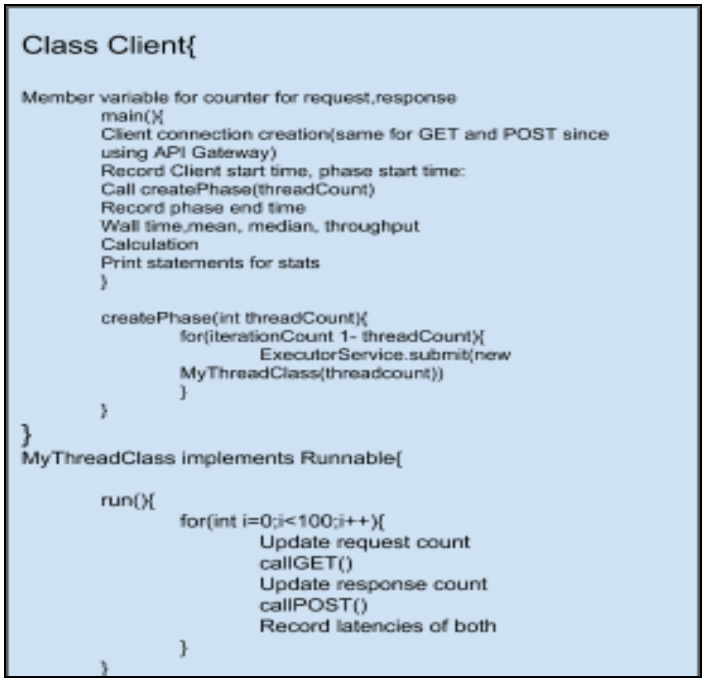
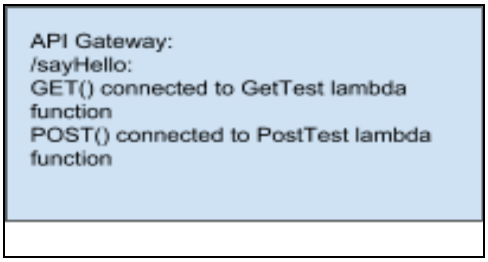
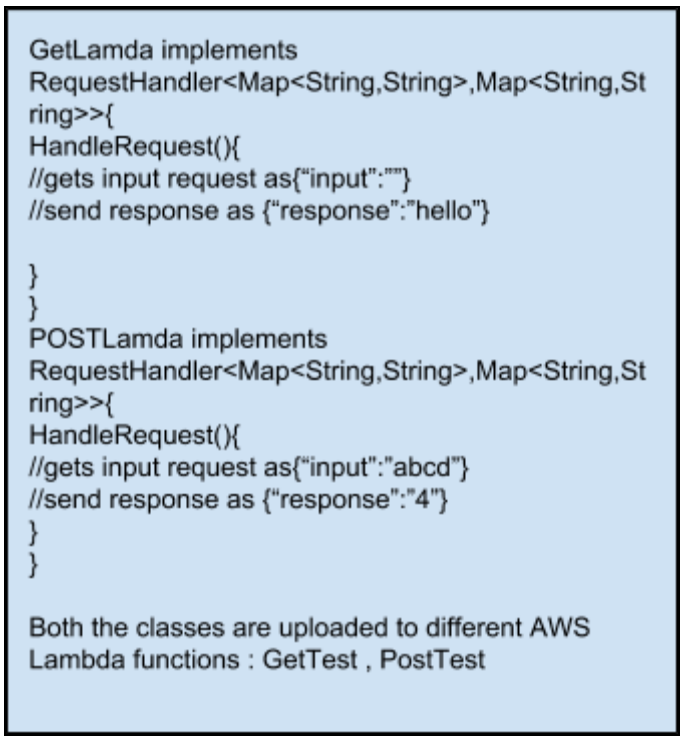
Design Diagram of Client-Server Program:







For calculation of latencies:
I have an array of latencies of size = (total number of threads from all phases*total number of iterations*2) . For every thread and every request I update the particular latency[index] by the request service time. I then sort this latency array and calculate the nth percentile value by

```
latency[ count(latency) * n/100]
```

So the value of 95th percentile is latency[latency.length * 0.95] .

