Applied Distributed Systems Project 2

Introduction:

This report contains the load testing and fault tolerance test results done on our Weather Data Service Application. We have used Kubernetes to create replicas of our system services and deployed it on minikube through a single bash file.

Roadblocks :

Due to memory constraints, we could only create 4 replicas of our Radar service since we allocated its container 3GB RAM

Testing environment:

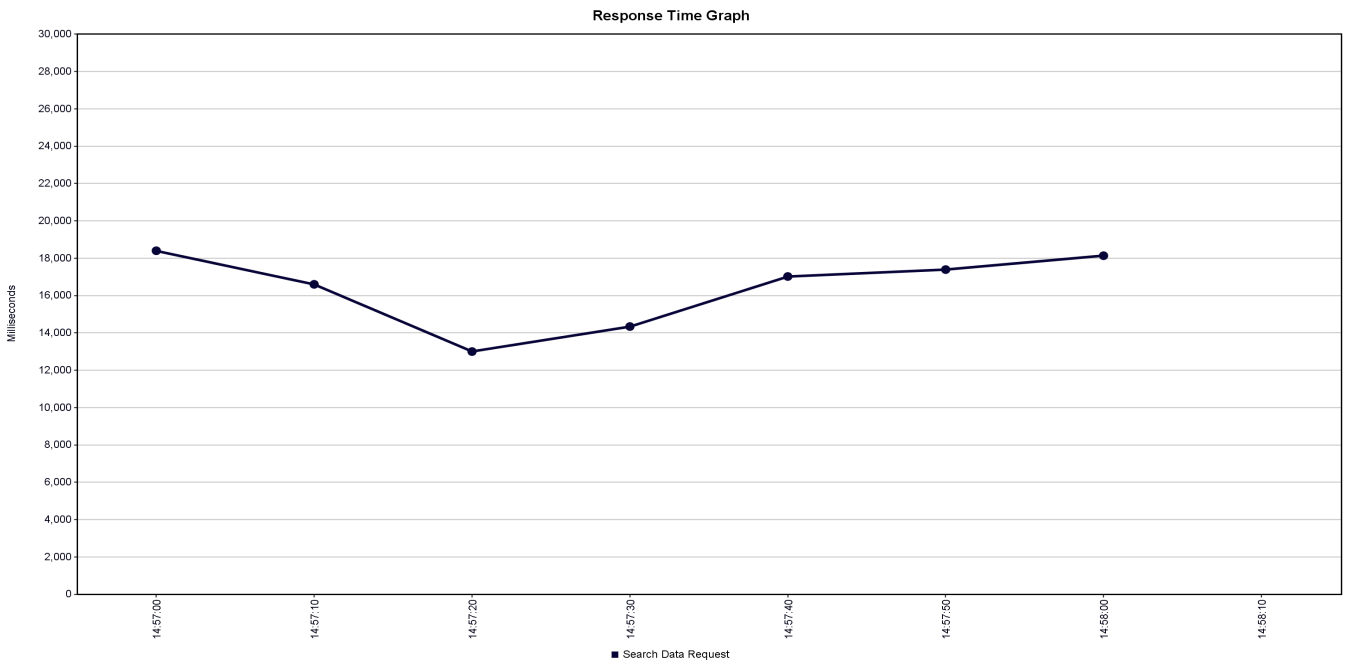
The testing for this application was done on Jmeter running in CLI and GUI modes. We have used laptops with 4 cores and 8 and 16 GB RAM's respectively.

Test Scenarios :

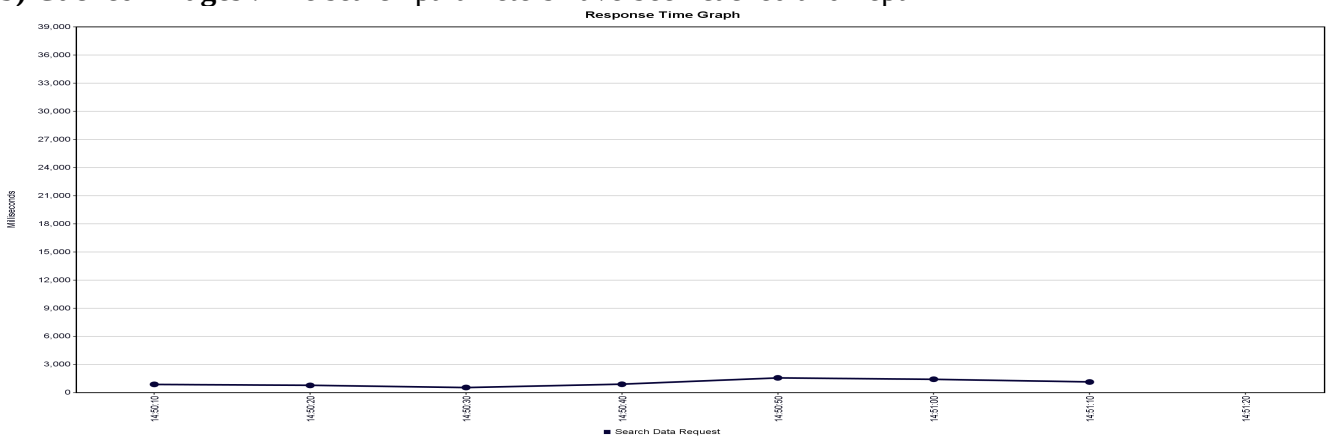
A] Radar Micro-service Tests

1: Radar Service Test with 1 instance
Test Context : Service Tested Independently.
Ramp Up Period : 100 seconds
Threads/Users : 5

Results :
a) Non-cached images : The search parameters were searched for the first time by the user



b) Cached Images : The search parameters have been cached and kept



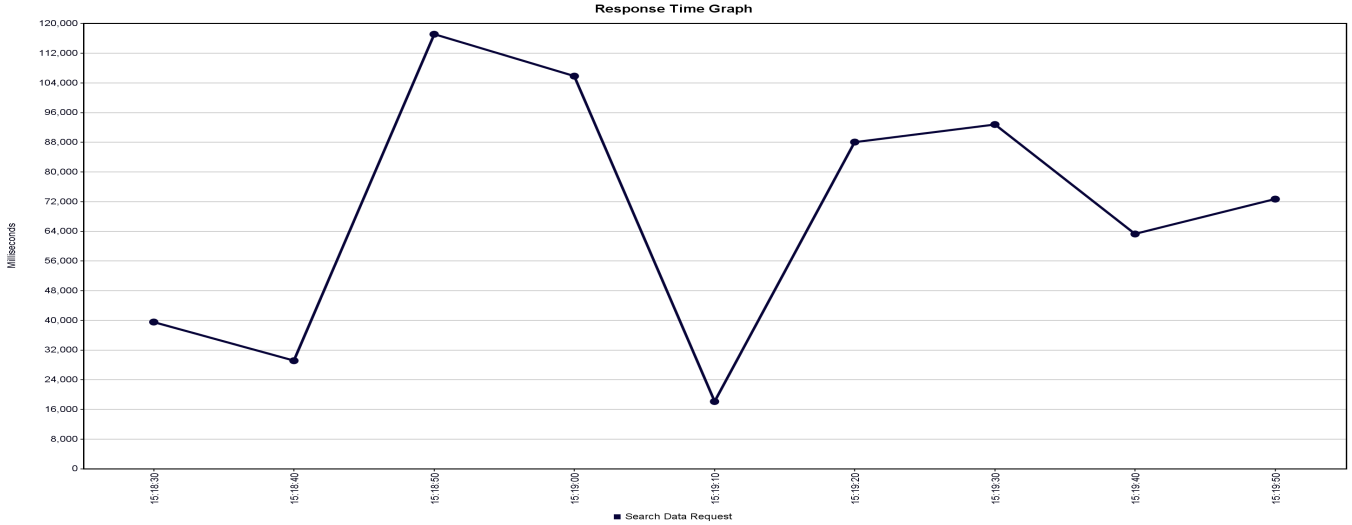
Tabular Results :

No of Users	Ramp Up	Comment
5	100 seconds	Steady response latency of 12 to 16 seconds
5	100 seconds	Response Latency of 1-3 seconds

Results Discussion: The Radar micro-service has a stead response latency of 12 to 16 seconds. Since the ramp up period is 100 seconds, each new request arrived in 20 second which gave it enough time to return a response within the time-frame and error rate was 0%. If the ramp up period were to be decreased, the error rate will increase which we will showcase in below tests

2: Radar Service Test with 3 instances
Test Context : Service Tested Independently.
Ramp Up Period : 100 seconds
Threads/Users : 10

