Ignition by inductive automation

Ignition Server Sizing and Architecture Guide

inductive automation

inductiveautomation.com 1-800-266-7798

Table of Contents

Introduction

Ignition is a development toolkit, with unlimited licensing and different modules, that give you the tools to build solutions. An Ignition project can be as small as a data collector for a few tags or as large as an enterprise solution with hundreds of devices, hundreds of thousands of tags, and hundreds of visualization clients. It is extremely important to find the right architecture and sizes of servers for your Ignition project so that the project will work as intended and have room for growth in the future.

This guide is intended to provide some tips to help you determine the correct architecture depending on your requirements. It is important to note that any architecture that you come up with needs to be fully tested and verified. Throughout that process you can observe the performance characteristics of the server in order to make any necessary adjustments to the architecture. There is no guarantee on performance since it is based on your design choices.

Ignition's performance and sizing is based on several different factors:

- Number of device connections
- Types of PLCs (driver)
- Number of tags
- Frequency of tag polling (1 second, 5 seconds, 1 minute, etc.)
- Number of tag value changes per second (% of change)
- Number of concurrent visualization clients
- SQL database throughput (historian, alarm journal, etc.)
- Deployment (physical machine, VM, etc.)
- Network & latency
- And more...

The features you use in Ignition also play a large role in performance. Features, such as the following, require processing time and memory:

- Device communication
- OPC-UA (client and server)
- Tags (OPC, Expression, SQL query, etc.)
- Historian (storage and retrieval)
- Alarming (status, journal, notification pipelines)
- Transaction groups (especially large numbers)
- Scheduled PDF reports
- Sequential Function Charts
- Scripting
    - Tag change scripts
    - Scripts running on the server (timer, message handlers, etc.)
- Visualization clients

  - Number of tag subscriptions on a screen
  - Polling queries (tag historian, alarming, custom queries, etc.)
  - Number of Gateway requests
- Gateway Network (remote services, EAM)
- MES functionality
- MQTT or cloud injection
- And more...

Ignition is heavily multithreaded and can handle configurations with all of the features above at reasonable limits. Some of the features above, such as device communication and tags, have predictable performance. Other features, such as visualization clients, have less predictable performance since they are based on how the user interacts with the system and how the project is configured. For example, we can allow a user to pull back a year of historical data on a trend and poll it at a one-second rate. We can go hours or days without anyone using that screen but then we could have a situation where 10 people are using it at the same time causing an increased load on the server. In most cases, a single server is sufficient to handle everything. However, when projects get large or when we push the limits of Ignition, we need to utilize multiple servers. Luckily, Ignition is modular and has the ability to scale-out by separating different modules and features onto dedicated servers. At the end of the day, those separate servers work as one Ignition unit and are completely transparent to the user, thanks to Ignition's Gateway Network and standards like OPC-UA, SQL, MQTT, and more.

The purpose of this guide is to walk you through projects of different sizes to understand their hardware requirements and architecture. Since Ignition has a lot of different features, this guide is going to focus on the number of devices, tags, and clients.

SQL Database

The SQL database (MySQL, Microsoft SQL Server, etc.) in all of these scenarios below should be on its own dedicated server. If you want to put it on the same server as Ignition, you will need more processing power and memory. Here are some basic database sizing tips:

Small Historian

2 Cores (2 GHz+), 2GB memory, SSD

0-100 value changes per second

Requires approximately 300GB disk space/year if 100% of the values are changing every second sustained (approximately 6GB with 2% change, smaller with slower rates)

2 Cores (2 GHz+), 4GB memory, SSD

100 - 500 value changes per second

Requires approximately 3TB disk space/year if 100% of the values are changing every second sustained (approximately 60GB with 2% change, smaller with slower rates)

Medium Historian

4 Cores (4 GHz+), 8GB memory, SSD

500 - 2,500 value changes per second

Requires approximately 7TB disk space/year if 100% of the values are changing every second sustained (approximately 150GB with 2% change, smaller with slower rates)

4 Cores (4 GHz+), 16GB memory, SSD

2,500 - 5,000 value changes per second

Requires approximately 15TB disk space/year if 100% of the values are changing every second sustained (approximately 300GB with 2% change, smaller with slower rates)

Large Historian

8 Cores (4 GHz+), 16GB memory, SSD

5,000 - 10,000 value changes per second

Requires approximately 30TB/year if 100% of the values are changing every second sustained (approximately 60GB with 2% change, smaller with slower rates)

