White Paper: Building Industrial Data Systems You Actually Own

Open Source, Open Standards, and Open Architecture in Industrial Automation

OPTO 22

Your Edge in Automation.™

Opto 22

43044 Business Park Drive • Temecula • CA 92590-3614
Phone: 800-321-6786 or 951-695-3000
Pre-sales Engineering is free.
Product Support is free.
www.opto22.com

Form 2456-260519

WHITE PAPER: BUILDING INDUSTRIAL DATA SYSTEMS YOU ACTUALLY OWN

Open Source, Open Standards, and Open Architecture in Industrial Automation

INTRODUCTION

You want better access to your machine data. Maybe you need to connect legacy PLCs to modern systems. Maybe you want to reduce software licensing costs. Maybe you are tired of a single vendor dictating the boundaries of your plant's architecture.

The reality is that most automation systems were built for a specific purpose, not for integration. PLCs control machines well. SCADA systems display data well. Historians store data well. But when you try to connect everything across lines, sites, or business systems, friction builds quickly. You add one machine and trigger a new license requirement. You change one PLC and break a data link upstream. Your data lives in systems that do not communicate with each other. And scaling to another site means rebuilding the same vertical structure from scratch.

Each system works well in isolation. Together, they create rigidity.

Enterprise IT teams solved this class of problem years ago. They build from modular components that share data through open protocols. They separate concerns cleanly. They scale without rebuilding from scratch. Industrial systems have historically not followed that model, but the gap is closing.

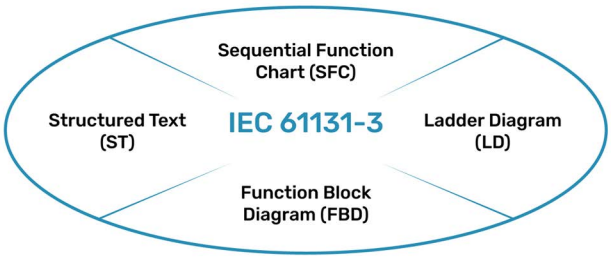
This paper explores a practical question: can you design an industrial data architecture that scales cleanly, reduces vendor lock-in, and avoids repeated licensing costs—without replacing the control systems that already work?

WHAT "OPEN SOURCE" MEANS IN INDUSTRIAL AUTOMATION

When people say "open," they often mean different things. In this paper, *open source* means software you can download for free, run without paying a runtime license, and inspect at the source code level. It does not mean there is no cost, no engineering effort, or no need for support. Free software removes license fees. It does not remove responsibility.

Let's also define three related but distinct ideas:

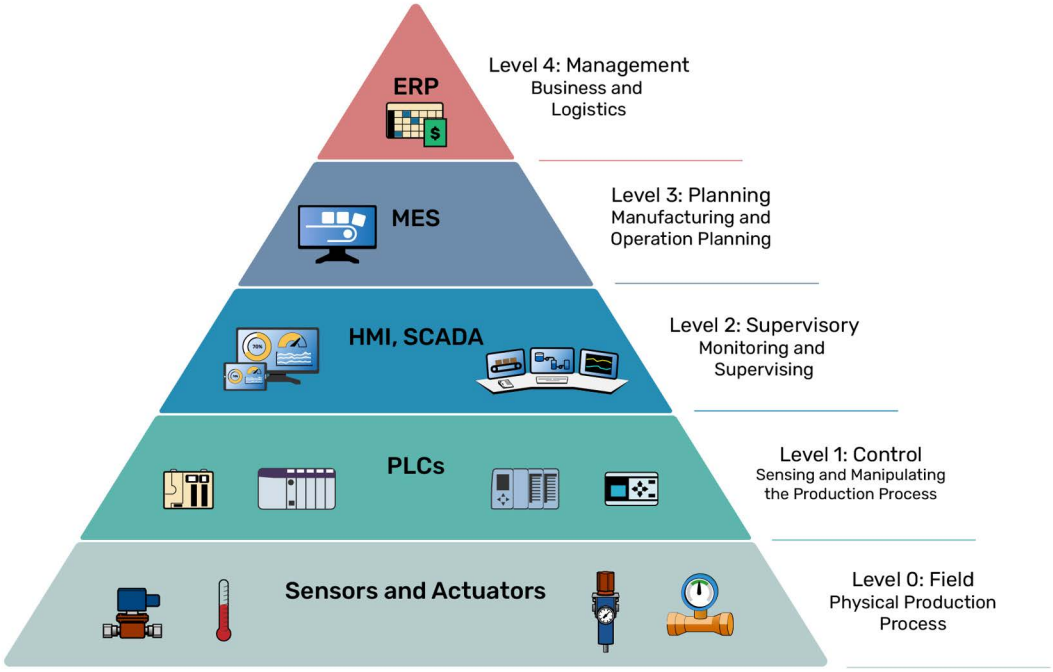
- **Open source** means the source code is public and you can run and modify it under its license.
- **Open standards** means the protocol or format is publicly defined and anyone can implement it. Examples include IEC 61131-3, MQTT, Sparkplug®, OPC UA®, Modbus®/TCP, HTTPS, and REST.
- **Open architecture** means the system uses standard interfaces so you can add or replace components without rebuilding everything. A 4–20 mA sensor works with any controller that accepts that signal, so you're not locked to one vendor. If a controller exposes a REST API (application programming interface), any system that can make an HTTP call can read data from it.



