thingworx®

Remote Monitoring of Assets in **Connected Factories**

Reference Benchmark

Document Version 3.0
October 2020

Copyright © 2020 PTC Inc. and/or Its Subsidiary Companies. All Rights Reserved.

User and training guides and related documentation from PTC Inc. and its subsidiary companies (collectively "PTC") are subject to the copyright laws of the United States and other countries and are provided under a license agreement that restricts copying, disclosure, and use of such documentation. PTC hereby grants to the licensed software user the right to make copies in printed form of this documentation if provided on software media, but only for internal/personal use and in accordance with the license agreement under which the applicable software is licensed. Any copy made shall include the PTC copyright notice and any other proprietary notice provided by PTC. Training materials may not be copied without the express written consent of PTC. This documentation may not be disclosed, transferred, modified, or reduced to any form, including electronic media, or transmitted or made publicly available by any means without the prior written consent of PTC and no authorization is granted to make copies for such purposes.

Information described herein is furnished for general information only, is subject to change without notice, and should not be construed as a warranty or commitment by PTC. PTC assumes no responsibility or liability for any errors or inaccuracies that may appear in this document.

The software described in this document is provided under written license agreement, contains valuable trade secrets and proprietary information, and is protected by the copyright laws of the United States and other countries. It may not be copied or distributed in any form or medium, disclosed to third parties, or used in any manner not provided for in the software licenses agreement except with written prior approval from PTC.

UNAUTHORIZED USE OF SOFTWARE OR ITS DOCUMENTATION CAN RESULT IN CIVIL DAMAGES AND CRIMINAL PROSECUTION. PTC regards software piracy as the crime it is, and we view offenders accordingly. We do not tolerate the piracy of PTC software products, and we pursue (both civilly and criminally) those who do so using all legal means available, including public and private surveillance resources. As part of these efforts, PTC uses data monitoring and scouring technologies to obtain and transmit data on users of illegal copies of our software. This data collection is not performed on users of legally licensed software from PTC and its authorized distributors. If you are using an illegal copy of our software and do not consent to the collection and transmission of such data (including to the United States), cease using the illegal version, and contact PTC to obtain a legally licensed copy.

Important Copyright, Trademark, Patent, and Licensing Information: See the About Box, or copyright notice, of your PTC software.

United States Governments Rights

PTC software products and software documentation are "commercial items" as that term is defined at 48 C.F.R. 2.101. Pursuant to Federal Acquisition Regulation (FAR) 12.212 (a)-(b) (Computer Software) (MAY 2014) for civilian agencies or the Defense Federal Acquisition Regulation Supplement (DFARS) at 227.7202-1 (a) (Policy) and 227.7202-3 (a) (Rights in commercial computer software or commercial computer software documentation) (FEB 2014) for the Department of Defense, PTC software products and software documentation are provided to the U.S. Government under the PTC commercial license agreement. Use, duplication or disclosure by the U.S. Government is subject solely to the terms and conditions set forth in the applicable PTC software license agreement.

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

Document Revision History

Revision Date	Version	Description of Change
July 2020	1.0	Initial Version
September 2020	2.0	Scenario 2 Added
October 2020	3.0	Scenario 3 Added

What is a Reference Benchmark?

As stated in the first in this series of benchmarks, a great way to evaluate how an IOT implementation will perform is to compare it against a reference, helping to:

- Understand the results and limitations in a known reference scenario
- Identify what differences exist between the implementation and the 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. The second in a series of documents, this document furthers the IOT EDC's goal of providing a rich catalog of baselines, each which can be used to inform the scalability and viability of different field implementations.

Benchmark Scenario Overview

This Reference Benchmark illustrates how Edge size affects the scalability of a Connected Factory use case. Variations in Edge size are made by adjusting the number of connected assets, the number of properties per asset, and the frequency at which these properties write to ThingWorx. In future benchmarks of this variety, the number of factories will be varied as well.

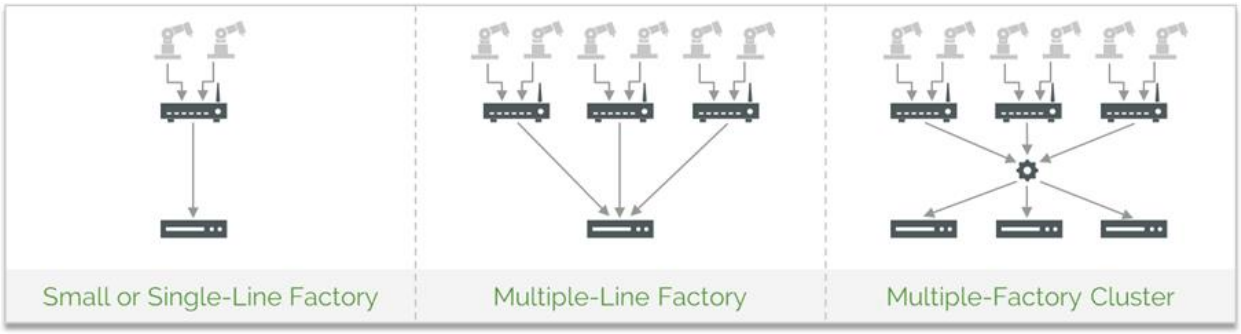


Figure 1 – The starting scenario (left), with the others representing potential future sections in this catalogue.

The focus in this benchmark centered around the throughput of data from Kepware Server to ThingWorx Foundation. As such, the business logic used here is the same as for the previous publication, meaning that this is still a Remote Monitoring of Assets use case, but this time in a Connected Factory setting. As before, the deployment architecture is held constant throughout these various tests and is not the subject of study here.

Scenario One released here contains the capabilities of a single Kepware Server configuration. The goal is to work towards a catalogue of Connected Factory use cases, each contained within this same document. Future configurations will make use of multiple Kepware Servers and multiple factory locations (across different regions, something not tested at the time of document publication).

Use-Case Overview

A healthy Connected Factory implementation will often look very similar in design and function to the Remote Monitoring scenario tested previously. However, in this case, there are fewer assets on the Edge side, each with more properties and more frequent property updates sent to the Foundation server.

