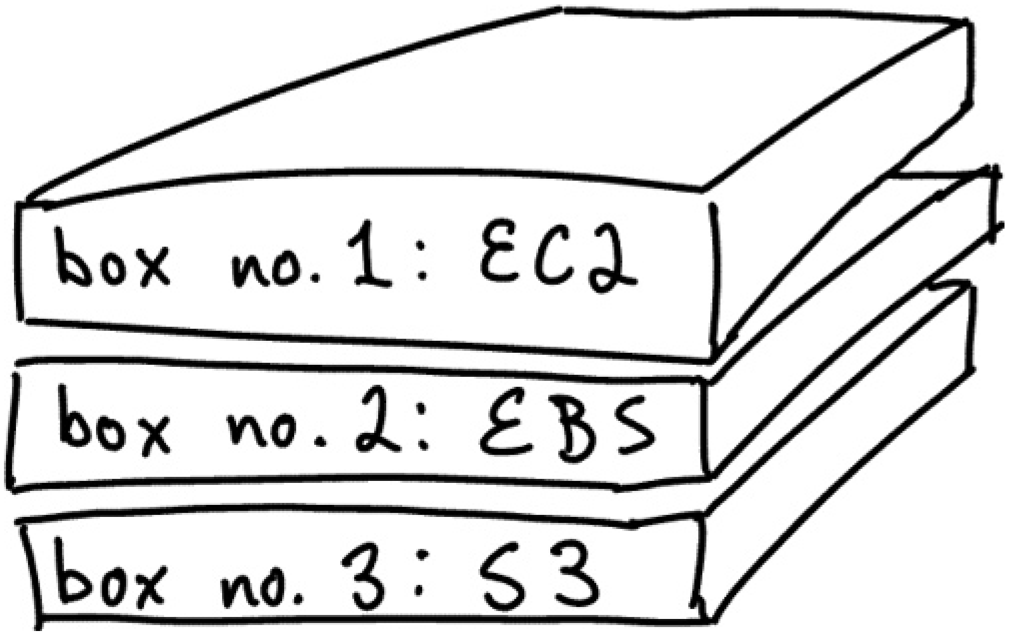

Student Name	Student Number
Bozhidar Lenchov	995959431
Shahriar Ninad	993690537
Kush Dua	996081957

1. General application architecture

Authenticate servlet is used for UI authentication/registration and redirects back to View Images page once authenticated or login page if unsuccessful.

For the image upload functionality, the FileUpload servlet takes care of saving the uploaded image to the server (i.e Amazon VM) temporarily, saves the image and 3 transformations on S3, and updates the database. The servlet then redirects back to the upload page, indicating if the upload was successful or not, allowing the user to upload more images, or retry the upload if the attempt was unsuccessful. The FileUploadStatic servlet takes in the same parameters (userID String and file binary image stream) but prints output as opposed to redirection (allowing for use of load generators).

LoadBalancer (static singleton custom library of functions) functions exist for updating worker pool statistics using the CloudWatch Metrics API and resizing the worker pool (increasing/decreasing). Since the servlets have to work with automatic workload generators that directly invoke the servlets (no login/session), load balance checks (seeing whether pool of workers needs to be changed) are also added at the beginning of the compute intensive servlets namely image upload (as an alternative to having a normal as well a static servlet, authentication (session/cookie check) and redirection back to login could only be performed if a referrer URL is set).

With automatic load balancing, the **manager first attempts to start the necessary workers from the pool of inactive ones** (removed from Load Balancer (LB) but not stopped) or create new instances using the worker AMI (ami-c669f7af) and place these new instances into a startup pool. After starting the instances, it **associates them with the Load Balancer (LB) in subsequent load balancing function** (in the LoadBalancer library) calls, since we assume the high load causing a resize continues. When an instance's state is observed to change to STARTED **after at least the minimum pool resize delay occurs** (e.g. another load balance can occur), it is moved from the startup pool to the active (worker) pool, and registered with the Load Balancer. **Therefore, after setting the pool size, we wait for instances to be in the Started state and these new workers are only added to LB when a file upload request (and a call to loadBalance function) takes place after the pool resize delay passes (same for auto LB). This is done to minimize chance of LB classifying them as inactive and having to wait even more for them to be functional in the LB.** This also complicates the code design, but does not rely on load balancer healthy/unhealthy threshold values in place (i.e. depending on set wait times before checking whether startinge up instances are not started (e.g. at most granular scale, every 60 seconds or whatever the set delay is), worker instances are only registered with LB once it is known they are fully operational).

