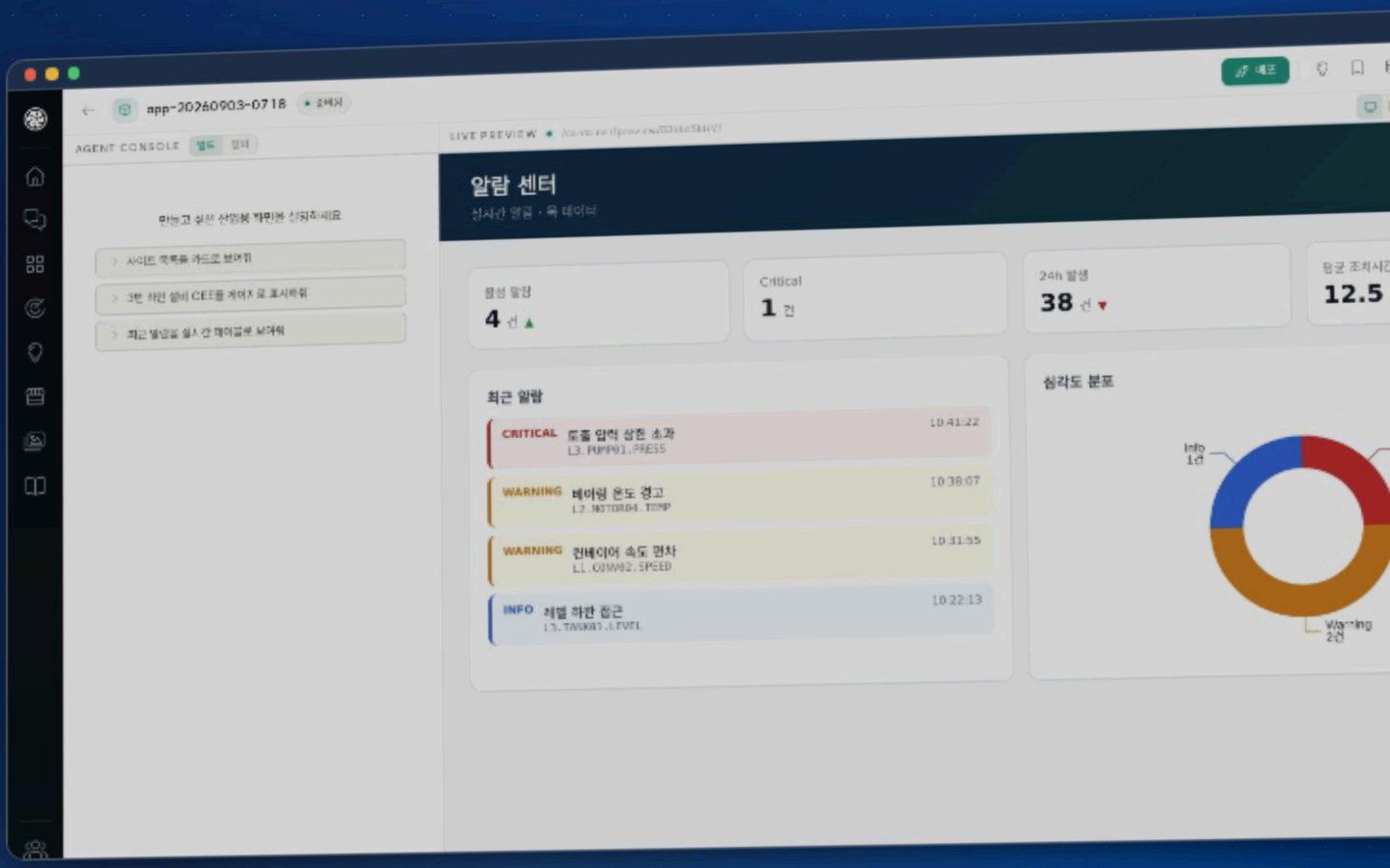
PlantPulse® **Studio**

INDUSTRIAL APP BUILDER

PRODUCT OVERVIEW & PROPOSAL

The screens the floor needs, built by the floor itself.

An **industrial AI app builder**: describe a dashboard, alarm or analytics screen in one chat sentence, connect it to live plant data and deploy it on the spot.