Real-time monitoring of each asset is facilitated through business logic that checks either one, two, or multiple properties before triggering an alarm. These alarms are added to a stream which is monitored by the operators via mashup. There is also logic that runs once every 30 minutes to roll up the statuses of all Factory Assets.

Remember that as before, the system must have enough resources to handle this steady workload in addition to any spikes in activity (which may in reality be even larger than any maximums given in the data here). A failed scenario is any in which there is any data loss or delays in event or user request processing, which in this benchmark, are the less limiting factors when compared to Edge throughput.

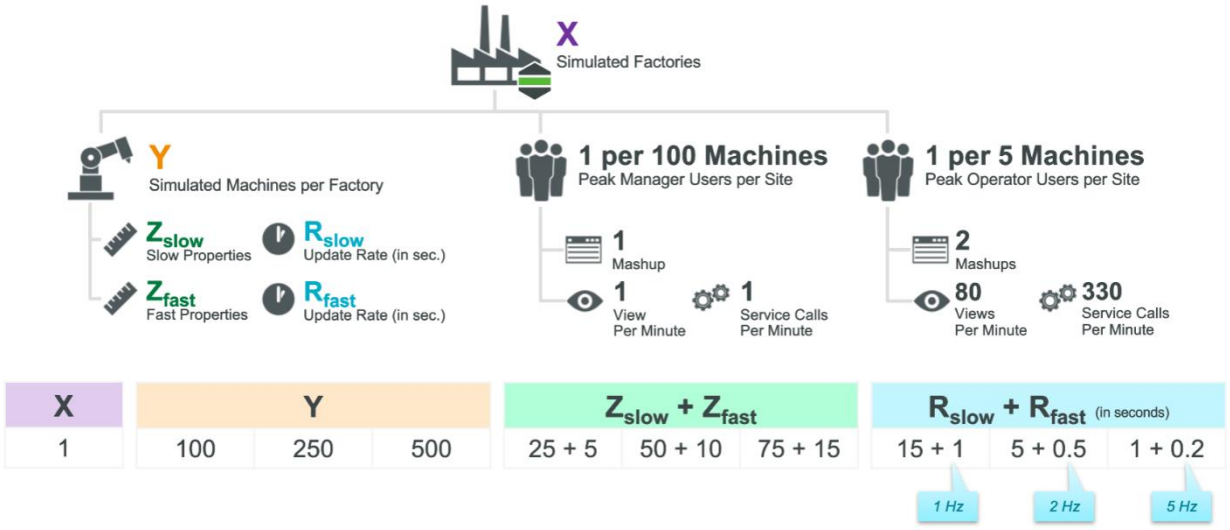


Figure 2 – This chart shows the benchmark scenario. Variations come from changing Y, Z, and R, which respectively represent the number of assets (Y) per Factory (X), the number of properties per asset (Z), and the scan rate or frequency of property updates (R). In Scenario One, X=1 throughout every test, meaning that there is only one factory in one region being simulated, but future scenarios will vary this as well.

User Load

This is still a Remote Monitoring of Assets use case, just in a factory setting, so the typical user workload is the same as it was in the previous benchmark: to view historical device data and respond to triggered alarms. However, this use case also has real-time monitoring, like seeing property values in a display as they come in (current state of properties, included in the operator view), and roll-up logic which runs less frequently and depicts the state of the entire factory (included in the manager view).

The operator mashup therefore contains real-time property information via the Property Display widget and historical property information via the Time Series Chart widget (with drop-down menus fueling both of these charts). There is also a Grid widget displaying all the alarms for a particular thing, and a List widget allowing operators to switch from one asset to another. A secondary mashup can be opened from this which allows operators to add notes, effectively acknowledging an alarm in the process. This mashup is called half as often, and the updates to the alarm tracking stream occur only 20% of the time.

The manager mashup shows the status of the entire factory, including a query to sort by factory and region (which does not apply in the first scenario) and a Grid widget containing all of the information about each factory: how many of the total things are connected (a percent) and how many unacknowledged alarms there are. The roll-up logic for this runs once per hour, populating a data table for more rapid querying.

Note that because this is a Connected Factory scenario, the number of operators and managers at the factory increases proportionally with in the number of assets. See Figure 2 above for a visual of the number of managers, the number of operators, and the corresponding traffic which they generated via their various activity.

Edge Load

There were two kinds of properties in this Connected Factory scenario: fast properties which had no logic upon ingestion, but high scan rates, and slow properties, those with lower scan rates and business logic which ran upon data change. Chart 1 below shows the manner by which the ingestion rate was manipulated for each test.

Assets (Y)	**Fast Prop** (Z_{fast})	**Slow Prop** (Z_{slow})	**Fast Freq.** (R_{fast})	**Slow Freq.** (R_{slow})	**Series Count** ($Y\ x\ (Z_{fast} + Z_{slow})$)	**Expected WPS** $Y\ x\ Z_{fast} \div R_{fast} + Yx\ Z_{slow} \div R_{slow}$
100	5	25	1 sec	15 sec	3,000	660
100	5	25	0.5 sec	5 sec	3,000	1,500
100	5	25	0.2 sec	1 sec	3,000	5,000
100	10	50	1 sec	15 sec	6,000	1,300
100	10	50	0.5 sec	5 sec	6,000	3,000
…	…	…	…	…	…	…
250	10	50	0.5 sec	5 sec	15,000	7,500
250	15	75	1 sec	15 sec	22,500	5,000
…	…	…	…	…	…	…
500	15	75	1 sec	15 sec	45,000	10,000

Chart 1 – A sample of the tests; the ingestion rate was manipulated by the variables in Figure 2.

Note that the scan rate on the ThingWorx Foundation server was 2 times faster to ensure that ThingWorx wouldn't miss any value changes before it was able to check. To protect against the possibility that tag value changes are missed between sample intervals, it is recommended to set the scan rate to twice as fast as the fastest rate of change expected from the tag being sampled, but not faster than 100ms. For example, if a tag is expected to change once per second, scan rate should be set to 500 ms."