Results: Response Time Graph

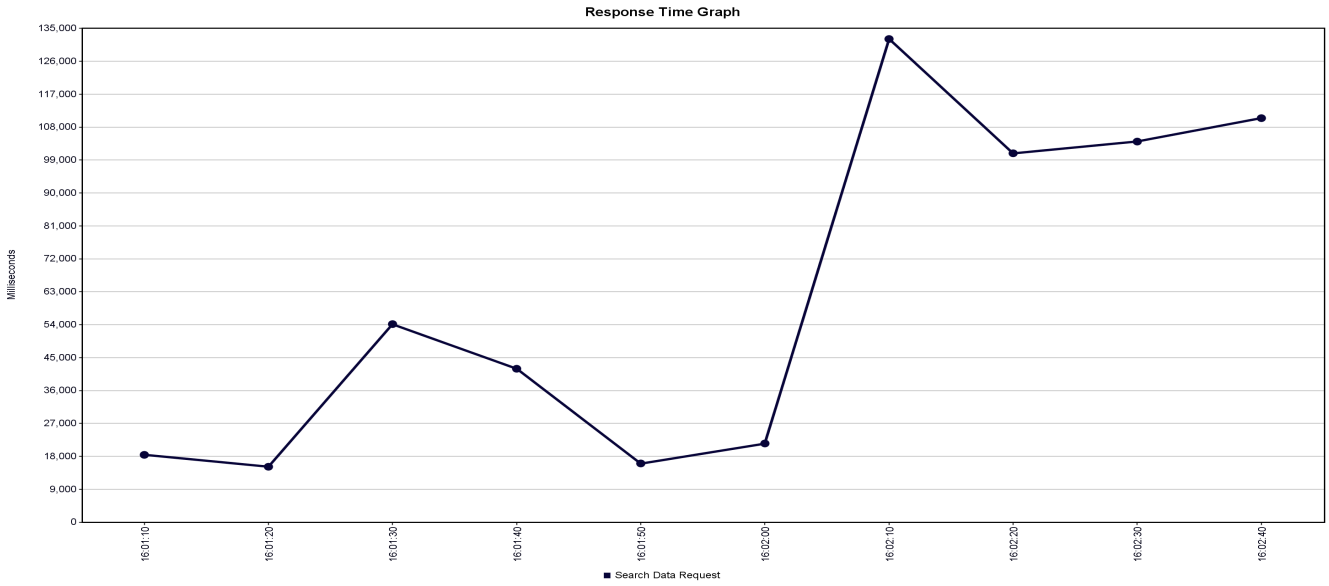


Error Rate : 30%
Throughput : 0.06545 requests/seconds
Discussion of Results: As visible in the graph above, since we haven't implemented a message queue, and since the ramp up period is lesser than the average response time of the service, we get a 30% failure rate.

3) Radar Service with 4 instances

Test Context : Service Tested Independently.
Ramp Up Period : 100 seconds
Threads/Users : 10

Results: Response Time Graph



Error Rate: 10%
Throughput : 0.05039 requests/seconds
Discussion of Results: Addition of an instance decreases the error rate even further down to 10%. 9 out of 10 requests succeeded. However, even with 4 instances running, system couldn't handle all requests which shows the importance of Message Queues.
The error rates are seen due to less resources available on the system and takes more time to compute. For the same reason, the throughput is also low

B] Authentication, Profile , History and Gateway micro-services test

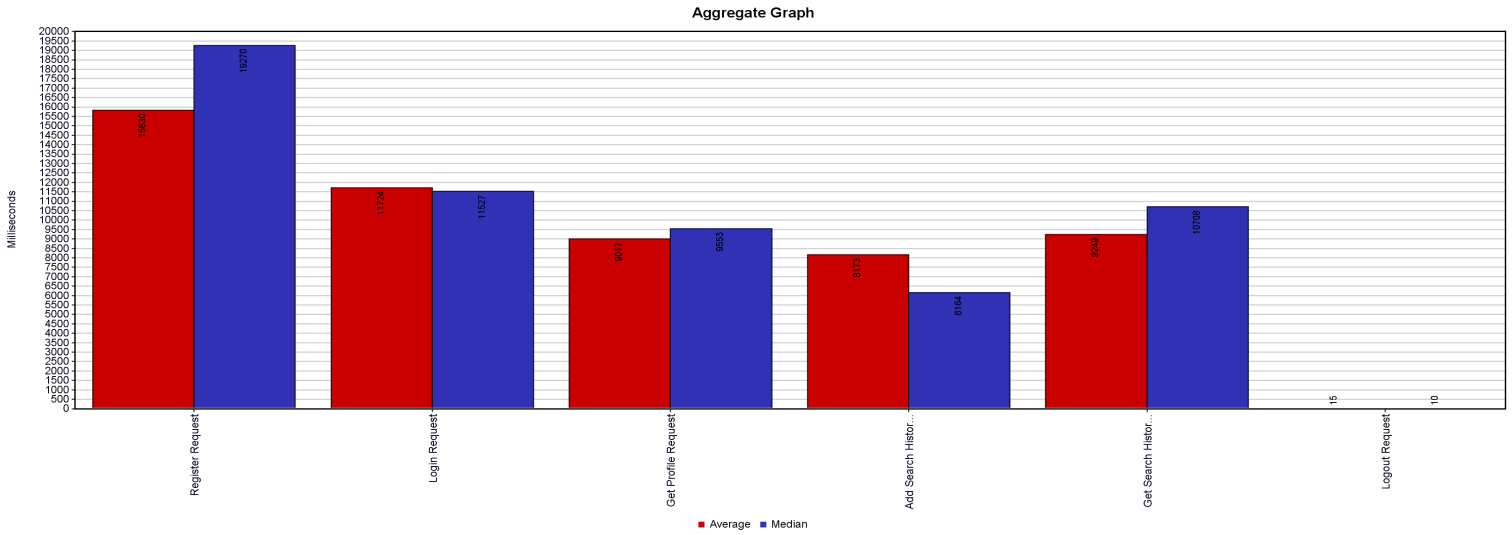
1: All micro-services with 3 instances

Test Plan Description :
a) Register Request : User will register using his email address, name and password.
b) Login Request: User will login into the application using his credentials
c) Set Cookie: After login, user gets a cookie as a response which is needed for further calls.
d) Get Profile Request: User profile details are returned.
e) Add Search History: The parameters searched for the weather service will be added to MySQL database
f) Get Search History: User search history will be returned
g) Logout: User will be logged out

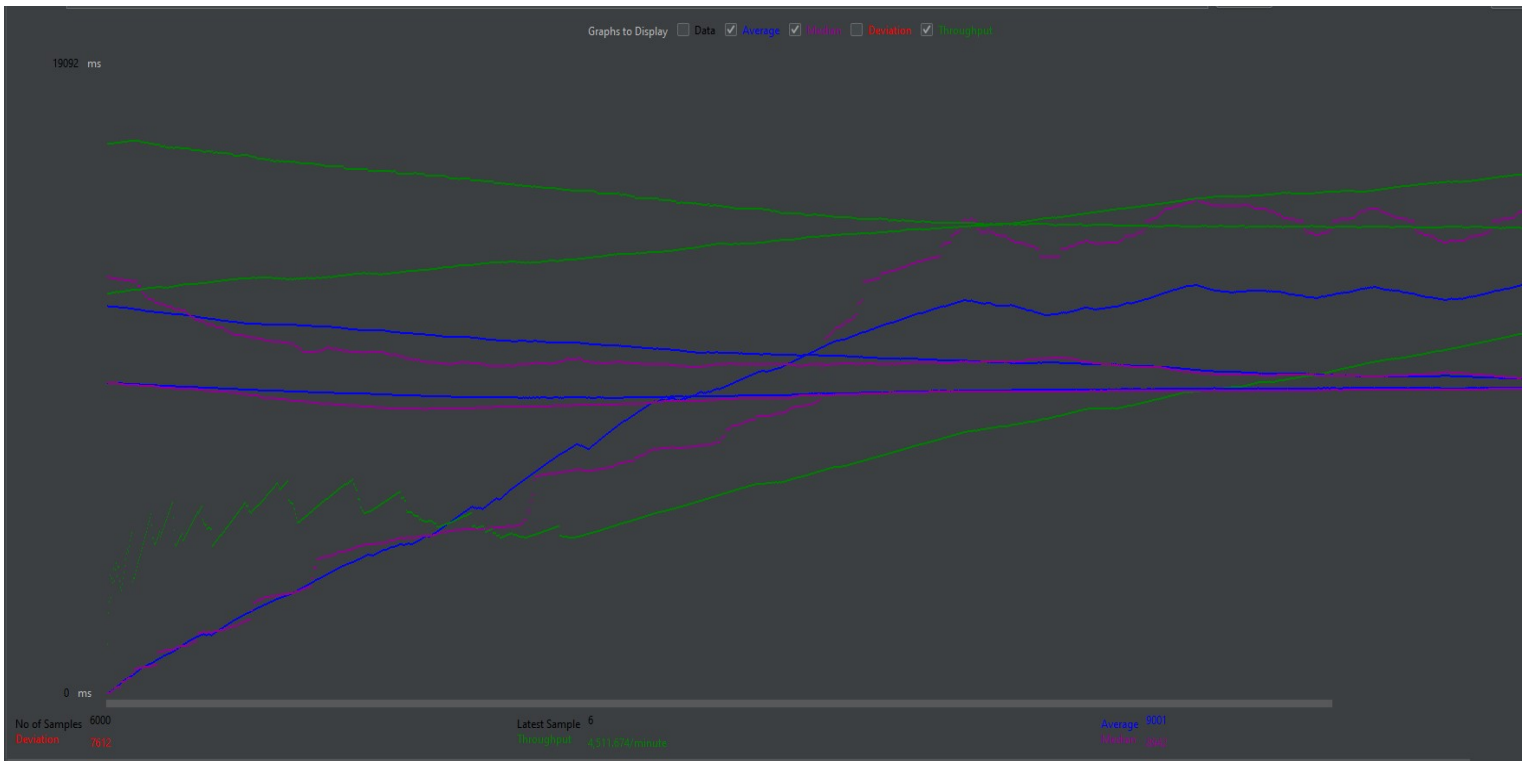
Ramp Up Period: 5 seconds
Threads/Users: 1000

