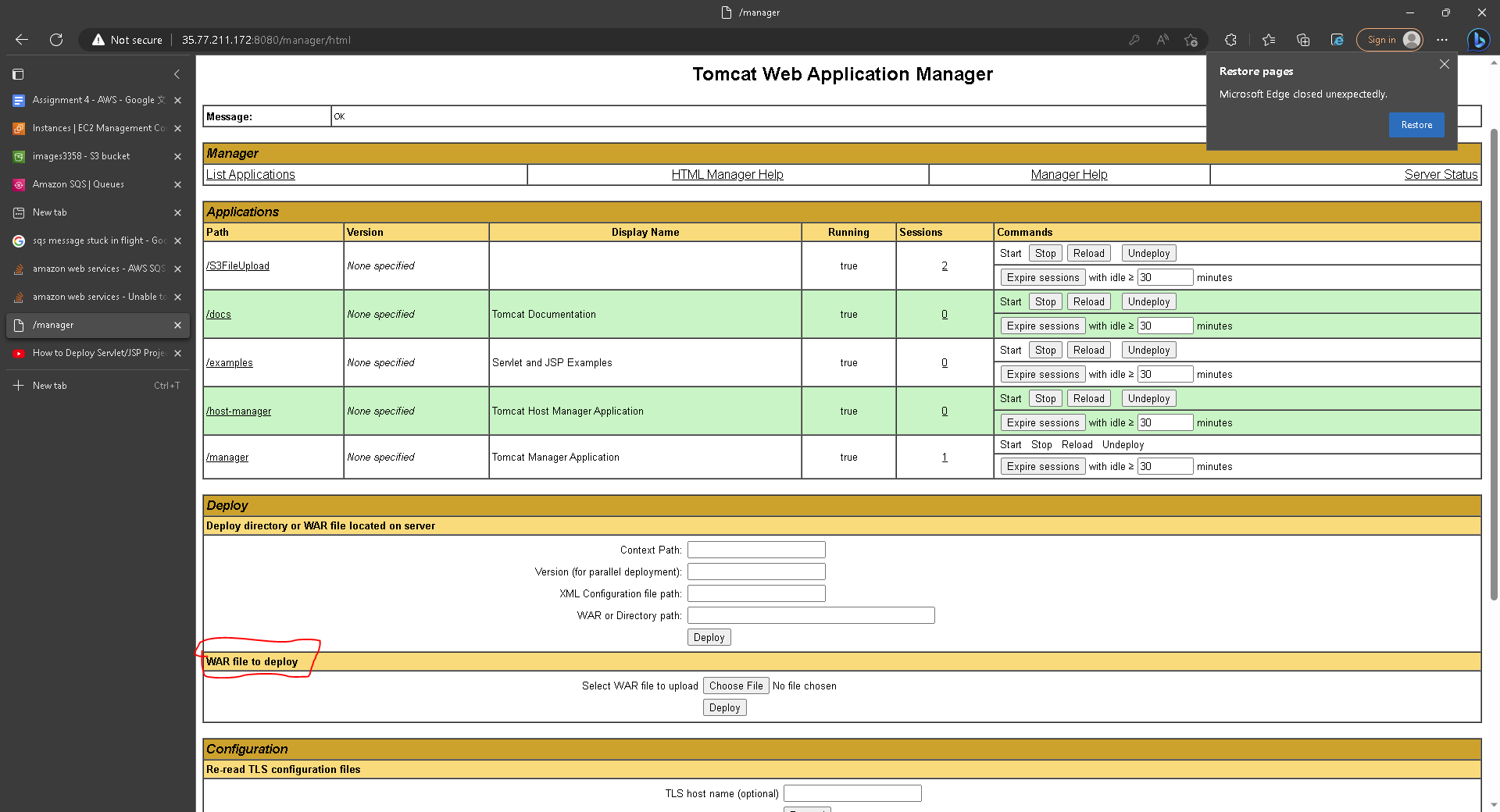
If one wants to deploy the codes themselves, they also need:

* AWS Toolkit in Eclipse
* Apache Tomcat 9.0.75
* Two SQS FIFO queues
* 2 EC2 instances configured with IAM role that grant access to S3 and SQS, with the web server adding an extra port
* A S3 bucket will ACL policy set to publicly accessible