IEC 61131-3 is an *open standard* for PLC control programming languages.

Open source, *open standards*, and *open architecture* are not the same thing. You can build an open architecture using commercial software, and you can build a closed architecture using open-source tools. What matters is how the parts connect.

In industrial systems, you care about keeping the line running, securely accessing your raw data without a vendor tool, keeping the system working ten or twenty years from now, and making changes without breaking something else. The question is whether open tools can operate within those constraints.



DEFINING (AND REDEFINING) THE STACK

The traditional automation stack (shown above) developed over decades. It reflects how systems were built when network bandwidth was limited, integration tools were scarce, and each layer addressed a specific problem.

At the lowest level, sensors and actuators connect to the process. As you go up the layers, PLCs read inputs and execute control logic. HMI and SCADA systems display data and provide operators with control interfaces. Historians store data over the long term. Higher-level systems—like MES (manufacturing execution systems), ERP (enterprise resource planning), and OEE (overall equipment effectiveness) platforms—consume historian data for reporting and business use. Data moves upward, one layer at a time.

If you need a temperature reading at the enterprise level, it typically passes through the PLC, then SCADA, then a historian before reaching any reporting or analytics system.

What changes is how data moves after it leaves the controller. Data no longer climbs the stack one layer at a time. It becomes shared infrastructure that multiple systems can access independently.

That model made sense when systems were isolated and integrations were limited. Each layer did its job, and data moved through fixed paths.

But this structure creates friction over time. Each layer can reshape data, require its own protocol drivers, add licensing costs, and create dependencies on the layer below it. The stack becomes rigid. Changing one layer risks affecting others. Scaling to another site often means rebuilding the same vertical structure.

Modern Messaging Changes the Layers

Publish-subscribe protocols like MQTT remove the requirement for data to move strictly upward through fixed layers. Instead of pushing data step by step, a controller or edge device publishes data once to a broker. Other systems subscribe directly to what they need.

The model shifts from vertical to shared:

Traditional:

Sensors → PLC → SCADA → Historian → Enterprise

Decoupled:

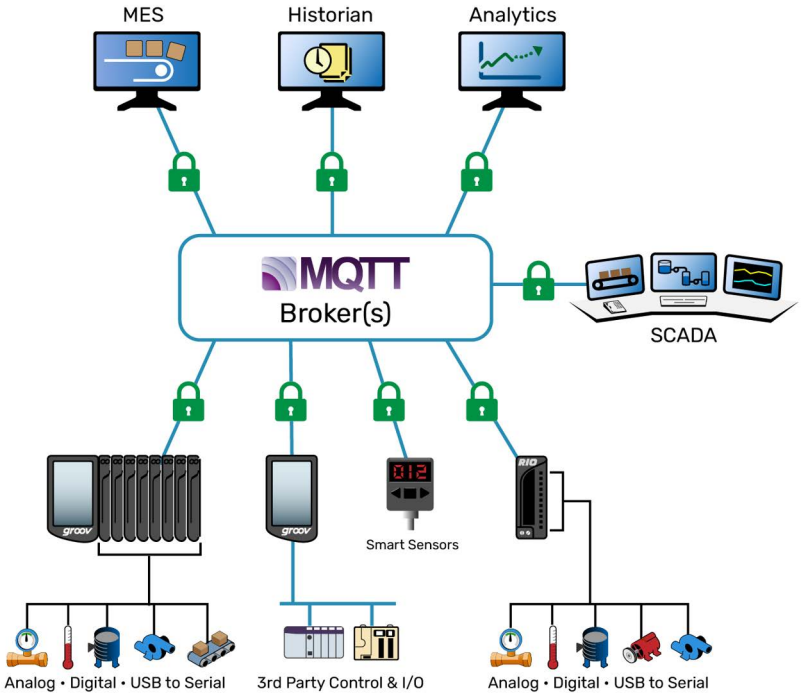
Sensors → PLC/ Edge → Messaging backbone (broker) → Multiple subscribers (SCADA, Historian, Enterprise)

Control stays where it is. What changes is how data moves after it leaves the controller. Data no longer climbs the stack one layer at a time. It becomes shared infrastructure that multiple systems can access independently.

That shift makes it possible to modernize or replace individual layers without rebuilding the entire system.

- You can keep your PLC and control logic in place but replace a proprietary historian with an open-source database.
- You can build a separate dashboard that reads from the same data stream instead of modifying your production SCADA project.
- You can connect analytics directly to published data without inserting another middleware layer.

Because systems subscribe to shared data instead of depending on each other, you can change one layer without forcing changes in the others.



With a publish-subscribe protocol like MQTT, a controller or edge device publishes data once to a broker. Other systems subscribe directly to the data they need.

Open-Source Tools Across the Stack

Once data is no longer forced to move layer by layer, you can evaluate each part of the stack independently. You do not replace everything.

You modernize selectively.

Sensors and Field Devices

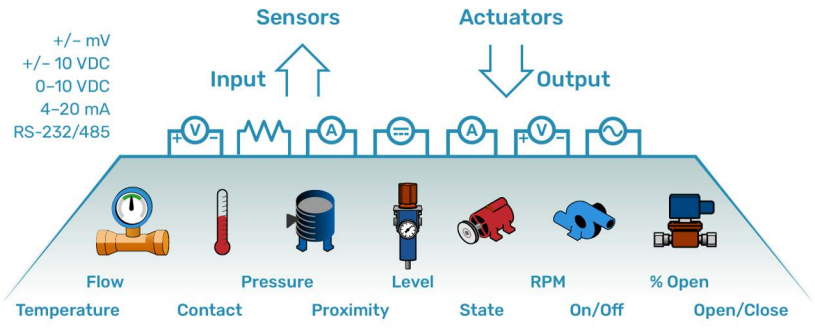
At the bottom of the stack are physical signals. Most industrial processes rely on standard electrical interfaces: 4–20 mA, 0–10 VDC, thermocouples, RTDs, thermistors, discrete I/O, and dry contacts.

Actuators are typically driven through discrete outputs, analog outputs, relays, and standard motor control interfaces.

These signals are not proprietary. They can be read or driven by any PLC or capable edge device.

While controller ecosystems may be vendor-specific, the field layer is largely standardized. That gives you flexibility at the hardware boundary.

Open architecture often starts here, using standard electrical interfaces that any compliant controller or edge device can support.



Open architecture often starts at the bottom of the stack, with standard electrical interfaces.

Control

The control layer handles deterministic tasks: scanning inputs and outputs, running control loops, managing interlocks, and executing sequences. In most plants, this role is filled by established PLC platforms. These systems are proven, reliable, and deeply integrated into maintenance workflows.

The question is not whether PLCs work. They do. The issue is how tightly they bind you to a specific ecosystem. Many traditional PLC platforms rely on vendor-developed protocols such as EtherNet/IP® or PROFINET®, proprietary tag structures and configuration formats, development software that requires paid licenses, per-tag or per-node communication fees, version-locked project files, and, in some cases, additional gateway hardware for integration outside the stack.

