thingworx®

SCP Remote Monitoring of Assets Reference Benchmark

Document Version 1.0
December 2019

PTC Inc., 121 Seaport Boulevard, Boston, MA 02210 USA

Document Revision History

Revision Date	Version	Description of Change
December 2019	1.0	Initial Version

What is a Reference Benchmark?

A great way to evaluate how an IOT implementation will perform for you is to compare your own scenario against a reference:

- Understand the results and limitations in a known reference scenario
- Identify what differences exist between your scenario and that reference
- Evaluate how those differences change the behavior of the system

The purpose of this document is to provide a known reference scenario that can be used for these purposes and is targeted at a reader familiar with ThingWorx architecture and implementations.

Over time the IOT Enterprise Deployment Center will be publishing a series of these Reference Benchmarks, starting from different use-cases and/or deployment architectures. Over time, this collection of Reference Benchmarks will create a rich catalog of baselines that can be used to inform the scalability of different field implementations.

Benchmark Scenario Overview

The goal of this Reference Benchmark was to identify a specific implementation architecture to handle a specific Remote Monitoring of Assets scenario.

Once that architecture was identified, it was then tested with different edge workload configurations to illustrate the scalability limits **of that specific architecture** as the number of edge devices or data properties is scaled.

While adjustments to the deployment architecture can typically solve many scale challenges, they are intentionally not in scope for this document. The goal is to establish a baseline reference for comparison. Modified deployment architectures for similar use-cases would appear in separate Reference Benchmark documents.

Use-Case Overview

A healthy Remote Monitoring of Assets implementation balances parallel workloads of ingesting edge data, rapid processing of business logic (checking data conditions or thresholds to generate alarms and events), and user responsiveness to data visualization requests.

In addition to handling these constant workloads, the system must have enough capacity in reserve to handle spikes in activity without causing data loss or significant delays in event processing or user requests.

User Load

In Remote Monitoring of Assets use-cases, typical user workload is to view historical device data for human analysis, or to respond to a triggered alarm.

To simulate this scenario, a simple Device Mashup was used to query historical data for a single asset into a table with 3-4 service calls. The load simulators then call this mashup a total of 32 times each minute, with each call requesting a different device to simulate many users working with the system in parallel.

Edge Load

The target edge configuration for this scenario was 15,000 assets, each with 200 properties per asset and a 3-minute (180 second) data refresh rate. These assets were simulated using the ThingWorx Java SDK to connect.

Once the correct implementation architecture was identified for this target edge, a total of 27 edge configurations were evaluated by varying the number of assets, properties, or the transmission frequency as indicated below:

Number of Things (T)	**Properties per Thing** (P)	**Frequency** (F)
5,000	50	45 seconds
* 15,000	100	90 seconds
30,000	* 200	* 180 seconds

Note: Asterisk (*) indicates target edge configuration used for deployment sizing

This variance created scenarios where the performance of similar edge data volumes can be compared to see how performance changes.

For example, of the 27 possible scenarios with these three variables, five result in the same expected Writes Per Second (WPS) from the Edge:

Things (T)	**Properties** (P)	**Frequency** (F)	**Series Count** (T × P)	**Expected WPS** (T × P) ÷ F
15,000	50	45 sec	750 K	16,667
15,000	100	90 sec	1.5 M	16,667
* 15,000	* 200	* 180 sec	3.0 M	16,667
30,000	50	90 sec	1.5 M	16,667
30,000	100	180 sec	3.0 M	16,667

Note: Asterisk (*) indicates target edge configuration used for deployment sizing

Business Logic, ThingModel and Property Set Approach

In Remote Monitoring of Assets use-cases, it is common for ingested edge data to be evaluated against thresholds (or if-then conditions) to determine if an alarm or event should be triggered. To simulate this, a set of threshold checks were run against incoming edge data, with a 5% chance of an event being triggered.