**If one wants to locally run my code and deploy my website, they must first build 2 FIFO SQS queues, a S3 bucket and an EC2 instance (an extra one for the web server). Then, the initialization of AWS objects like AmazonS3 and AmazonSQS should be renamed to the correct instance names. For EC2 instances, one needs to supply credentials or grant IAM role to access S3 and SQS. For S3 bucket, Access control list (ACL) is set to allow public access. For SQS, check Content-based deduplication.**

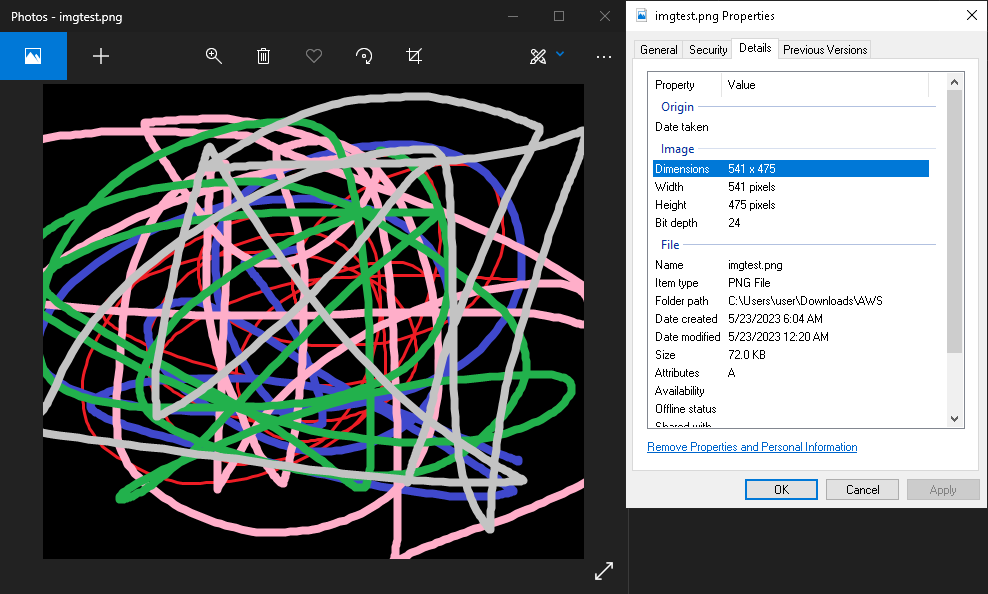
**For the EC2 web server, I used an Ubuntu instance and downloaded Tomcat and Java environment there. Upon setting up permissions with “sudo chmod +x” to startup.sh, shutdown.sh and catalina.sh, one can launch the server by “./startup.sh”. Before the manager can be accessed, one needs to create an additional port available to everyone in the EC2 security policy. After everything is done, the Apache Tomcat manager page can be seen and one can then deploy the .war web project there.**