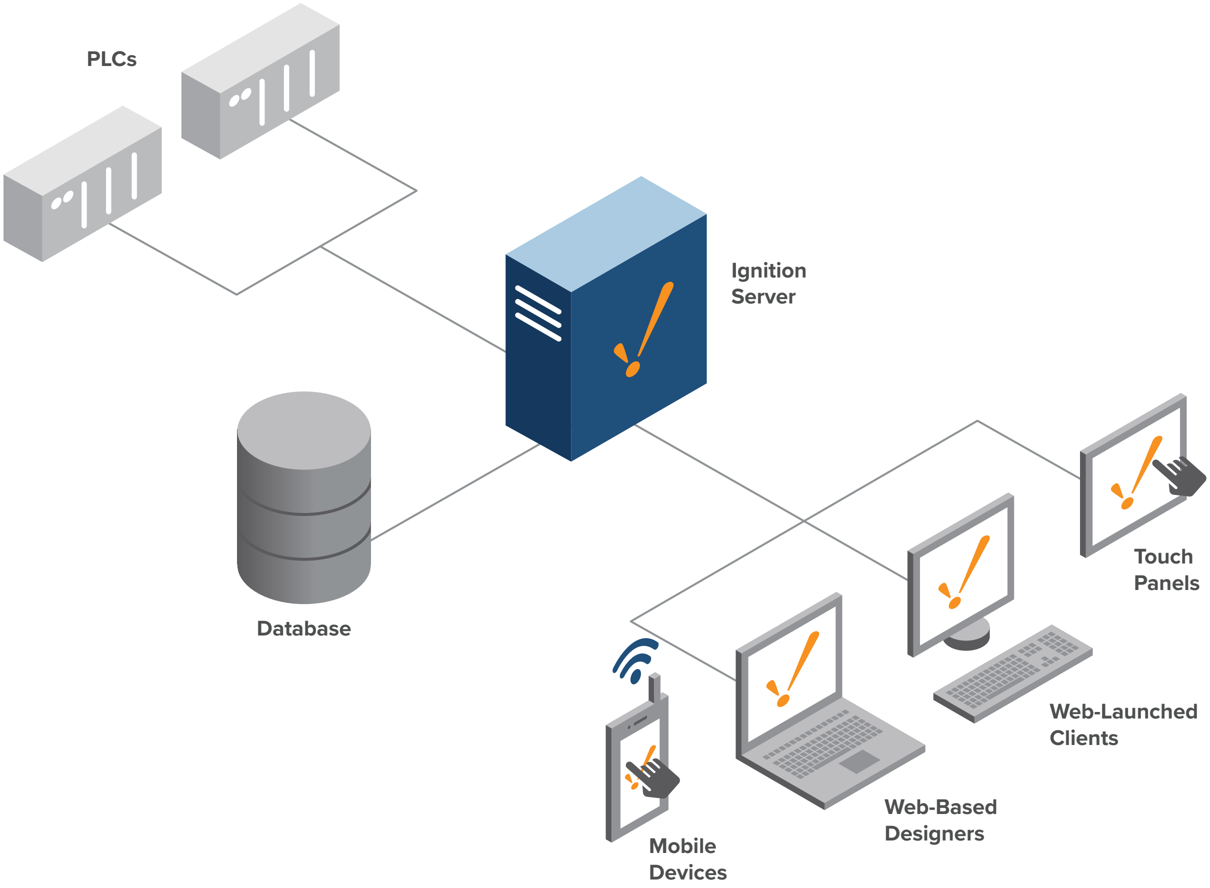
Note: Some databases on a properly tuned server can max out at 10,000 value changes per second. For example, MySQL will max out server performance at 10,000 value changes per second, so it is advisable to stay within 5,000. Microsoft SQL Server can handle up to 20,000-30,000 value changes per second with the latest version. However, throughput and query speed is dependent on hardware and database setup. Speed of retrieving data in Ignition can also be impacted with higher storage throughput. You can achieve higher throughput by installing another Ignition instance on the same server as the database and use Ignition's remote tag history feature through the Gateway Network. This will avoid sending a large amount of raw data (SQL inserts) through the network and will, instead, send compressed data over the network, ultimately resulting in the same inserts but executed locally.

Ignition Memory

The amount of memory Ignition has allocated in Ignition's ignition.conf configuration file can be adjusted. You need to set the proper amount of memory for each of the scenarios below. Typically, you can leave 1-2GB of memory to the Operating System (OS) and use the rest for Ignition. See our documentation for more details on setting Ignition's memory:

https://docs.inductiveautomation.com/display/DOC81/Gateway+Configuration+File+Reference#GatewayConfigurationFileReference-GatewayMemoryAllocation

Standard Architecture (Single Server)



Ignition's most common architecture consists of a single on-premise Ignition server connected to a SQL database, PLCs, and clients. In this case, all functionality is configured on the same Ignition server. Ignition's licensing is unlimited. However, Ignition is limited by the size of the hardware. With larger hardware, Ignition can handle more device connections, tags, and clients.

Small Ignition Project

Perfect for Edge Panel, Edge IIoT, HMI, small SCADA, data collection, alarming.

ARM, 1GB memory, SSD

1-2 devices

500 tags

2 concurrent clients †

2 Cores (2 GHz+), 2GB memory, SSD

1-10 devices

2,500 tags

5 concurrent clients †

2 Cores (3 GHz+), 4GB memory, SSD

1-10 devices

5,000 tags

10 concurrent clients †

Medium Ignition Project

Medium full SCADA systems or MES systems.

4 Cores (3 GHz+), 4GB memory, SSD

1-25 devices

10,000 tags

20 concurrent clients †

4 Cores (4 GHz+), 8GB memory, SSD

1-50 devices

25,000 tags (40% tag history change at max)

25 concurrent clients †

4 Cores (4 GHz+), 16GB memory, SSD

1-100 devices

50,000 tags (20% tag history change at max)

50 concurrent clients †

Large Ignition Project

Large full SCADA systems or MES systems.

