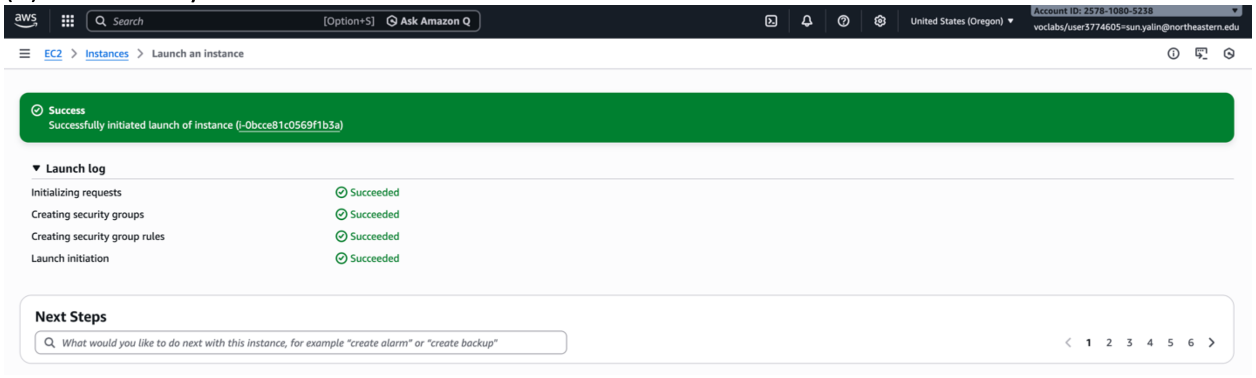
Assignment 1b

Yalin Sun

Part I

(1) Successfully launch an EC2 instance



aws
Search
[Option+S]
Ask Amazon Q
United States (Oregon)
Account ID: 2578-1080-5238
voclabs/user3774605=sun.yalin@northeastern.edu
EC2 > Instances > Launch an instance

Success
Successfully initiated launch of instance (i-0bcce81c0569f1b3a)