****



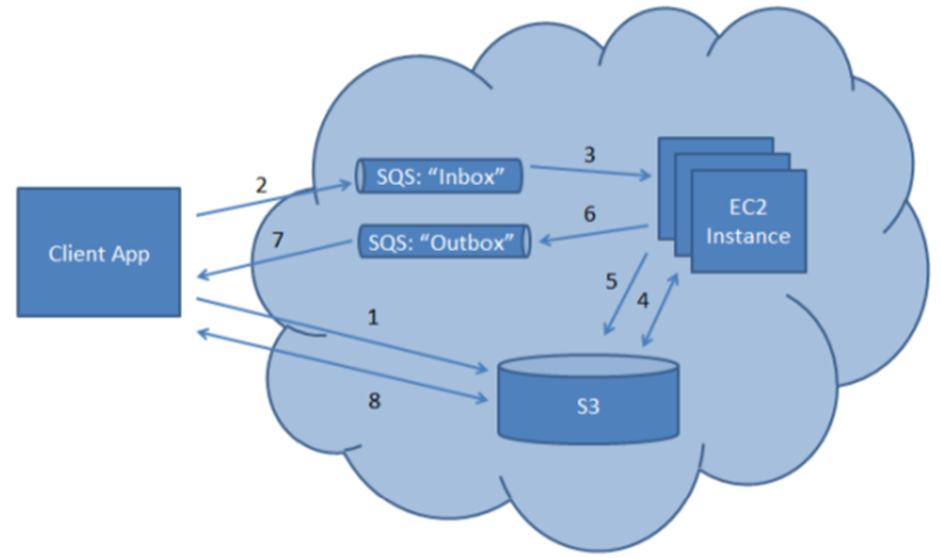
**Runtime screenshots**

Initial image & its size

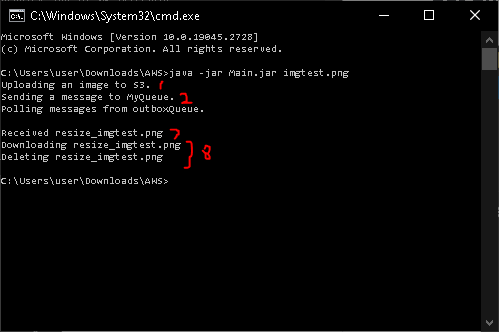


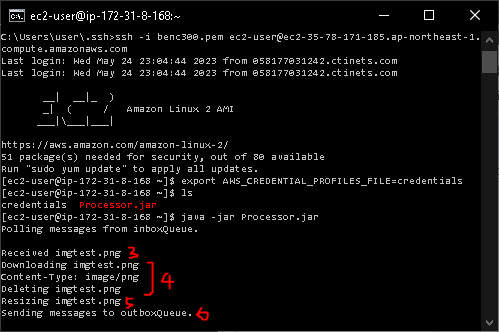
Local console

Please refer to the architecture diagram as a reference to each step.



**\* IMPORTANT: Please use the filename directly as input (i.e. without path), and do not include any spaces and special characters in the file name. Also, please do not use include “resize\_” in the file name to avoid confusion.**





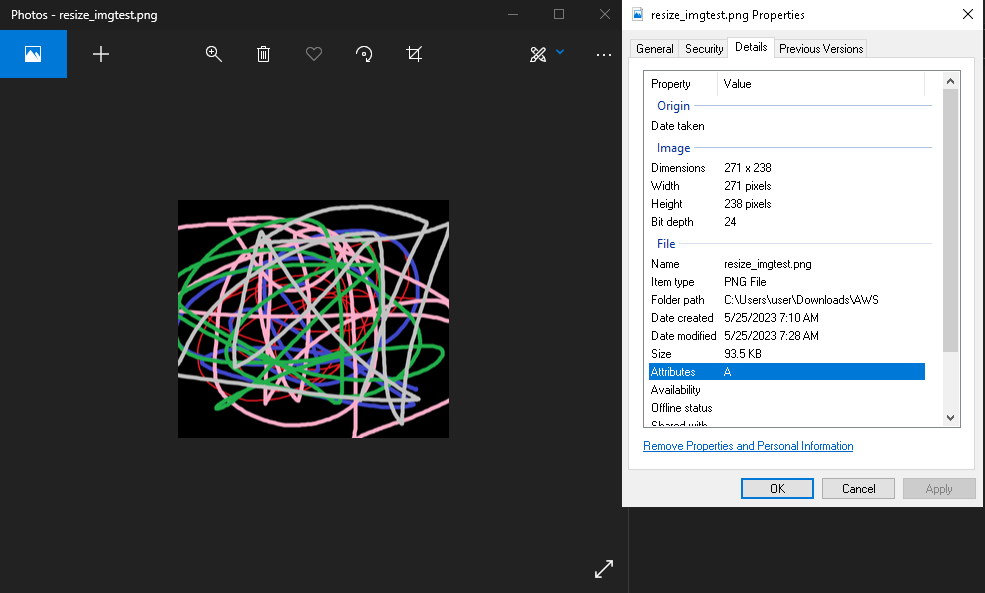
Note:

3 & 7: Received implies consumption of the message. (The message body is the file name.)