8 Cores (4 GHz+), 8GB memory, SSD

1-100 devices

75,000 tags (10% tag history change at max)

50 concurrent clients †

8 Cores (4 GHz+), 16GB memory, SSD

1-100 devices
100,000 tags (10% tag history change at max)
75 concurrent clients [†]

16 Cores (4 GHz+), 32GB memory, SSD

1-100 devices
150,000 tags (6% tag history change at max)
100 concurrent clients [†]

[†] *Results can vary based on design choices. The examples are based on typical usage. Faster polling rates, increased value changes, and utilization of other Ignition features, such as scripts, Transaction Groups, SFCs, and more, can significantly impact the overall performance.*

Scale-Out Architecture

In larger systems, it is often easier to have multiple Ignition installations to help split the load between the frontend tasks and the backend tasks. This is perfect for single large systems that aren't split up into different sites. In cases where systems are split into different sites, the Hub and Spoke Architecture is usually a better fit.

In the Scale-Out architecture, you have at least one Gateway that handles backend communications. The backend Gateway deals with all PLC and device communications. The frontend Gateway handles all of the Clients, serving up the data pulled from the backend Gateway. This is made possible through the Gateway Network, connecting Gateways to each other, and allowing tags to be shared through remote tag providers.

Gateway Network

The Gateway Network allows you to connect multiple Gateways together over a wide area network and opens up many distributed features between Gateways.

The Gateway Network provides the following features:

- A dedicated HTTP data channel that can handle multiple streams of message data.
- The ability to set up a node to act as a proxy for another node.
- Security settings that restrict incoming connections based on a whitelist or on manual approval of the connection. Incoming connections can also be disabled entirely.
- An available SSL mode. When enabled, connections must send SSL certificates to prove their identity. A connection will not be accepted until its SSL certificate is approved. Optionally, client certificates can be configured to be required as well as server certificates.

Gateway Network Features

The Gateway Network opens up certain services for use that make managing multiple Gateways and effective communication with each of the Gateways easy. It also has special security that can restrict certain services from happening in certain zones of the Gateway Network.

Enterprise Administration

The Enterprise Administration Module (EAM) uses the Gateway Network for message and file transfer and can monitor network connections for availability. The EAM reports whenever communications are lost via alarm events and system tags.

Distributed Services

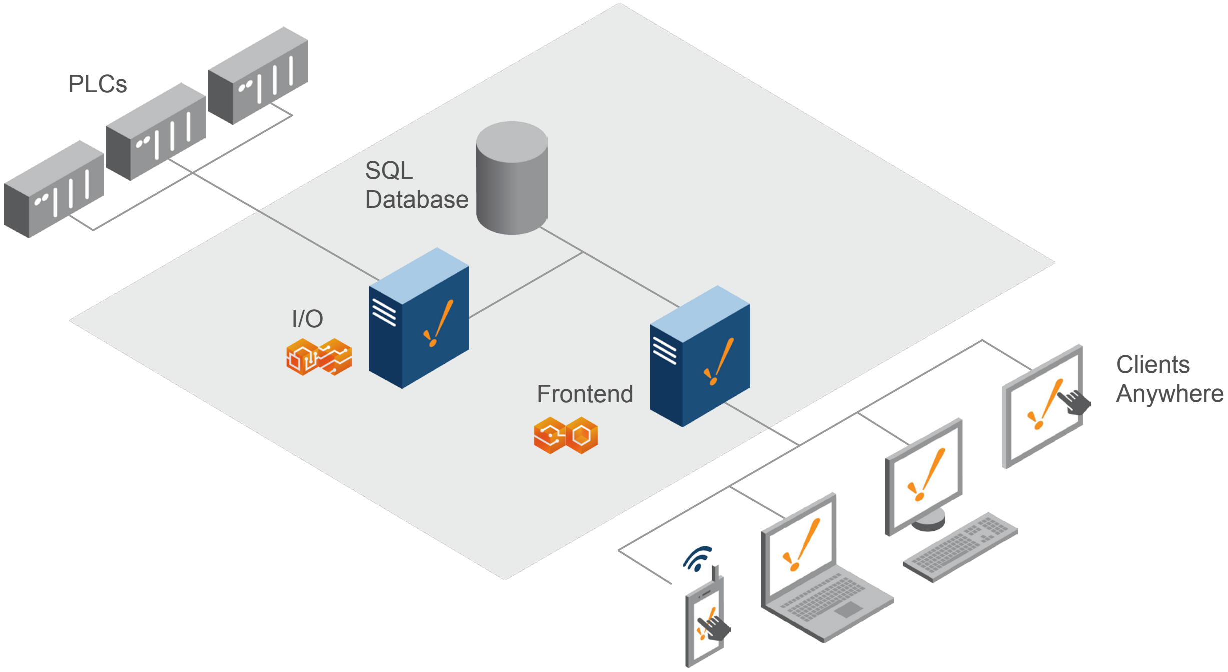
Distributed services included the following:

- Remote Realtime Tag Providers: Provides the ability to read/write tags from a remote Ignition server. Perfect for frontend servers that need to see the tags on the backend. This includes reading and writing to tags, seeing alarms, and even querying historical data.
- Remote Historical Tag Providers: Provides the ability to send historical data to another Ignition server for storage to a SQL database. Also provides the ability to remotely query historical data when a connection to the SQL database is not available.
- Remote Alarming: Provides the ability to send alarm notifications to another Ignition server. Perfect when email, voice, or SMS is only available on a central Ignition server.
- Remote Alarm Journal: Provides the ability to send alarm history to another Ignition server for storage to a SQL database.
- Remote Audit Logs: Provides the ability to send audit logs to another Ignition server for storage to a SQL database.

Security Zones and Service Security

Security Zones can be set up to lock down or prevent access to certain parts of Gateways within the Gateway Network. You can configure the service level security on each system to define what is allowed from remote Ignition servers.