These systems are reliable for control, but they tie data access and integration to the vendor's tools and licensing model.

An Alternative: Open Control Platforms

An open control layer still performs the same tasks as a traditional, proprietary PLC ecosystem, but how you develop, extend, and integrate around that control changes. Integration options are no longer limited to specific drivers, protocols, or licensed software.

An open control platform takes a different approach:

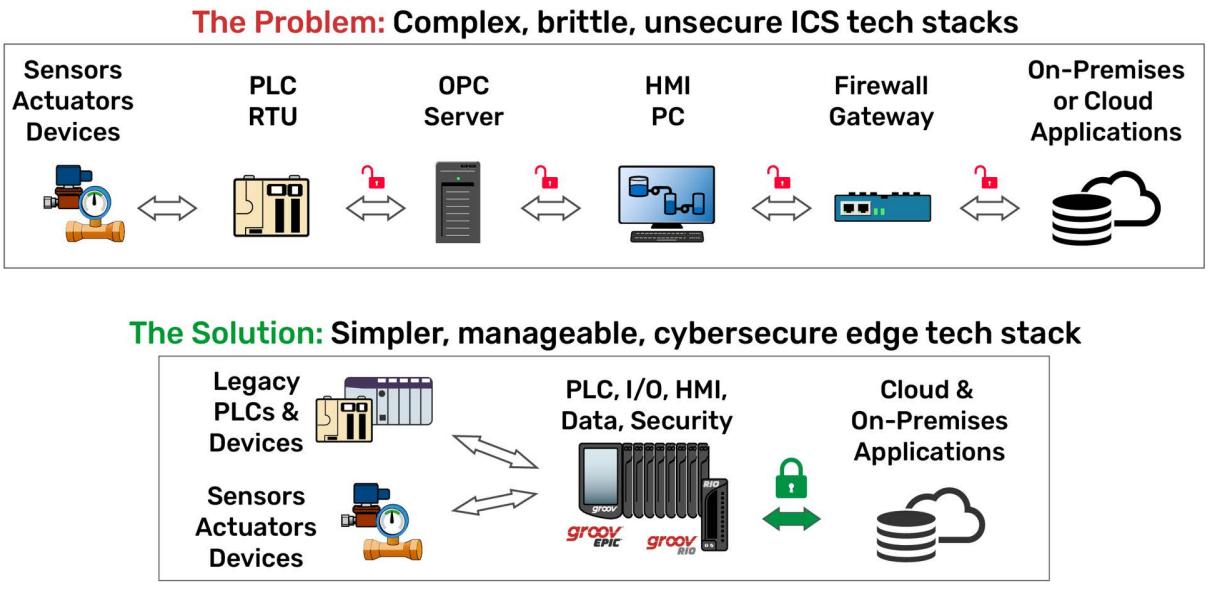
- PLC programming inside an IEC 61131-3 compliant integrated development environment (IDE) such as CODESYS®
- Linux®-based systems with optional secure shell (SSH) access
- Development in common computer languages like C, C++, Python™, or JavaScript®
- Standard APIs and open protocol support for accessing control data
- Version control using tools like Git

The controller remains responsible for real-time control. The difference is that you are no longer forced to operate inside a closed development ecosystem.

The Edge: Expanding the Role of Control

In the traditional stack, the PLC already lived at the edge. It connected to field devices and passed data upward. When additional integration was needed, plants added gateways, OPC servers, or vendor middleware.

What has changed is capability. Modern edge devices are not just protocol bridges. They can provide local processing and real-time control, secure network segmentation with firewalls and VPNs, built-in support for open protocols like MQTT Sparkplug, OPC UA, and RESTful



Modern edge devices can make a complex, unsecure industrial control system (ICS) simpler and more cybersecure.

APIs, and services to model and contextualize data before publishing.

