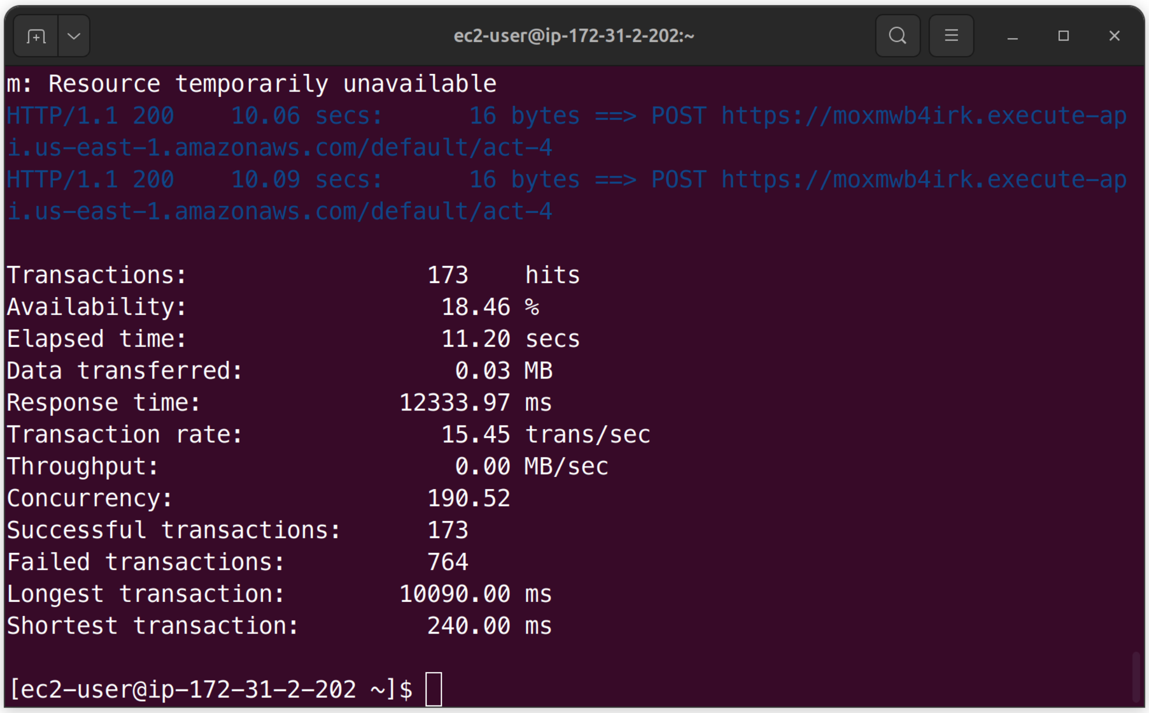
Chatrin Yoonchalard 6631304721

1. Run siege using the same number of clients (supposedly large) as Activity 2 and 3. Observe the response times, and the variations in response times across all requests. Do all requests see the same response times? Are there variations? Explain what you observe and why (same response time or variation)?

Ans: With the same number of concurrent clients (937 clients) the response time from this activity is lower than previous activity (as shown in the image below). I think its because the parallelism that Serverless(lambda) provide the AWS lambda spins up multiple separate containers to handle those requests.



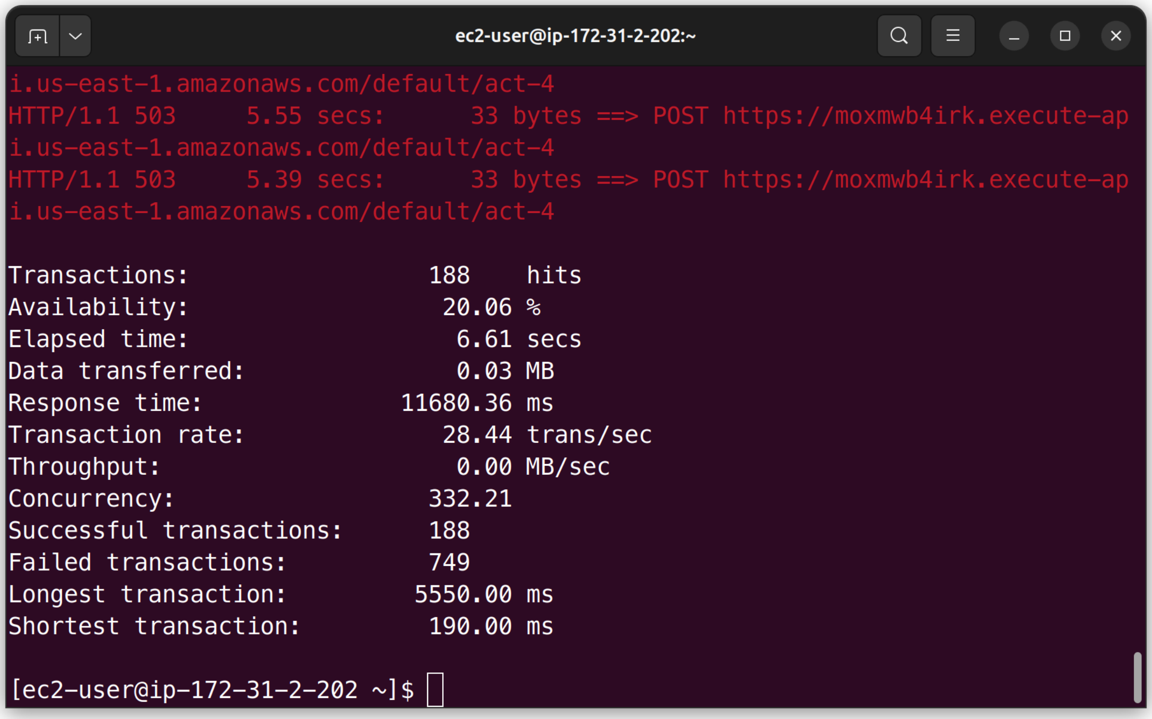
```
m: Resource temporarily unavailable
HTTP/1.1 200    10.06 secs:      16 bytes ==> POST https://moxmwb4irk.execute-ap
i.us-east-1.amazonaws.com/default/act-4
HTTP/1.1 200    10.09 secs:      16 bytes ==> POST https://moxmwb4irk.execute-ap
i.us-east-1.amazonaws.com/default/act-4

Transactions:                    173    hits
Availability:                     18.46 %
Elapsed time:                     11.20 secs
Data transferred:                  0.03 MB
Response time:                 12333.97 ms
Transaction rate:                 15.45 trans/sec
Throughput:                        0.00 MB/sec
Concurrency:                     190.52
Successful transactions:         173
Failed transactions:             764
Longest transaction:           10090.00 ms
Shortest transaction:            240.00 ms

[ec2-user@ip-172-31-2-202 ~]$ 
```