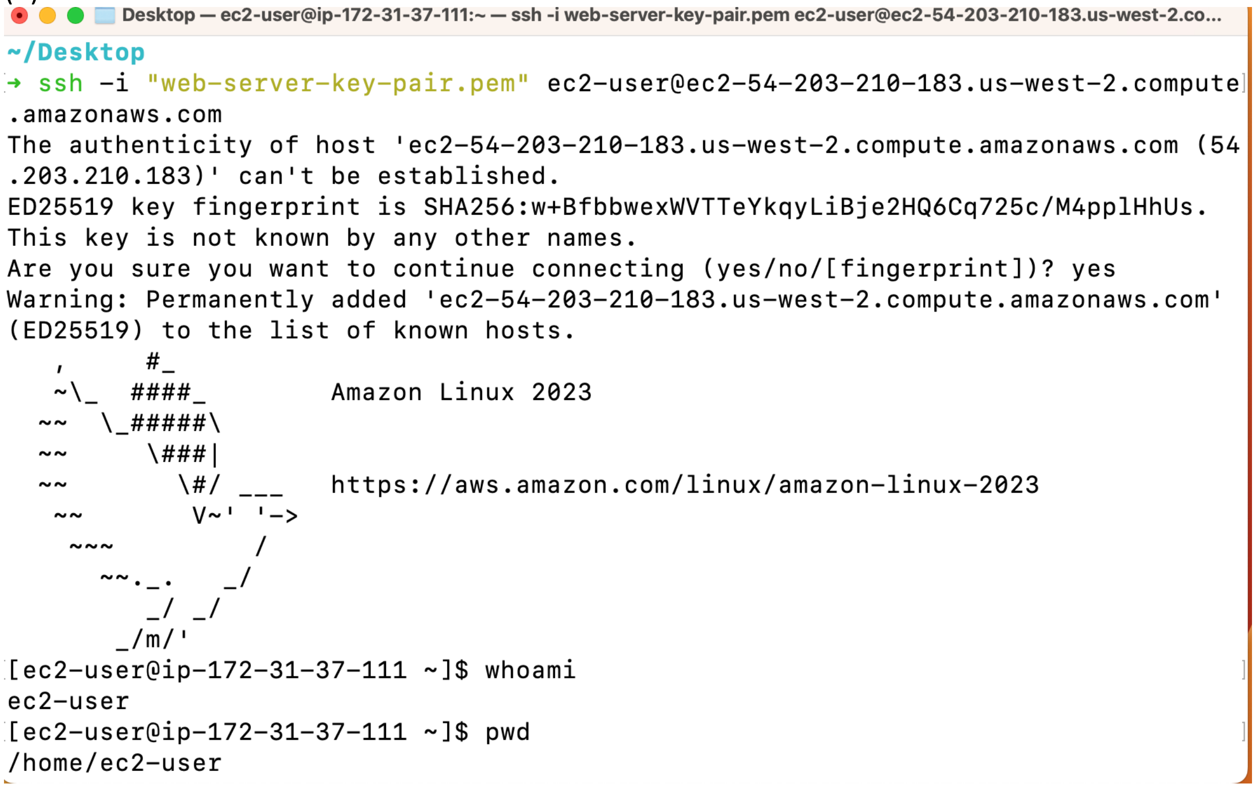
Launch log

Initializing requests	Succeeded
Creating security groups	Succeeded
Creating security group rules	Succeeded
Launch initiation	Succeeded

Next Steps

What would you like to do next with this instance, for example "create alarm" or "create backup"
1 2 3 4 5 6

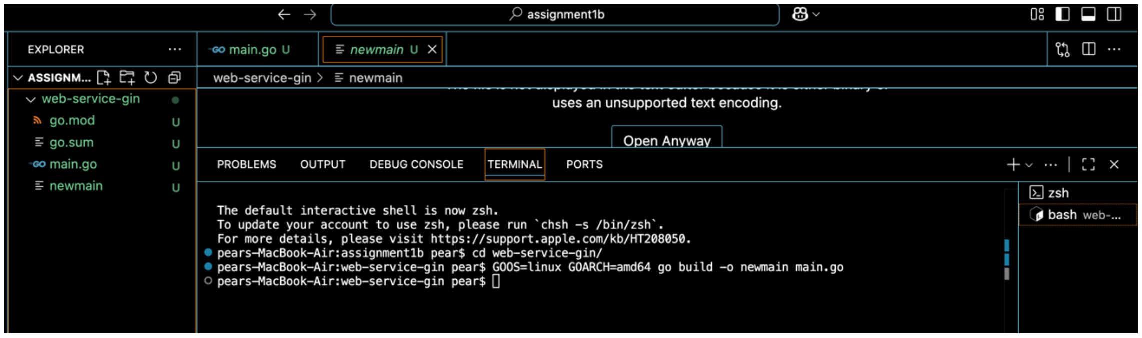
(2) Connected to the EC2 instance



Desktop — ec2-user@ip-172-31-37-111:~ — ssh -i web-server-key-pair.pem ec2-user@ec2-54-203-210-183.us-west-2.co...

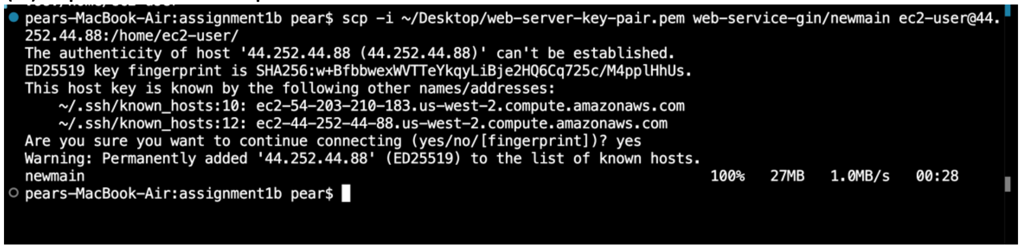
```
~/Desktop
➜ ssh -i "web-server-key-pair.pem" ec2-user@ec2-54-203-210-183.us-west-2.compute
.amazonaws.com
The authenticity of host 'ec2-54-203-210-183.us-west-2.compute.amazonaws.com (54
.203.210.183)' can't be established.
ED25519 key fingerprint is SHA256:w+BfbbwexWVTTeYkqyLiBje2HQ6Cq725c/M4pplHhUs.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-54-203-210-183.us-west-2.compute.amazonaws.com'
(ED25519) to the list of known hosts.
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-37-111 ~]$ whoami
ec2-user
[ec2-user@ip-172-31-37-111 ~]$ pwd
/home/ec2-user
```

