ECM3408 Enterprise Computing Continuous Assessment 2019-20

Candidate number: 089004

1. Microservice Identification

For realising the e-mail delivery service, there are three microservices required which can be seen below. It is important to note that for each server, there will be an MSA and MTA. The BBS server is accessible by all servers and is crucial in linking them all together.

- Mail Submission Agent (MSA)

Once a user has created an email, the MSA places the email into the outbox of the user's mail system. From here, the MTA can access it and can proceed with the next steps. The MSA can also be used to show the emails that are sitting within a user's inbox and outbox at any time. It can also be used to make adjustments to the emails, such as marking them as read or deleting them (these functions can be accessed by the MTA too: when sending an email for example, the email would be deleted from the outbox once sent). Finally, the MSA can be contacted by the MTA when a new email is received, in order to add it to the user's inbox.

- Message Transfer Agent (MTA)

The MTA uses the MSA to read and delete a message from the user's outbox, then sending it across the network to another email server – this is done at regular intervals. In order to know which email server to send it to, the MTA works with the BBS in order to find the network address based of the source or destination email address (for example, wilma@here.com – the domain 'here.com' being the key part). Once the email is sent across the network, the MTA uses its MSA to add the message to another user's inbox. An MTA communicates with other MTA's on different servers, and plays a key part in the sending and receiving of messages.

- Blue Book Server (BBS)

The BBS keeps track of all the network addresses for the email servers. These may be obtained by supplying the BBS with the source or destination address of an email message, and it will return the network address. This essentially ties each network together, allowing the MTA to send emails.

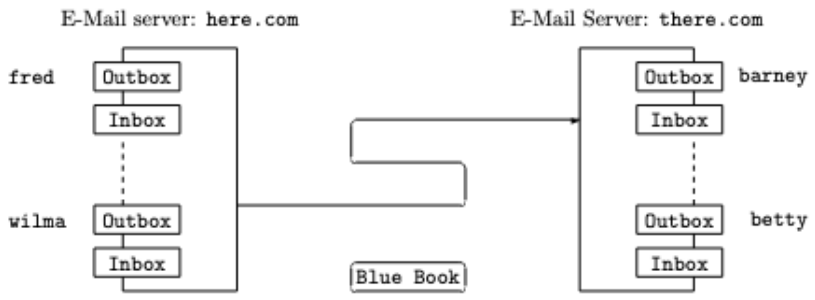
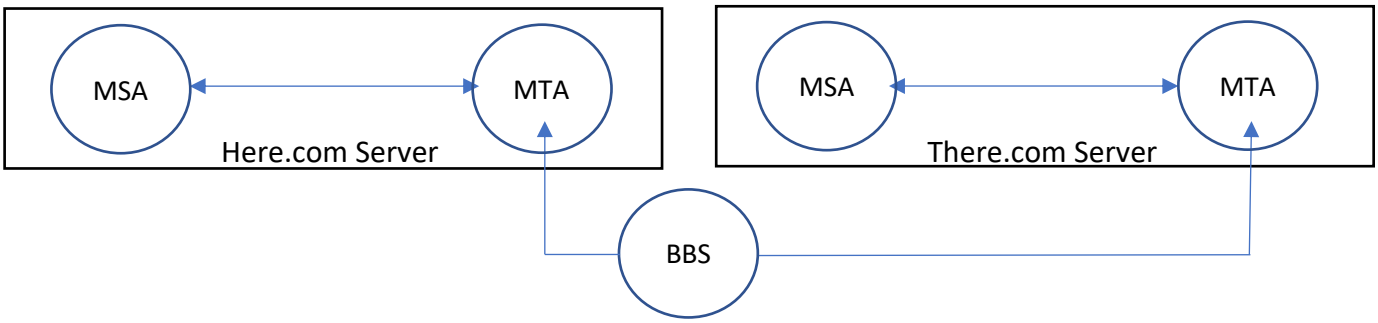


Figure 1: E-Mail servers.



2. Microservice Implementation

For the implementation of the microservices, I kept the MTA, MSA and BBS separate. I chose to run all three on different IP addresses, so that they can talk to each other independently. They all use the same internal docker ports to make things a bit simpler, however when accessed via localhost, they use the following.

