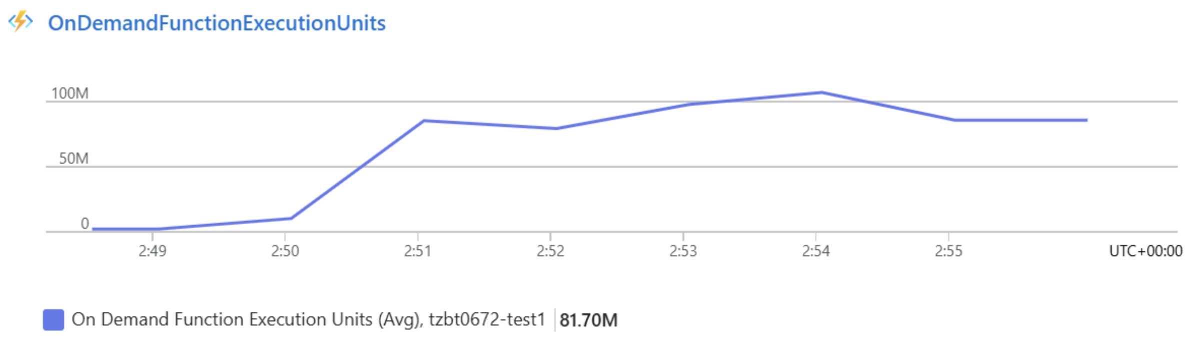
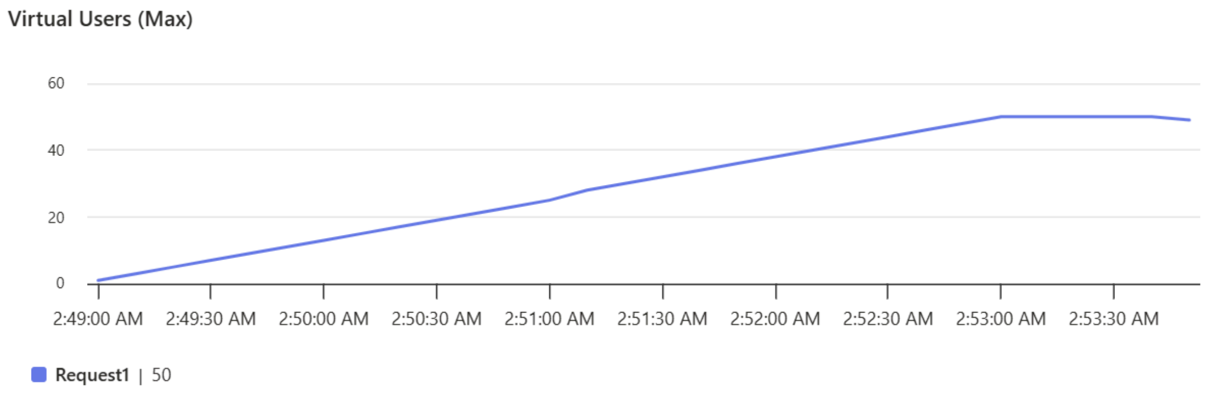