Simulation Parameters and KPIs

Each simulation consisted of a four-hour execution of various Edge configurations, with the same business logic and user workload in place throughout. To confirm the success of the tests, the same KPIs were monitored as before:

	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

However, determining if a test failed or not this time required checking the Kepware Server console output as well. This is because when the throughput between Kepware Server and ThingWorx was too high, the message queue within Kepware Server built up and exceeded its limit. When it began rejecting new messages to the queue, the below error began printing repeatedly in the Kepware Server console output, and depending on the settings, in the ThingWorx Application Log as well:

```
One or more value change updates lost due to insufficient space in the
connection buffer. | Number of lost updates = #####.
```

Most of the failed tests shown here resulted from data loss. Note that Kepware Server does have a mechanism to handle disconnects or queue overflows, dealing with this data loss a bit better, but this "Store and Forward" feature has performance considerations entirely of its own, and so was not used here (see the *Store and Forward* section on page 47 of this Kepware Server guide for details).

In Connected Factory use cases, the "Ingestion" KPI takes center stage as a result of throughput limitations from Kepware Server to ThingWorx. The business logic is less of a limiting factor, though that is in part because of how basic the logic is in this benchmark scenario, and not one used to deem any of these tests as failures. However, if the series count is high enough, say because there are many more properties or many more things than used here, then the business logic considerations may become more important, as well as the sizing of the Foundation server. Issues with handling business logic can cause higher CPU usage, resulting in overall delays and back pressure, which reduces the throughput from Kepware Server to ThingWorx Foundation as well.

Scenario One: One Kepware Server in ThingWorx 8.5

Implementation Architecture

The goal of this scenario was to mimic a Connected Factory in which one Kepware Server represented a whole factory, located on premise (on the same network) as the ThingWorx Foundation server. This simulation did make use of Azure for hosting, but no other considerations for region locality or network delays in communication were considered. This test is primarily a single-region, on-premise architecture.

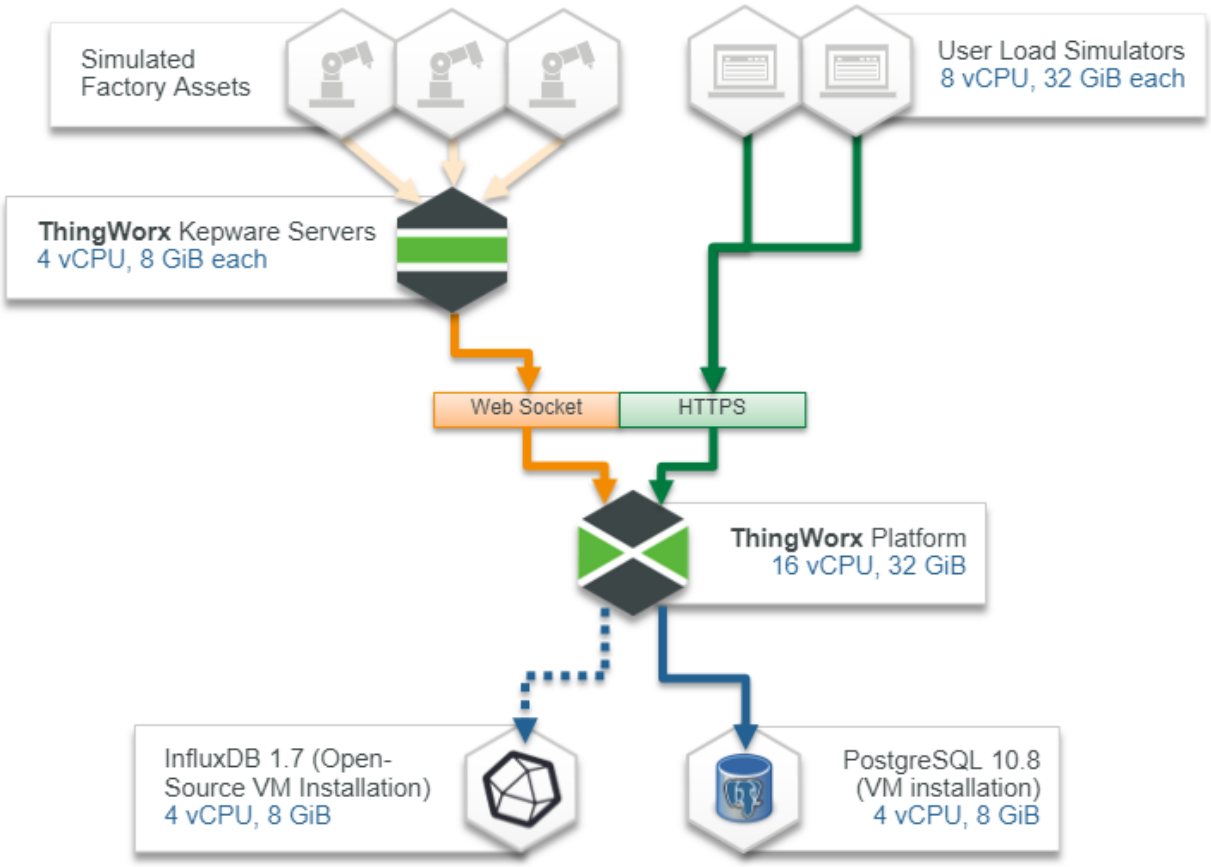


Figure 3 – The architecture: multiple Factory Assets from one Factory location connect to the Foundation server via Kepware Server. Future sections in this catalogue will include multiple Kepware Servers.

Each of the results matrices shown below is grouped by the frequency of property updates, same as before. However, now there are two different update frequencies, so each chart contains both of those frequencies in the top level of organization. All slow properties in that chart will use the larger "S" frequency, and all fast properties the smaller "F" frequency, regardless of other variations.