1 sentence
natural language → real app

6
built-in templates

3 modes
plan · auto · accept

September 2026

Industrial AI app builder · On-premises / air-gapped

kopens®

CONTENTS

Contents

01	Why now — the floor's problem	3
02	What PlantPulse Studio is	4
03	What you can build	5
04	What gets built (real generated apps)	6–7
05	Key strengths ①②③	8–10
06	Key strengths ④⑤⑥	11–12
07	Architecture — topology	13
08	Technology stack	14
09	Builder engine · semantic layer	15
10	How it is used (live screens)	16–18
11	Security · governance	19
12	Deployment · delivery model	20
13	References · verification	21
14	Deployment plan and schedule	22

One-page summary

> When a plant needs “one more screen”, the path used to be **request form → contractor → weeks**. PlantPulse Studio replaces it with **one sentence → minutes**.

One sentence
Describe it in plain language and a real React app is built

Live data
Connects to the plant API automatically names → tags resolved for you

Air-gapped
On-premises · air-gapped install signed, verified image bundles

THE KEY DIFFERENCE

It is not a drag-and-drop dashboard tool. An AI agent writes and edits real source code (React · TypeScript), runs type checks and visual verification itself and fixes what it finds. The result is **source the customer owns**, delivered through Git — the asset stays even if the tool goes.

What this document covers

What the product is and what makes it different

Pages 3–12. The floor's problem, product definition, real generated apps, six key strengths.

How it is built

Pages 13–15. Topology, technology stack, builder engine and semantic layer.

How it is actually used

Pages 16–18. Five steps captured from a running instance.

What deployment looks like

Pages 19–22. Security, delivery options, references, schedule.

This document is **product and technical evaluation material**. Industry case studies and quantified outcomes are provided separately through your sales contact.

01 · Background

Why now — the floor's problem

Equipment data is already piling up. The problem is **not the data but the screens**. The one screen the floor needs today always arrives weeks later.

PROBLEM 1

Weeks from request to screen

One line — “I just want to see line 3 pressure on its own” — goes through spec → quote → development → acceptance. By then the floor's attention has moved to another machine.

PROBLEM 2

Requirements distort in transit

The people who know the floor cannot code; the people who code do not know the floor. While documents bridge that gap, “what was wanted” and “what was delivered” drift apart.

PROBLEM 3

Even small fixes go to a contractor

Changing one colour or one threshold needs an owner and a schedule. So the screen sets as it was built, and nobody fixes it.

PROBLEM 4

The limits of off-the-shelf dashboards

They cannot step outside their widget set. The moment a process-specific calculation, screen or interaction is needed, it becomes a development project again.

So what is needed

> A tool where the person on the floor **describes it in their own words**, gets **a working screen** on the spot, and **fixes it on the spot** if it is not right.

With one condition: it has to work in an industrial plant. Closed networks that cloud SaaS cannot enter, the authentication and data conventions of the existing plant system, and the verification question — “can we trust what the AI built?” — all have to be solved together. PlantPulse Studio was designed with those three as givens.

02 · Introduction

What PlantPulse Studio is

A product for building, deploying and operating industrial apps through chat. Describe the screen you want in one sentence; an AI agent writes the actual project source, runs it in a container to show you immediately, and deploys it as-is once you approve.