Results:

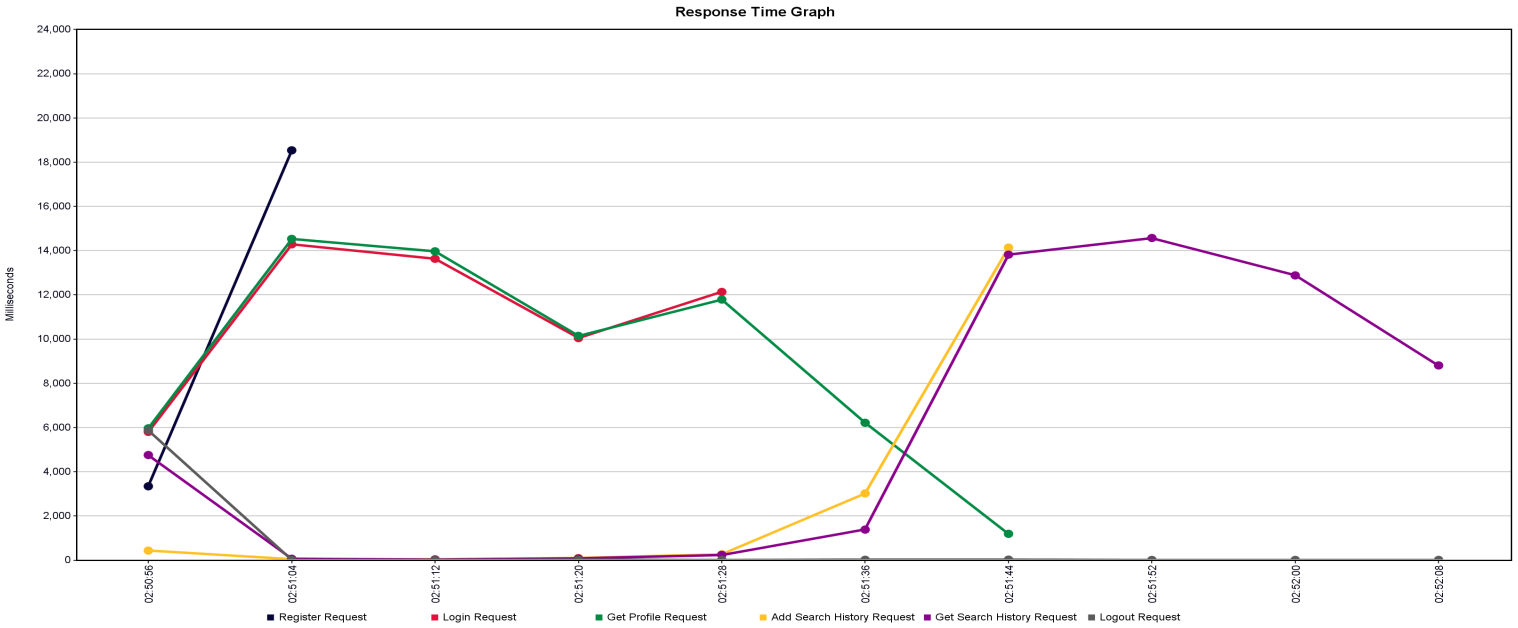
I) Aggregate Graph – Response times of all services



II) Real time running Graph Result



III) Response-Time Results



IV) Tabular Results

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**6000**	**0**	**0.00%**	**9001.55**	**4**	**33565**	**8949.00**	**19747.70**	**22317.75**	**25535.78**	**75.19**	**82.45**	**27.69**
Add Search History Request	1000	0	0.00%	8173.28	33	33565	6168.00	19177.10	24980.75	31136.55	13.53	14.00	7.70
Get Profile Request	1000	0	0.00%	9017.12	5	20287	9557.00	14079.80	14699.90	17108.60	22.63	24.50	8.02
Get Search History Request	1000	0	0.00%	9249.01	25	32888	10732.00	18645.70	20656.35	28492.15	13.20	14.08	4.85
Login Request	1000	0	0.00%	11724.63	680	20522	11533.50	15990.50	18154.45	20225.90	22.80	29.44	5.81
Logout Request	1000	0	0.00%	15.21	4	318	10.00	26.00	36.00	106.94	13.21	13.51	4.63
Register Request	1000	0	0.00%	15830.05	116	24498	19273.50	24005.10	24261.75	24436.95	34.47	37.29	10.80

Discussion of Results: Since the ramp up period is only 5 seconds, 200 users will be active and sending concurrent register request first as part of the test plan, Hence, the register requests finish the earliest as visible from the Response Time Graphs.