To simulate typical logic, some thresholds were based on multiple values. For example:

$$IF\ (X > 10)\ AND\ (Y < 5)\ THEN\ (Create\ an\ Alarm)$$

As both X and Y could change independently on the edge device, different business logic approaches can yield different results:

Approach	Description	Pros	Cons
Individual Properties	Rules process as individual data items are ingested	Works with existing edge agents	Multi-property rules process multiple times and in a non-deterministic order Transition false-positives possible
Property Sets	At the edge, changed properties are encapsulated in a JSON or Infotable and processed as a set on the server	Rules process once per update from the edge device Transition false-positives not possible	May require edge agent logic to group changed properties Additional server business logic to unpack and store grouped data into properties
Timer Based Execution	Rules process at a regular time interval instead of triggering by data arrival	Works with existing edge agents Decouples ingestion from business logic	Delays based on ingestion vs. execution frequency Transition false-positives possible

These execution ordering and transition state challenges become more apparent as logic complexity increases. For example, if your logic uses five different properties to determine an alarm, the quantity and order of executions can vary significantly.

This benchmark used a Property Set approach: rather than sending individual changes, the edge agent would group any changed properties into an InfoTable and send that object to the platform.

Upon receiving the InfoTable, the platform would process business rules against those changes (pulling any unchanged or current values from memory), and "unpack" the changed properties in the InfoTable into individual logged properties.

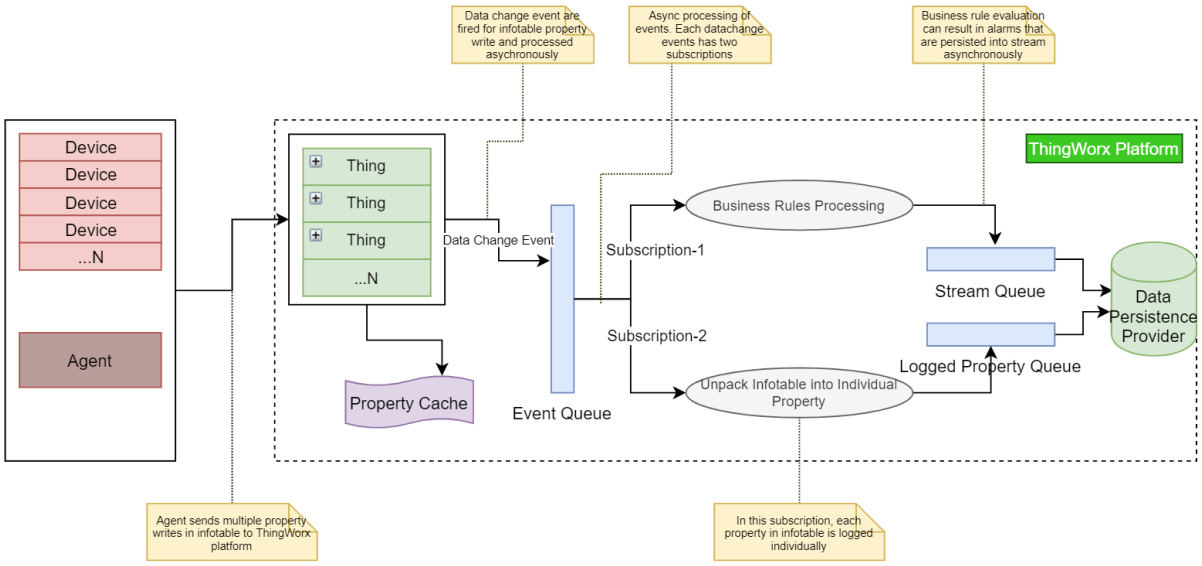

Figure 1 - High level illustration of ThingModel used for this benchmark.

This approach will not be the best choice for every IOT use-case or solution. The complexity of your logic and edge connectivity are important factors to consider.

Implementation Architecture

The following deployment model was used for this simulation leveraging Microsoft Azure, based on the size needed for the target edge configuration (15,000 assets, 200 properties per asset, 180 second transmission frequency) plus the expected business logic and user load.

This simulation was performed using ThingWorx Platform version 8.4.4.