Server - Here:
- MSA: localhost:3000
- MTA: localhost:3001

Server - There:
- MSA: localhost:3010
- MTA: localhost:3011

Independent
- BBS: localhost:3002

To access the microservices when hosted in their own docker containers, the following ports need to be used. I found that using the following command useful to access them when the docker containers are running, as this simulates how each docker container talks with each other within their networked environment: 'docker exec running-bbs curl http://192.168.1.7:8888/emails' for example returns all the emails in there here.com MSA inbox/outbox.

Server - Here:
- MSA: http://192.168.1.7:8888/
- MTA: http://192.168.1.8:8888/

Server - There:
- MSA: http://192.168.1.2:8888/
- MTA: http://192.168.1.3:8888/

Independent
- BBS: http://192.168.1.9:8888/

Please find below the best order to run the executables in, in order for them to function correctly.

1) Submission/start_network.sh
2) Here/msa/run.sh
3) Here/mta/run.sh

4) Here/mta/run_updater.sh (optional)
5) Here/bbs/run.sh
6) There/msa.run/sh
7) There/mta.run.sh
8) There/mta/run_updater.sh (optional)
9) There/bbs/run.sh

3. Microservice Deployment

I have created Dockerfiles and bash scripts for each microservice (MTA, MSA, BBS). Simply by running the bash script, it will build the docker image and run it. Assigning the applicable port and IP addresses.

If the updater bash script is activated (run_updater.sh), the MTA for each server will run every 5 seconds to check for new emails in the outbox. If it finds an email, it will send it. This can be activated for both MTAs (for here.com and there.com) if required – there is a separate executable bash script for each.

The MSA can accept a POST request in order to send an email. The format of the emails is as follows. The required values for a successful POST request are To, From, Subject, Body.

```
type email struct {
  ID          string    `json:"ID"`
  To          string    `json:"To"`
  From        string    `json:"From"`
  Subject     string    `json:"Subject"`
  Body        string    `json:"Body"`
  Time        time.Time `json:"Time"`
  Status      string    `json:"Status"`
  Location    string    `json:"Location"`
}
```

The MSA can also take calls to view the outbox, inbox, all emails, a single email, mark an email as read, delete an email, and add a new email to the inbox.

The MTA can send and receive emails – both can be called using a GET or POST request, respectively.

The BBS can be called to find an MSA/MTA address, it can share a list of all servers that it knows, it can create a server, return one server, update a server or delete a server from its storage.

```
type server struct {
  ID                string `json:"ID"`
  Address           string `json:"Address"`
  MsaAddress        string `json:"MsaAddress"`
  MtaAddress        string `json:"MtaAddress"`
}
```

Docker problems that were overcome(!):

The problem:
Unfortunately, I ran into a rather large error when running the docker containers on my Mac. All three docker containers function, with building and running not being a problem at all. A problem does occur however when running the http://localhost:3001/send GET request - this GET request activates the push method within the MTA, which makes another GET request to http://localhost:3000/outbox/emails , which returns the emails sitting in the outbox ready to be sent. I can only replicate this error when the Docker containers are activated. Strangely, when running the three .go files on my computer locally, they work fine and there are no issues when making the GET requests as noted above. Similarly, the issue does not happen when running Docker containers for the MSA and BBS, with the MTA running locally.

```
Get http://localhost:8080/outbox/emails: dial tcp 127.0.0.1:8080
: connect: connection refused
```

I tried changing the ports that the Mux router runs on for all files, closing docker containers, restarting my computer/docker etc, as well as writing completely new docker containers for blank files, adding mux routing capabilities with some methods to call Get requests on and I still got the same error. At this point I was very unsure what else to try and was quite frustrated to say the least!

How I overcame the problem:
After a lot of testing and digging, I realised that I had not set up a network to connect my docker containers to. After learning more about how networks are created and how they work, I created one to connect them all to. After that didn't quite work how I thought it would, I then started testing docker network commands, which led me to realise that my ports weren't forwarding how I thought they would either. Again, after some more digging I realised that I could not simply call 'http://localhost:3000' from within my golang microservice (MTA, MSA etc) to communicate with another of my microservices. Slowly I realised that a specific IP address was required so that I could set each microservice up to communicate with each other properly. From there, everything seemed to work well.

To conclude, it is possible to create two email servers from the code I have supplied (here.com and there.com), and you are able to send and receive emails between the two. I have also included a lot of error handling and messages, as these are very useful for both production and testing scenarios. Further, I have provided bash scripts to set up the docker

network, create the images and run the containers, as well as the regular send email checker(s) contained in the bash scripts.