Conversely, on worker pool decrease, the stopped instances are removed from the LB and initially placed in the inactive worker pool. After a minimum configurable timeout in the LoadBalancer library code (60s default), on a subsequent loadBalance function call, instances are stopped (and kept in inactive worker pool for future reactivation).

There is only 1 manager however who is able to resize the pool (BozhidarsInstance), but the code could easily be extended to support extension by workers, as long as there is a timeout period after resizing, during which no resizing occurs. Because of the centralized failure point and somewhat intensive nature of the resize requests, the manager is left out of the worker pool and thus not able to serve user requests directed at the load balancer URL. It has however a fully-functional project repository on its instance, and can be used to configure operational parameters or browse the site as a user, if someone connects to it directly.

Apart from that the architecture of the webapp is basic, as the functionality is limited. A header enforcing authentication and being responsible for redirect request parameter handling is included on each UI page. Login functionality is implemented through cookies and session variables as implemented in the Authenticate servlet.

2. Instructions for starting the application (including AMI(s) to run)

BozhidarsInstance (i-84ec54f7) is setup as the manager module instantiating workers. It has an Elastic IP of 54.235.69.246 which needs to be associated with it on startup. Pulling latest changes from repository, setup of the webapp and starting tomcat is automatically done as a cron job using the ~/startup.sh script . An AMI image of this instance was taken and used as a starting point for the worker instances. For proper LB setup, the manager does not need to be added (as it will not serve any user requests), but a default worker instance named ECE1779 Default Worker (i-e1478c93) needs to be started, removed and readded (to make sure it is functional with LB) before any user requests are performed.

In step form,
1. Launch BozhidarsInstance (i-84ec54f7).
2. Associate Elastic IP 54.235.69.246 with BozhidarsInstance.
3. Launch ECE1779 Default Worker instance (i-e1478c93).
4. Remove Load Balancer pool members.
5. When ECE1779 Default Worker instance is fully started after ~ 1 minute, add it to Load Balancer pool.
6. Wait an addition 2-5 minutes for metrics to kick in and worker and manager to be available in Manager UI.

The site can then be accessed from http://ece1779project1group1-282770155.us-east-1.elb.amazonaws.com:8080/ece1779-img-project1/ where ece1779project1group1-282770155.us-east-1.elb.amazonaws.com on port 8080 is the A record DNS name where the Load Balancer is listening (or http://54.235.175.246:8080/ece1779-img-project1/ with equivalent values translated into IP).

For normal testing, the above URL and links work just fine. For testing upload servlet with load generators, please use ece1779-img-project1/servlet/FileUploadStatic servlet which does not redirect.

3. Credentials

Test account of username “test”, password “test” is setup. The account used for manager UI access is user “root” password “root”. The user key for SSH for the manager and default worker instance can be made available upon request.

4. Anything else you think I will need to understand your code and how it works.

As a general note, in our testing we noticed reporting wasn’t very fast (e.g. took a good 10 minutes after starting up instances to see monitoring results appear through API). For seeing whether they were started and added to LB the most reliable way is through the online console.

To disable load balancing, either turn up the pool expansion threshold to 1,000% (the maximum we allow in the servlet storing these config values) or disable any calls to LoadBalancerLibrary.getInstance().loadBalance(...) and the manager’s LoadBalance servlet (in particular call to manager’s LoadBalance servlet from FileUpload and FileUploadStatic servlets).

In terms of LB config value limits, the following are hardcoded in the Threshold.java servlet:

CPU Grow Threshold Max = 1000 (%)
CPU Shrink Threshold Max = 200 (%)
Ratio Expand Pool Max = 5 (x)
Ratio Shrink Pool Max = 5 (x)
Delay between pool resize = 60 (s)

Testing on a local tomcat server, manual setting of pool size works (verified on EC2). Auto LB on EC2 untested, but using 95% same codepath so should work, assuming LoadBalance servlet invoked by human/automated clients when uploading.

For any questions, comments or issues, please don't hesitate to contact Bozhidar at bozhidar.lenchov@mail.utoronto.ca .

Thank you and enjoy our creation :)
-Shahriar, Bozhidar and Kush