Ignition Guide



# Ignition!®

# Phased Deployment Methods

inductive automation.

(800) 266-7798

www.inductiveautomation.com

# Phased Deployment Methods

## Overview

The phased deployment strategy outlined in this document emphasizes systematic rollout of system components while utilizing template-driven development and code reusability. By leveraging common design patterns and standardized templates, teams can accelerate deployments while maintaining consistency across each environment. Data centers must frequently deploy additional phases at sites already in production; while this document considers the example use case of data centers, the concepts discussed could be applied in any industry with similar requirements.

## Phased Commissioning Challenges

Before stepping into this, let's first understand some of the challenges of deploying and commissioning a new site.

Teams need to have the ability to:

- Test low-level equipment (sensors, actuators, etc.) off-site.
- Bring those various pieces of equipment online with the primary production system at a later date.
- Easily roll project edits back into the main project.
- Avoid alarms on systems that have not yet been validated with the primary production system.

## Ignition Concepts

To mitigate some of the challenges listed above, teams can employ a few Ignition design and development strategies to their overall solution.

### UDTs (User Defined Types)

Also referred to as Complex Tags, UDTs offer the ability to leverage object-oriented data design principles in Ignition. These are very useful for rapid development/deployment of systems. Teams can easily modify the UDT definition and, due to its inherited nature, the changes get propagated to all the instances in the system.

Example:
UDTs could be created for an Uninterruptible Power Supply (UPS) or an Air Handling Unit (AHU).

## View Templates

Creating views for any project can be a time-consuming process. Ignition's Embedded View and Flex Repeater components enable teams to create a view once, and then reuse that view repeatedly throughout a project. Any changes to the original view are propagated to all of the view instances used within the system.

Example:
Create a parameterized KPI or Current/Voltage widget to re-use throughout the project.

## Project Inheritance

Project Inheritance allows one project to inherit resources from another project. An inheritable project can be created and function as a "library" to share views, scripts, etc., with multiple other projects across the architecture. This continues to extend the concept of using view templates across the larger system, where making changes in a parent project view will propagate to all the instances.

Example:
Create a parameterized KPI or Current/Voltage widget to re-use throughout multiple sites.

## Object-Oriented Approach

Piecing it all together, it's like a traditional object-oriented approach, where UDTs and View templates are used in combination to create larger "objects." For example, a View could be created to visualize the data of a UPS and repeated multiple times on a single view to show the data and statuses of multiple UPSes throughout a data center.

## Ignition Edge

When developing resources for an eventual Ignition Edge deployment, it is beneficial to think ahead to what the Edge deployment will look like out in the field, as well as how it will connect to a standard deployment. Many developers will spin up a new gateway and start development using default configuration. The following are some default configuration options that should be changed before development begins so that rework is not required after deployment.

### Device Connections - Edge

Create devices with names that reflect the actual implementation. For example, rather than creating a device connection named "Test PLC 1," create a device with the name that will be used in the field (i.e., "Hill Top PLC"). Once the transition has been completed, the device connection can be edited so that the hostname of the device utilizes the field device rather than the test device. The name remains the same so that other items that reference that name do not need to be changed.

### Gateway Tag Provider - Standard

Create Tag Providers with names that reflect the actual implementation. For example, rather than creating a Tag Provider connection named “Test Site A,” create a Tag Provider with the name that will be used in the field (i.e., “Hill Top Data Center”). Once the transition has been completed, rename the Standard Tag Provider so that a new Remote Tag Provider can be created with the same name. This allows for the tags to be coming from the spoke (the edge device at that site) rather than from the testing server without having to change any references to those tags on screen bindings or scripts.

### Database Connections - Standard

Create Database Connections with names that reflect the actual implementation. For example, rather than creating a Tag Provider connection named “Testing DB,” create a database connection with the name that will be used on the Hub, in this example “Hill Top Data Center DB.” Like Device Connections, the hostname can be changed but the name should stay the same so that anything referencing that database does not need to be changed.

## Phased Commissioning Approach

Utilizing a phased commissioning approach begins with establishing a solid backbone for the project. This involves creating a structured database and defining a tag structure to organize and reference system components effectively. Once this foundation is in place, metadata, such as disabled or active bits, can be utilized to enhance functionality. This helps filter the status of equipment and distinguish between commissioned and enabled states, while also managing dynamic views for active alarms, navigation, and recurring display elements. Considering your commissioning approach in the project design is key to ensuring a smooth and controlled deployment.