ThingWorx Model Configuration

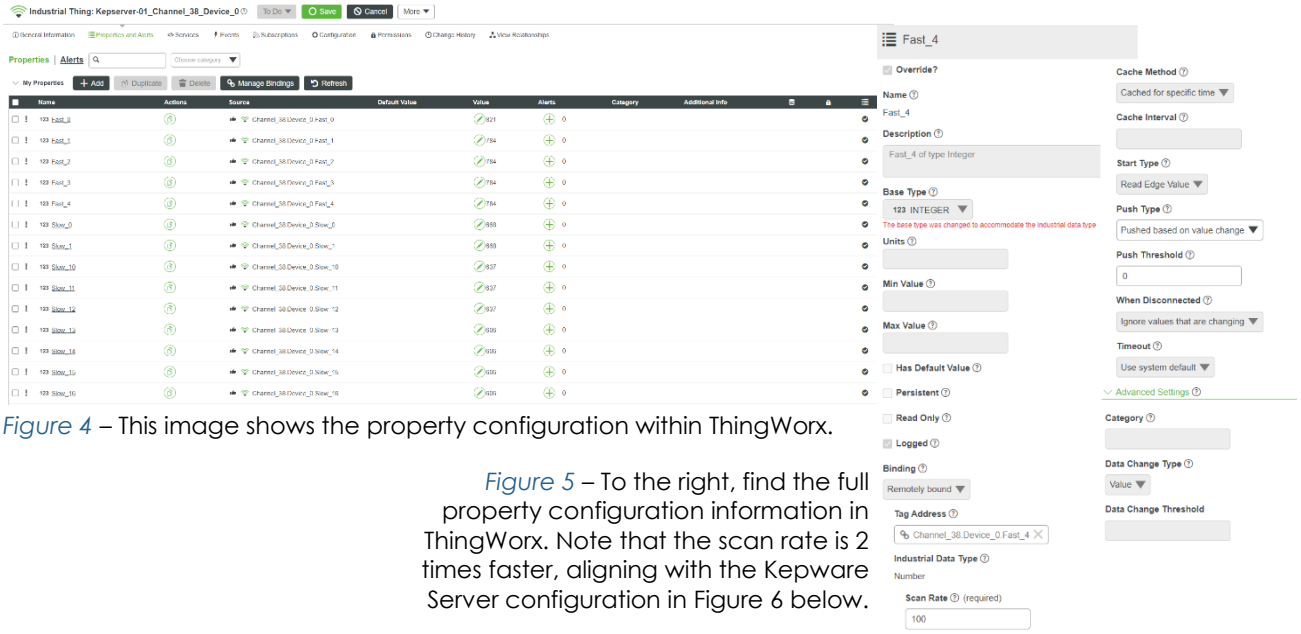
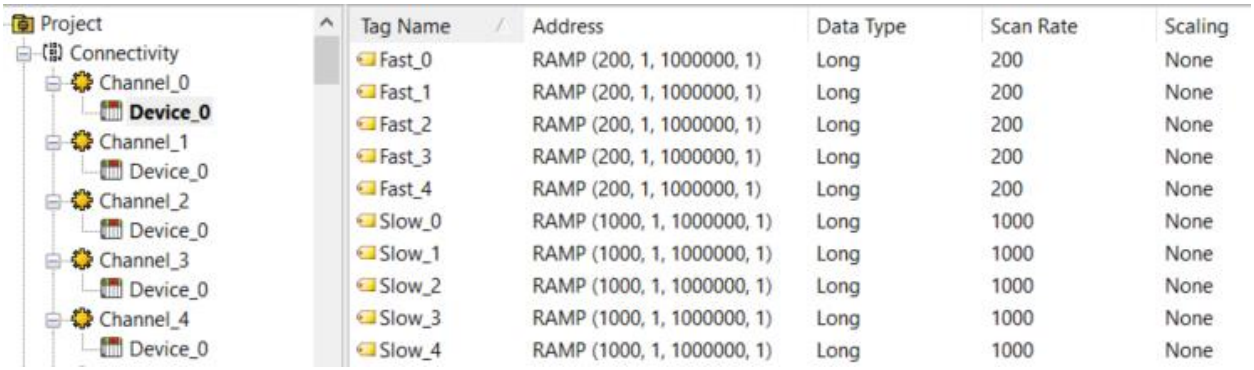


Figure 4 – This image shows the property configuration within ThingWorx.

Figure 5 – To the right, find the full property configuration information in ThingWorx. Note that the scan rate is 2 times faster, aligning with the Kepware Server configuration in Figure 6 below.

Kepware Server Configuration

For these tests the Simulation driver component of Kepware Server was used to "create" changing data to send to Thingworx. This provided a level of data throughput that can be measured for these tests, but note that this data does not fully represent the real-world scenario of polling data from industrial controllers and PLCs. Data collection over networks involves external factors that vary the consistency of new data being received by Kepware Server. Each simulation device in Kepware Server is analogous to a Thing in the Thing Model, while each tag in Kepware Servers configuration represents a property for that Thing. The tags were generated automatically by a python script which was run on a different server (with specifications shown in Figure 3).



Tag Name	Address	Data Type	Scan Rate	Scaling
Fast_0	RAMP (200, 1, 1000000, 1)	Long	200	None
Fast_1	RAMP (200, 1, 1000000, 1)	Long	200	None
Fast_2	RAMP (200, 1, 1000000, 1)	Long	200	None
Fast_3	RAMP (200, 1, 1000000, 1)	Long	200	None
Fast_4	RAMP (200, 1, 1000000, 1)	Long	200	None
Slow_0	RAMP (1000, 1, 1000000, 1)	Long	1000	None
Slow_1	RAMP (1000, 1, 1000000, 1)	Long	1000	None
Slow_2	RAMP (1000, 1, 1000000, 1)	Long	1000	None
Slow_3	RAMP (1000, 1, 1000000, 1)	Long	1000	None
Slow_4	RAMP (1000, 1, 1000000, 1)	Long	1000	None

Figure 6 – A screenshot from Kepware Server showing the tag configuration. This run had 30 properties total, 5 fast and 25 slow. Note that while a scan rate can be set within Kepware Server, when integrated with ThingWorx this value will be overridden by the Scan Rate set in the ThingWorx Model Configuration (as shown in Figure 5).