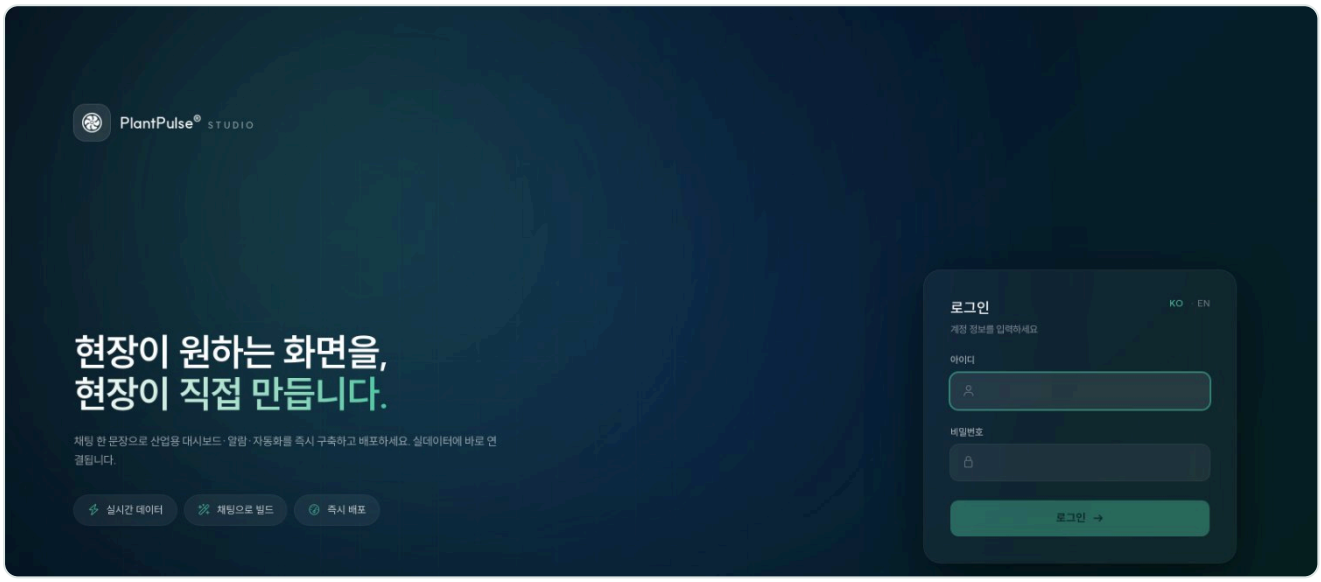


Figure 1. Login screen — white-label structure where the customer's logo and product name can be swapped in. (Captured from a live instance.)

INPUT
One sentence
"Build a live temperature and pressure dashboard for line 3. Turn red above the threshold."

PROCESSING
The agent writes code
It edits a real React · TypeScript project and corrects itself through type checks and visual verification.

OUTPUT
A working app
Check it in the live preview at once; one click records a version and deploys it.

How it differs from the old way

	Off-the-shelf dashboard tool	PlantPulse Studio
How it is built	Widget placement · settings	Natural language → generated source code
Expressive limit	Within the supplied widget set	Anything code can do
Ownership of the result	Settings inside the tool	Customer-owned source (delivered via Git)
Changes	Reconfigure, or a contractor	Ask in chat → applied at once
Data connection	Manual mapping	Names → tags resolved automatically

03 · Use cases

What you can build

Start from a blank screen, or pick one of the shapes the floor uses most as a template. Either way, what gets built is **a real, editable app**.