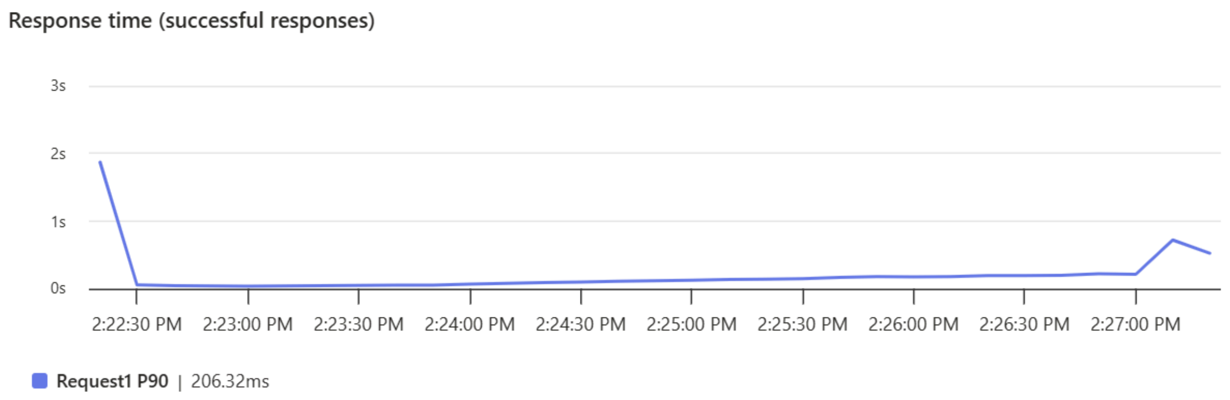
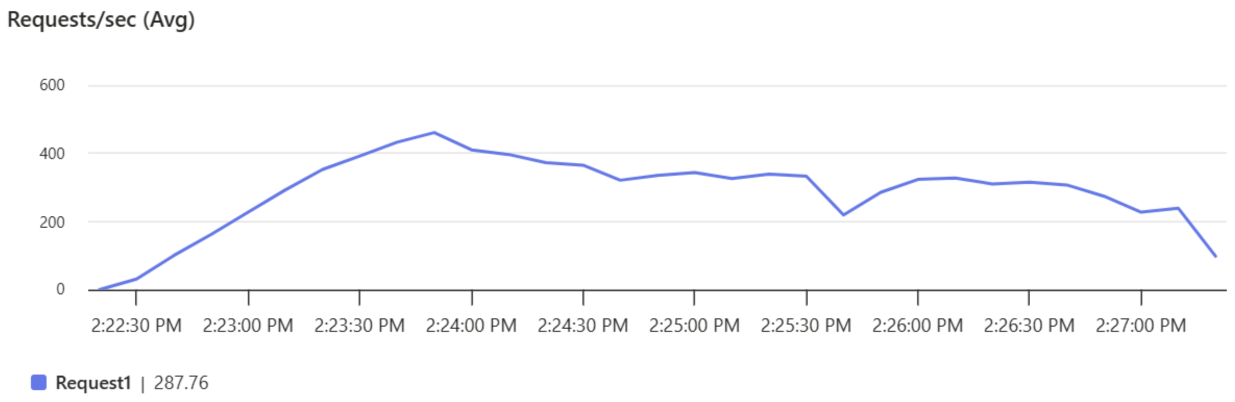
Task 1