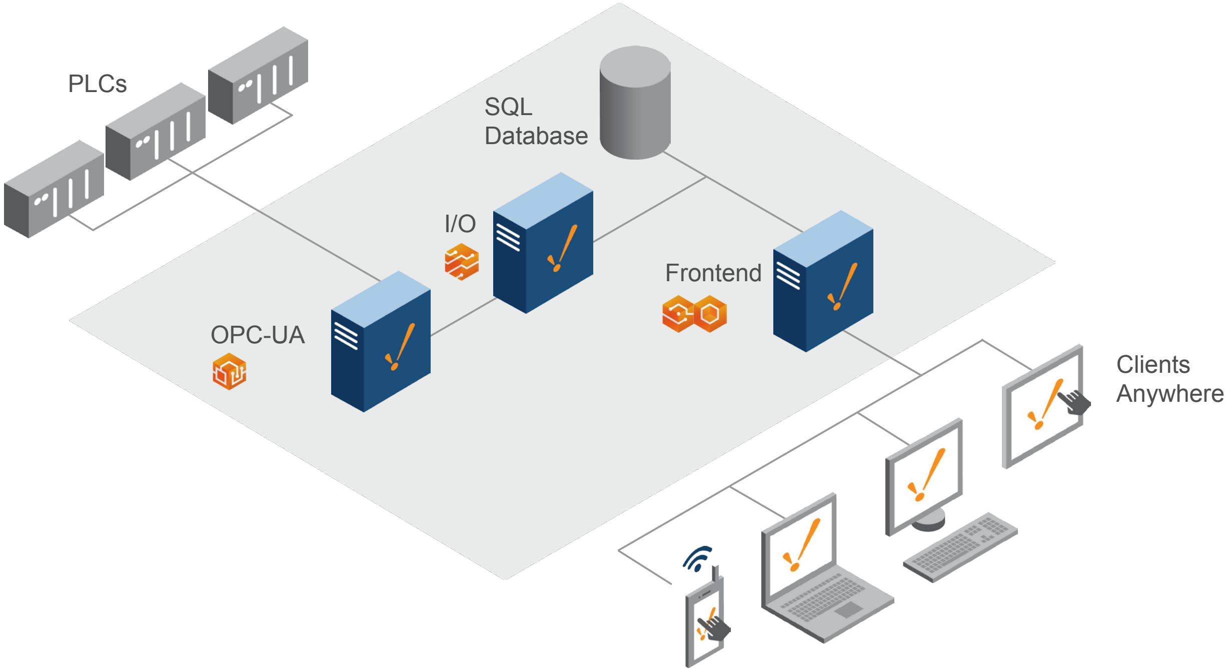
One I/O / One Frontend



With this architecture, you have one Ignition server communicating to the PLCs and performing backend tasks such as polling live values, historian, and alarming. The backend server is responsible for logging data to the SQL database. The frontend server handles all of the visualization clients (Vision and Perspective) and should also communicate to the SQL database for querying historical data.

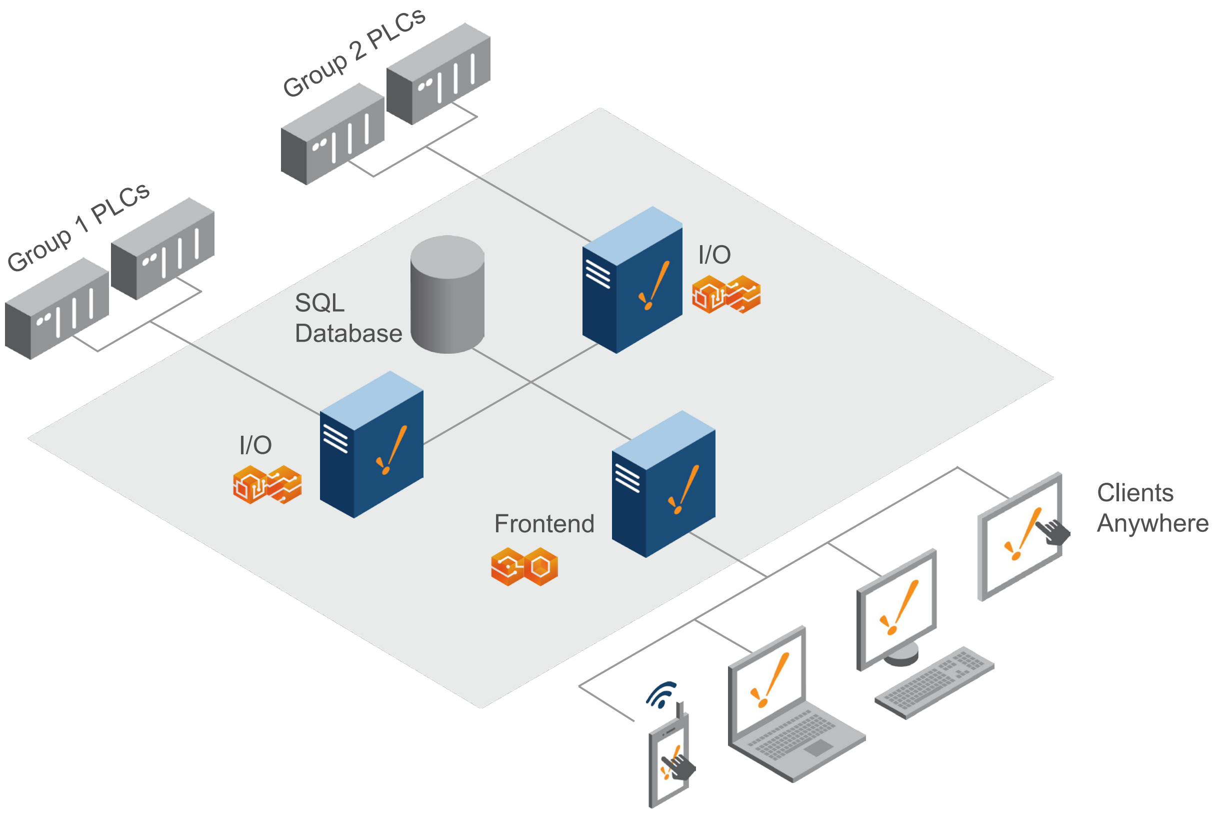
With this architecture, you can handle more device connections, tags, and clients since you have two servers. You can easily expand to multiple I/O servers and frontend servers as the size of the project increases, allowing for easy scalability.