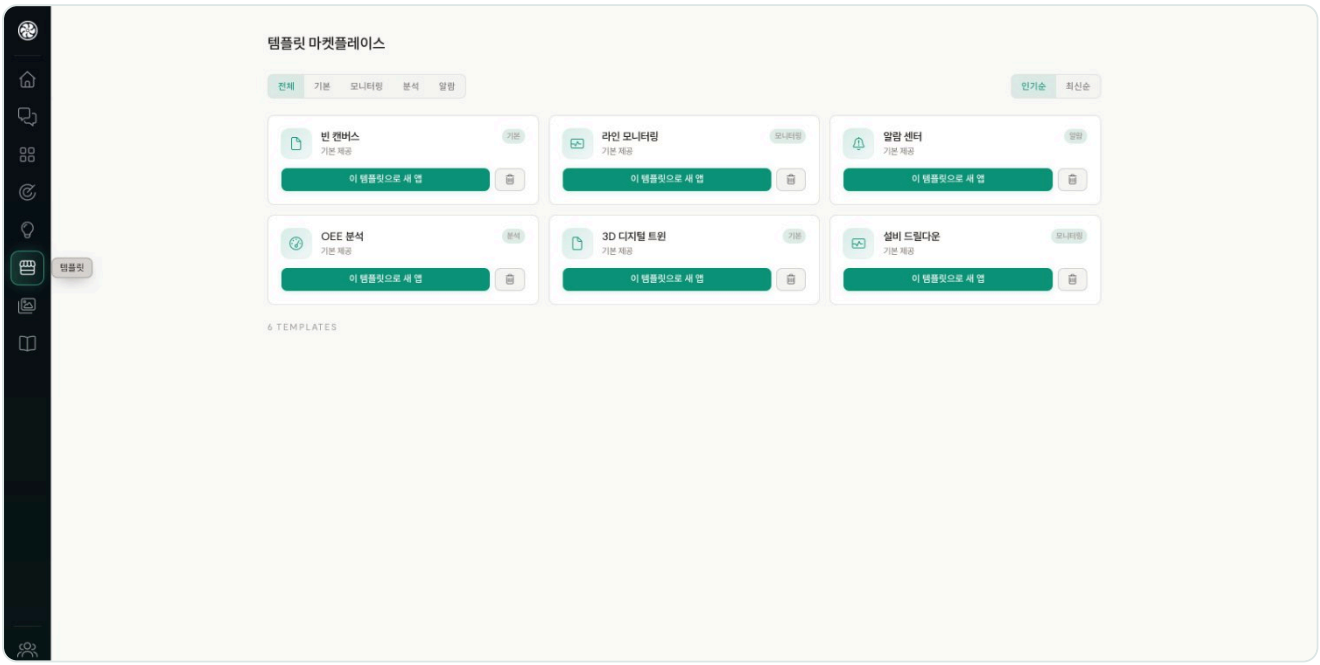


Figure 2. Template marketplace — line monitoring, alarm centre, OEE analytics, 3D digital twin and equipment drill-down come built in, and screens built in-house can be added as company templates.

Typical scenarios

Live line monitoring

Current values and threshold state per line and machine on one screen. Used as-is for shift handover.

Alarm centre

Severity filters, recent alarm list, occurrence trend, with time-to-action KPIs alongside.

Equipment OEE · productivity analytics

Availability, performance and quality combined per machine. Period comparison and underperformer identification.

Equipment drill-down

Explore along the asset tree from plant → line → machine → tag.

Quality · process reports

Daily, weekly and monthly aggregates. A print layout can be requested too.

Small floor utilities

Inspection checklists, shift logs, quick calculators — the things too small to outsource.

> **“Screens that exist only in our process” fit best.** The more unique the formula or layout — the ones off-the-shelf tools cannot cover — the bigger the advantage of this approach.

03 · Use cases

What gets built ① — dashboards

Everything below is **a real app that was generated and captured** — not a mock-up.

Live monitoring

KPI tiles + trend charts + status list. The shape most often used for shift handover.

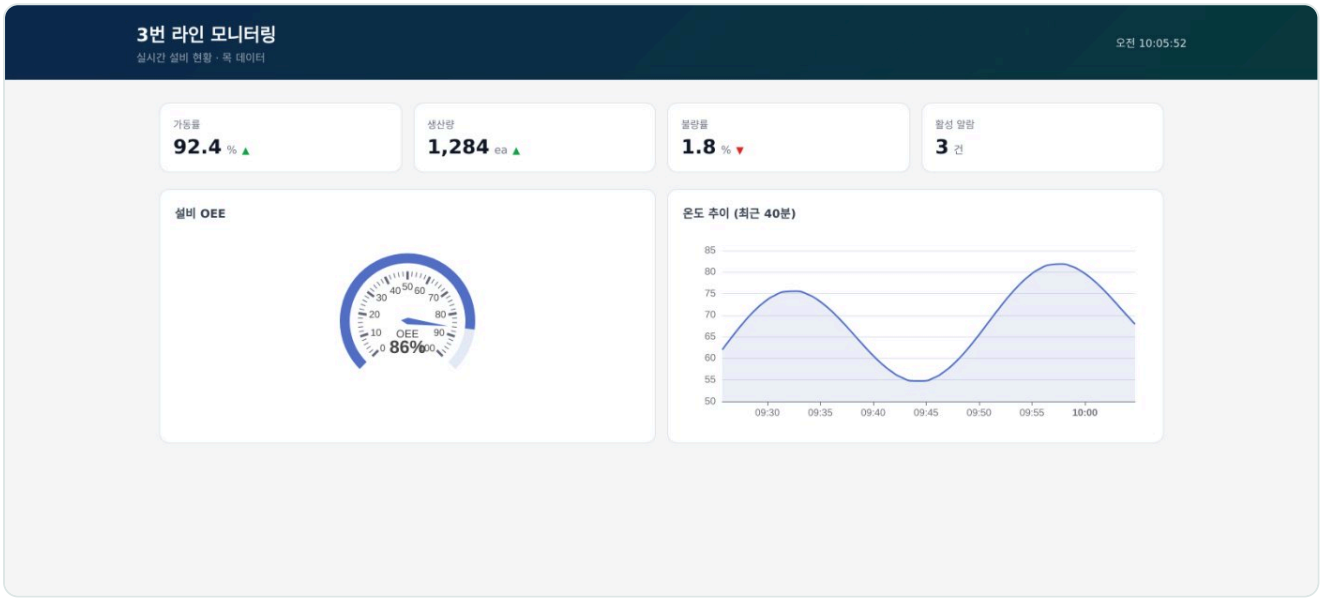


Figure 3. "Line 3 monitoring" — utilisation, output, defect rate and active alarms as top KPIs, equipment OEE and temperature trends as charts below. *Demo data.*