For task one, I used a Java azure function with an http trigger. Once triggered, the function would generate a base weather value for all parts of the sensor reading (temperature, wind speed, etc.) and then for each of the twenty sensors, it would be slightly modified. After all twenty sensors had their data generated, the sensor data would be packaged into a json and returned to the user through an http response. This function was later augmented to also write these sensor readings to the azure sql database sensor readings table. Performance was measured using the load testing tool in azure. I mainly looked at response time, which scaled linearly with the increased requests per second. Initially there were scaling issues but after adding DB connection caching/pooling, the bottleneck resolved. As users and requests increase, the response time increases at a linear rate as expected by the high performance scalability that is typical in azure functions. In the beginning there is a high response time but this can be attributed to startup lag. There were no errors reported during the load test. [1]





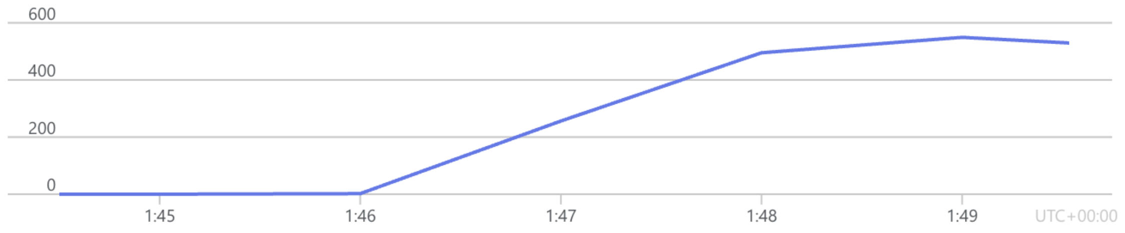




Task 2

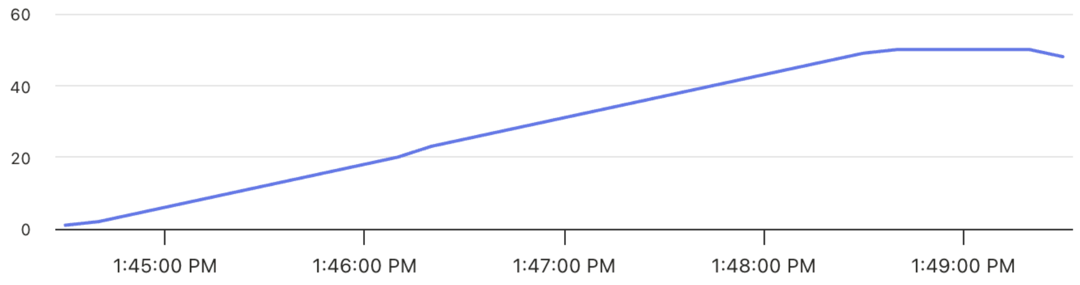
For Task 2 we used a SQL input binding to perform all of the minimum, maximum, and average processes and we returned the statistics for each metric in the response. We calculate the minimum, maximum, and the average of the temperature, wind, humidity, and Co2. The performance of the statistics function was very solid. We measured the average execution time versus the number of virtual users. This is expected as the only operation that is occurring in this function is a database retrieval and the processing of the statistics using SQL aggregate functions. The initial spike in response time is likely due to regular startup latency that drops in the first 20 seconds of the 5 minute test. The response time generally correlates linearly with the number of virtual users. The 90th percentile response time is 108ms. This demonstrates the scalability of Azure Functions. [1]

OnDemandFunctionExecutionCount



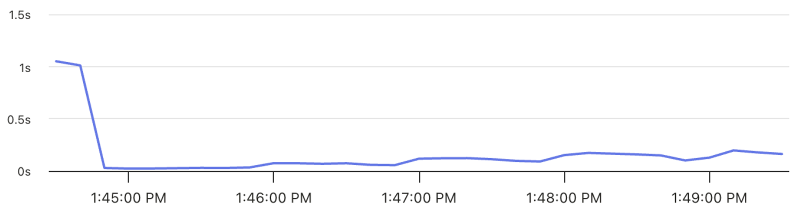
🟦 On Demand Function Execution Count (Avg), helloworldtriggerpython | 460.38

Virtual Users (Max)



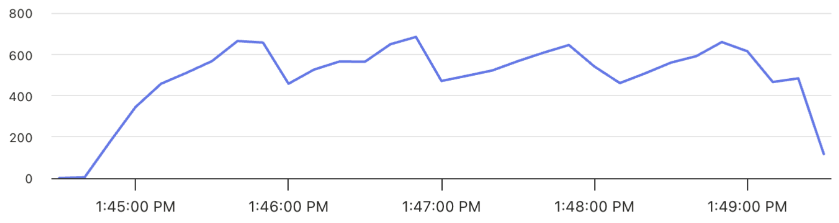
🟦 Request1 | 50

Response time (successful responses)

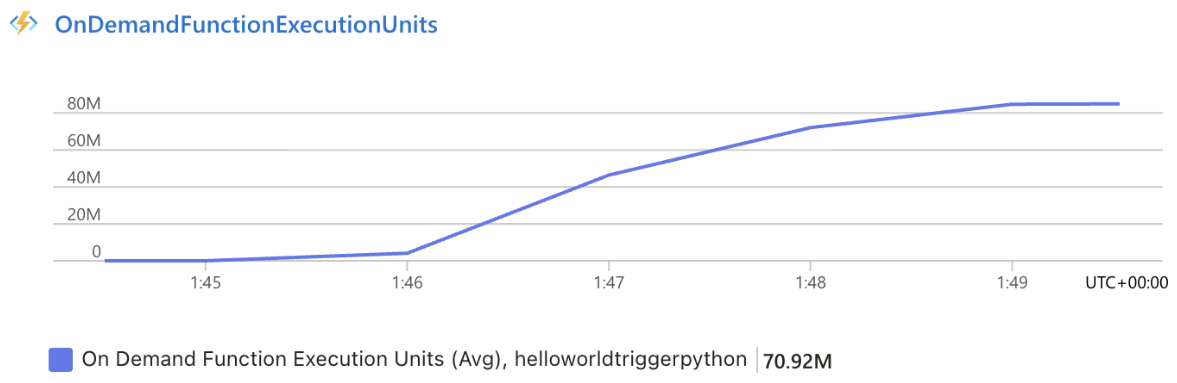


🟦 Request1 P90 | 156.9ms

Requests/sec (Avg)