(3) Cross compile Go-Gin code



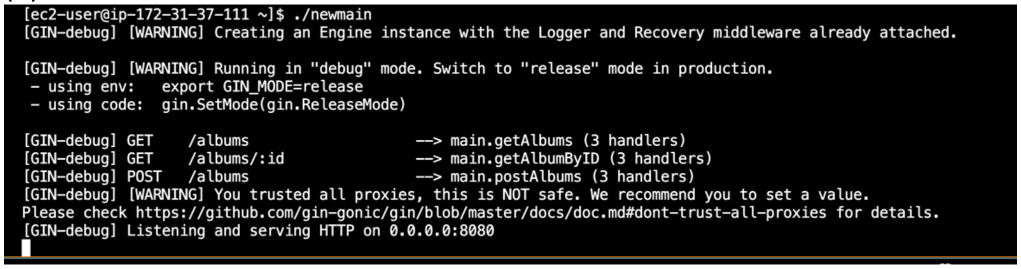
```
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
pears-MacBook-Air:assignment1b pear$ cd web-service-gin/
pears-MacBook-Air:web-service-gin pear$ GOOS=linux GOARCH=amd64 go build -o newmain main.go
pears-MacBook-Air:web-service-gin pear$
```

(4) Upload the compiled file



```
pears-MacBook-Air:assignment1b pear$ scp -i ~/Desktop/web-server-key-pair.pem web-service-gin/newmain ec2-user@44.
252.44.88:/home/ec2-user/
The authenticity of host '44.252.44.88 (44.252.44.88)' can't be established.
ED25519 key fingerprint is SHA256:w+BfbbwexWVTTeYkqyLiBje2HQ6Cq725c/M4pplHhUs.
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:10: ec2-54-203-210-183.us-west-2.compute.amazonaws.com
    ~/.ssh/known_hosts:12: ec2-44-252-44-88.us-west-2.compute.amazonaws.com
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '44.252.44.88' (ED25519) to the list of known hosts.
newmain                                                                  100%   27MB   1.0MB/s   00:28
pears-MacBook-Air:assignment1b pear$
```

(5) Run the Go server

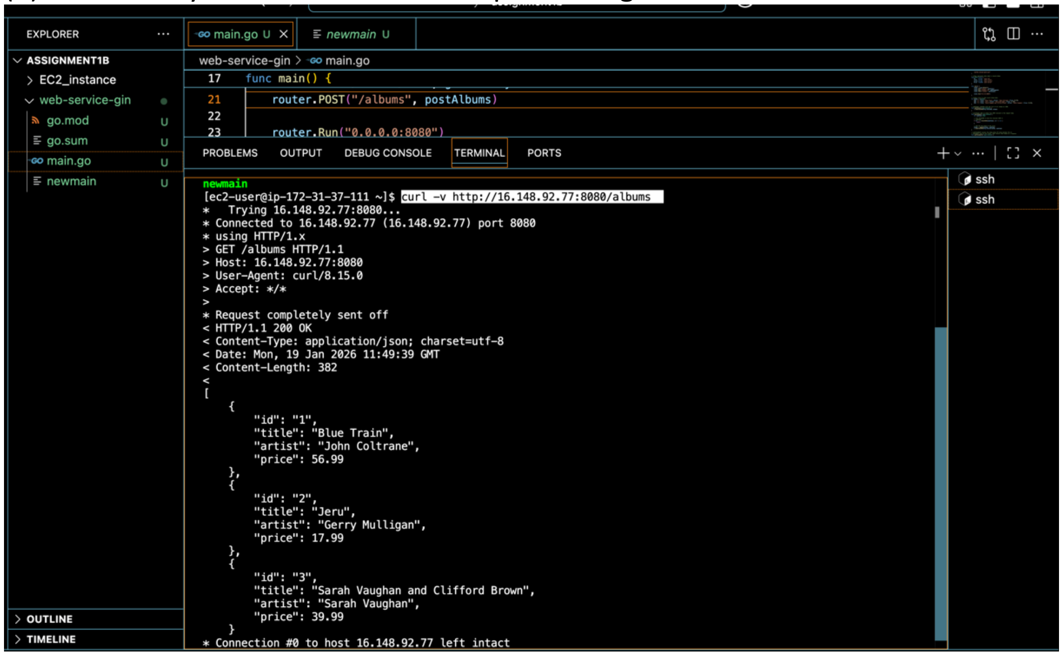


```
[ec2-user@ip-172-31-37-111 ~]$ ./newmain
[GIN-debug] [WARNING] Creating an Engine instance with the Logger and Recovery middleware already attached.

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
 - using env:   export GIN_MODE=release
 - using code:  gin.SetMode(gin.ReleaseMode)

[GIN-debug] GET    /albums                   --> main.getAlbums (3 handlers)
[GIN-debug] GET    /albums/:id               --> main.getAlbumById (3 handlers)
[GIN-debug] POST   /albums                   --> main.postAlbums (3 handlers)
[GIN-debug] [WARNING] You trusted all proxies, this is NOT safe. We recommend you to set a value.
Please check https://github.com/gin-gonic/gin/blob/master/docs/doc.md#dont-trust-all-proxies for details.
[GIN-debug] Listening and serving HTTP on 0.0.0.0:8080
```