Analytics

Breaks a metric into its components, showing each part's contribution and the trend over time.

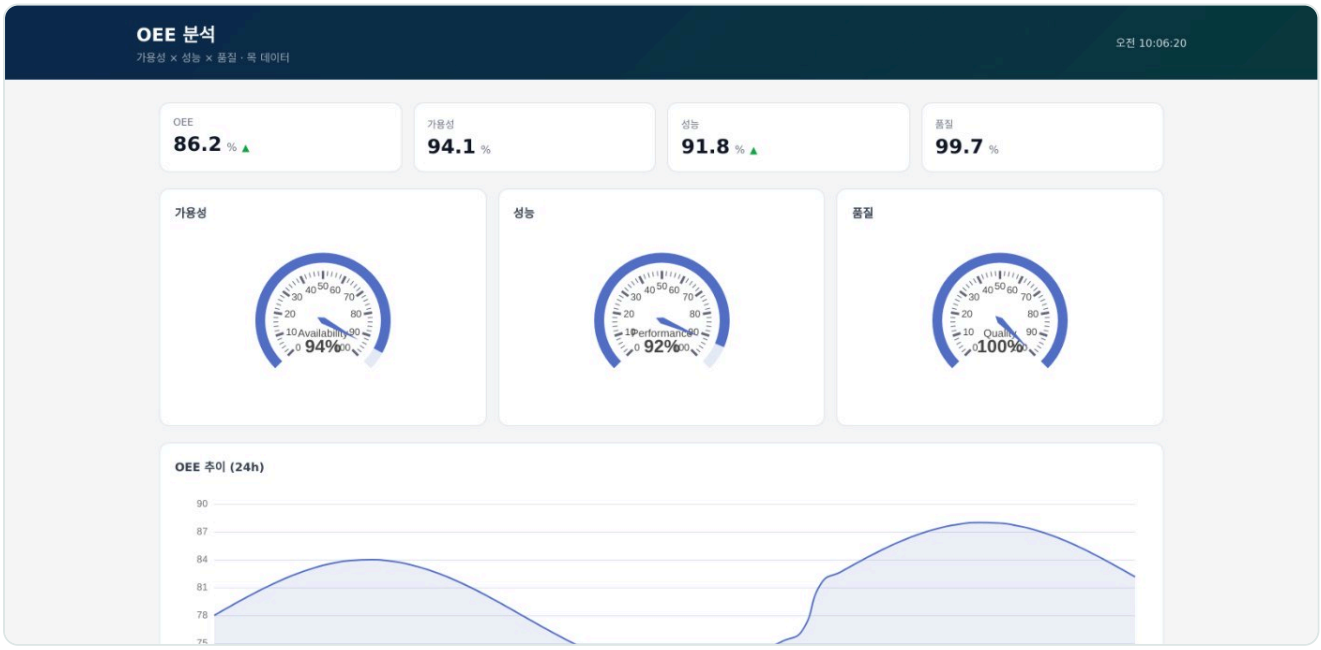


Figure 4. "OEE analytics" — overall OEE with the availability · performance · quality breakdown and a 24-hour trend. *Demo data.*

03 · Use cases

What gets built ② — twin · explorer

Digital twin

The line layout drawn in 3D with equipment state shown by colour. Drag to rotate, scroll to zoom, click a machine to open its detail.