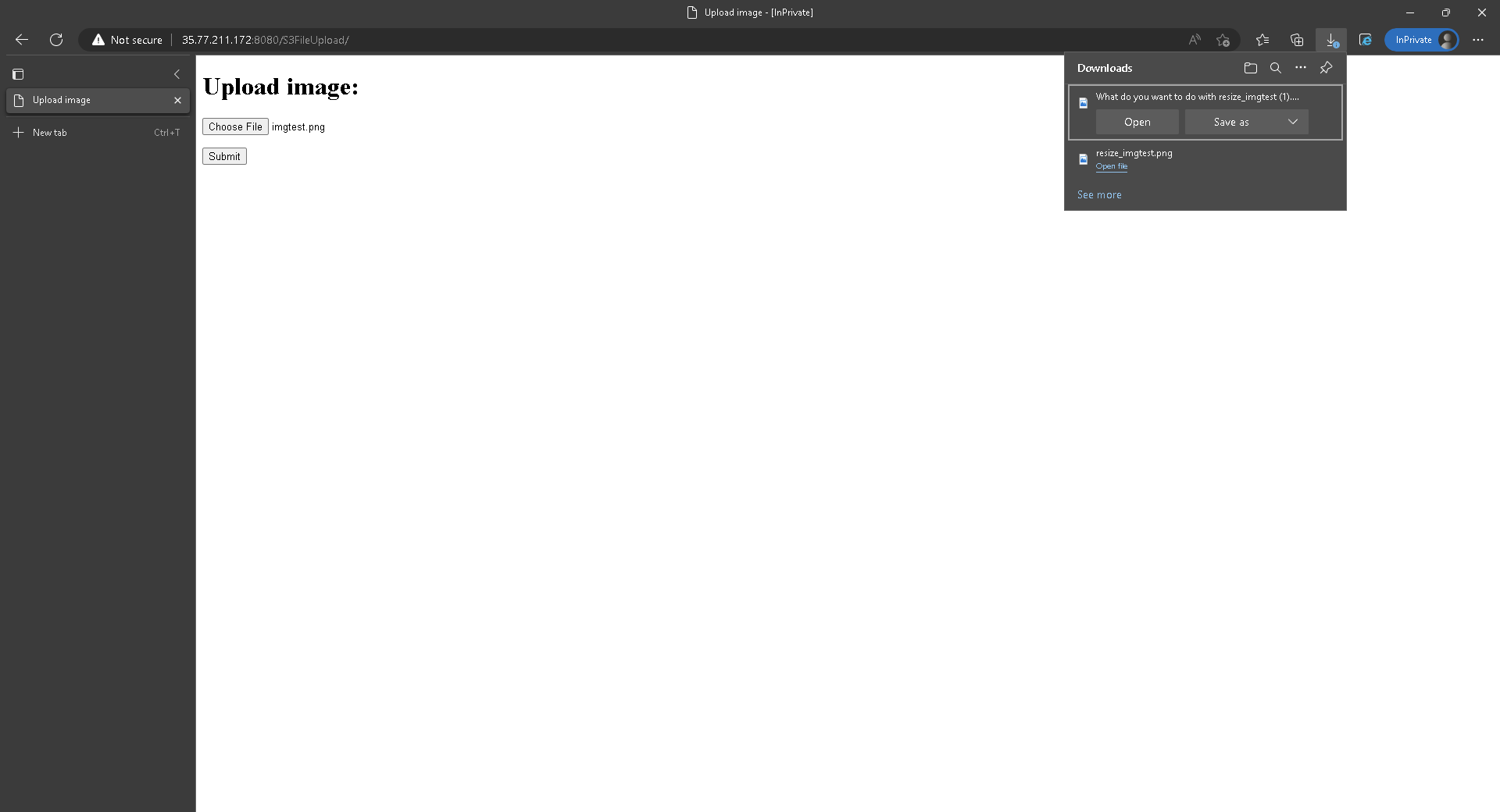
5: Resizing implies sending the resized image to S3.

**\* The above EC2 console view sorely demonstrates what is going on inside EC2 instances after running Processor.jar. In the actual case, I used the “screen” command to run the EC2 instances indefinitely even after I exited the SSH session.**