URL of git repository:
https://github.com/Survi15/Fall2018 :
LambdaSurvi - has the project used for creating the Lambda Functions. GetLambda class is for lambda function creation used in GET API. PostLambda class is for lambda function creation used in POST API.
Assignment1 - has the project used for creating the server , client application. RestClient is the class used to access the server deployed at EC2 instance.
ClientForLambda class is the client class used to access the AWS lambda function.
HelloWorld class is the class used to create the REST server

To run RestClient use arguments as: Number of threads iteration DNS IP PORT
For example : 20 100 ec2-54-200-226-64.us-west-2.compute.amazonaws.com 8080
To run ClientForLambda use arguments as: Number of threads iteration
For example : 20 100

Step 4 Screenshots:

```
client: 20 100 ec2-54-200-226-64.us-west-2.compute.amazonaws.com 8080
Client start time: 1537816214019 millis
Warmup phase:All threads running...
Warmup phase complete time: 9.919 seconds
Load phase:All threads running...
Load phase complete time: 12.688 seconds
Peak phase:All threads running...
Peak phase complete time: 18.309 seconds
Cooldown phase:All threads running...
Cooldown phase complete time: 7.566 seconds
Client end time: 1537816262503 milliseconds
========================================
Total requests sent: 7400
Total response received: 7400
Total wall time: 48.484 seconds
BUILD SUCCESSFUL (total time: 48 seconds)
```

```
debug:
client: 100 100 ec2-54-200-226-64.us-west-2.compute.amazonaws.com 8080
Client start time: 1537816380705 millis
Warmup phase:All threads running...
Warmup phase complete time: 15.760 seconds
Load phase:All threads running...
Load phase complete time: 45.524 seconds
Peak phase:All threads running...
Peak phase complete time: 95.865 seconds
Cooldown phase:All threads running...
Cooldown phase complete time: 21.711 seconds
Client end time: 1537816559571 milliseconds
========================================
Total requests sent: 37000
Total response received: 37000
Total wall time: 178.866 seconds
BUILD SUCCESSFUL (total time: 2 minutes 59 seconds)
```

Step 5 screenshots:

```
debug:
client: 20 100 ec2-54-200-226-64.us-west-2.compute.amazonaws.com 8080
Client start time: 1537818507707 millis
Warmup phase:All threads running...
Warmup phase complete time: 10.483 seconds
Load phase:All threads running...
Load phase complete time: 13.463 seconds
Peak phase:All threads running...
Peak phase complete time: 22.553 seconds
Cooldown phase:All threads running...
Cooldown phase complete time: 9.072 seconds
Client end time: 1537818563284 milliseconds
========================================
Total requests sent: 7400
Total response received: 7400
Total wall time: 55.577 seconds
Total number of successful requests: 7400
Overall throughput across all phases: 133.1486054428469 seconds
Mean of all latencies: 0.08140351358272538 seconds
Median of all latencies: 0.05400000140070915 seconds
latency of 95th Percentile: 0.2770000100135803 seconds
latency of 99th Percentile: 0.4779999852180481 seconds
BUILD SUCCESSFUL (total time: 55 seconds)
```

```
client: 100 100 ec2-54-200-226-64.us-west-2.compute.amazonaws.com 8080
Client start time: 1537819135589 millis
Warmup phase:All threads running...
Warmup phase complete time: 17.084 seconds
Load phase:All threads running...
Load phase complete time: 48.017 seconds
Peak phase:All threads running...
Peak phase complete time: 124.700 seconds
Cooldown phase:All threads running...
Cooldown phase complete time: 27.253 seconds
Client end time: 1537819352650 milliseconds
========================================
Total requests sent: 37000
Total response received: 37000
Total wall time: 217.061 seconds
Total number of successful requests: 37000
Overall throughput across all phases: 170.45899175483697 seconds
Mean of all latencies: 0.3971206487681012 seconds
Median of all latencies: 0.12399999797344208 seconds
latency of 95th Percentile: 1.4279999732971191 seconds
latency of 99th Percentile: 3.680999994277954 seconds
BUILD SUCCESSFUL (total time: 3 minutes 37 seconds)
```