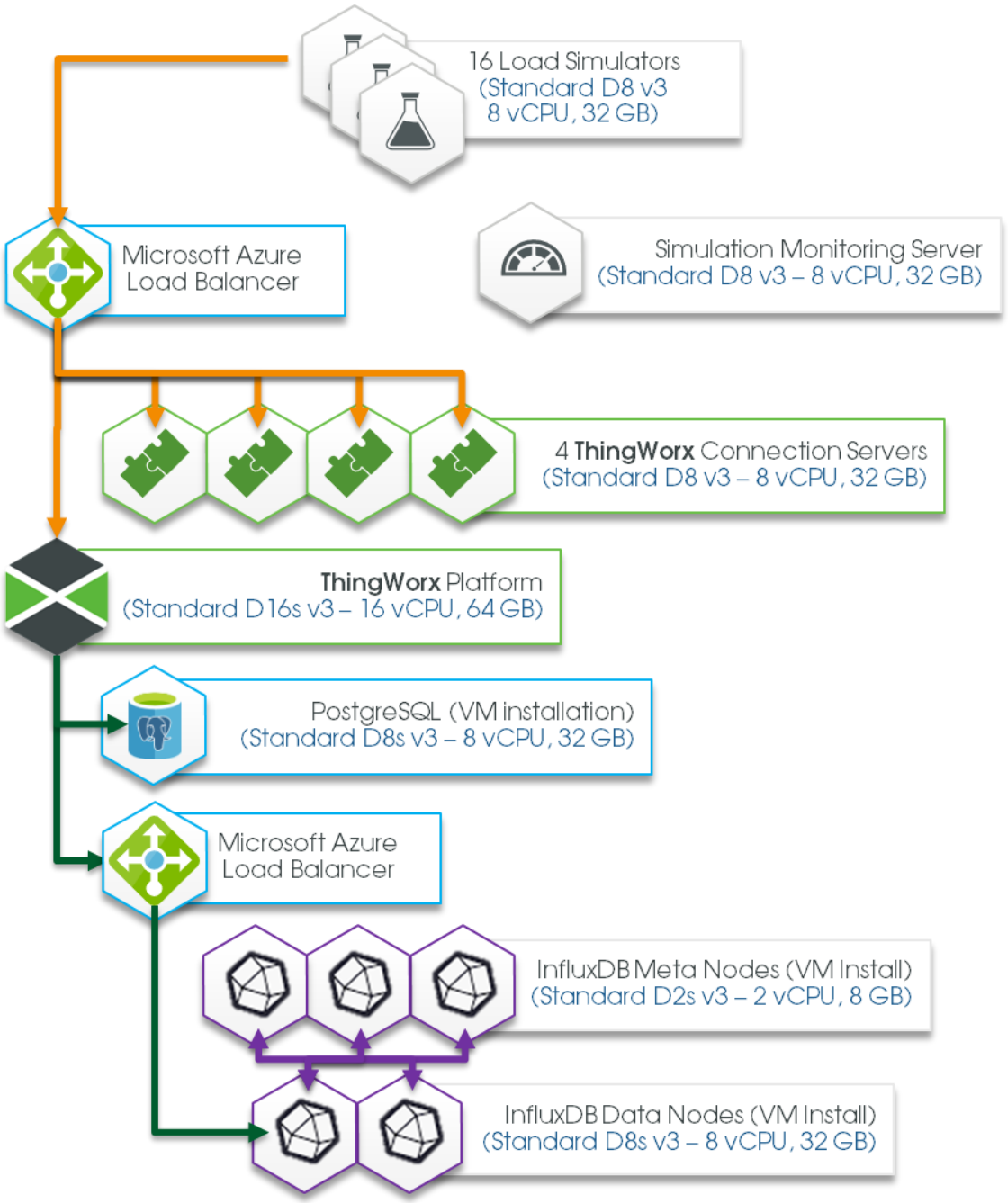

Figure 2 - Architecture diagram for this simulation

Simulation Parameters and KPIs

To check these three factors, the following KPIs were monitored:

	Ingestion	Processing	Visualization
Primary KPI	Value Stream Writes Per Second	Event Rate	HTTP Requests Per Second
Secondary KPIs	Value Stream Queue Size "Lost" data points (failed writes)	Platform CPU Utilization Event Queue Size (i.e. backlog)	HTTP Request Response times "Bad" HTTP Requests

The simulation consisted of a four-hour execution of each edge configuration with the same business logic and user workload in place.

In addition to the four-hour simulations, a seven-day simulation was conducted to ensure the architecture remained stable with the target edge configuration (15,000 assets, 200 properties per asset, 180 second transmission frequency).