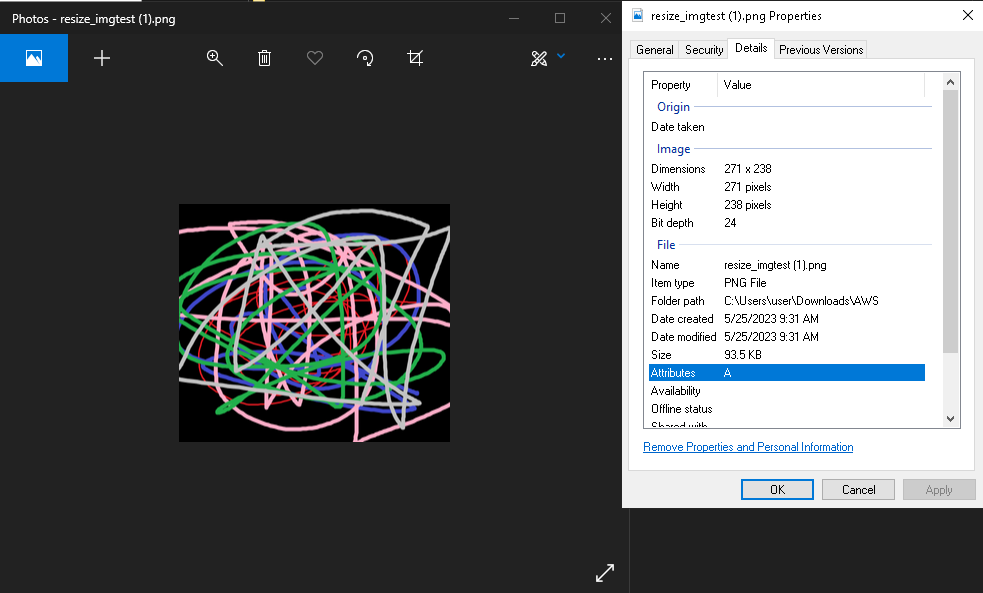
Resulting image



Website: <http://35.77.211.172:8080/S3FileUpload/>

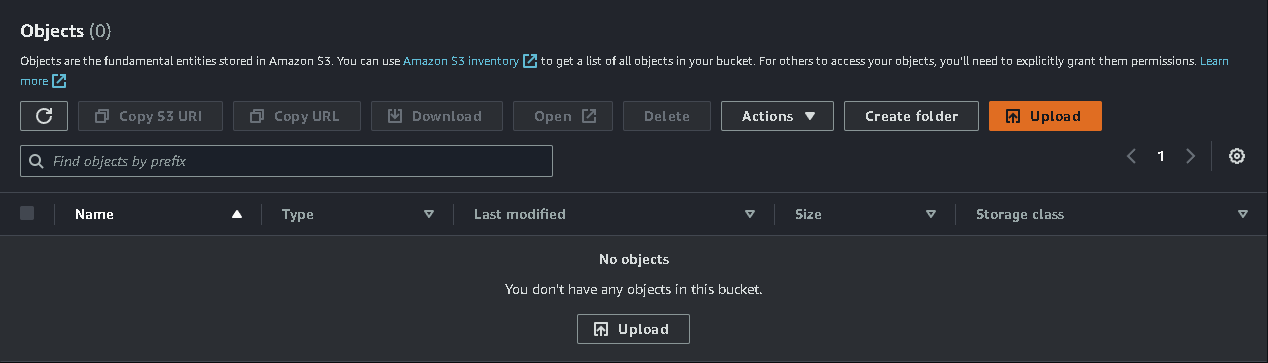


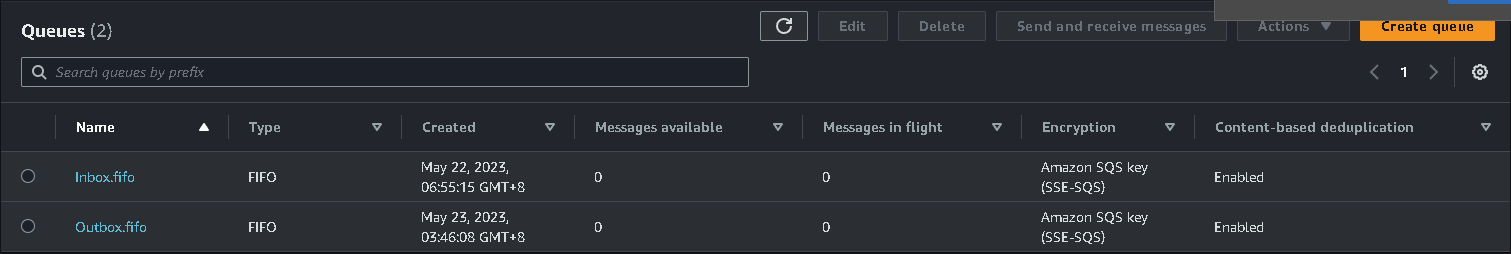
After uploading the image to my website and waiting for roughly a minute, the download for the resized image will show up.



The downloaded image is the same as the previous resized image.

On successful flow, both SQS queues and the S3 bucket should be empty.