In some architectures, this capability sits alongside the PLC. In others, modern edge programmable industrial controllers (EPICs) allow the controller itself to perform all of these roles. A modern edge controller can execute IEC 61131-3 control logic while also running tools like Node-RED or Linux services, and publishing data using MQTT, OPC UA, and REST. The PLC functionality becomes one responsibility of the edge device, not its only purpose.

Data Movement: Building a Shared Data Backbone

Once data leaves the controller or edge device, the question becomes how it moves. In the traditional stack, data moves upward through tightly coupled, individually configured poll/response connections. A PLC feeds SCADA. SCADA feeds a historian. The historian feeds higher-level systems, like MES and ERP.

That approach works—many plants run today because of it—but working is not the same as optimal. The traditional model served its era well. Today, the tools exist to do better, and the cost of not adopting them compounds over time as integration demands increase.

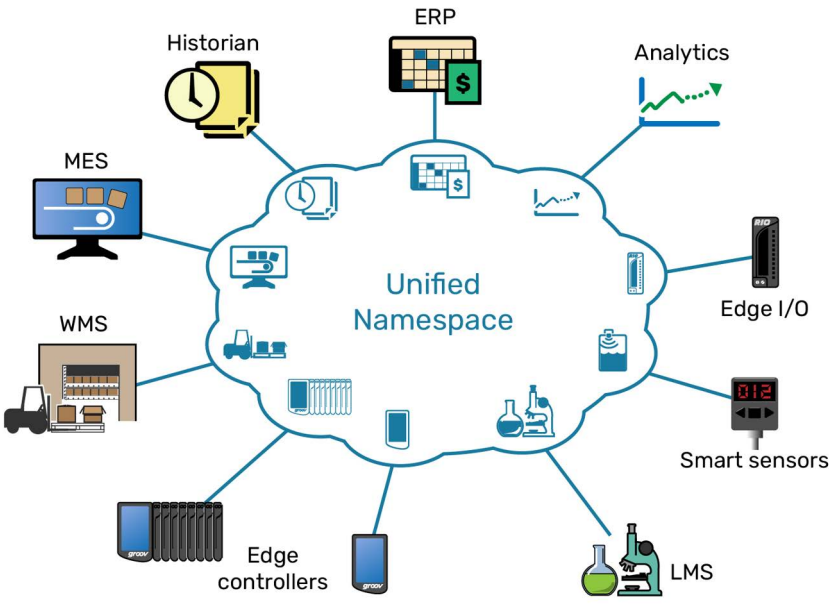
Adopting a new architecture often follows a J-curve. There is initial friction as teams learn new tools and legacy habits must change. There may be a temporary dip in comfort or productivity. But once the new model is in place, the gains compound. Flexibility increases, costs stabilize, integration becomes easier. The challenge is not technical; it's the willingness to commit to the transition.

In a publish-subscribe model, a controller or edge device publishes data once to a broker. Any authorized system subscribes to what it needs. Open-source brokers commonly used in this layer include Eclipse Mosquitto® (lightweight, stable, widely deployed), EMQX® Open Source Edition (high performance with clustering support), and HiveMQ® Community Edition (popular commercial vendor with a free edition).

These brokers are free to download and run. They do not require per-tag licenses. They do not care which vendor produced the data. When structured correctly, this messaging backbone becomes what many call a Unified Namespace (UNS).

A UNS is not a product. It's an architectural pattern. All devices publish structured data to a central broker. Topic hierarchies represent equipment, lines, or sites. Any application subscribes to the data it needs, and no application depends directly on another application.

The result is decoupling. Storage does not depend on SCADA. Dashboards do not depend on historians. Analytics do not require new drivers. You publish once and subscribe anywhere.



A Unified Namespace (UNS) is not a product; it's an architectural pattern. All devices publish structured data to a central broker.

Storage: Replacing the Proprietary Historian

Once data is available on a shared backbone, storage no longer needs to sit directly behind SCADA. Traditional historians often require licensed connectors, per-tag pricing, and vendor-specific tools. They work, but access and scale come at a cost.

Open-source databases provide a practical alternative. Widely used options include InfluxDB® OSS, TimescaleDB®, PostgreSQL®, MariaDB®, and MySQL® Community.