2. Wait for at least 20 minutes. Rerun siege. Observe the response times, and the variations in response times across all requests. Are the results the same as question 1? Do you observe any start up delays? Are they shorter or longer than before?

Ans: Even after waiting for one night, I did not observe a significant startup delay (Cold Start). In the **us-east-1 (N. Virginia)** region, AWS manages a vast pool of resources and utilizes **highly optimized micro-VMs (Firecracker)** that allow **Python** runtimes to initialize in milliseconds. Because my function is lightweight with minimal dependencies, the "Cold Start" was likely under 100ms—too fast to be distinguished from standard network latency in my Siege results.



```
i.us-east-1.amazonaws.com/default/act-4
HTTP/1.1 503     5.55 secs:      33 bytes ==> POST https://moxmwb4irk.execute-ap
i.us-east-1.amazonaws.com/default/act-4
HTTP/1.1 503     5.39 secs:      33 bytes ==> POST https://moxmwb4irk.execute-ap
i.us-east-1.amazonaws.com/default/act-4

Transactions:                      188    hits
Availability:                      20.06 %
Elapsed time:                       6.61 secs
Data transferred:                   0.03 MB
Response time:                  11680.36 ms
Transaction rate:                  28.44 trans/sec
Throughput:                         0.00 MB/sec
Concurrency:                      332.21
Successful transactions:          188
Failed transactions:              749
Longest transaction:             5550.00 ms
Shortest transaction:             190.00 ms

[ec2-user@ip-172-31-2-202 ~]$ 
```

3. How much did this experiment cost you? How does this compare to if you were to use EC2 to run your application? How does this compare to if you were to use Elastic Beanstalk to run your application?

Ans: This experiment cost approximately $0 to $0.00003, falling entirely within the AWS Lambda Free Tier. In contrast, using IaaS (EC2) or PaaS (Elastic Beanstalk) would have been significantly more expensive for this specific test.

vs. EC2 (IaaS): EC2 follows a provisioned capacity model where you pay for the instance per hour regardless of actual traffic. Even a small t3.micro instance would cost ~$0.01 per hour.

vs. Elastic Beanstalk (PaaS): Beanstalk often requires an Elastic Load Balancer (ELB) and at least one EC2 instance to be active, resulting in a minimum idle cost of ~$25/month.

Conclusion: Serverless is more cost-effective for this experiment because it eliminates idle costs. Since the application does not require constant server availability, Lambda only charges for the exact duration of code execution during the Siege test, whereas IaaS and PaaS charge for "uptime" even when no requests are being processed.

---

4. Apparently, with AWS Lambda, auto-scaling happens without user configuration. Compare the pro's and con's with configuring auto-scaling for IaaS and PaaS in Activity 3.

Ans: Here the comparison table

Feature	Serverless (Lambda)	IaaS (EC2)	PaaS (Beanstalk)
Scaling Trigger	Every single request.	Metric-based (CPU/RAM).	Metric-based (CPU/Network).
Configuration	None (Automatic).	High (Auto Scaling Group + Policies).	Medium (Tuning).
Speed	Milliseconds.	Minutes (Boot time).	Minutes (Boot time).

5. Compare the ease of development and ease of deployment between getting an app ready to work on AWS Lambda vs. if you were to get it to work on EC2.

Ans: The **AWS Lambda** offers ease of deployment through a code first approach. We simply just upload our code, and AWS handles the underlying infrastructure(runtime, OS, ...). However it can be harder to debug.

In contrast, **EC2** requires significantly more time for **initial development and deployment**, as you must manually configure the Linux environment, install dependencies (like Python, Nginx, or Docker), and manage security groups.

But EC2 provides **greater flexibility** and a traditional debugging experience where you can SSH into the server to inspect the environment in real-time.