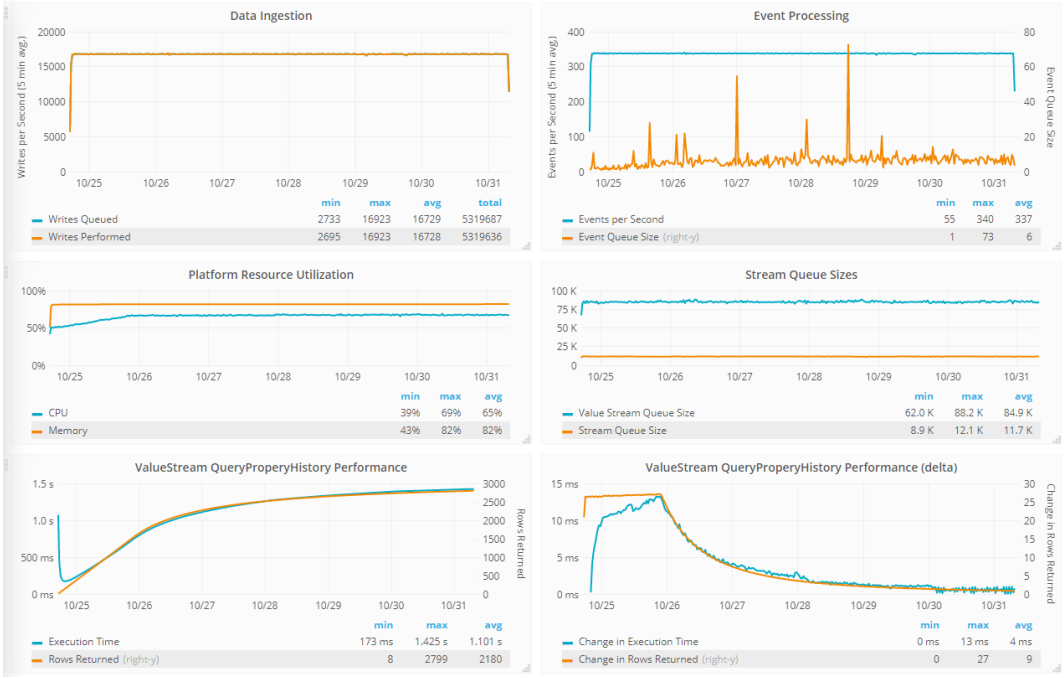

Figure 3 – Telemetry from seven-day "soak test" of 15,000 asset, 200 property, 180 second frequency simulation
Note stable queue sizes, data ingestion at expected level and constant CPU utilization with headroom for spikes.
Mashup response times for mashup did increase as the number of rows returned grew,
but stabilized below 1.5 seconds as mashup query limits were reached.

Simulation Matrix 1 – 180 Second Transmission Frequency

180s			Frequency (F) / Number of Things (T)		
			5,000	**15,000**	**30,000**
Frequency (F)	**Properties per Thing (P)**	**50**	**WPS:** 1,389 **CPU Min/Avg/Max:** 6% / 15% / 19% **Memory Min/Avg/Max:** 4% / 6% / 6%	**WPS:** 4,167 **CPU Min/Avg/Max:** 17% / 23% / 24% **Memory Min/Avg/Max:** 7% / 21% / 22%	**WPS:** 8,333 **CPU Min/Avg/Max:** 47% / 50% / 55% **Memory Min/Avg/Max:** 38% / 39% / 39%
		100	**WPS:** 2,778 **CPU Min/Avg/Max:** 13% / 14% / 15% **Memory Min/Avg/Max:** 9% / 20% / 30%	**WPS:** 8,333 **CPU Min/Avg/Max:** 23% / 31% / 33% **Memory Min/Avg/Max:** 7% / 52% / 76%	**WPS:** 16,667 **CPU Min/Avg/Max:** 63% / 80% / **87%** **Memory Min/Avg/Max:** 30% / 79% / 82%
		200	**WPS:** 5,556 **CPU Min/Avg/Max:** 5% / 19% / 21% **Memory Min/Avg/Max:** 5% / 15% / 16%	**WPS:** 16,667 **CPU Min/Avg/Max:** 47% / 49% / 52% **Memory Min/Avg/Max:** 21% / 78% / 81%	**WPS: 24,387** (should be 33,333) **CPU Min/Avg/Max:** 5% / 76% / **96%** **Memory Min/Avg/Max:** 11% / 78% / 84%

Analysis

The architecture is oversized for scenarios below 7,500 writes-per-second (in blue above). A smaller Azure VM size for the platform (D8s_v3) may work, and InfluxDB may not be needed if PostgreSQL is resized to compensate.