These tools are free to download and run, support standard SQL, and integrate directly with dashboards and analytics platforms without per-tag charges.

With an open architecture, visualization no longer needs to be built inside a SCADA system. Dashboards and HMIs subscribe to shared data.

This layer is often the safest place to modernize first. You can subscribe to your messaging backbone, store data in an open database, and evaluate performance—all without touching your control system.

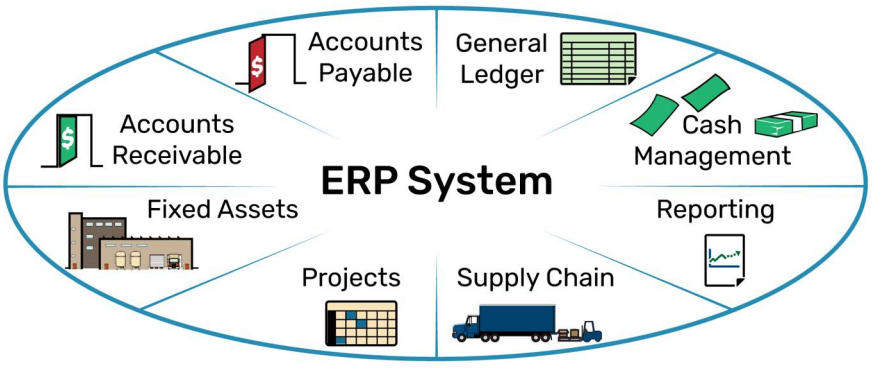
Visualization and Dashboards

Once your data lives in an open database or is available through a messaging backbone, visualization no longer needs to be confined to your SCADA system. Traditionally, dashboards and HMIs are built inside a licensed SCADA platform. Adding users, screens, or remote access often requires additional licensing and modifications inside a production environment.

An open architecture separates visualization from control. Grafana®, for example, connects directly to databases like PostgreSQL, TimescaleDB, or InfluxDB using standard connectors. No proprietary drivers. No per-tag licenses.

The open HMI or dashboard changes how you deploy visibility. You can build dashboards without modifying your SCADA project; provide read-only access to operations, maintenance, or management; create site-level or enterprise-level views; and deploy new dashboards without touching control logic. SCADA continues to handle real-time operator control. Open dashboards handle trending, reporting, and analytics. Visualization becomes another subscriber to shared data.

Applications and Business Systems



In an open architecture built on a shared data backbone, ERP, MES, OEE, cloud storage, and other applications connect as needed.

At the top of the stack are the systems that consume operational data: MES, ERP, OEE platforms, maintenance systems, reporting tools, and cloud storage. In the traditional model, these depend on SCADA exports, historian queries, or vendor-specific connectors. Each integration is custom and tightly coupled.

In an open architecture built on a shared data backbone, these systems become another subscriber. They can query an open

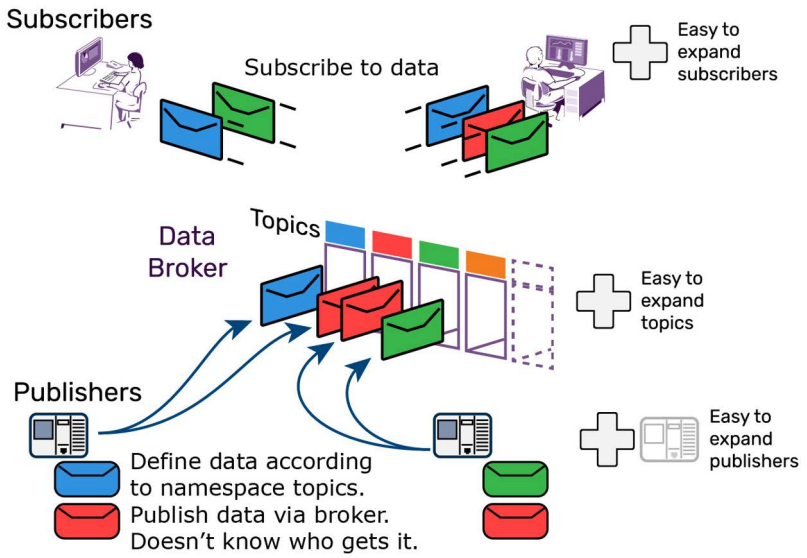
database using standard SQL, subscribe to MQTT topics, or access data through REST APIs.