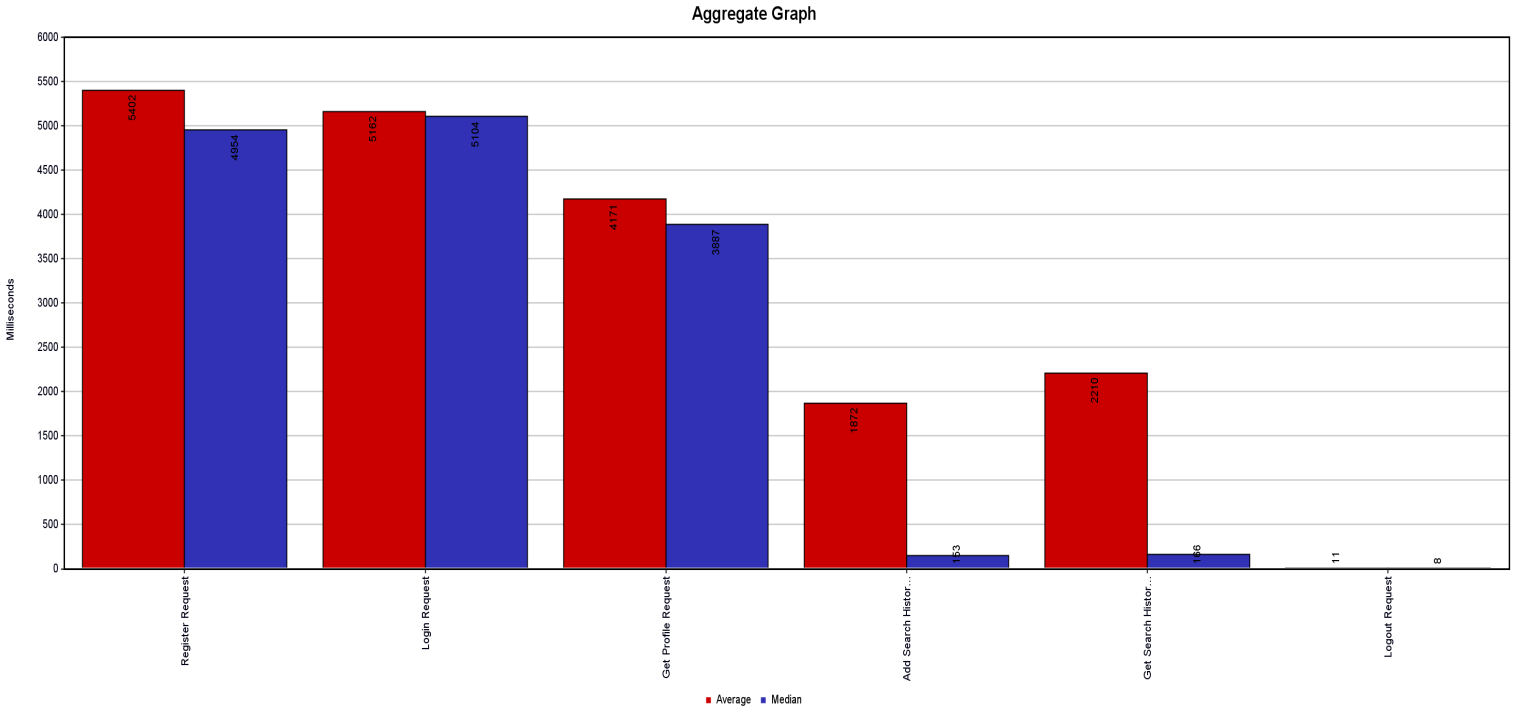
2: All Microservices with 3 instances

Ramp Up Period : 30 seconds

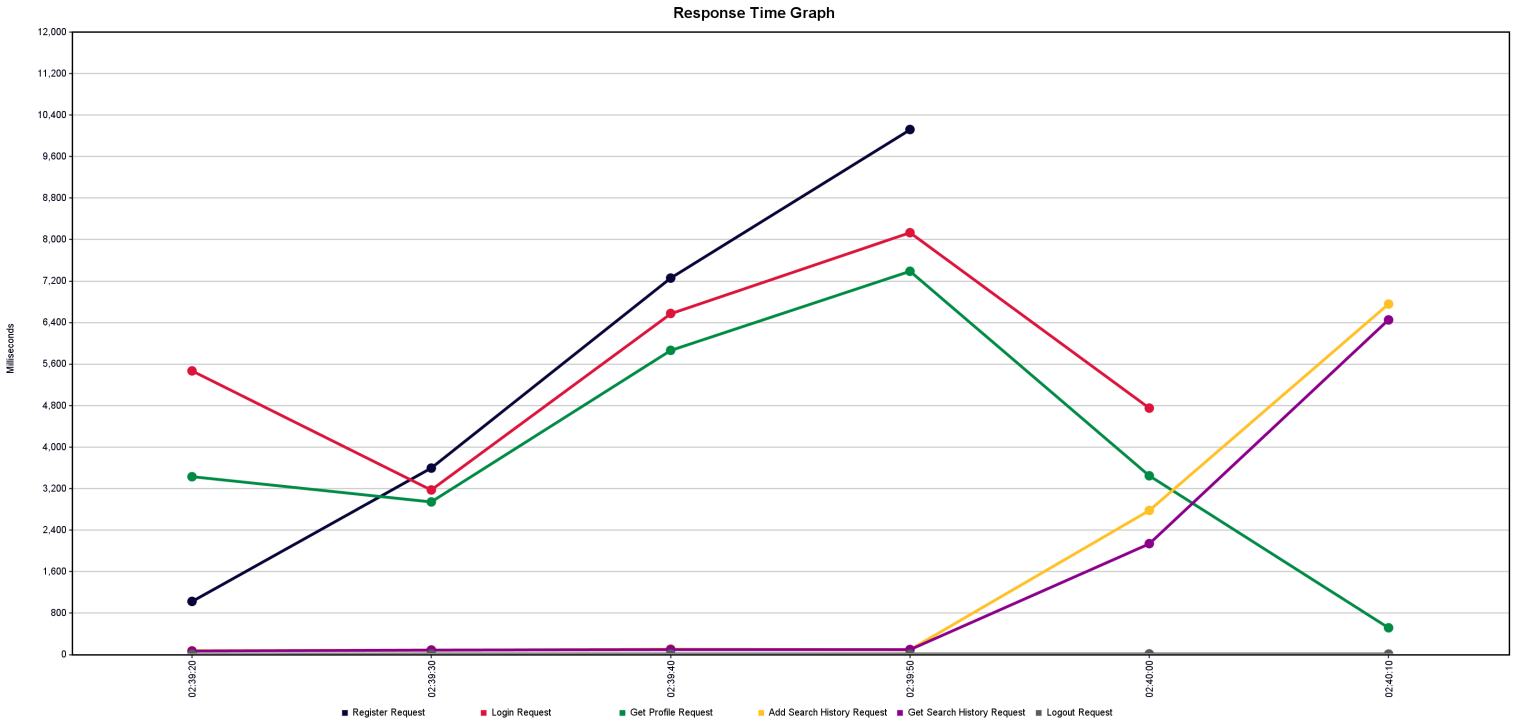
Threads/Users: 1000

Results:

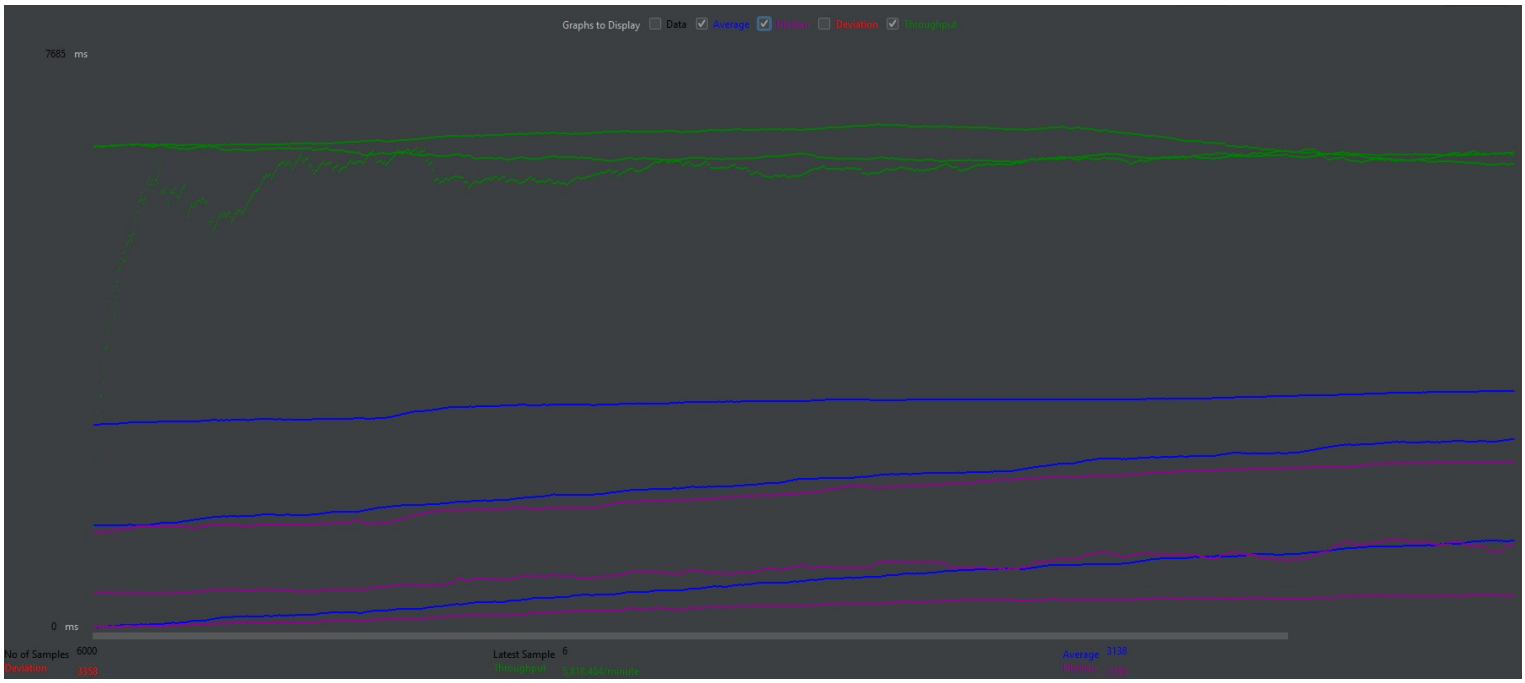
I) Aggregate Graph



II) Response Time Graph



III) Real-time Throughput Graph Results



Error Rate: 0%
Throughput : The table shows the throughput

Label	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Register Request	5402	45	16564	3508.26	0.000%	23.97	25.93	7.51	1108
Login Request	5162	36	13837	3013.04	0.000%	21.12	27.26	5.38	1322
Get Profile Request	4171	6	12563	2717.04	0.000%	21	22.75	7.45	1108.9
Add Search History Request	1872	30	16996	2699.1	0.000%	16.74	17.31	9.53	1059
Get Search History Request	2210	24	14420	3018.17	0.000%	16.21	18.1	5.95	1143.6
Logout Request	11	3	129	9.87	0.000%	16.22	16.59	5.69	1047
TOTAL	3138	3	16996	3358.91	0.000%	96.97	107.15	35.71	1131.4