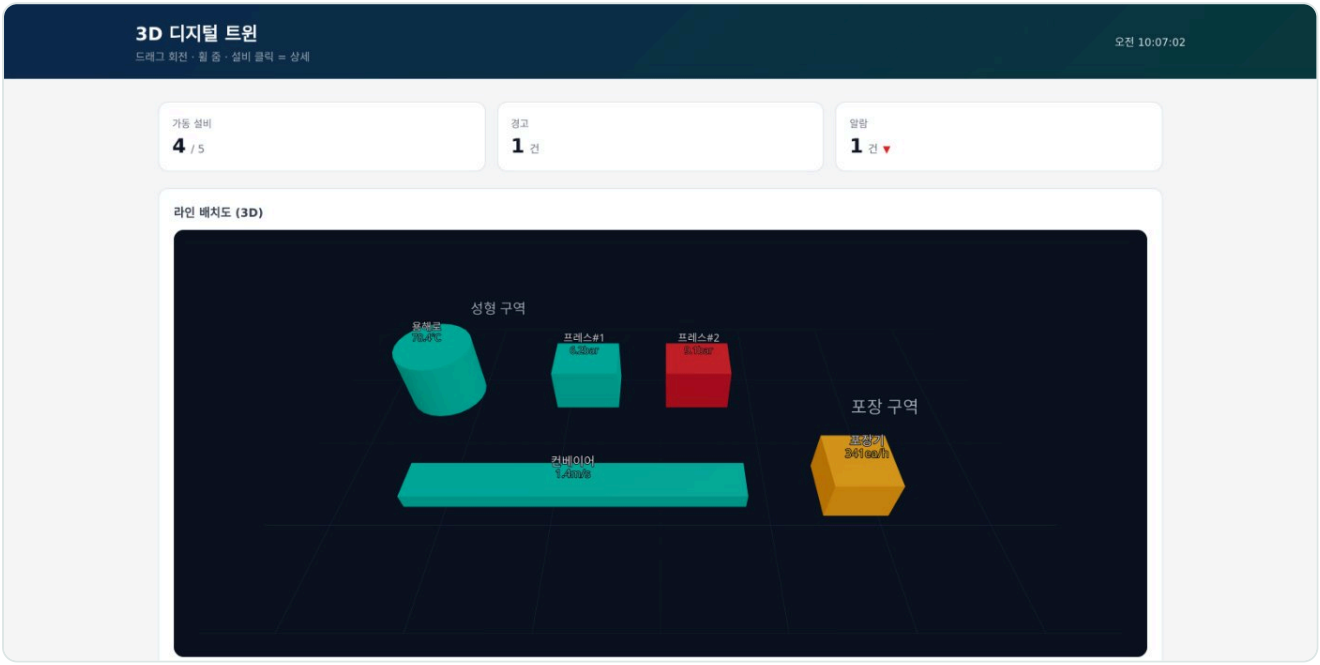


Figure 5. "3D digital twin" — the moulding and packaging areas as a 3D layout with current values (temperature · pressure · speed · output) above each machine. Abnormal machines are colour-coded.

Explorer · table

Drill along the asset tree and check tag values in a table — the basic skeleton of a list → search → detail business screen.



Figure 6. "Equipment drill-down" — pick a machine in the asset tree on the left and its state, utilisation, active alarms and tag list appear as a table.

> **The shape is not fixed.** The four above are just the most common skeletons; in practice it builds **what you say** — admin screens with create/edit/delete, print reports and kiosk-style signage come out the same way.

04 · Key strengths

① Screens are “written”, not “assembled”

PlantPulse Studio's agent does not place widgets — it **reads, writes and runs the source files of a real project**. So the ceiling on what can be expressed is code, not the tool.

What happens inside one turn

1. **Workspace preparation** — each request gets an independent project copied from a starter template and `git init`-ed.
2. **Tool use** — the agent calls `read_file` · `write_file` · `list_dir` · `run_command` · `platform_get` · `semantic_search` directly.
3. **Immediate execution** — a dev server runs in a session-specific container, so file changes appear in the live preview at once.
4. **Automatic commit** — a commit is recorded at the end of every turn. History can be viewed and reverted at any time.

Three working modes

Plan

The agent proposes a plan first and touches code only after approval. Suited to large changes.

Auto

Executes immediately without a plan. Suited to iterative tweaks and small adjustments.

Accept edits

Review the changes first, then choose to commit or revert. For when care is needed.

The result is the customer's asset

The generated app is not a set of settings inside the tool but **a complete source project**. It can be pushed to your GitLab/GitHub (a repository is created per app), and your own developers can take it over afterwards. **Remove the tool and the asset remains** — the decisive difference from a SaaS dashboard.