No proprietary middleware. No per-tag drivers. You build the data infrastructure once, and applications connect as needed. Adding a new reporting tool does not require modifying SCADA. Connecting a new enterprise system does not require a new licensed gateway.

The top of the stack becomes modular instead of custom-built.

SECURITY IN AN OPEN ARCHITECTURE

A common concern with open architectures is security. If systems are more connected and data flows more freely, does that increase risk?



Open architectures built around a shared data backbone make it easier to scale the system by adding edge publishers, data topics, and data subscribers.

Open architectures do not inherently sacrifice security, but they do require deliberate design. Security is not a product you install. It is a set of practices applied at every layer. In a well-designed open architecture, for example:

- Data transport is encrypted using TLS.
- MQTT brokers support authentication and access control lists that restrict which clients can publish or subscribe to specific topics.
- Edge devices can enforce network segmentation, acting as firewalls between the plant network and upstream systems.
- RESTful APIs and OPC UA connections support certificate-based authentication.
- Role-based access controls limit who can view, modify, or configure each layer.

Modern edge controllers often include built-in security tools such as firewalls, VPN servers, and user management—capabilities that were once reserved for dedicated IT infrastructure. When the edge device itself enforces network boundaries, you gain segmentation without adding standalone appliances.

The architectural principle matters here: in a publish-subscribe model, consuming systems never connect directly to control devices. The broker mediates all communication. That separation provides a natural security boundary that is harder to achieve in traditional architectures where SCADA or business systems poll controllers directly.

Open does not mean exposed. It means transparent, auditable, and built on standards that the broader security community actively tests and improves.

REAL-WORLD PROOF

The following case studies illustrate how different companies have adopted open architectures built around shared data backbones. These examples use Opto 22 *groov* edge hardware, but the architectural principles—publish-subscribe data movement, open protocols, standard APIs, decoupled layers—apply regardless of the specific platform. What matters is the pattern, not the product.

Open architectures do not inherently sacrifice security, but they do require deliberate design, with a set of security practices applied at every layer.

ALTA Refrigeration—Open Edge Architecture at Scale

ALTA standardized its EXPERT refrigeration systems on Opto 22 *groov* EPIC controllers. They wrote primary control logic in C++ over SSH (secure shell), used a company SDK (software development kit) for I/O control, exposed data through REST APIs and Modbus/TCP, served their own HTML/JavaScript interfaces, and moved site data upstream using MQTT with TLS.

Each unit publishes structured data. Sites communicate securely to a central system. No per-tag historian licensing; no SCADA runtime per machine. Their nationwide service model depends on open interfaces and publish-subscribe data movement. The architecture scales because the data layer is decoupled from control.

Read the Alta Refrigeration case study.

Gurtler Industries—OEM Software Built on Open SDKs

Gurtler needed advanced data collection and visualization for their dispensing systems. Instead of buying into a proprietary PLC development ecosystem, they integrated both Opto 22 legacy SNAP PAC controllers and modern *groov* EPIC edge controllers directly using a free .NET SDK, built their application layer in .NET, logged and accessed data via standard SQL systems, and implemented remote monitoring without per-client runtime fees.

They retained ownership of their control logic and integration layer. Their product differentiation lives in software they control, not in licenses they rent.

Read the Gurtler Industries case study.

Costa Farms®—Edge Overlay and Cloud Integration Using Open Protocols

Costa Farms needed production visibility across mixed OEM machinery. They deployed configurable Opto 22 *groov* RIO edge I/O devices as a standardized data layer. They captured machine signals via analog and digital I/O, used Node-RED at the edge for logic and data shaping, published data using MQTT, wrote data upstream via SQL and HTTPS, and integrated with Microsoft Azure® and Power BI®.

Instead of negotiating for PLC software access, they built a parallel open data layer. Each machine became a data publisher, and applications subscribed upstream. This is open architecture without removing existing control.

Read the Costa Farms case study.