Data Flow Diagram

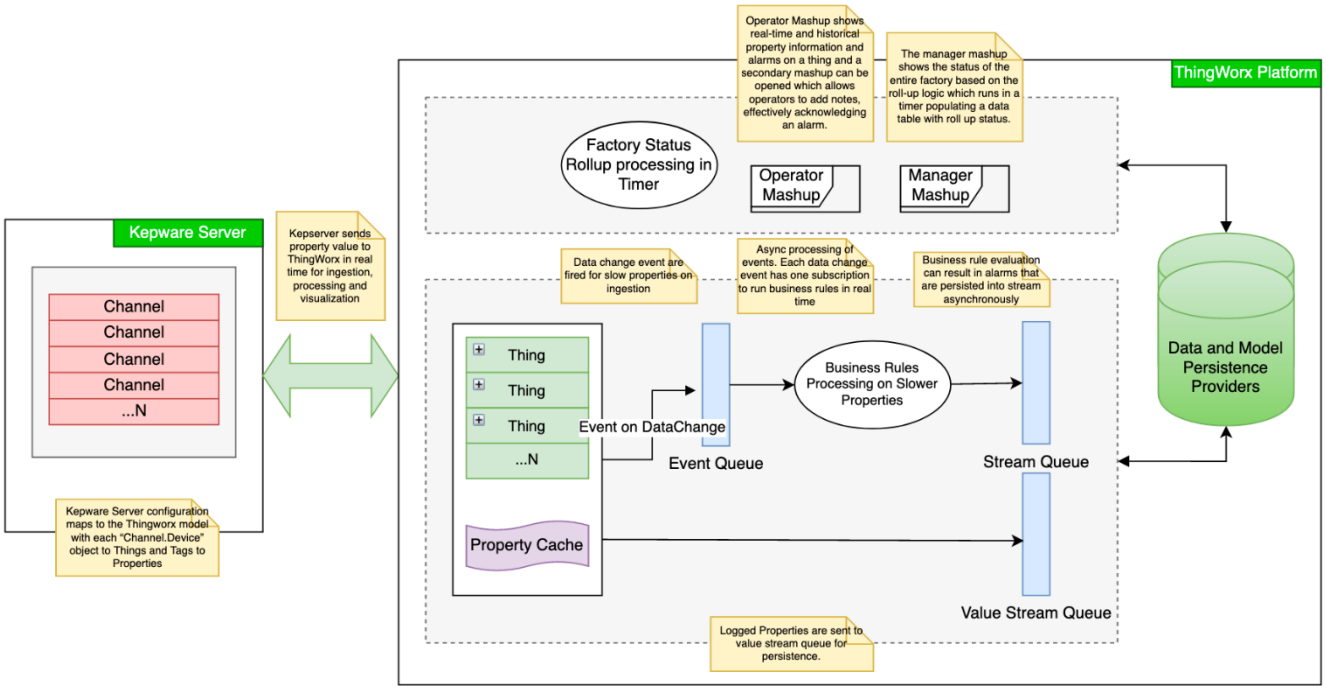


Figure 7 – This diagram shows the data flow for the entire application, from Kepware Server through the Foundation Server and into the data base.

Matrix 1 – Slow (15s slow properties, 1s fast)

S: 15s F: 1000ms		Frequency (R) / Number of Things (Y)	100	250	500
Frequency (R)	Properties per Thing (Z)	25+5	**WPS:** 667 **CPU Min/Avg/Max:** 6% / 8% / 13% **Memory Min/Avg/Max:** 4.5% / 6% / 6.2%	**WPS:** 1,670 **CPU Min/Avg/Max:** 8% / 10% / 16% **Memory Min/Avg/Max:** 9.3% / 9.4% / 9.5%	**WPS:** 3,340 **CPU Min/Avg/Max:** 13% / 15% / 19% **Memory Min/Avg/Max:** 38% / 38.5% / 39%
		50+10	**WPS:** 1330 **CPU Min/Avg/Max:** 9% / 11% / 15% **Memory Min/Avg/Max:** 28.2% / 28.3% / 28.4%	**WPS:** 3,340 **CPU Min/Avg/Max:** 15% / 17% / 21% **Memory Min/Avg/Max:** 46.4% / 46.4% / 46.5%	**WPS:** 6,400 **CPU Min/Avg/Max:** 9% / 24% / 41% **Memory Min/Avg/Max:** 4.8% / 16.1% / 16.3%
		75+15	**WPS:** 2,000 **CPU Min/Avg/Max:** 11% / 14% / 19% **Memory Min/Avg/Max:** 38.2% / 38.3% / 38.3%	**WPS:** 5,000 **CPU Min/Avg/Max:** 23% / 28% / 47% **Memory Min/Avg/Max:** 28.5% / 28.5% / 28.6%	**WPS:** 10,000 **CPU Min/Avg/Max:** 44% / 49% / 58% **Memory Min/Avg/Max:** 28% / 51% / 58%

Analysis

These runs represent the slowest overall frequency. All of these tests pass, the blue ones very easily (meaning the hardware sizing may be more than required) and the green ones perfectly.

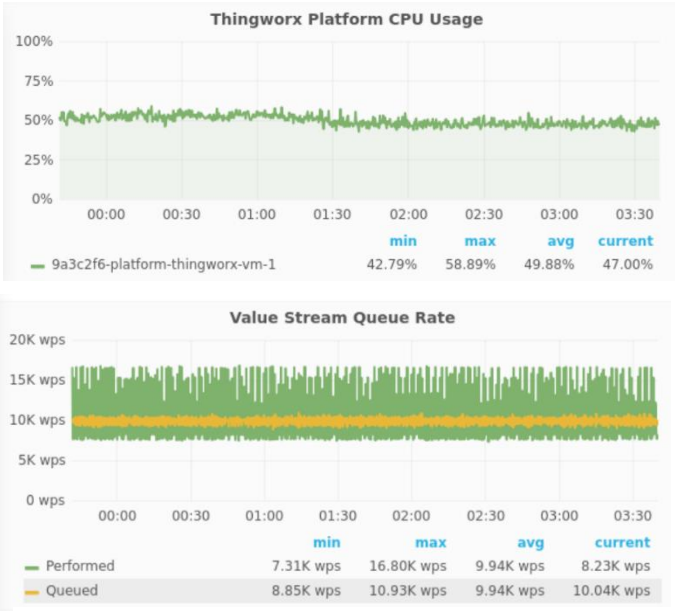


Figure 8 – Grafana Charts from largest run in Matrix 1

There was one data point reported as lost in the largest test here, which is well within acceptable standards for most customers. With steady CPU usage well below the maximum, there was room for handling spikes and variations in performance, even though none occurred here (shown in Figure 5). The Value Stream queue rate fluctuated up and down in response to the batched requests coming in from Kepware Server, but despite the variations, the average queue rate was very close to the target. There was only one lost point of data in the entire 4-hour test, across all 500 things. These results demonstrate that throughput is the greater limiting factor for Connected Factory tests because the business logic had no bearing on the success or failure of each test.