One OPC-UA Server / One I/O / One Frontend



With this architecture, a dedicated OPC-UA server that handles all of the PLC communication was introduced. Essentially, you are breaking apart OPC-UA from a single server. This architecture lets one handle a larger set of devices. The I/O server simply handles the tag value changes. You can easily handle a larger set of tags, in regards to history and alarming, with additional OPC-UA servers for different sets of PLCs.

Two I/O / One Frontend



With this architecture, you have two Ignition servers communicating to different sets of PLCs, allowing for communication to a larger set of devices and tags. For example, it is possible to handle approximately 250,000 tags from 200 devices. You can easily add additional I/O servers as necessary.

Tag Provider Best Practice

With the scale-out architecture, it is important to provide proper names to your tag providers and the names should be consistent across the I/O and the frontend. You should avoid using the default provider that comes in a standard installation. It is common to create a new realtime tag provider on the I/O server with a name that describes that I/O server and is distinguished from other I/O servers. On the frontend server, you should create a remote tag provider with the same name as the I/O server so the two are consistent.

Tag Paths

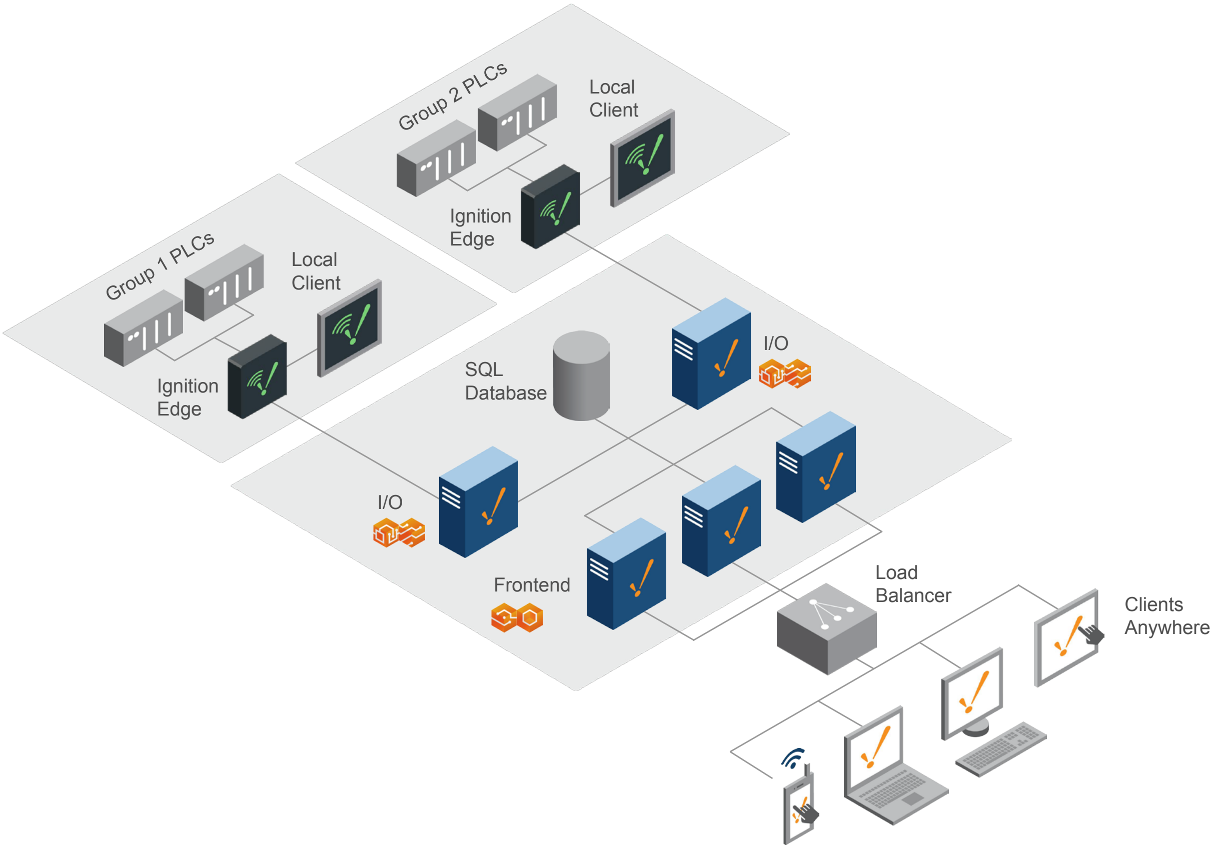
It is recommended to use fully-qualified tag paths in your projects, especially with visualization templates and screens. A fully-qualified path allows Ignition to know exactly where the tag is, including the tag provider name.