## Typical Architectures

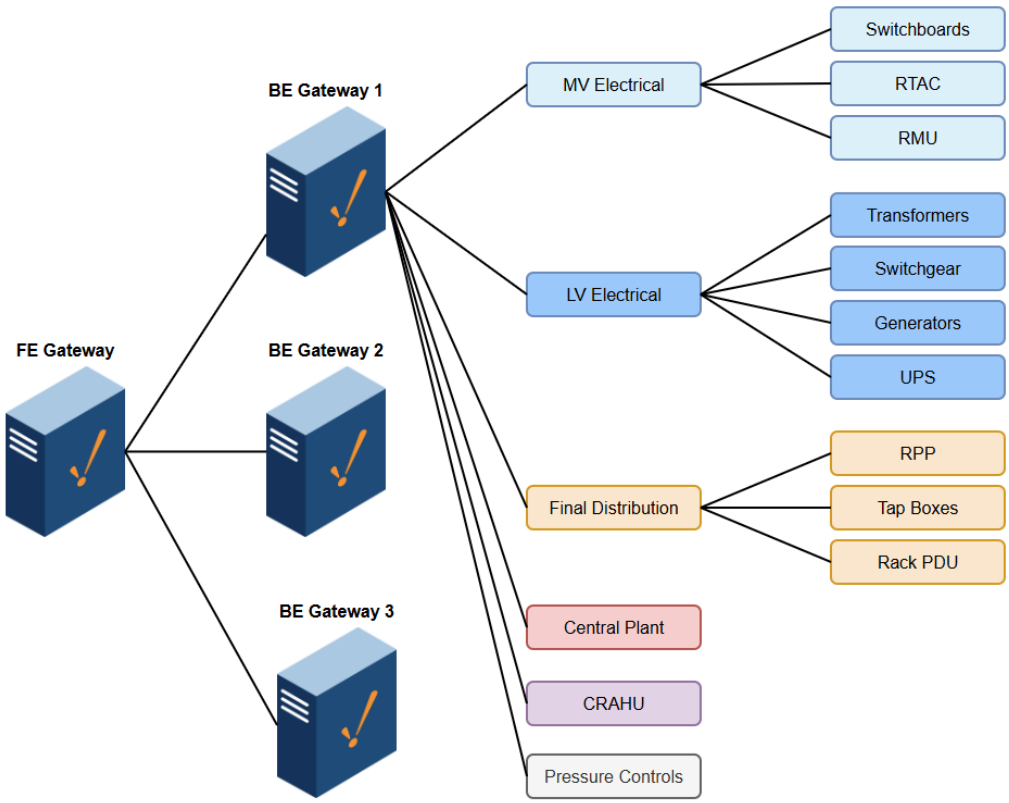
The facility may only need one gateway. However, data centers typically have large amounts of data throughput. Therefore, scale-out architectures are common in order to split the load of a single gateway into data collection (back-end) gateways and visualization (front-end) gateways. Additionally, multiple gateways can be tied together with an enterprise-type architecture that allows multiple sites to be monitored simultaneously from one central location.

To review some of the common architectures, please refer to the following resources:

- Architectures
- Server Sizing and Architecture Guide

## Example Factory Acceptance Test (FAT) Process

- The initial phase would include commissioning the systems connected to back-end (BE) Gateway 1.
- During this initial commissioning phase, a temporary front-end (FE) gateway that has the same project as the production FE gateway can be used to provide local controls without connecting the BE gateway online to the production FE gateway.
    - As an alternative, a separate temporary “commissioning” project could be used on the production FE gateway if connection can be established.
- Once commissioning is complete, the template project can be updated on the production FE gateway with any updates made during commissioning.
    - Additionally, update the equipment records in the database as having passed commissioning.
- Repeat this process for N phases.



# Additional Resources

## Webinars/Videos

- Developing HMIs
- Testing Tips
- Deployment with Kubernetes

## Case Studies

- Smart Factory Journey
- Scaling Ignition
- Dynamic UI Based On Database Configuration