Matrix 2 – Fast (5s slow properties, 500ms fast)

S: 5s F: 500 ms		Frequency (R) / Number of Things (Y)		
		100	250	500
Frequency (R) / Properties per Thing (Z)	25+5	**WPS:** 1,500 **CPU Min/Avg/Max:** 9% / 11% / 16% **Memory Min/Avg/Max:** 4.5% / 4.8% / 21.7%	**WPS:** 3,750 **CPU Min/Avg/Max:** 14% / 17% / 21% **Memory Min/Avg/Max:** 9.1% / 9.4% / 9.5%	**WPS:** 7,510 **CPU Min/Avg/Max:** 26% / 28% / 32% **Memory Min/Avg/Max:** 46.3% / 46.4% / 46.5%
	50+10	**WPS:** 3,000 **CPU Min/Avg/Max:** 15% / 18% / 22% **Memory Min/Avg/Max:** 9.4% / 9.4% / 9.5%	**WPS:** 7,510 **CPU Min/Avg/Max:** 31% / 34% / 39% **Memory Min/Avg/Max:** 28.4% / 28.4% / 28.4%	**WPS:** 15,020 **CPU Min/Avg/Max:** 61% / 63% / 70% **Memory Min/Avg/Max:** 46.5% / 50.7% / 65.6%
	75+15	**WPS:** 4,500 **CPU Min/Avg/Max:** 24% / 26% / 30% **Memory Min/Avg/Max:** 15.7% / 15.8% / 15.8%	**WPS:** **11,260** (expected 11,300) **CPU Min/Avg/Max:** 54% / 58% / 62% **Memory Min/Avg/Max:** 38.4% / **73.2%** / **75.9%**	Not executed

Analysis

These runs represent the medium overall frequency of tests in this scenario. The largest 250 thing test in this category barely passed as there were only a couple of data points reported as lost by Kepware Server. Likewise, the Thingworx Foundation Server reported average and maximum memory usages that, while higher than would be ideal for a production system, would likely still be able to handle spikes in performance. The largest 500 thing run in this matrix passed with no data loss and no excessive memory or CPU consumption by the Foundation Server whatsoever.

The results here demonstrate that throughput capabilities between Kepware Server and Thingworx Foundation Server are maximized when the number of things is balanced with the size of the messages. The more properties there are, the larger each message itself must be, and therefore the more packets overall the IP layer will need to send across the limited-size pipe. More things with fewer properties yields more messages overall, but each message is smaller, so the throughput may actually be better when there are more things and less properties, as shown by the largest 500 thing run here.

Note that in this case, a 15k ingestion run was successful, but this may not always be the case. In general, 10k wps is the recommended limit for a single Kepware Server instance, but performance can vary widely from one thing model to the next. It is recommended to leave room for fluctuations that still come in below this limit as well.

Matrix 3 – Faster (1s slow properties, 200ms fast)

S: 1s F: 200ms	Frequency (R) / Number of Things (Y)	100	250	500
Frequency (R) / Properties per Thing (Z)	25+5	**WPS:** 4,970 **CPU Min/Avg/Max:** 23% / 25% / 33% **Memory Min/Avg/Max:** 38.3% / 38.4% / 38.4%	**WPS: 10,710** (expected 12,500) **CPU Min/Avg/Max:** 6% / 49% / 68% **Memory Min/Avg/Max:** 28.4% / 28.5% / 28.6%	Not executed
	50+10	**WPS: 9,930** (expected 10,000) **CPU Min/Avg/Max:** 52% / 54% / 59% **Memory Min/Avg/Max:** 28.3% / 28.3% / 28.4%	Not executed	Not executed
	75+15	**WPS: 10,240** (expected 15,000) **CPU Min/Avg/Max:** 65% / **71%** / **77%** **Memory Min/Avg/Max:** 38.3% / 51% / 52.9%	Not executed	Not executed

Analysis

This category represents the highest frequency of property updates in this scenario. For small numbers of things and properties, the higher scan rates are ok, as seen by the smallest test here (marked in green). The thing model is the limiting factor here again, in all but the largest test (which exceeded healthy levels of CPU performance on the ThingWorx Foundation Server). It is possible in this largest test that without the back pressure of the business logic, or with more resources allocated to the Platform to better handle that logic, that the ingestion KPI could be met, but it still exceeds the recommended use for Kepware Server because a steady throughput that high is risky, leaving no room for spikes in activity.

Note the same pattern found in this chart: the larger number of things, fewer properties test (250 things, 30 properties) resulted in a higher ingestion rate than the smaller number of things, more properties test, even though the former has more messages (one might therefore expect it to have a lower throughput). However, when there are more properties in play, each individual message is itself larger and may be broken down further by the IP layer. This will result in even more messages overall. The more messages there are, the lower the throughput. That's why some of the tests with expected ingestion rates of 10k fail, even while the ingestion rate exceeds this level in other tests. The pipe has a limited size, and the more messages are shoved into that pipe at a time, the slower they all move.

Conclusions

The big take-away from this scenario is that the chosen thing model has a major impact on throughput. This is because the ThingWorx Native Interface within the Kepware Server application batches updates on a per thing basis. It generally avoids batching multiple updates to meet the real-time monitoring needs of Connected Operations use cases, but if the property updates are happening on the same thing at the same time, then they are sent as one message. So the more things there are as well as the higher the frequency of property updates on those things, the more messages per second are needed, and therefore the more risk there is for data loss as the application scales up.