For example, a non-fully-qualified tag path looks like the following:
Path/to/my/tag

With that path, you don't know which tag provider it comes from. An Ignition project will use the project's default tag provider, which is only one provider.

A fully-qualified tag path looks like the following:
[TagProviderName]Path/to/my/tag

Notice the path includes the tag provider. You can easily provide a parameter in a visualization template that can change the tag provider and the tag path, allowing us to point to any tag from any provider.



The best thing about the Scale-Out architecture is that it is easy to scale up Ignition as your system grows. In the image above, more frontend Gateways were added to help handle an increase in clients, and a Load Balancer to automatically distribute the clients between them. When using a load balancer, it is important to turn on "sticky" sessions to ensure the connection stays consistent for at least one hour. The frontend server has to be completely stateless. It gets its data from the I/O servers and the SQL database. Using this architecture with an appropriate number of frontend servers, you can handle thousands of concurrent clients along with a high number of devices and tags.

Examples of Load Balancers:

- F5 Load Balancer
- HAProxy
- AWS / Azure LB

Value Changes Are Key

When it comes to tags, value changes are the most important metric to look at. This is defined as a change in value that is more than the configured deadband. Deadbands can be absolute or percentage based. For example, you have an analog value with a deadband of 0.1. A change from 7.89 to 7.88 would not be considered a change. However, a change from 7.89 to 7.9 would be considered a change. Usually, any change with discrete values is considered a change, since we are dealing with a state. Deadbands are configurable on a per tag basis.

The reason value changes are key, is due to the fact that we have to process the value change, for alarms, the historian, and more. That processing requires memory and compute time. The more tags we have changing per second, the more processing. An Ignition server at the high end can handle approximately 50,000 value changes per second processing through the tag system, alarming, historian, and clients. However, as mentioned above, some SQL databases have a limit of 10,000 value changes per second on a dedicated instance. So, your job is to try to reduce the number of value changes per second. Ignition can handle more devices, tags, and clients through optimization and reducing value changes.

Let's look at some different scenarios that help to understand what a server can handle. Each scenario represents a number of tags that max out performance on a server.

- 10,000 tags at 1 second rate with 100% changing = 10,000 values/sec
- 50,000 tags at 5 second rate with 100% changing = 10,000 values/sec
- 100,000 tags at 1 second rate with 10% changing = 10,000 values/sec
- 500,000 tags at 5 second rate with 10% changing = 10,000 values/sec
- 500 devices with 100 tags each = 50,000 tags @ 5 second rate with 100% changing = 10,000 values / sec

As you can see, more tags can be handled by optimizing poll rates & deadbands. It is possible to have a high number of device connections and tags through optimization.

Optimizations

You can really optimize your system by paying close attention to the number of value changes happening every second. The more you tune the system, the more you can handle, and the better performance you will get. Here are some things to consider:

Polling rates

Tag groups (or scan classes) dictate how often Ignition polls data from PLCs. Make sure you are using proper polling rates. Not everything has to be at a 1 second rate. Some values can be polled slower than others since they don't change very often. Try to determine all of the possible polling rates you require. For example, you might need a set of tags at a 1 second rate for alarming or history while the rest of the tags can be polled at 5 second or minute intervals.

Deadbands

Deadbands log fewer data points, particularly when value changes are too small to be significant. They are especially useful in a process that inspects with a high frequency, but is most interested in capturing value changes exceeding a defined magnitude or percentage. Deadbands can be either absolute or percentage-based. Absolute is based on an absolute change in the value whereas percentage deadbands mean the value has to change by a particular percentage. There are also two deadband modes: digital and analog. Digital deadband is stored when a new value is +/- the deadband value away from the previously stored value. Analog mode's values are evaluated with a modified 'Sliding Window' algorithm that tries to log the least amount of points necessary to draw the graph correctly.

There are deadbands on tags for both realtime and historical. Realtime deadbands means Ignition will not process the new value at all unless it changes according to the configuration. Historical deadbands apply after and dictate when a value will get logged to the historian. That allows one to see more granular data on realtime screens but log fewer points.

Without proper deadbands, systems will log 'noise' on analog signals. It never makes sense to have more sensitivity for logging than a sensor's rated precision.

The better you tune your deadbands, the fewer value changes you will have in the system. You can avoid fluttering on analog values and costly historian storage.

Leased and Driven Tag Groups

Tag groups tell Ignition how often to read (poll) the PLC. Tag groups dictate the execution of tags, which is especially important for large and high-performance systems. It is recommended that you organize tag groups by polling strategies. For any given tag it is common to use a fast tag group for status and control, and a slower tag group for history.

Driven tag groups are a great option for history where faster logging is conditionally required. Tag groups do not set your historical logging rate, but influence the frequency at which Ignition will see the data change.

Leased tag groups are a great option when you only want to poll tags when it is needed on a screen. There are a lot of values in PLCs that aren't used for history or alarming, only for viewing on a screen. We don't need to poll those values all of the time.