(6) Successfully sent the HTTP GET request and get the result

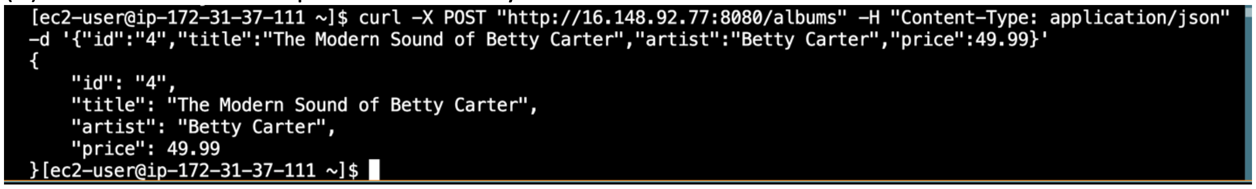


```
func main() {
    router.POST("/albums", postAlbums)

    router.Run("0.0.0.0:8080")
```

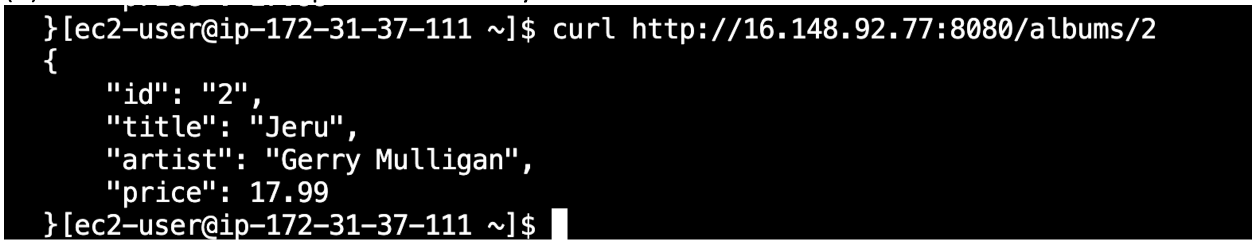
```
newmain
[ec2-user@ip-172-31-37-111 ~]$ curl -v http://16.148.92.77:8080/albums
*   Trying 16.148.92.77:8080...
* Connected to 16.148.92.77 (16.148.92.77) port 8080
* using HTTP/1.x
> GET /albums HTTP/1.1
> Host: 16.148.92.77:8080
> User-Agent: curl/8.15.0
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 200 OK
< Content-Type: application/json; charset=utf-8
< Date: Mon, 19 Jan 2026 11:49:39 GMT
< Content-Length: 382
<
[
    {
        "id": "1",
        "title": "Blue Train",
        "artist": "John Coltrane",
        "price": 56.99
    },
    {
        "id": "2",
        "title": "Jeru",
        "artist": "Gerry Mulligan",
        "price": 17.99
    },
    {
        "id": "3",
        "title": "Sarah Vaughan and Clifford Brown",
        "artist": "Sarah Vaughan",
        "price": 39.99
    }
* Connection #0 to host 16.148.92.77 left intact
```

(7) sent HTTP POST request successfully



```
[ec2-user@ip-172-31-37-111 ~]$ curl -X POST "http://16.148.92.77:8080/albums" -H "Content-Type: application/json"
-d '{"id":"4","title":"The Modern Sound of Betty Carter","artist":"Betty Carter","price":49.99}'
{
    "id": "4",
    "title": "The Modern Sound of Betty Carter",
    "artist": "Betty Carter",
    "price": 49.99
}[ec2-user@ip-172-31-37-111 ~]$ 
```

(8) sent HTTP GET-id request successfully



```
}[ec2-user@ip-172-31-37-111 ~]$ curl http://16.148.92.77:8080/albums/2
{
    "id": "2",
    "title": "Jeru",
    "artist": "Gerry Mulligan",
    "price": 17.99
}[ec2-user@ip-172-31-37-111 ~]$ 
```

Part II

1. What is a Virtual Machine? How is it running a program on VM different from running it locally?

What is VM?