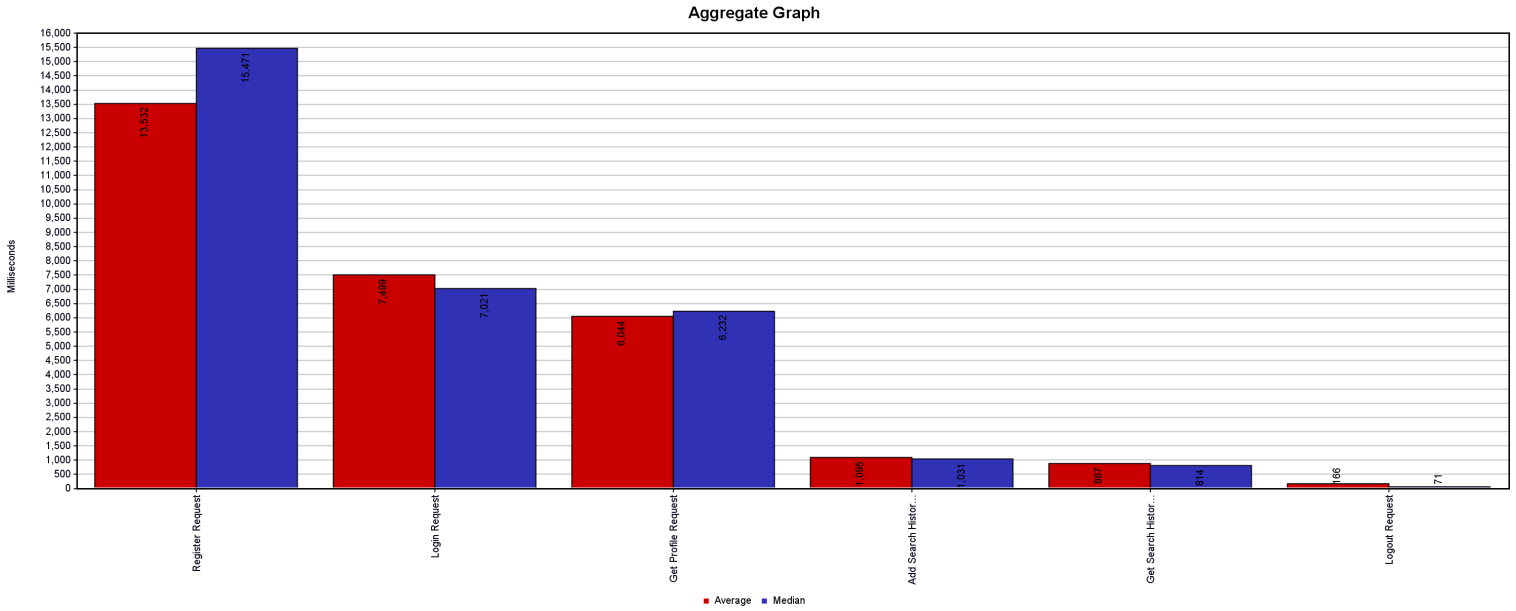
Discussion of Results: Since the ramp up period now increased, we get a better throughput of the application with no spikes visible in the real-time graph above. The response times also decreased for all requests due to the increased ramp-up period

3: All Microservices with 5 instances

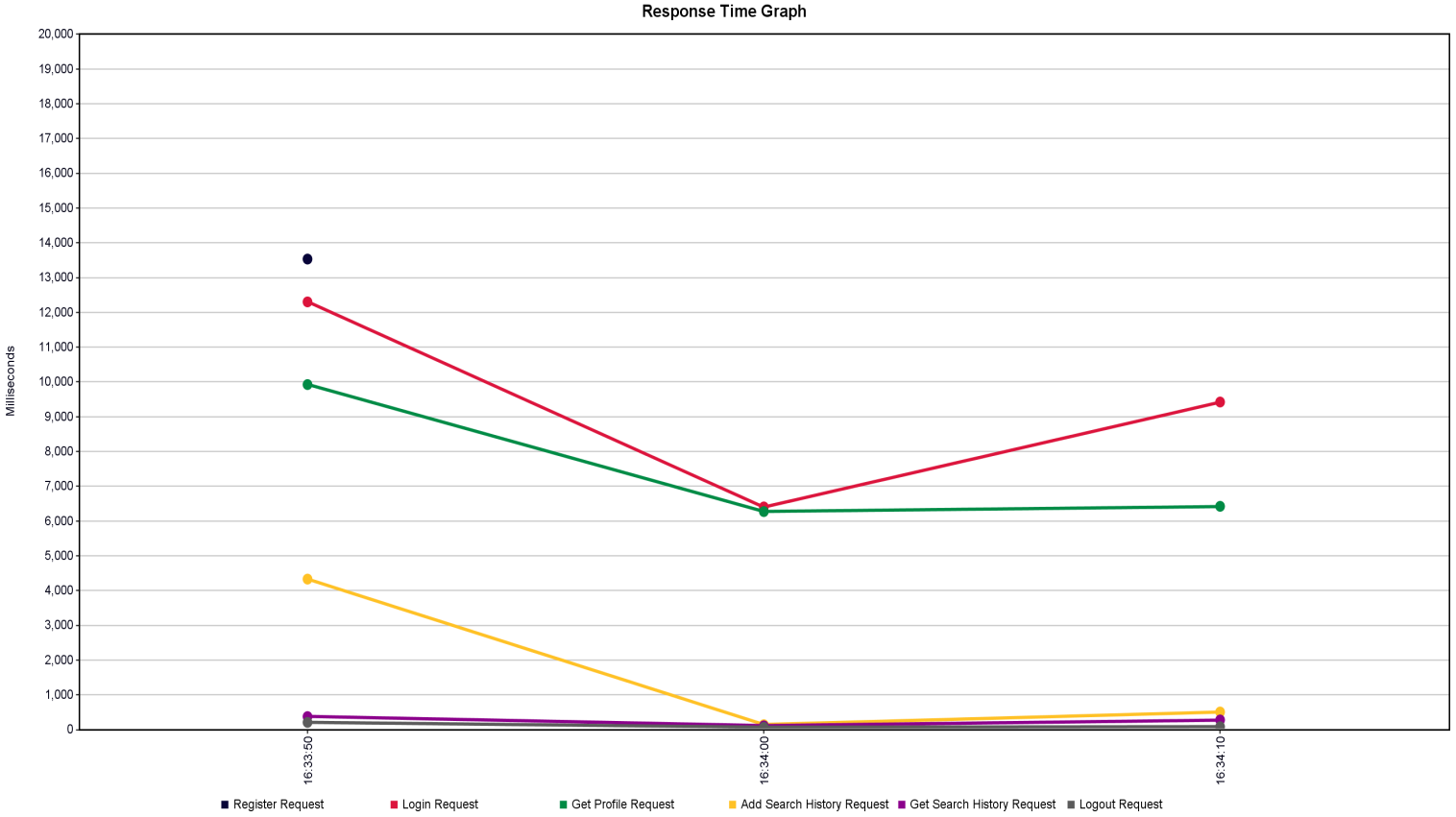
```
λ kubectl get pods
NAME                                      READY   STATUS    RESTARTS      AGE
api-gateway-deployment-576fbdf7b6-22jjj   1/1     Running   0             38m
api-gateway-deployment-576fbdf7b6-j5bpg   1/1     Running   0             27m
api-gateway-deployment-576fbdf7b6-mtjlt   1/1     Running   0             27m
api-gateway-deployment-576fbdf7b6-nf98r   1/1     Running   0             27m
api-gateway-deployment-576fbdf7b6-r9jgd   1/1     Running   0             27m
auth-deployment-69dc4d448d-5mbz4          1/1     Running   0             27m
auth-deployment-69dc4d448d-9hzbj          1/1     Running   0             27m
auth-deployment-69dc4d448d-kjqqb          1/1     Running   0             27m
auth-deployment-69dc4d448d-w5gvp          1/1     Running   0             27m
auth-deployment-69dc4d448d-zh5w6          1/1     Running   0             38m
history-deployment-55777cb8b4-2tvh7       1/1     Running   4 (31m ago)   38m
history-deployment-55777cb8b4-f2w8m       1/1     Running   0             26m
history-deployment-55777cb8b4-lhwls       1/1     Running   0             26m
history-deployment-55777cb8b4-pbxgc       1/1     Running   0             26m
history-deployment-55777cb8b4-vlqx6       1/1     Running   0             26m
mongo-profile-deployment-648fc4c7d5-88dbv 1/1     Running   0             38m
mysql-history-deployment-657c9678f9-z65p8 1/1     Running   0             38m
profile-deployment-6445877bbb-6pnn5       1/1     Running   0             38m
profile-deployment-6445877bbb-8zn2q       1/1     Running   0             27m
profile-deployment-6445877bbb-dp4dd       1/1     Running   0             27m
profile-deployment-6445877bbb-nkbhm       1/1     Running   0             27m
profile-deployment-6445877bbb-w4588       1/1     Running   0             27m
```

Ramp Up Period: 1 second
Threads/Users: 1000

Results:
I) Aggregate Graph:



II) Response Time Results :



Error Rate: 0%

Throughput: The following table shows the throughput

Label	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes

Register Request	13532	685	19597	5064.310.000%	49.37	53.42	15.47	1108
Login Request	7499	1285	14278	2789.020.000%	31.76	41.01	8.1	1322
Get Profile Request	6044	27	12429	2806.350.000%	35.39	38.33	12.55	1108.9
Add Search History Request	1095	15	4065	903.510.000%	47.42	49.05	27	1059
Get Search History Request	887	9	4394	752.10.000%	46.91	51.07	17.22	1114.9
Logout Request	166	3	1489	221.990.000%	46.9	47.95	16.44	1047
TOTAL	4871	3	19597	5451.360.000%	166.55	183.24	61.34	1126.6