Both 16,667 write-per-second scenarios completed but CPU utilization on the 30,000 asset run (in yellow above) was above 85% under steady load. This indicates a bottleneck risk during a spike of edge or user activity.

The failed run (in red above) did not reach the expected writes-per-second, and did not keep up with events due to insufficient CPU. A larger Azure VM for the platform (D32s_v3) would need to be tested.

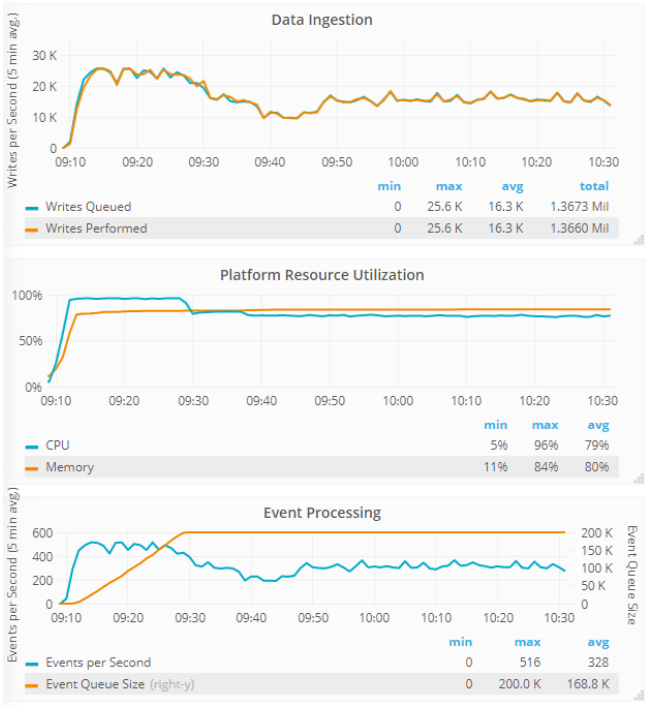

Figure 4 - Runaway Event Queue in failed 30,000 asset scenario. Note CPU 95+% until data loss begins, WPS never reaches 33K.

Simulation Matrix 2 – 90 Second Transmission Frequency

90s			Frequency (F)		
			Number of Things (T)		
			5,000	15,000	30,000
Frequency (F)	Properties per Thing (P)	50	**WPS:** 2,778 **CPU Min/Avg/Max:** 19% / 26% / 28% **Memory Min/Avg/Max:** 6% / 6% / 6%	**WPS:** 8,333 **CPU Min/Avg/Max:** 44% / 46% / 52% **Memory Min/Avg/Max:** 20% / 50% / 58%	**WPS: 7.583** (should be 33,333) **CPU Min/Avg/Max:** 61% / 67% / **97%** **Memory Min/Avg/Max:** 26% / 67% / 83%
		100	**WPS:** 5,556 **CPU Min/Avg/Max:** 23% / 26% / 37% **Memory Min/Avg/Max:** 6% / 14% / 16%	**WPS:** 16,667 **CPU Min/Avg/Max:** 46% / 75% / 81% **Memory Min/Avg/Max:** 30% / 79% / 82%	Not executed
		200	**WPS:** 11,111 **CPU Min/Avg/Max:** 31% / 33% / 35% **Memory Min/Avg/Max:** 15% / 55% / 79%	**WPS: 25,070** (should be 33,333) **CPU Min/Avg/Max:** 74% / 77% / **97%** **Memory Min/Avg/Max:** 70% / 83% / 83%	Not executed

Analysis

Similar to the 180-second simulations, the scenarios below 7,500 writes-per-second might run on the next smallest Azure VM size for the platform (D8s_v3) and may not require InfluxDB if PostgreSQL is resized to compensate.

Also similar to the failed scenario from the 180-second simulations, the two 33,333 writes-per-second scenarios exhibited a similar failure, each losing edge data and filling their event queues within the first ten minutes of the simulation.