A virtual machine (VM) is a digital version of a physical machine that functions as an isolated system. It has its own operating system, CPU, memory, storage, and network resources, which are provided through a virtualization layer called a hypervisor.

Difference Between Running a Program on a VM and Running it Locally?

(1) Resource Usage

When a program runs locally, it directly uses the computer's physical hardware and operating system resources.

When a program runs on a virtual machine, it runs inside the VM's operating system, and the VM's resources are allocated and virtualized from the underlying physical machine.

(2) Isolation

When running locally, programs share the same operating system with other processes, so failures or misconfigurations may affect the entire system.

When running on a VM, the program is isolated within the virtual machine, and crashes or configuration errors usually do not impact the host machine.

(3) Environment Consistency

Local execution is highly dependent on the host machine's operating system, library versions, and system configuration, which can lead to environment-related issues.

A VM provides a fixed and reproducible environment, making it well-suited for development, testing, and maintaining consistency between development and production environments.

(4) Performance and Flexibility

Running programs locally generally offers better performance and lower latency.

Running programs on a VM may introduce some performance overhead due to virtualization, but it offers greater flexibility, such as easy creation, deletion, and migration of virtual machines, which is especially useful in cloud computing.

2. You allowed the ssh connection from your IP address to the EC2 instance. What do you need to do if your IP address changes?

When I allowed SSH access to the EC2 instance, the security group was configured to permit connections only from my current IP address. If my IP address changes, I will no longer be able to connect to the instance using SSH.

To fix this, I need to update the security group inbound rules for the EC2 instance. Specifically, I must replace the old IP address with my new IP address for port 22. After updating the rule, SSH access will be restored.

3. Your EC2 will be assigned a different IP everytime it starts running from stopped, terminated, or restarted state. See how Elastic IP address can help, at a cost.

By default, an EC2 instance is assigned a new public IP address each time it is stopped and started, terminated and recreated, or restarted in certain cases. This means that the instance's public IP is not stable, which can cause problems if applications, users, or SSH configurations rely on a fixed IP address.

An Elastic IP (EIP) solves this problem by providing a static public IP address that can be associated with an EC2 instance. Once an Elastic IP is attached, the instance keeps the same public IP address even if it is stopped and restarted. This is useful for services that require a consistent endpoint, such as SSH access, web servers, or APIs.

However, Elastic IPs come at a cost. AWS charges for Elastic IPs that are allocated but not actively associated with a running EC2 instance. This encourages users to release unused Elastic IPs to avoid unnecessary charges.

4. What level of service are you "paying" for with your selected instance type? See instance type. Why would you care about these specs?

With the selected instance type t2.micro, I am paying for a low-cost, general-purpose level of service designed for light workloads. The t2.micro instance is part of the AWS Free Tier, which makes it suitable for learning, testing, and small applications with limited resource requirements.

The t2.micro instance provides a small amount of CPU, memory, and network performance, using a burstable performance model. This means it can handle short bursts of higher CPU usage, but sustained heavy workloads may be throttled once CPU credits are exhausted.

These specifications matter because they directly affect how applications perform. CPU, memory, and network limits determine how many requests the instance can handle, how responsive the application is, and whether it can support more demanding workloads. Understanding these specs helps in choosing the right instance type to balance cost, performance, and scalability, especially when moving from development to production.

5. What was different between getting your little backend going on GCP and AWS?

Getting my small backend running on GCP and AWS felt different mainly in terms of setup complexity, tooling, and defaults.

On GCP, the setup felt more streamlined and beginner-friendly. Many services are tightly integrated, and some configurations (such as networking and firewall rules) are handled more automatically. Deploying and testing the backend required fewer manual steps, which made it easier to get something running quickly.

On AWS, the process was more manual and required a deeper understanding of infrastructure details. I had to explicitly configure EC2 instances, security groups, SSH access, and networking rules before the backend could be accessed. While this made the setup more time-consuming and initially more complex, it also provided more fine-grained control over security and system configuration.