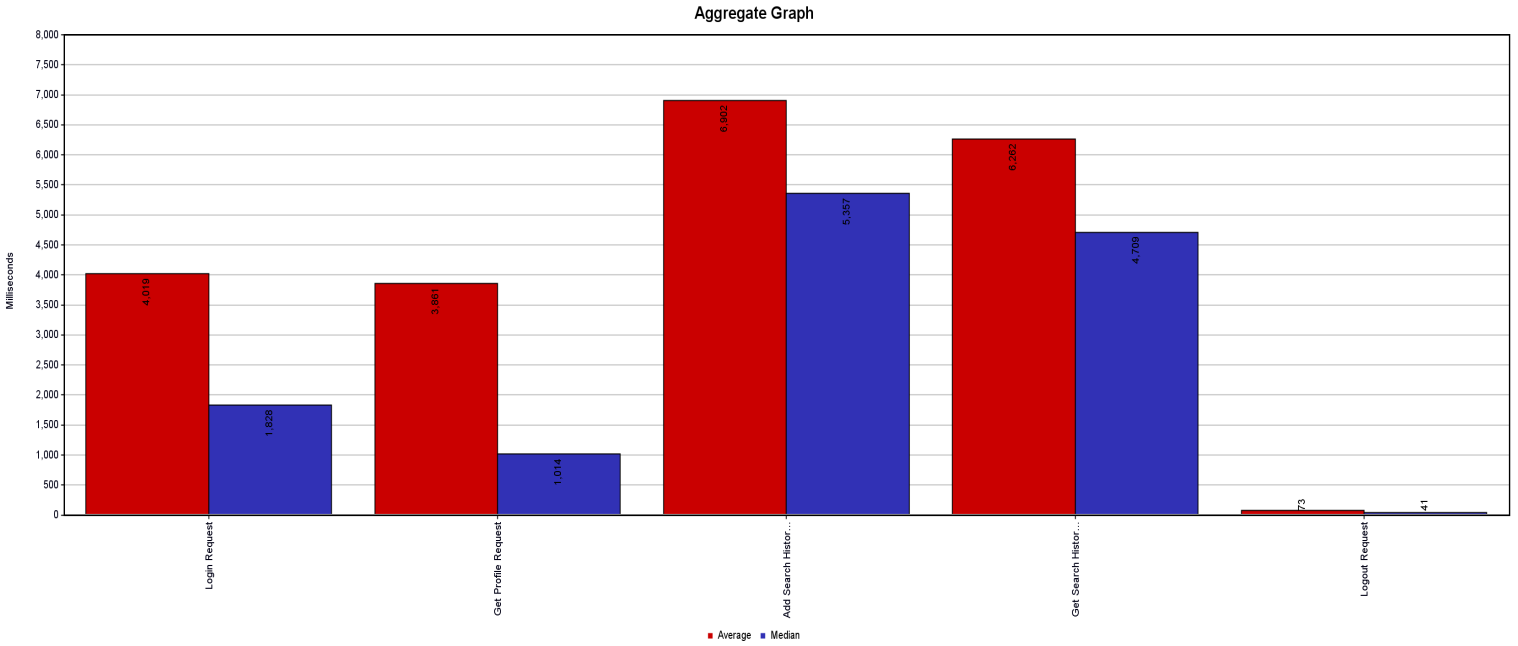
Discussion of Results: Since all users concurrently access the system, the register api is severely overwhelmed which is why it has the most response time.

3: All Micro-services with 5 instances (Pre-existing Users)

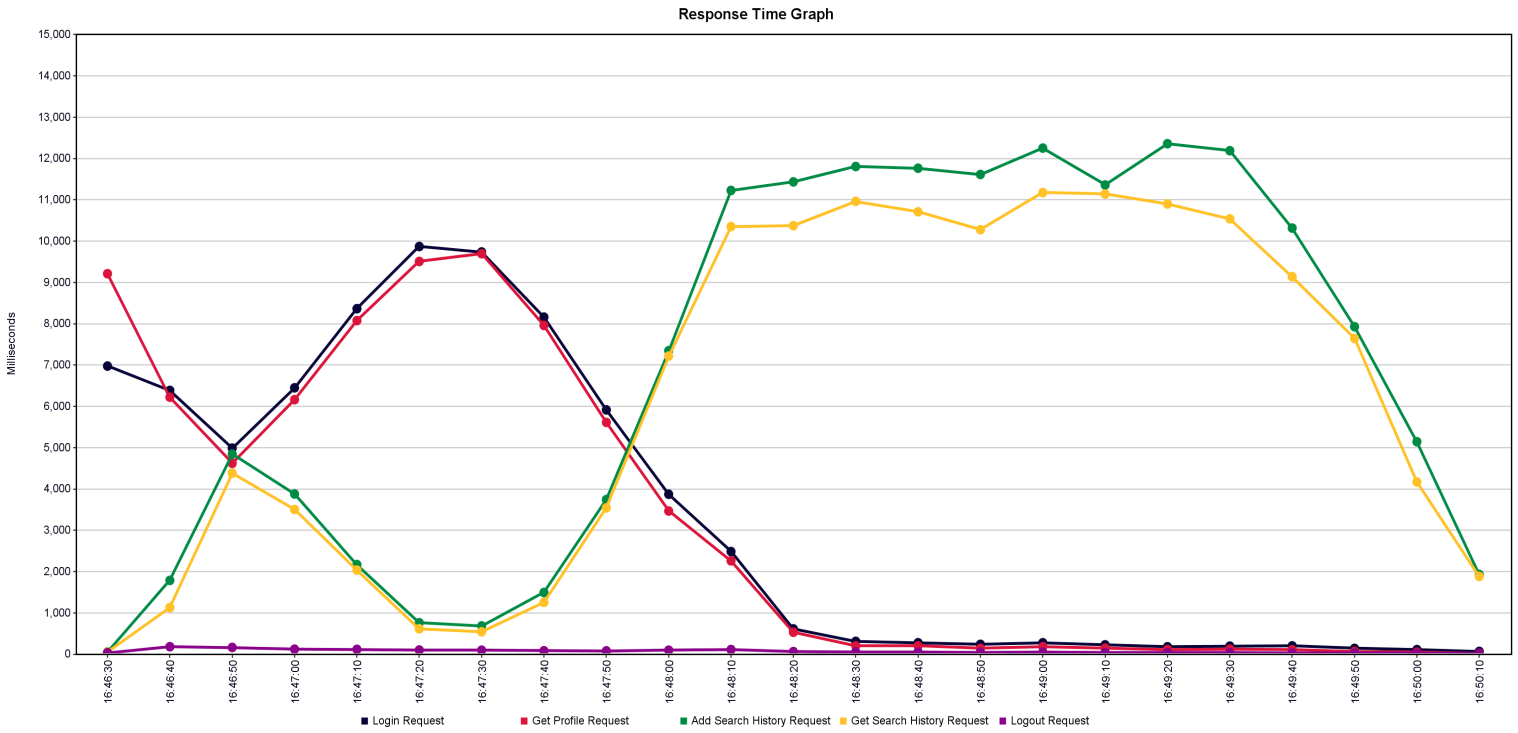
Ramp Up Period : 3 seconds
Loop Count: 10
Threads/Users: 1000

Results:

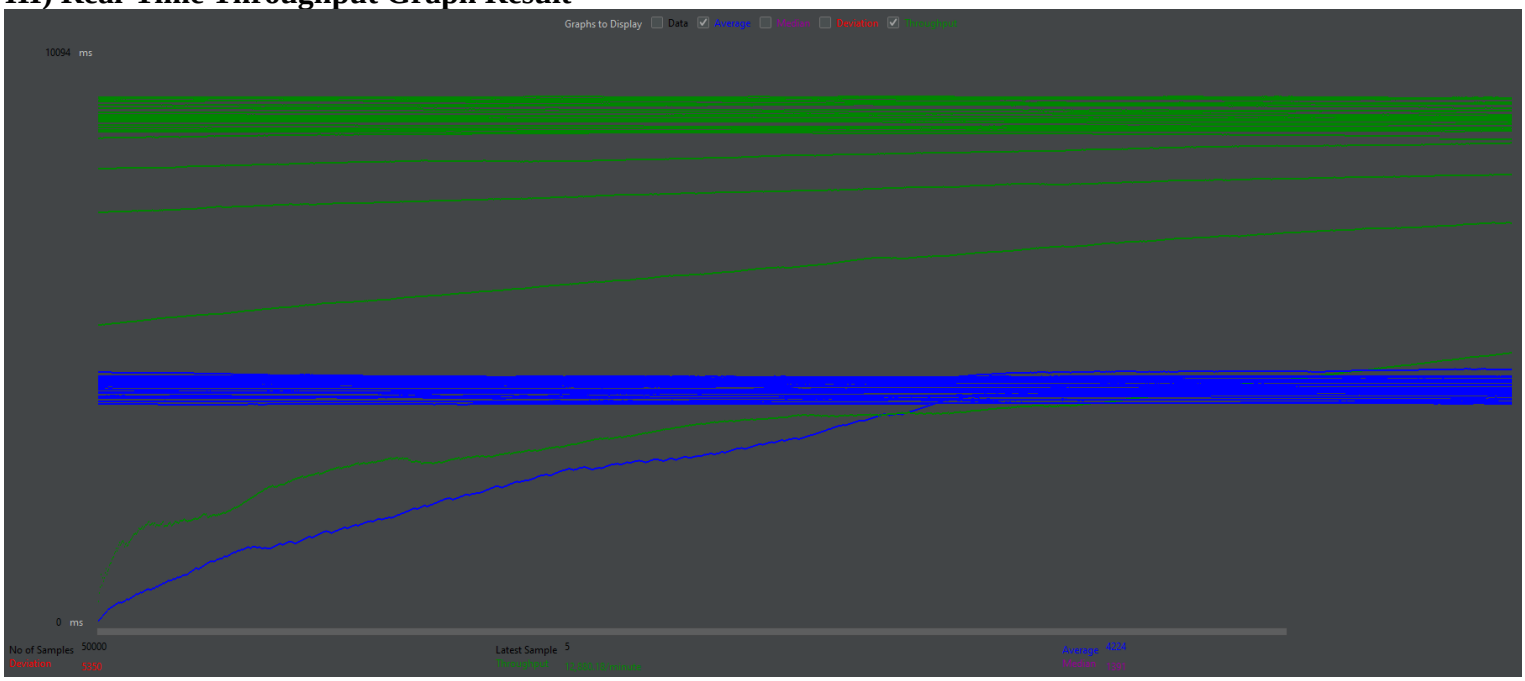
I) Aggregate Graph



II) Response Time Graph



III) Real-Time Throughput Graph Result



Error Rate : 0%

Throughput : The below table shows the throughput of the services

Label	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Login Request	4019	29	19479	4470.58	0.000%	42.94	55.44	10.95	1322
Get Profile Request	3861	4	18051	4482.18	0.000%	42.96	46.52	15.23	1108.9

Add Search History Request	6902	14	37829	6156.87	0.000%	43.04	44.52	24.5	1059.2
Get Search History Request	6262	10	37596	6035.41	0.000%	43.06	54.82	15.81	1303.5
Logout Request	73	3	1598	91.38	0.000%	43.12	44.09	15.12	1047
TOTAL	4224	3	37829	5350.65	0.000%	214.67	244.88	81.42	1168.1

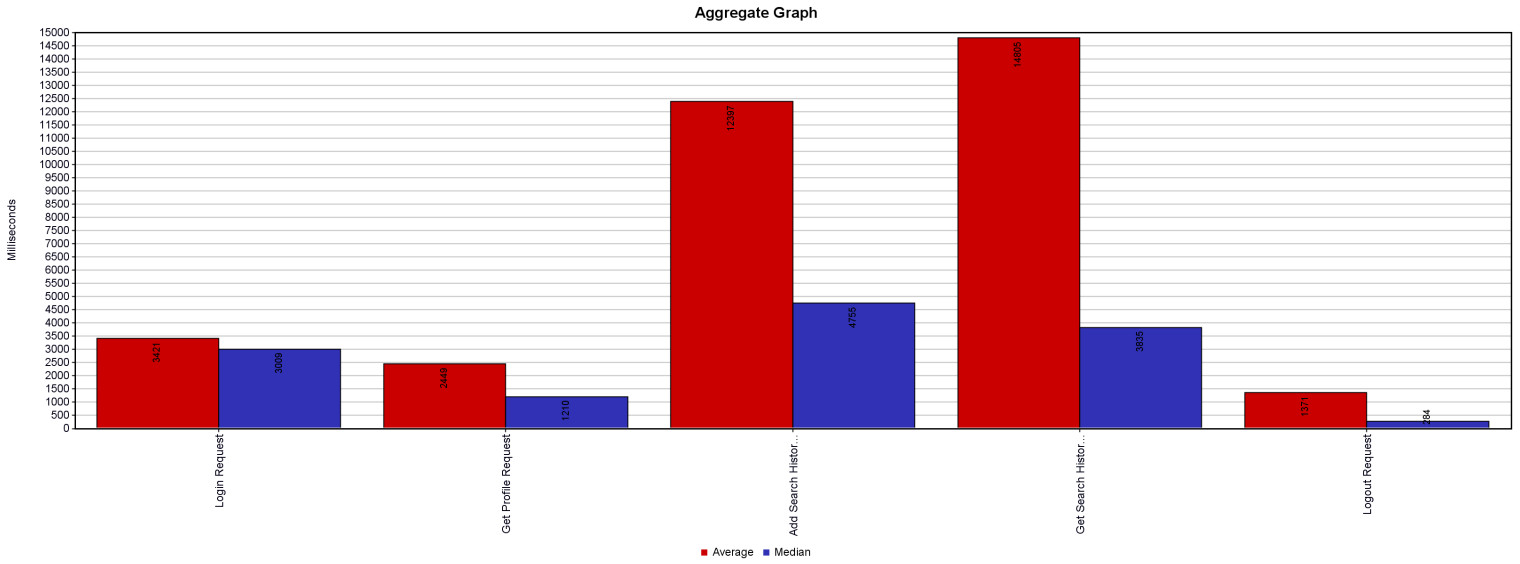
Discussion of Results: Since the ramp up period was increased, we got a steady through as visible in the live graph results. The number of requests being served simultaneously was steady and hence all requests executed correctly.

C] Fault Tolerance and Elastic Scalability

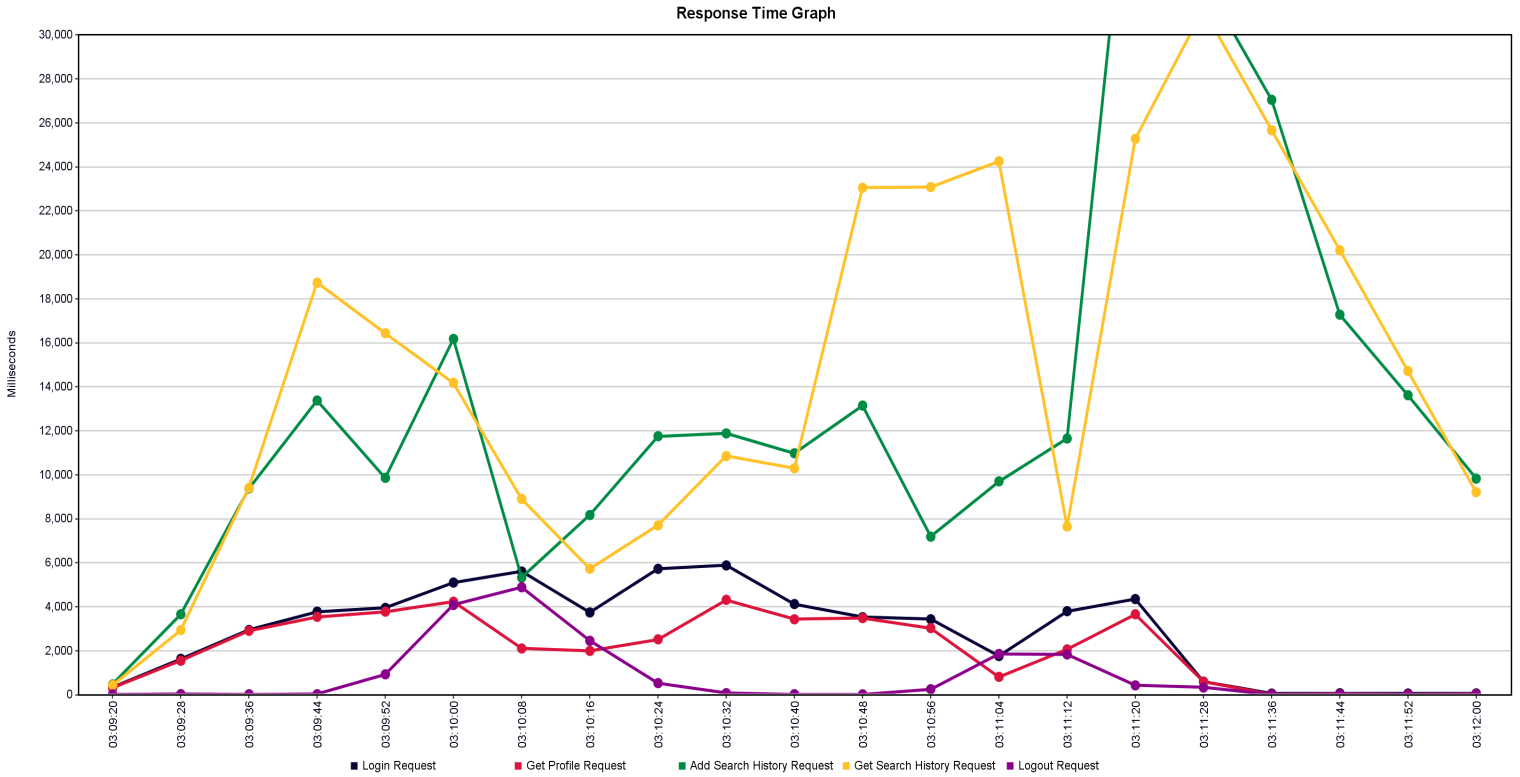
Loop Count: Infinite
Thread/Users: 1000
Ramp Up Period: 30 seconds