It is important to note that leased and driven tag groups have an impact when using Ignition's OPC-UA server and drivers as subscriptions change quite often. A change in subscription requires Ignition's device driver to reoptimize per device connection, which re-reads every tag configured on that connection at the moment any leased or driven tag changes between the fast and slow rates. Because of this, when using Ignition's OPC-UA server and drivers with leased or driven tag groups it is recommended to create two device connections to the PLC if allowed. One connection for the direct tag groups and one connection for leased and driven tag groups. That way leased and driven tag group subscription changes don't have an effect on the direct tag groups and avoid costly re-optimizations and stale overlays.

Event-Driven

It is extremely important to only execute logic on change, especially with expressions and SQL queries. This will avoid additional computation when values haven't changed. Ignition 8.0+ introduced new execution modes on tags, of which "Event-Driven" is notable. Event-driven only fires the expression or SQL query when a dependency changes (i.e., tag value event or alarm event) versus running at a specific rate all the time. Imagine having an expression tag in a UDT that performs a switch statement on an integer value to provide a human-readable state. The state doesn't change very often. Let's say you have 2,000 UDT instances resulting in 2,000 expressions. Event-driven will only ever fire the expression when the state changes, which is very infrequent. Alternatively, fixed rates could fire 2,000 expressions every second, which could be a lot of unnecessary computations.

It is recommended to use event-driven on expressions, derived tags, and SQL query tags. It is also recommended to set a tag's history mode to "On Change" as well.

Scripts

You can write scripts in several places within Ignition. It is very important to understand where and when those scripts are running. Avoid runScript expressions unless absolutely necessary. Avoid lots of timer scripts on the Gateway or in the client. Avoid lots of tag change scripts. Ultimately, try to reduce the number of scripts you have on both the server and clients. You can view Ignition's Gateway status page for more information on running scripts and execution engines. Expressions run much faster than scripts, so use expressions instead of scripts wherever possible. Use scripting where it makes sense, as scripting is very powerful and useful, but be mindful of the processing power the scripts you write require.

Avoid Polling Queries and Use Caching

It is really easy to set up polling queries to the historian, alarm journal, audit logs, or custom SQL queries in a client. It is recommended to avoid polling queries as that places additional load on the Ignition server, especially when there are lots of clients open. For example, if you have two polling queries running every second in a client and 25 clients open, you will have 50 queries running every second. Try to reduce the number of polling queries or turn them off altogether. You can easily put a refresh button on the User Interface (UI) that can run the query on-demand. Both Vision and Perspective have an easy to use function to refresh any binding.

Named queries can opt-in to caching the results on the Gateway. This means if another request to the same Named Query comes in, the Gateway can return the cached result instead of having the database executing the query again. This will use more memory on the Gateway (to maintain the results) but could result in fewer queries running against the database. When polling is necessary or if you have a lot of clients open, use named queries with caching to provide better performance.

Tag Historian Stale Data Detection

This Ignition feature, if enabled, tracks tag group executions to determine the difference between unchanging values, and values that are flat due to the system not running. Essentially, it tracks when the system is not running, commonly due to clock drift issues, and records the time when the system wasn’t running. This feature is on by default. Although this can be useful at times, many systems with this feature enabled end up with lots of rows in the sqlth_sce table in the database inadvertently causing performance issues querying historical data. The performance issues are caused by Ignition performing more SQL queries with higher numbers of rows in the table.

Unless absolutely necessary, we recommend turning this feature off and simply letting the system track values as they change. If there are clock drift or system issues, those should be dealt with as well, and treated as separate issues to resolve. To turn off the feature, first, disable the stale data detection setting for your history providers in the Ignition Gateway configuration page under Tags → History. Edit your provider and under Advanced Settings disable “Enable Stale Data Detection”. Next, set the history mode on all of your historical tags to “On Change”. Lastly, check your SQL database to see if you have a high number of rows in the sqlth_sce table with the following query:

```
SELECT COUNT(*) FROM sqlth_sce
```

If you have hundreds of rows, or more, consider consolidating the rows into one row per scan class and rate. You can use this query to find the ideal minimum number of rows:

```
SELECT scid, MIN(start_time) start_time, MAX(end_time) end_time, rate FROM sqlth_sce GROUP BY
scid, rate
```

Vision Client Tag Poll Rate

Vision clients and the Ignition designer have to communicate to the Ignition Gateway in order to get tag values. Of course, you only need to get the data you are interested in seeing on screen. In that case, Vision clients and the designer create a subscription on the tags that they need. However, the Gateway cannot notify the client/designer when a tag changes so the client/designer must poll the Gateway. The Client poll rate (in milliseconds) is the rate at which a Vision client or Ignition designer polls the Gateway for updates to its subscribed tags. By default, the setting is set to 250ms. That means every Vision client and designer will poll the Gateway every 250ms to see if any of the tags they are subscribed to has changed. If nothing has changed, we get an empty result. The rate is quite fast but typically not a problem. However, if you have a high number of Vision clients it can cause additional load on the server. You can easily reduce this load by changing the setting to 1 second. So, if you have 100 clients open, you will get 400 requests per second for tag changes by default. If you change the setting to 1 second, you will get only 100 per second. The setting for this is in the designer under Project → Properties on the Project → General tab.

Gateway Network Optimizations

The Gateway Network is amazing but it can be easy to send lots of data unintentionally. It is important to know what and how much data is going through the network. Ignition provides a diagnostic page in the status section of the Gateway to see incoming and outgoing traffic by the Gateway Network service. It is recommended to optimize the communication to ensure Gateway Network health.