6. What testing did you do, and how did you do it?

To test my backend service, I performed both local testing and remote testing to make sure the application was working correctly in different environments.

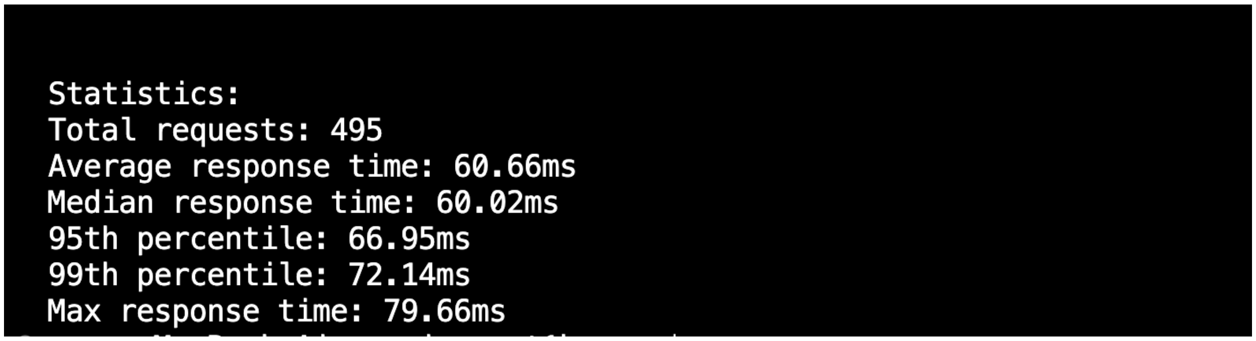
First, I tested the backend locally by running the server on my machine and sending HTTP requests to the endpoints using tools such as curl. This allowed me to quickly verify that the routes were correctly implemented and that the server responded as expected.

Next, after deploying the backend to the EC2 instance, I tested it remotely by sending requests from another machine using the EC2 instance's public IP address and port number. I used curl again to confirm that the server was reachable over the network and that the responses matched the local results. When issues occurred, such as connection timeouts, I checked security group rules, instance status, and server binding settings to debug the problem.

These testing steps helped ensure that the backend was functioning correctly both locally and in the cloud environment.