Test Procedure : The Loop count was set to infinite.
The micro-services containers were stopped 1 at a time
The services were then restarted and the response time was monitored

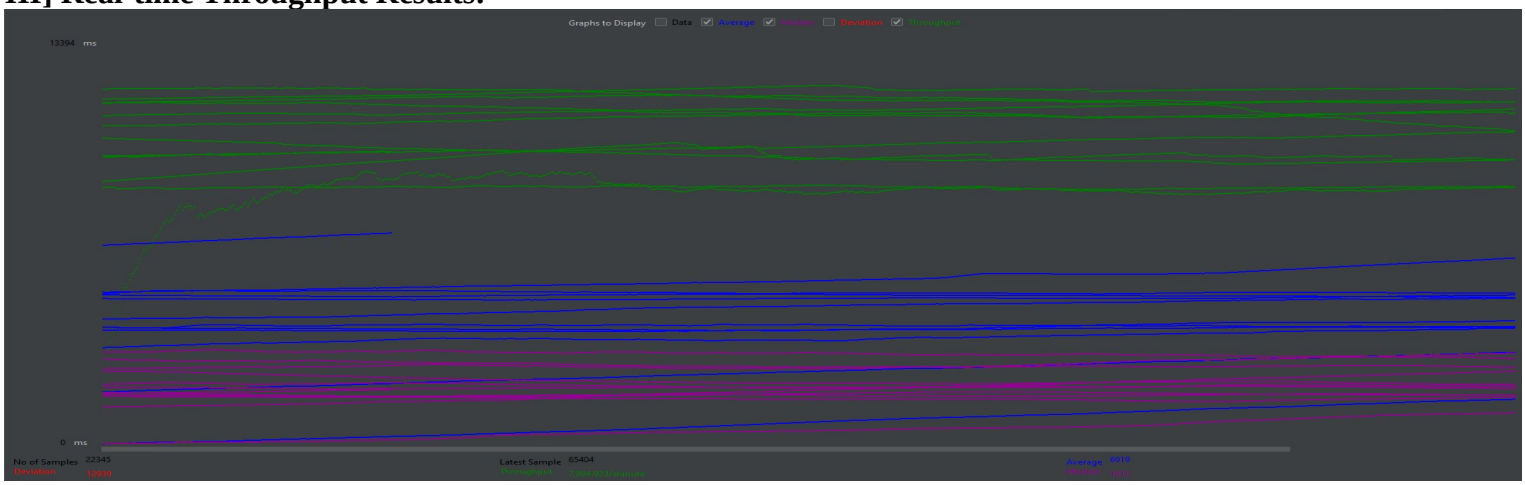
Results:
I] Aggregate Results



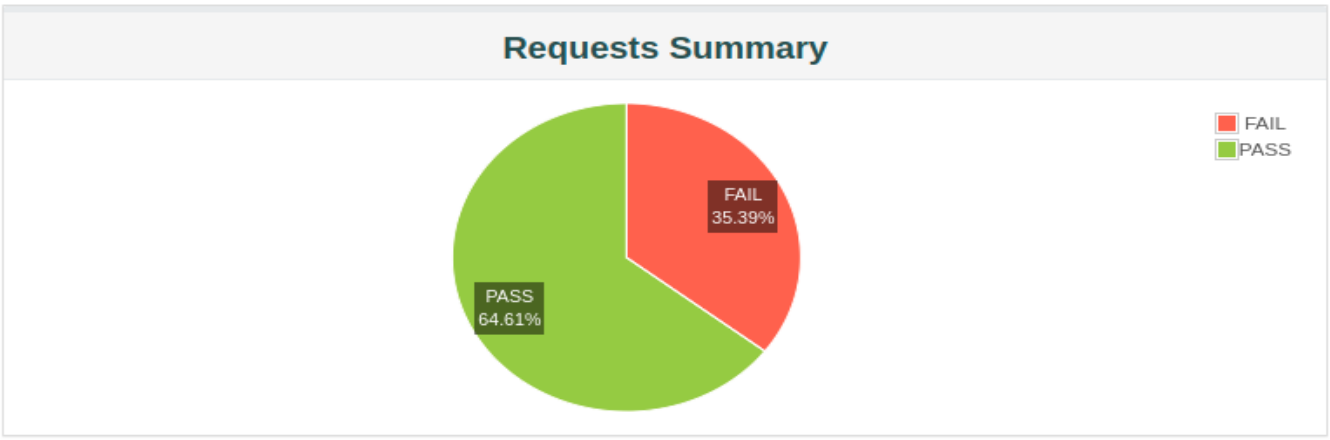
II] Response Time Graph



III] Real-time Throughput Results:



IV] Error Rate Pie Chart

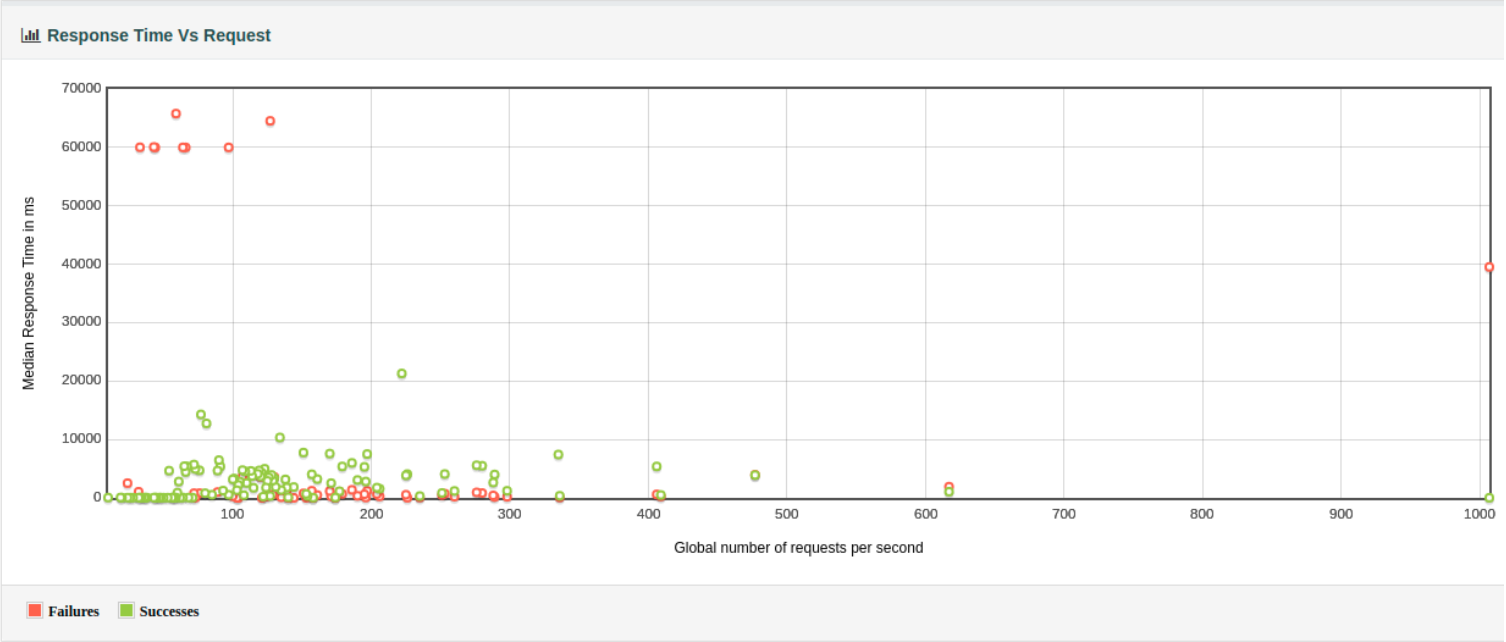


V] Throughput Statistics

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**22345**	**7909**	**35.39%**	**6919.94**	**2**	**98728**	**1893.00**	**27486.00**	**42555.45**	**60167.99**	**131.58**	**143.45**	**41.38**
Add Search History Request	4791	2704	56.44%	12397.78	2	60120	4755.00	40058.40	47760.80	58150.60	28.22	31.11	12.56
Get Profile Request	4791	1440	30.06%	2449.28	2	15216	1210.00	6119.00	8270.20	12223.32	28.33	29.65	8.57
Get Search History Request	4181	2215	52.98%	14805.32	2	98728	3835.00	46542.60	60145.90	74490.50	24.64	27.25	6.74
Login Request	4791	846	17.66%	3421.56	5	15795	3009.00	7330.60	10187.20	12797.24	28.32	33.60	7.22
Logout Request	3791	704	18.57%	1371.71	2	7298	284.00	4713.40	5311.20	5908.32	22.44	22.24	6.40

VI] Request vs Response Time Scatter Plot



Discussion of Results:

As we can see, the error rate of the system was 35% which means our application seemed to work even when the services stopped. Once the services came back up, the response time came down which is clearly visible in the graphs.