Remote Tag Provider Alarm Subscription

The remote tag provider has the ability to query alarms through the Gateway Network for use in the alarm status component. You can either turn alarming off or set the mode for how you want to retrieve alarms. It is recommended to use the “Subscribed” mode for optimal communication. In 'subscribed', the state will be subscribed, and updates will be sent asynchronously. Subscribed provides better performance but uses more memory. This is especially important with lots of Gateway Network connections and high numbers of alarms.

Remote Tag Provider History Mode

The remote tag provider has the ability to query history through the Gateway Network. It is recommended to connect a frontend Gateway to the SQL database directly and use historical tag paths to query the data. However, if you are using realtime tag paths to query history, it is recommended to set the remote tag provider to use the “Database” mode for history access. That way you can avoid having to ask the I/O server for historical data. Rather, you can short-circuit and query the database directly, requiring a direct connection to the SQL database. If a connection to the SQL database is not available, try to reduce the amount of history queries and avoid costly queries with large time ranges and data.

Running Ignition on Virtual Machines

Summary

One of the main deployment environments today for Ignition is on Virtual Machines (VMs). A large percentage of enterprise customers use Ignition on VMs rather than bare-metal physical servers. Although VMs by design are slower than running on bare metal, as long as the correct VM configuration is properly applied, Ignition should run well in these environments.

Software

Ignition will run on any hypervisor that properly emulates hardware and an Operating System. When Ignition is running on a VM, it is most commonly used on VMWare, mainly because of VMWare's ubiquity. Inductive Automation does not recommend VMWare above other hypervisors, and as long as resource allocation is appropriate, has not seen hypervisor-specific issues in any recent virtualization software

Resource Allocation

CPU and memory allocation should be determined by the engineers creating the Ignition projects. Small systems may be 4 vCPUs with 8GB of RAM. Very large systems could be 32 vCPUs or more with 64+GB RAM.

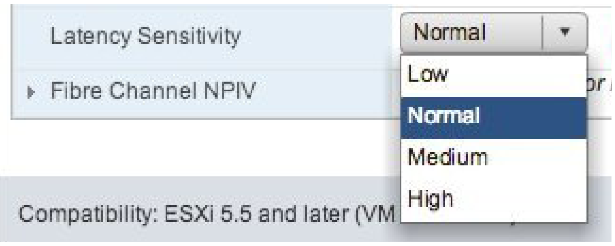
When Ignition is running small systems with only a handful of clients and tags, normal resource allocation is often acceptable. When it is running critical applications or running any kind of significant load, dedicated resources are required for the system to stay performant.

Dedicated Resources

When first specing an Ignition Gateway's needs, always mention the need for dedicated vCPUs and memory. If an Ignition Gateway starts small, it might not be initially needed, but as it grows dedicated resources will be required to avoid issues. If dedicated resources aren't initially configured, there may later be significant pushback from IT departments who weren't anticipating this need.

VMWare Specific Settings

In VMWare, this is most commonly configured by setting Latency Sensitivity to High.



Latency Sensitivity Normal

Fibre Channel NPIV

Compatibility: ESXi 5.5 and later (VM

After configuring this, check to make sure the vCPU allocation is 100% dedicated to the Ignition VM. If it's not, it means the whole VM Host is over-provisioned, and this will need to be fixed.

Common Questions

Q: My CPU utilization isn't too high. Do I still need Dedicated Resources?

A: Yes, it's likely. Unless you're running a system that doesn't do much, your Ignition Gateway is probably handling a lot of burst processing. This means that traffic is coming over the network, threads are waking up asynchronously, or other things are happening where Ignition instantly wants to use CPU cycles. Without dedicated resources, this can take up to 50-500 uS.If there are 1000 operations like this happening per second, which isn't unreasonable on a heavily loaded Ignition system, the wake-up delay could be up to 5%-50% of one vCPU.

Q: How do I know if I need dedicated resources?

A: Inductive Automation normally recommends starting with dedicated resources. Considering how critical Ignition is for most companies, the value it brings, and the investment in the purchase, it doesn't make sense to try to go light on resources.

If you've already started with shared resources, there is one main sign that dedicated resources are becoming necessary.

> **Clock Drift.** This is an Ignition metric on the status pages and is a measure of when the Operating System runs a task that's scheduled. Ignition schedules a task to run in 1s. Under most circumstances, that task will run in 1000ms exactly. If it doesn't, there is a problem with the Operating System getting the CPU cycles it needs to run that task. It's either a CPU allocation problem, a very heavily loaded system, other software affecting the Operating System, or the hypervisor not providing resources or enough resources quickly enough. Dedicated resources almost always clear up Clock Drift. Note that having too few vCPUs allocated can also contribute to Clock Drift.

Delays in common tasks, slow processing, historian issues if the database and devices aren't overloaded, and other sluggish behavior can all be signs that dedicated resources are needed as well.

Q: I already have dedicated resources, and I've allocated a large number of vCPUs, but my Ignition system is still overloaded. What do I do?

A: At some point, a single Ignition server will reach its limit, regardless of CPU and memory allocations. Inductive Automation's scale-out architecture is the general recommendation for moving beyond the limits of a single system.

References

VMWare 5.5 Latency Sensitivity

https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/latency-sensitive-perf-vsphere55-white-paper.pdf

VMWare 6.7 Latency Sensitivity

https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/performance/vsphere-esxi-vcenter-server-67-performance-best-practices.pdf