Simulation Matrix 3 – 45 Second Transmission Frequency

45s			Frequency (F) / Number of Things (T) 5,000	15,000	30,000
Frequency (F)	Properties per Thing (P)	50	**WPS:** 5,556 **CPU Min/Avg/Max:** 29% / 32% / 47% **Memory Min/Avg/Max:** 4% / 40% / 43%	**WPS: 8,233** (should be 16,667) **CPU Min/Avg/Max:** 61% / 66% / **97%** **Memory Min/Avg/Max:** 21% / 75% / 82%	Not executed
		100	**WPS:** 11,111 **CPU Min/Avg/Max:** 43% / 48% / 53% **Memory Min/Avg/Max:** 13% / 57% / 75%	Not executed	Not executed
		200	**WPS:** 22,222 **CPU Min/Avg/Max:** 70% / 79% / **87%** **Memory Min/Avg/Max:** 37% / 79% / 81%	Not executed	Not executed

Analysis

The 22,222 writes-per-second scenario (in yellow above) was successful during the duration of the simulation, but showed an increasing max CPU utilization and slowly growing event queue backlog.

This scenario would likely fail during a spike of edge activity or user requests and could fail after several days of steady-state workload based on the growing backlog.

At this higher transmission frequency, the 16,667 writes-per-second simulation (in red above) exhibited the same failure signature as the 33,333 writes-per-second scenarios on the prior tables: 95+% platform CPU load with runaway event queue growth and edge data loss.

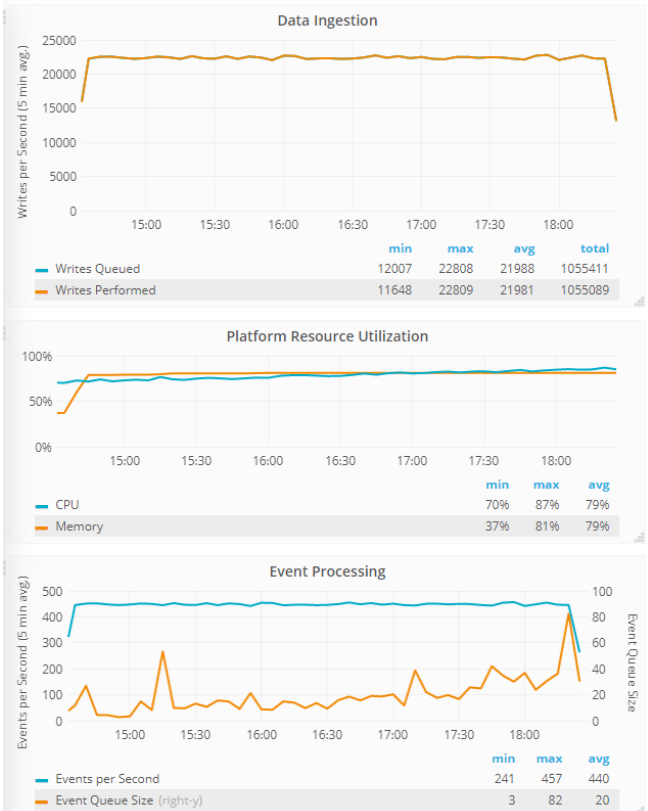

Figure 5 - Telemetry from 22,222 WPS run. Note slowly increasing CPU utilization and Event Queue Size. Failure likely beyond four-hour simulation timeframe.

Analysis and Conclusions

The primary limiting factor for these simulations was platform CPU capacity required to process business logic as data was being ingested.

When incoming data volume increased past a constant ~17,000 writes per second, this deployment architecture was unable to keep up with the frequency and duration of business logic execution and could not recover from the growing backlog.

To validate this conclusion, an additional run was performed using the "30,000 asset, 200 property, 180 sec. frequency" edge configuration with the same simulated business logic and a larger D32s_v3 platform instance and additional InfluxDB Enterprise data nodes.

This was successful for the duration of the simulation but may require additional capacity and/or tuning for longer durations. This larger deployment architecture may be explored in a follow-up benchmark document.

The complexity and timing of business logic operations performed on incoming data is another variable to consider.

Limiting the amount of business logic conducted during data ingestion can help to control this by moving more complex and/or less time-sensitive analytical operations to a more scheduled model.