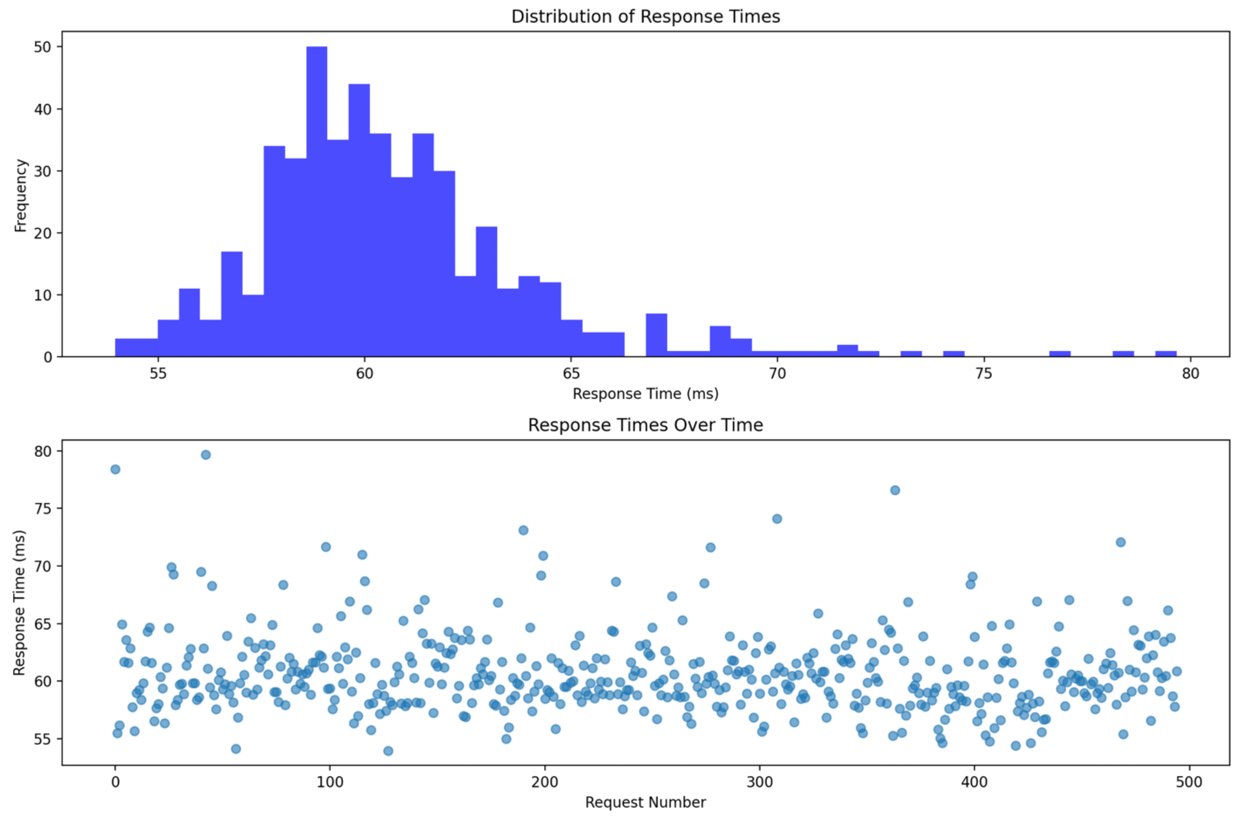
Part III

(1) Screenshot of the running results:



```
Statistics:
Total requests: 495
Average response time: 60.66ms
Median response time: 60.02ms
95th percentile: 66.95ms
99th percentile: 72.14ms
Max response time: 79.66ms
```

(2) Plot



(3) Answers to the questions:

1. Distribution Shape: Does your histogram show a long tail? What percentage of requests fall into the "slow" category?

The histogram shows a mostly normal distribution with a slight long tail on the right side. Most response times are clustered between 58 ms and 62 ms, while a small number of requests take longer, reaching up to around 80 ms.

Based on the statistics, the 95th percentile is 66.95 ms, which means about 5% of requests can be considered “slow” compared to the majority. Requests above 70 ms represent roughly 1–2% of total requests, indicating that slow responses are infrequent.

2. Consistency: Are the response times consistent throughout the 30-second window, or do you see patterns or spikes?

The response times are generally consistent throughout the 30-second test window. The scatter plot shows that most requests stay close to the average of around 60 ms, without a clear upward or downward trend over time.

Although there are a few isolated spikes where response times increase to around 70–80 ms, these spikes are sporadic and do not form a repeating pattern. This suggests that the system performance remains stable during the test.

3. Percentiles: What is the difference between the median and the 95th percentile response times? What does this indicate?

Median (50th percentile): 60.02 ms

95th percentile: 66.95 ms

Difference: approximately 6.9 ms

This relatively small difference indicates low variability in response times. Most requests are handled within a narrow time range, and there is no significant performance degradation for slower requests.

4. Infrastructure Impact: How might running a single container on a basic EC2 instance contribute to response time variability?

Running a single container on a basic EC2 instance (such as a t2.micro or t3.micro) can contribute to minor variability due to:

(1) Limited CPU resources and burstable performance

(2) Virtualization and container overhead

(3) Single-threaded or limited concurrency handling

These constraints can cause occasional delays when the instance experiences brief CPU contention or scheduling delays, which explains the small latency spikes observed in the results.

5. Scaling Implications: What would likely happen if there were 100 concurrent users instead of sequential requests?

With 100 concurrent users, the response times would likely increase significantly, especially at the higher percentiles. Since the service is running on a single container with limited resources, requests would begin to queue, leading to higher latency and potentially timeouts.

While the average response time might remain reasonable at first, the 95th and 99th percentile response times would likely increase sharply, indicating reduced performance under high concurrency.

6. Network vs. Processing: Are the longer response times due to network latency, server processing time, or both? How could this be investigated further?

The longer response times are likely caused by both network latency and server processing time, with server-side processing being the more significant factor. The relatively consistent baseline latency suggests stable network conditions, while occasional spikes point to processing or resource contention.

To investigate further, one could:

(1) Measure response times using localhost on the EC2 instance to isolate network latency

(2) Add server-side logging to measure request handling time

(3) Monitor EC2 metrics such as CPU utilization and network traffic

These steps would help determine how much latency comes from networking versus application processing.