American Metal Processing—Control, SQL, and REST Under One Architecture

AMP modernized aging furnace systems using Opto 22 *groov* EPIC edge controllers. They rebuilt control logic in an open IDE (integrated development environment), created operator interfaces in a free web-based HMI, logged process data directly to an internal SQL database, and used Node-RED and REST API calls for work order and recipe integration. No standalone proprietary historians. Control, visualization, data logging, and enterprise integration all run on open interfaces. They kept source access, database ownership, and avoided paying an integrator for the keys to their own system.

Read the American Metal Processing case study.

Moriroku® Technology North America—Unified Namespace at Scale

MTNA's Vision 2030 initiative required unifying more than 100 injection molding machines across multiple plants. They deployed an inexpensive *groov* RIO edge device on each machine, captured critical signals (including full-auto status, mold open/close, and water flow), published structured data using MQTT Sparkplug B, routed all machine data into a centralized broker, and fed data into Inductive Automation®'s Ignition® running in Microsoft

Azure to build a Unified Namespace as the enterprise data backbone.

Each machine publishes once. MES, QMS (quality management system), and other systems subscribe. No polling chains. No vertical stack dependency. No per-machine SCADA rebuild. They scaled from one pilot machine to over 100 in just four months using a repeatable commissioning model. The UNS allowed new applications to plug into live, contextualized data without modifying control systems. This is open architecture implemented intentionally.

Read the Moriroku case study.

A FINAL WORD: OWNERSHIP

At its core, this white paper is about control beyond the PLC. Not machine control. Architectural control.

When your data requires an expensive vendor tool to access it, you do not fully own it. When adding a machine requires negotiating new licenses, you do not control your expansion. When changing one layer risks breaking three others, you do not control your system.

Open source and open standards shift that balance. You stop building around what a vendor allows and start building around what your operation needs.

This change does not require replacing every PLC tomorrow. It requires making different decisions going forward.

For some operations, the first step is pulling machine data directly into MQTT brokers, databases, or analytics tools instead of routing everything through vendor-specific software. For others, it may mean telling machine builders and integrators that future equipment must support open protocols and direct data access.

The goal is to stop expanding closed ecosystems and start building systems your operation can adapt and control over time. Every new machine can either extend dependency or increase independence. Every new integration can either tighten coupling or reduce it. Ownership compounds.

The technology to build systems you control already exists. The only remaining question is whether you are ready to use it.

The challenge is not technical; it's the willingness to commit to the transition.

Ready to talk architecture? Opto 22's engineers can help walk you through how open edge controllers, publish-subscribe data backbones, and open-source tools can apply to your specific environment.

Talk to an Opto 22 engineer.

ABOUT OPTO 22

Opto 22 was started in 1974 by a co-inventor of the solid-state relay (SSR), who discovered a way to make SSRs more reliable.

Opto 22 has consistently built products on open standards rather than on proprietary technologies. The company developed the red-white-yellow-black color-coding system for input/output (I/O) modules and the open Optomux® protocol, and pioneered Ethernet-based I/O.

Famous worldwide for its reliable industrial I/O, the company in 2018 introduced *groov* EPIC® (edge programmable industrial controller). EPIC has an open-source Linux® OS and provides connectivity to PLCs, software, and online services, plus data handling and visualization, in addition to real-time control.

groov RIO Ethernet-based edge I/O modules, introduced in 2020, include I/O and IIoT software in a compact industrial package that goes anywhere.

All Opto 22 products are manufactured and supported in the U.S.A. Most solid-state SSRs and I/O modules are guaranteed for life.

The company is especially trusted for its continuing policy of providing free product support, free online training, and free pre-sales engineering assistance.

For more information, visit opto22.com or contact **Opto 22 Pre-Sales Engineering**:

Phone: **800-321-6786** (toll-free in the U.S. and Canada) or **951-695-3000**
Email: systemseng@opto22.com

OPTO 22 www.opto22.com
43044 Business Park Dr. Temecula, CA 92590-3614

SALES sales@opto22.com
800-321-6786 • 1+951-695-3000

SUPPORT support@opto22.com
800-835-6786 • 1+951-695-3080