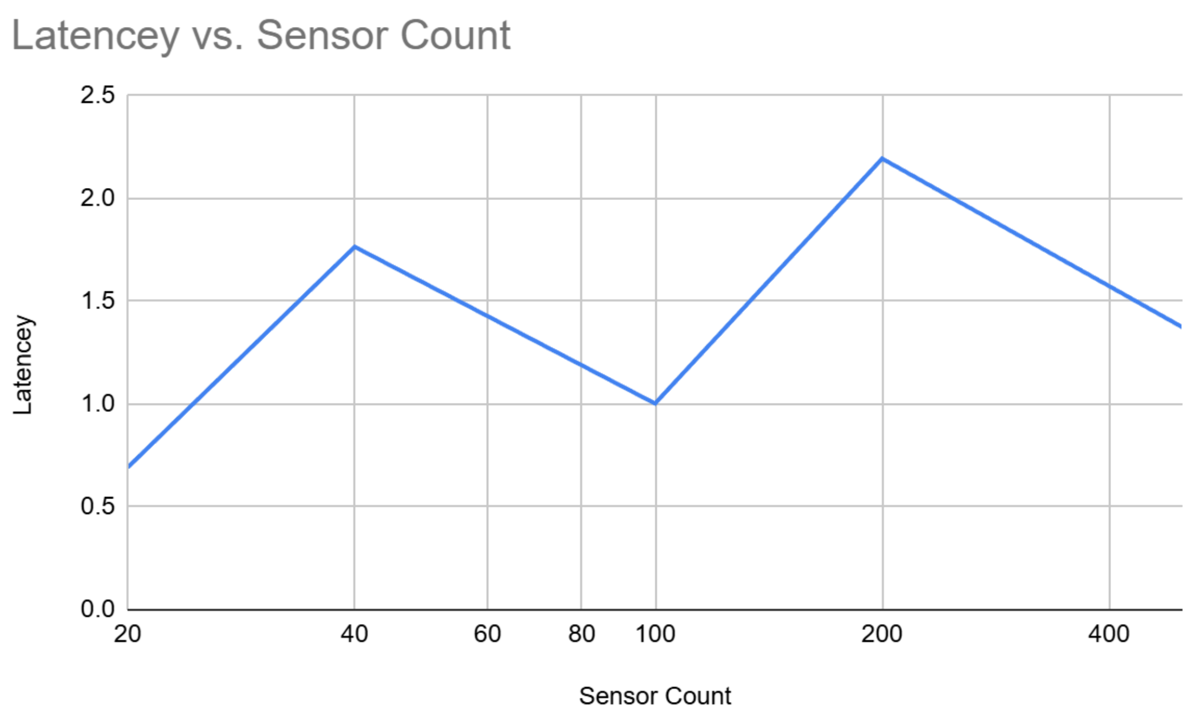


🟦 Request1 | 487.94



Task 3

For the task 3 scenario, we propose using Azure triggers to facilitate automatic timed interval running for function one, and SQL database trigger for the second function. Function one will make use of the azure timer trigger, where we set the timed interval to one minute [2]. Once running, the function to generate sensor data will automatically run every minute. The generated data is then written to the Azure SQL database in the SensorReadings table. For the second function, we first need to enable SQL table change tracking in the preview editor on Azure [3]. After tracking is enabled, the function will be triggered every time there is a change to the SensorReadings table.



Performance was measured by measuring the difference in time between the task 1 function completion and task 2 function completion. As you can see, the latency (time between these two moments) generally increases, as expected, as the sensor count goes up. We found that at higher sensor counts per request, the sql table change trigger had buffering issues and caused timeouts.

References

[1] Ninallam, “Create load tests in Azure Functions - Azure Load Testing,” Microsoft Learn. https://learn.microsoft.com/en-us/azure/app-testing/load-testing/how-to-create-load-test-function-app

[2] Ggailey, “Timer trigger for Azure Functions,” Microsoft Learn. https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-timer?tabs=python-v2%2Cisolated-process%2Cnodejs-v4&pivots=programming-language-java

[3] JetterMcTedder, “Azure SQL trigger for Functions,” Microsoft Learn. https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-azure-sql-trigger?tabs=isolated-process%2Cpython-v2%2Cportal&pivots=programming-language-csharp