On the other hand, the more property updates must be sent at once, the larger the messages are sized, and therefore the more likely they are to be broken down and made into more messages by the IP layer. Therefore, selecting a thing model is all about balancing the size of the messages sent by Kepware Server with the number of total messages per second. Ensure there is no offset between measurements of property values on the physical devices themselves, i.e. synch the scan rates, to manipulate the number of properties which will be batched together at a time.

Consider if it is better to use more things with less properties or less things with more properties. For instance, if there is a conveyor belt that has a scale as well as a motor, it can be better to represent the scale and the motor by using differently named properties (e.g. "motor_speed" or "scale_weight") on a single conveyor thing, as opposed to making separate "conveyor motor" and "conveyor scale" things. However, consider that if there are enough properties to each conveyer component that they might be better off as their own things, each having fewer properties overall.

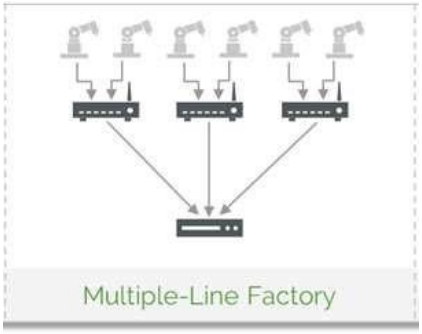
Reducing the overall number of messages sent across the network is the way to maximize throughput from Kepware Server to ThingWorx, which will mean different things for different customers. Generally, it is best to plan to scale horizontally, using more than one Kepware Server when the ingestion rate scales up over 10k or so wps at a single factory (which will be explored in detail in the next scenario added to this document). However, there are many factors that should also be considered when it comes to application architecture, including network structure and bandwidth, hardware/processing availability and installation, maintenance, and use case requirements. There certainly aren't any "one size fits all" recommendations that can be made, and discussion is very welcome on <link>our community<link>.

Scenario Two: Multiple Kepware Servers in ThingWorx 8.5

Implementation Architecture

Building on the prior scenario, additional tests were executed with multiple Kepware Server instances: a common deployment pattern for multi-line factory implementations. Tests below 10,000 writes per second were not repeated with multiple Kepware Servers.

ThingWorx deployment sizing was not changed during these tests to demonstrate the limits of a given configuration. Changes that may improve the results of a failed test (such as adding CPUs or Memory) will be mentioned but not validated as part of this benchmark.



Matrix 1 – Slow (15s slow properties, 1s fast)

S: 15s F: 1000ms			Frequency (R)		
			Number of Things (Y)		
			100	250	500
Frequency (R)	Properties per Thing (Z)	75+15	**Target WPS:** 2,000 Not executed *(Requires 1 Kepware Server)*	**Target WPS:** 5,000 Not executed *(Requires 1 Kepware Server)*	**WPS:** 10,000 **CPU Min/Avg/Max:** 43% / 47% / 52% **Memory Min/Avg/Max:** 38% / 55% / 78% *(Note: 2 Kepware Servers)*

Analysis

At these data frequencies, ThingWorx Foundation performance was largely unchanged whether one or two Kepware Servers were used to provide steady-state edge load.

However, in general the two Kepware Server configuration is preferable. The single Kepware Server running near the recommended maximum data rate and thing count would be susceptible to data loss in a more real-world scenario, where the data rate could easily spike above 10,000 writes per second for a period of time.

While the smaller scenarios from this matrix were not repeated with multiple Kepware Servers, it might be worth considering multiple Kepware Servers for smaller solutions, especially if lower bandwidth and/or higher latency network conditions exist between ThingWorx and Kepware Server, or anything that could reduce the maximum steady-state throughput between the Foundation and the Edge.

Matrix 2 – Fast (5s slow properties, 500ms fast)

S: 5s F: 500 ms		Frequency (R)		
		Number of Things (Y)		
Frequency (R) / Properties per Thing (Z)		100	250	500
50+10		**Target WPS:** 3,000 Not executed *(Requires 1 Kepware Server)*	**Target WPS:** 7,500 Not executed *(Requires 1 Kepware Server)*	**WPS:** 15,010 **CPU Min/Avg/Max:** 60% / 63% / 67% **Memory Min/Avg/Max:** 46% / 50% / **73%** *(Note: 2 Kepware Servers)*
75+15		**Target WPS:** 4,500 Not executed *(Requires 1 Kepware Server)*	**WPS:** 11,280 (expected 11,300) **CPU Min/Avg/Max:** 53% / 57% / 61% **Memory Min/Avg/Max:** 38% / 38% / 38% *(Note: 2 Kepware Servers)*	**WPS: 14,390** (expected 22,500) **CPU Min/Avg/Max:** **91% / 94% / 96%** **Memory Min/Avg/Max:** **79% / 82% / 83%** *(Note: 3 Kepware Servers)*

Analysis

At this data frequency, the three simulation configurations above 10,000 writes per second were repeated. The 15,000 WPS test was successful, but the maximum ThingWorx Foundation memory consumption exceeded 70%, the safe threshold identified for these simulations.

A three Kepware Server configuration was used for the 22,500 WPS scenario above. This run was unsuccessful – at these data rates, the server sizing for ThingWorx Foundation was not large enough to handle the increased load – both CPU and Memory Utilization were far above acceptable, and a significant amount of data loss was observed.

Thread dumps confirmed the high CPU was caused by the volume of business logic at these data rates.