Step 6 Screenshots:

```
client: 20 100
Client start time: 1538102612396 millis
Warmup phase:All threads running...
Warmup phase complete time: 10.275 seconds
Load phase:All threads running...
Load phase complete time: 8.576 seconds
Peak phase:All threads running...
Peak phase complete time: 8.859 seconds
Cooldown phase:All threads running...
Cooldown phase complete time: 8.475 seconds
Client end time: 1538102648585 milliseconds
========================================
Total requests sent: 7400
Total response received: 7400
Total wall time: 36.189 seconds
Total number of successful requests: 7400
Overall throughput across all phases: 204.48202958048026 seconds
Mean of all latencies: 0.03760189193401586 seconds
Median of all latencies: 0.03799999877810478 seconds
latency of 95th Percentile: 0.05000000074505806 seconds
latency of 99th Percentile: 0.09099999815225601 seconds
BUILD SUCCESSFUL (total time: 36 seconds)
```

```
debug:
client: 100 100
Client start time: 1538102861060 millis
Warmup phase:All threads running...
Warmup phase complete time: 10.591 seconds
Load phase:All threads running...
Load phase complete time: 13.284 seconds
Peak phase:All threads running...
Peak phase complete time: 26.370 seconds
Cooldown phase:All threads running...
Cooldown phase complete time: 9.161 seconds
Client end time: 1538102920475 milliseconds
========================================
Total requests sent: 37000
Total response received: 37000
Total wall time: 59.415 seconds
Total number of successful requests: 37000
Overall throughput across all phases: 622.7383561367669 seconds
Mean of all latencies: 0.08487778392106898 seconds
Median of all latencies: 0.06800000369548798 seconds
latency of 95th Percentile: 0.16200000047683716 seconds
latency of 99th Percentile: 0.5070000290870667 seconds
BUILD SUCCESSFUL (total time: 59 seconds)
```

Stress Test and Challenges:

1. Making code efficient enough to handle maximum concurrent threads was the first challenge. I had previously designed my client class such that there are 4 different executor services for the 4 different phases. These executor services also required 4 different latches to ensure proper termination of each thread. But because of the poor design, I could only use a maximum of 120 threads. Thus, I decided to use only one executor service and make the phase creations more modular by creating a PhaseCreate() function which could take in the required number of threads as argument and reuse a single executor service.
2. The second improvement was using a single client connection instead of one client connection per thread. This improved the performance of my code measurably(~ around 250 threads). And made sure the connection of the client is closed to avoid any open client connection.
3. The third improvement was using “response.readEntity()” function which ensures to close the response socket after fetching a response from server.
4. Capturing latencies: I am currently using an array to store all the latencies. This array is then sorted and the calculation of 95 and 95th percentile latency is reported. I feel it’s a poor design for now but I plan to improve this on the future assignments. This may be the reason I can use only 255 threads at max before I get connection timeout exceptions.
5. Adjusting API gateway model templates and mapping to fetch the response from the lambda function was also a bit tricky. I used a JSON of the format {“input”:”value”} in the lambda

function for both GET and POST since I understand API Gateway can only take application/json content type.

6. Though Oracle documentation suggests using executor.awaitTermination() along with executor.shutdown() I experienced a few seconds of delay when I kept both the functions. So for now I only have the executor.shutdown() function to ensure an executor does not take any tasks more than the specified thread count of a particular phase.
7. Currently I can go run around 255 threads with 100 iterations each. Following is the result screenshot of the same.