**Code description**

S3 bucket name: “images3358”

Inbox.fifo URL: “https://sqs.ap-northeast-1.amazonaws.com/713581367265/Inbox.fifo”

Outbox.fifo URL: “https://sqs.ap-northeast-1.amazonaws.com/713581367265/Outbox.fifo”

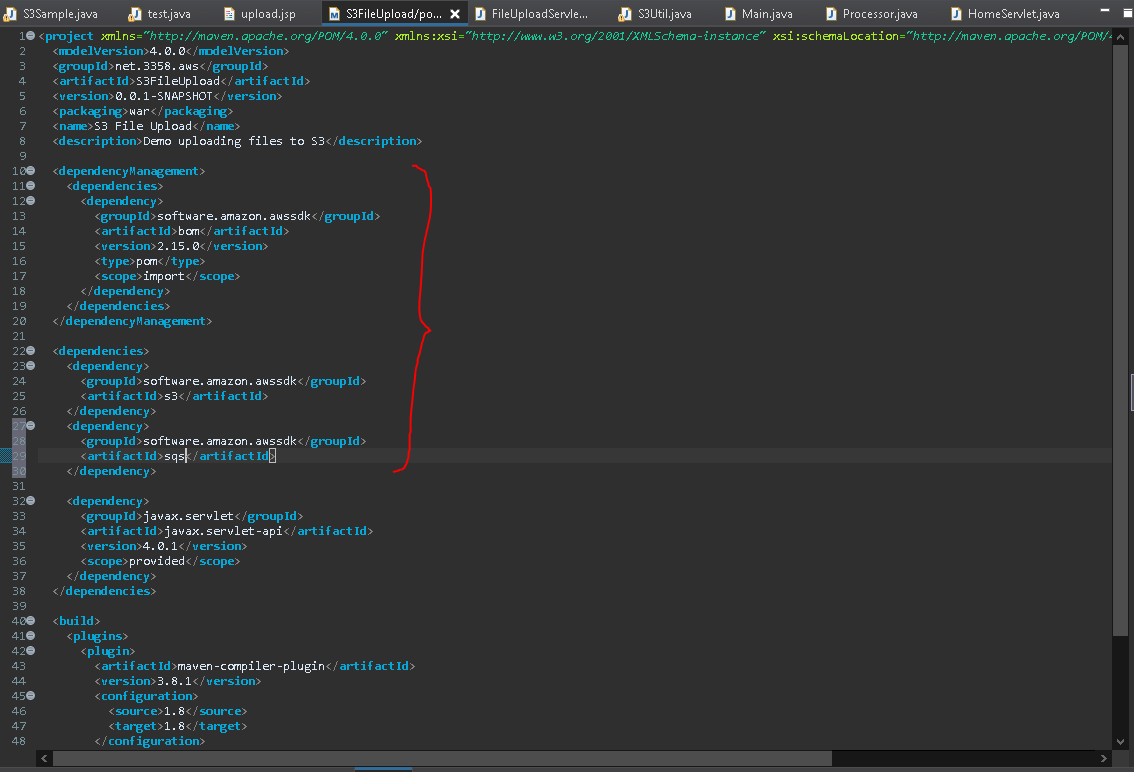
The flow of the local code (Main.jar) is as follows:

1. Initialize S3 and SQS classes
2. Send an image with filename as key to S3 and a message with the file name to the inbox SQS queue
3. Poll messages from the outbox SQS queue and look for the “resize\_”+fileName message.
4. Once consumed the “resize\_”+fileName message, get and delete the resized image with “resize\_”+fileName key from S3.

The flow of the EC2 code (Processor.jar) is as follows:

1. Initialize S3 and SQS classes
2. Poll messages from the inbox SQS queue and look for the fileName message.
3. Once consumed the “resize\_”+fileName message, download locally and delete the image with the fileName key.
4. Resize the image. This is done by running the linux “convert resize” command in Processor.jar
5. Send and rename the resized image to S3.
6. Send a message with body “resize\_”+fileName to the outbox SQS queue..

The flow of the servlet is similar to that of Main.jar, but it uses S3Client and SqsClient in software.amazon.awssdk, S3Client and SqsClient to send requests/response instead of AmazonS3 and AmazonSQS in com.amazonaws. The web project is made in Eclipse and is exported as a .war file.

The page is coded in upload.jsp. Upon uploading a file and clicking the submit button, it calls the servlet FileUploadServlet.java (which then calls a helper class S3Util.java to send upload requests) to send upload and download requests. The file submission is done with InputStream and file download is done with FileWriter (together with response.setHeader("Content-Disposition", "attachment; filename=\""+ fileName + "\"")). Moreover, the pom.xml needs to be configured to enable importing from software.amazon.awssdk (see below). For more details, please refer to the source code.

The source code of S3FileUpload.war can be viewed in the S3FileUpload folder.

.