Matrix 3 – Faster (1s slow properties, 200ms fast)

S: 1s F: 200ms		Frequency (R) / Number of Things (Y)		
		100	250	500
Frequency (R) / Properties per Thing (Z)	25+5	Target WPS: 5,000 Not executed *(Requires 1 Kepware Server)*	**WPS: 12,410** (expected 12,500) **CPU Min/Avg/Max:** 50% / 54% / 62% **Memory Min/Avg/Max:** 15% / 49% / 50% *(Note: 2 Kepware Servers)*	Target WPS: 25,000 Not executed *(Requires 3 Kepware Servers + larger Foundation instance)*
	50+10	**WPS: 9,930** (expected 10,000) **CPU Min/Avg/Max:** 52% / 54% / 59% **Memory Min/Avg/Max:** 28% / 28% / 28% *(Note: 2 Kepware Servers)*	Target WPS: 25,000 Not executed *(Requires 3 Kepware Servers + larger Foundation instance)*	Target WPS: 50,000 Not executed *(Requires 5-6 Kepware Servers + larger Foundation instance)*
	75+15	**WPS: 11,080** (expected 15,000) **CPU Min/Avg/Max:** **89% / 94% / 97%** **Memory Min/Avg/Max:** **80% / 82% / 83%** *(Note: 2 Kepware Servers)*	Target WPS: 37,000 Not executed *(Requires 4 Kepware Servers + larger Foundation instance)*	Target WPS: 75,000 Not executed *(Requires 8 Kepware Servers + larger Foundation instance)*

Analysis

Much like the single Kepware Server tests performed previously, at these rapid data rates, Kepware Server's simulation tooling introduces some data loss, as both the 10,000 and 12,500 WPS test above illustrate.

While these two tests are marked in yellow, given the ThingWorx Foundation CPU and Memory consumption results on these two tests and that the rate of simulation-induced data loss is similar in the comparable tests with one Kepware Server these two setups are likely stable in a real-world scenario.

Much like the failed test on the previous page, the 15,000 WPS test above experienced data loss beyond that introduced by the simulation tooling and also exhausted almost all CPU and Memory resources on the ThingWorx Foundation virtual machine.

Conclusions

With a single Kepware Server, the ThingWorx Foundation deployment architecture selected for this reference benchmark was more than adequate for Edge data loads up to 10,000 writes per second.

By adding a second Kepware Server this load was able to be increased beyond 10,000 writes per second, but as ingestion approached 15,000 WPS, the increased load from business logic processing began to encounter insufficient CPU and memory resources.

To overcome the CPU and memory capacity issues from these failed tests, changes to the deployment architecture or the application complexity would be needed. Options to consider could include one or more of the following:

- Vertical scale (or "sizing up") by adding CPU and Memory to the ThingWorx Foundation VM. Faster physical CPUs could also be considered if available.

  Note: *To confirm this analysis, the failed tests were repeated with a larger (32 vCPU, 64 GiB RAM) ThingWorx Foundation VM and were successful.*

- Horizontal scale (or "sizing out") by deploying a ThingWorx cluster with multiple active nodes operating in parallel to distribute business logic and user load.

  Note: *ThingWorx cluster configurations require ThingWorx 9.0 or later.*

- If adjusting the hardware footprint is not possible, reducing the frequency or complexity of the business logic within the ThingWorx application could also be considered (for example: trigger more complex, multi-property rules on a timer instead of automatically with every data change).

Scenario Three: One Kepware Server in ThingWorx 9.0

The goal of this scenario is to confirm the same performance in **ThingWorx 9.0** as seen in scenario one, where one Kepware Server represented a single factory in version 8.5.

S: 15s
F: 1000ms

Frequency (R)	Properties per Thing (Z)	Number of Things (Y): 250	Number of Things (Y): 500
Frequency (R)	50+10	**WPS:** 3,330 **CPU Min/Avg/Max:** 11.4% / 16.6% / 22.4% **Memory Min/Avg/Max:** 16.3% / 16.4% / 16.5%	**WPS:** 6,670 **CPU Min/Avg/Max:** 21.8% / 25.6% / 29.5% **Memory Min/Avg/Max:** 16.3% / 25.8% / 39.4%
Frequency (R)	75+15	**WPS:** 5,010 **CPU Min/Avg/Max:** 28.8% / 54.7% / 58.6% **Memory Min/Avg/Max:** 17.5% / 22.6% / 27.8%	**WPS:** 10,000 **CPU Min/Avg/Max:** 35% / 39.2% / 46% **Memory Min/Avg/Max:** 28.5% / 28.7% / 28.8%

S: 5s
F: 500 ms

Frequency	Properties	Number of Things (Y): 250	Number of Things (Y): 500
Frequency	50+10	**WPS:** 7510 **CPU Min/Avg/Max:** 24% / 28% / 32% **Memory Min/Avg/Max:** 16.4%% / 16.4% / 16.5%	**WPS: 12,080** (should be 15,000) **CPU Min/Avg/Max:** 35.8% / 44.7% / 53.4% **Memory Min/Avg/Max:** 28.9% / 29.1% / 29.2%

S: 1s
F: 200ms

Frequency (R)	Properties per Thing (Z)	Number of Things (Y): 100	Number of Things (Y): 250
Frequency (R)	25+5	**WPS:** 4,970 **CPU Min/Avg/Max:** 38.3% / 38.4% / 38.4% **Memory Min/Avg/Max:** 5% / 5.3% / 5.3%	**WPS: 12,340** (should be 12,500) **CPU Min/Avg/Max:** 40.9% / 46% / 50.7% **Memory Min/Avg/Max:** 16.3% / 16.9% / 16.9%
Frequency (R)	50+10	**WPS: 9,910** (should be 10,000) **CPU Min/Avg/Max:** 41.4% / 44.8% / 47.8% **Memory Min/Avg/Max:** 28.8% / 28.9% / 30%	Not executed

Matrix 1 - Slow (15s slow properties, 1s fast)

The lower frequency tests performed the same in 9.0. Even the 10k ingestion test, which lies very close to the boundary for a single Kepware Server, passed with no errors.

Matrix 2 – Fast (5s slow properties, 500ms fast)

These showed similar results, but the 500 thing, 50-10 property test had data loss in 9.0. However, the write rate is much higher than PTC recommends for a single Kepware Server anyway.

Matrix 3 – Faster (1s slow properties, 200ms fast)

The fastest tests had similar results as well. The larger tests ran with more success with two Kepware Servers (data not shown here).

Conclusions

ThingWorx 9.0 is similarly capable of ingesting data using Kepware Server. A single instance can still achieve up to 10k wps. **Future scenarios will now make use of ThingWorx 9.0**