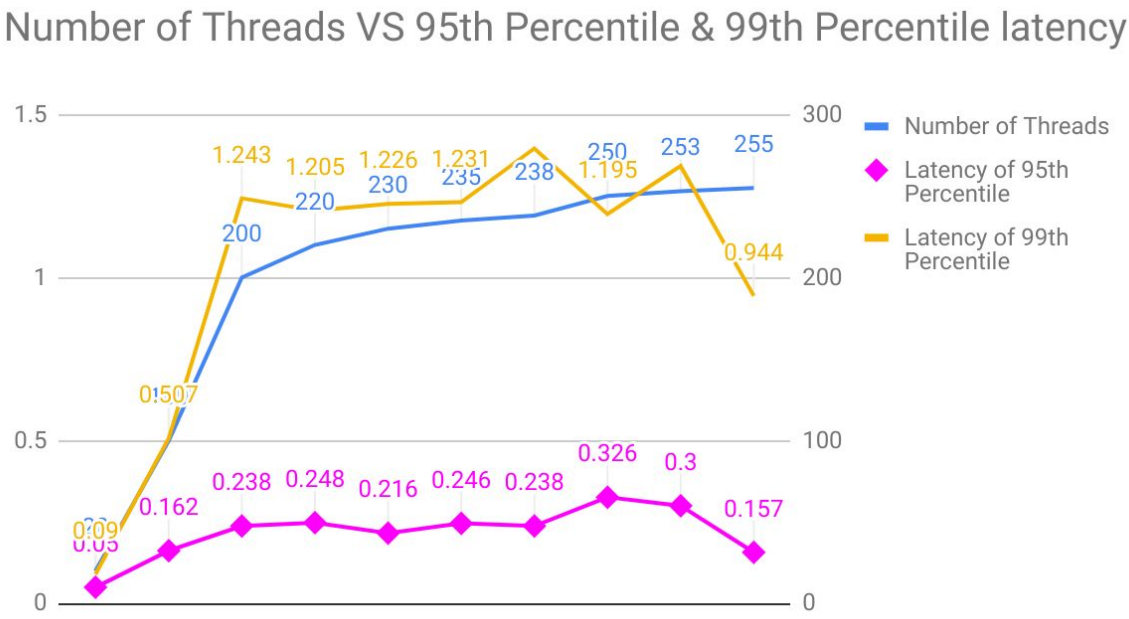
```
Client: Thread Count: 255, Iteration Count: 100
Client starting time: 1538273229175 milliseconds
Warmup phase:All threads running...
Warmup phase complete time: 12.059 seconds
Load phase:All threads running...
Load phase complete time: 59.504 seconds
Peak phase:All threads running...
Peak phase complete time: 83.082 seconds
Cooldown phase:All threads running...
Cooldown phase complete time: 17.501 seconds
Client end time: 1538273401329 milliseconds
========================================
Total requests sent: 94000
Total response received: 94000
Total wall time: 172.154 seconds
Total number of successful requests: 94000
Overall throughput across all phases: 546.0227250065036 seconds
Mean of all latencies: 0.16179439399973986 seconds
Median of all latencies: 0.0560000017285347 seconds
latency of 95th Percentile: 0.15700000524520874 seconds
latency of 99th Percentile: 0.9440000057220459 seconds
```

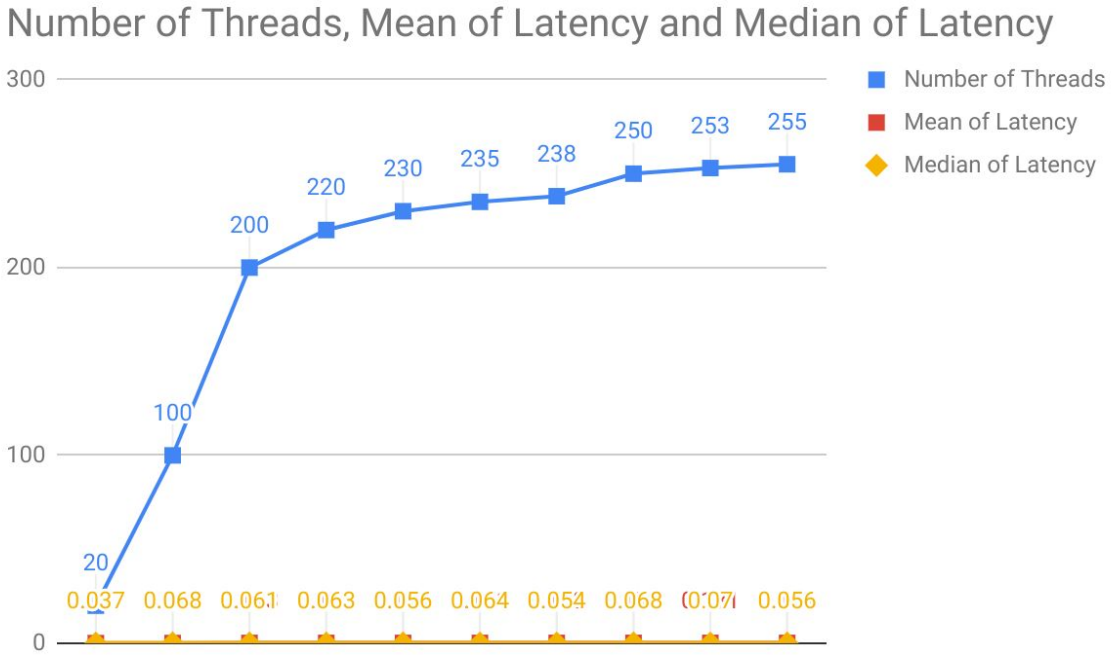
Charts and Experimentation:

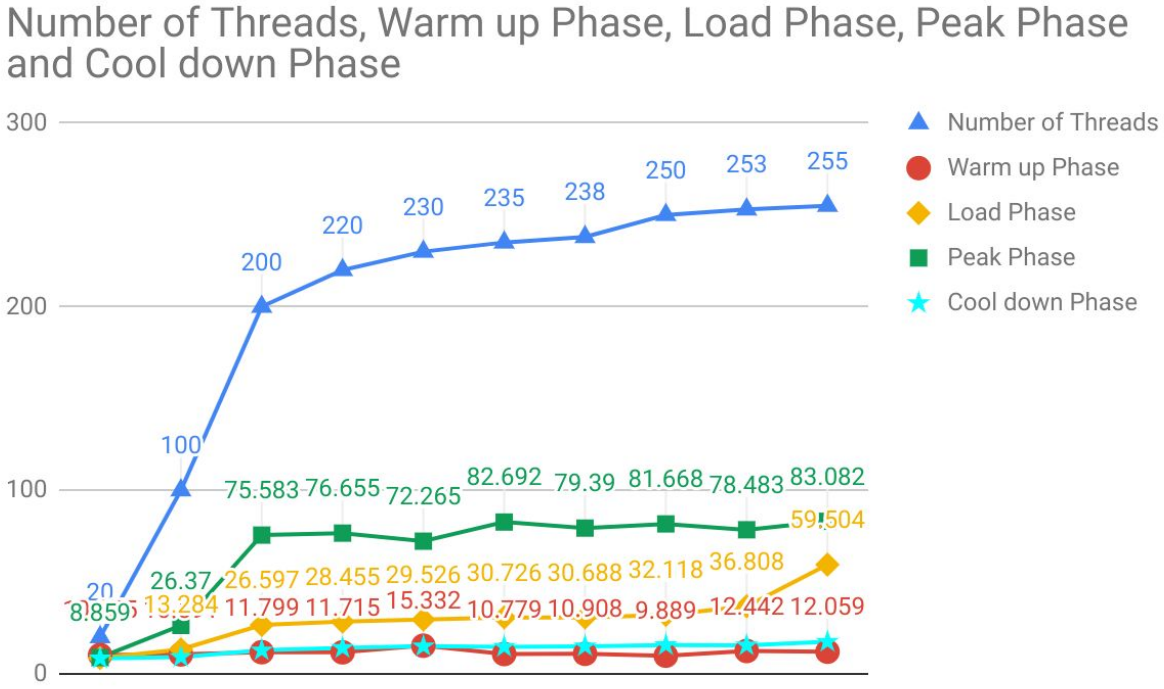
Below is the table of measurements:

Number of Threads	Wall Time(s)	Latency of 95th Percentile(s)	Latency of 99th Percentile(s)	Mean of Latency(s)	Median of Latency(s)	Warm up Phase(s)	Load Phase(s)	Peak Phase(s)	Cool down Phase(s)
20	36.189	0.05	0.09	0.037	0.037	10.275	8.576	8.859	8.475
100	59.415	0.162	0.507	0.084	0.068	10.591	13.284	26.37	9.161
200	126.828	0.238	1.243	0.143	0.061	11.799	26.597	75.583	12.835
220	130.952	0.248	1.205	0.155	0.063	11.715	28.455	76.655	14.096
230	132.187	0.216	1.226	0.154	0.056	15.332	29.526	72.265	15.059
235	135.935	0.246	1.231	0.161	0.064	10.779	30.726	82.692	14.731
238	136.034	0.238	1.396	0.163	0.054	10.908	30.688	79.39	15.039
250	139.327	0.326	1.195	0.173	0.068	9.889	32.118	81.668	15.622
253	143.292	0.3	1.342	0.181	0.07	12.442	36.808	78.483	15.546
255	172.154	0.157	0.944	0.161	0.056	12.059	59.504	83.082	17.501

The anomaly in the above table is in the 255th thread. Surprisingly the latency of 95th and 99th thread in this case was lesser than 253rd thread. This could be for various reasons including network connection, traffic to AWS lambda at the time of the stress test. I tried testing with more number of threads but my application freezes beyond 255 threads. Below shown are some interesting charts from the table above.






Number of Threads, Warm up Phase, Load Phase, Peak Phase and Cool down Phase
300
200
100
0
Number of Threads
Warm up Phase
Load Phase
Peak Phase
Cool down Phase
20
100
200
220
230
235
238
250
253
255
75.583
76.655
72.265
82.692
79.39
81.668
78.483
83.082
59.504
26.37
26.597
28.455
29.526
30.726
30.688
32.118
36.808
8.859
13.284
11.799
11.715
15.332
10.779
10.908
9.889
12.442
12.059