Project

Due Date: April 30, 2021 11pm EST

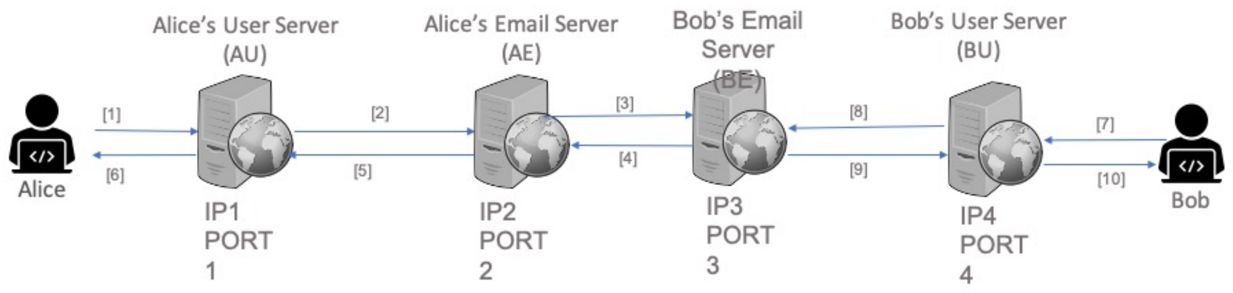
Objectives: Get hands on experience of network protocols (HTTP, SMTP, POP3) using Docker and Kubernetes.

Deliverables:

- Submit all your code to NYU Classes (as a .zip file).
- You can do this project in groups of 2. Each person should have a dedicated responsibility in the project.
- Also make sure to commit your programming question to Github. We will use the last commit time for grading.
  - In Github create a folder for the programming exercise and name it "**project**". Make sure to upload your code in there.
  - Maker sure to submit your code to NYU Classes as well.
- **No tolerance for copying and cheating.**
- Late submission policy (no exception)
  - 1 day – 50% off
  - 2 day – 75% off
  - 3 day – 100% off

Note: Changes/fixes are reflected with yellow coloring.

In this project you will implement simple email clients/servers (SMTP, POP3) and HTTP servers using Docker and Kubernetes. Figure below shows the end-to-end scenario that you need to implement for this project.



End to end scenario

[1] Alice sends email to Bob via her proxy User Server (AU) by sending an HTTP GET request: http://IP1:PORT1/email?from=IP2:PORT2&to=IP3:PORT3&message=hi. Make sure to use the same argument name as specified in here.
[2] AU will parse the arguments in the query and using SMTP protocol will send the message to AE as specified in "from" argument above.
[3] Once AE received the message, it will send the message to BE ("to") using SMTP protocol. Once BE received the message it will save the message to a local file.
[4] If everything is successful, BE will return a success message to AE.
[5] AE will return a success response to AU
[6] AU can now return HTTP 200 to Alice.
[7] Now Bob visits BU by: http://IP4:PORT4/email?from=IP3:PORT3
[8] BU initiate and retrieves the messages from BE ("from") using POP3 protocol
[9] BE returns the full list of messages to BU.
[10] BU returns the list of messages as a list: [msg1, msg2, …] with HTTP 200.

Notes

1. Alice to AU interaction happens via HTTP protocol.
2. AU to AE and AE to BE interactions happen with SMTP protocol.
3. Bob to BU interaction happens via HTTP protocol.
4. BU to BE interaction happens via POP3 protocol.
5. If one of the arguments of Alice's or Bob's initial HTTP requests missing, respond with HTTP 400.
6. For SMTP interactions, do not use any SMTP library, your program should be able to parse SMTP commands. You are supposed to implement SMTP yourself. Use the standard SMTP commands: "HELO", "MAIL FROM", "RCPT TO", "DATA", ".", "QUIT". Also use the standard codes for success status: "250", "354", "221". Check the book and the slides for more details. Both Alice's and Bob's email servers should be able to parse these commands in a message and understand.

7. When AU communicates with AE using SMTP, you can pass the IP and port of BE to AE (so that it can initiate SMTP with BE) using the RCPT TO command in SMTP. So instead of using Bob's email address, you can simply pass the IP of BE in RCPT TO command. Note that BE's IP and port number is given in Alice's initial HTTP GET request, i.e., "to" argument.
8. For POP3 interactions, do not use any POP3 library, your program should be able to parse and understands POP3 commands. Use the standard POP3 commands: "list", "retr", "dele", "quit" and responses: "+OK", "-ERR". Check the book and the slides for more details. You can skip the authorization phase for POP3.
9. In the scenario above, BE needs to support both SMTP and POP3 protocol. To be able to use the same port (PORT3) for both protocols, after the initial TCP handshake, you can assume that the first message specify which protocol is going to be used for the TCP connection. Hence first TCP handshake, then protocol to be specified by the client (SMTP vs. POP3), then based on the protocol the server would assume that the rest of the interaction will be based on that agreed protocol.
10. Do not hardcode IPs and port numbers. The steps defined above and the arguments should be sufficient to automate the flow.

You should implement the four components described above ("AU", "AE", "BE" and "BU") using Docker containers (4 Docker files). You can test the scenario above in your local set up. Please see the tips section for details about Docker and how to create a network where multiple containers can communicate with each other.

In addition, your set up should also work in Kubernetes. You can push each image ("AU", "AE", "BE" and "BU") to Docker Hub (as in previous programming assignment) and create a deployment in Kubernetes. You can take a look at this link to learn more about how to create a deployment file combining multiple Kubernetes resources into one file: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#creating-a-deployment . Provide a single file (containing your deployment and service) that can be used to deploy your whole application with: "kubectl apply -f deploy_email.yml". Be sure to expose your ports with nodePort. Name the single file as "**deploy_email.yml"**. Also note that K8s does not allow nodePort except for the range of 30000-32767.

Please submit the following:

- Create **project** folder in Github and create **four** separate folders within it: "**AU**", "**AE**", **"BE"** and "**BU**".
- For each folder provide **Dockerfile** and all the files that Dockerfile needs in order to run the specific server.
- "**deploy_email.yml**" deployment file for Kubernetes, places at the "**project**" folder.
- Add a readme.txt file and explain the contribution of each person in the project if done in a group.

- Push all your code to the **project** folder in your Github repo (with your netid).
- Finally make the zip of “**project**” folder and upload to NYU Classes.
- Your final commit date to the folder in Github will be used to grade the submission.

Tips:

To build an image from Dockerfile, use the following command (change the tag accordingly and notice the . at the end which specifies the context of the build):

```
docker build -t bulutmf/ae:latest .
```

To get servers running within Docker containers communicate with each other, you can create a Docker network with the following command:

```
docker network create N_NAME
```

Once created run your containers by specifying the network name with the following command (change the parameters accordingly):

```
docker run --network N_NAME --name C_NAME -p 53533:53533/udp -it bulutmf/ae:latest
```

Containers that are running within the same network, should be able to communicate with each other. You can learn the IP address of your container by inspecting the network that you created with the following command:

```
docker inspect N_NAME
```

Grading Rubric:

- Instructions followed properly (10pts)
  - (1pts) Code submitted to Github
  - (4pts) Folder name project exist and AU, AE, BU, BE are all exist under it.
  - (2pts) Code submitted to NYU Classes
  - (3pts) Dockerfile exist for each service (AU, AE, BU, BE)
- (60pts) End-to-end scenario described for the project works using a local development set up with docker.
  - AU (10 pts), AE (15 pts), BU (15 pts), BE (20 pts)
- (30pts) Deployment of provided **deploy_email.yml** to K8s with: “kubectl apply -f deploy_dns.yml” works for the end to end scenario.