ALSO INCLUDED
App forking (clone and vary an existing app) · automatic app documentation · version retention and rollback · QR/embed sharing · access roles (admin/builder/viewer) · gallery publishing.

04 · Key strengths

② Say “line 3” and it understands

The real difficulty of an industrial screen is not drawing it but **connecting it**. Finding which of tens of thousands of tags is “line 3 discharge pressure” and mapping it by hand eats most of the time.

Semantic layer

PlantPulse Studio normalises and caches the plant's asset model, tags, aliases, units and hierarchy, and **resolves the names that appear in natural language into real identifiers**.

What the user said	What the system resolved
“line 3”	The set of asset IDs for that line
“discharge pressure”	The pressure tag ID per machine
“from yesterday until now”	An absolute time range in the configured time zone

> **The AI does not guess this.** Name → identifier conversion is rule-based and **deterministic**. The same input always gives the same result, and mappings never drift silently.

Division of labour with the platform

Role	Owner
Live data · search · relationship traversal	The existing PlantPulse Platform (holds the full database)
Name resolution · screen generation	PlantPulse Studio
Authentication · permissions	Delegated to the platform, or Studio's own accounts (for the floor)

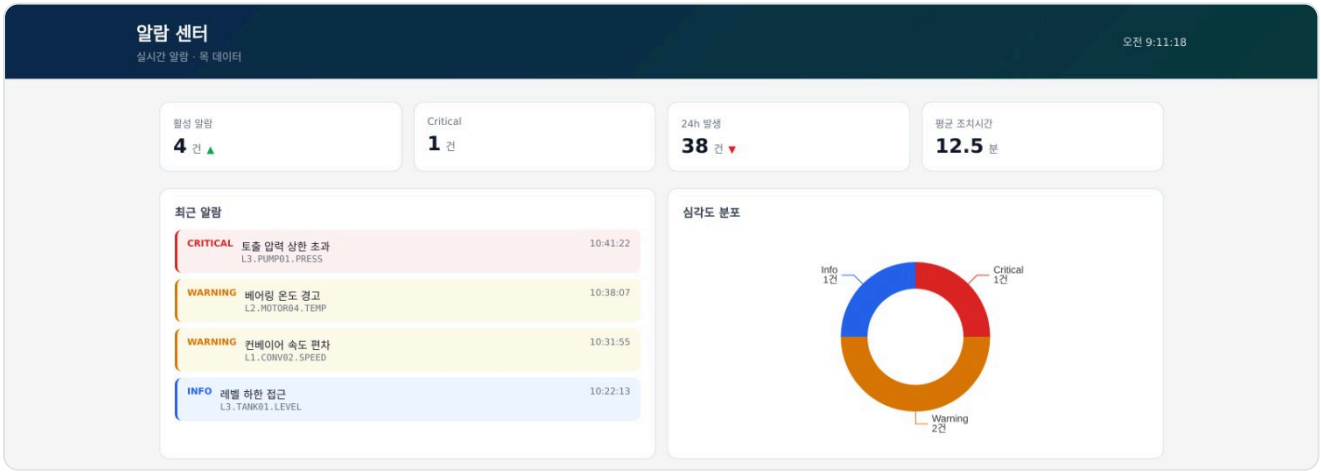
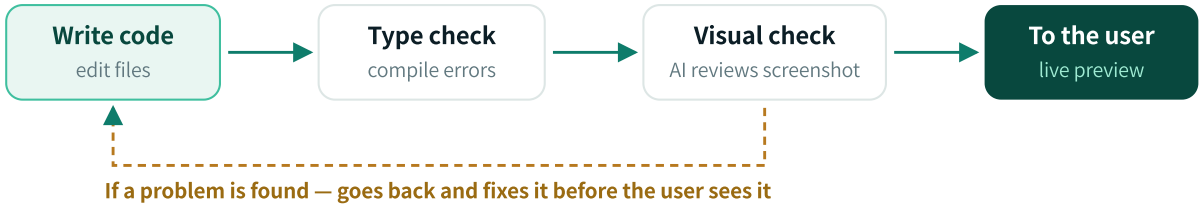


Figure 3. An alarm centre generated from one conversation — list, distribution and time-to-action KPIs by severity. The real tag naming convention (`L3.PUMP01.PRESS`) appears on screen as-is. *This capture uses demo data.*

04 · Key strengths

③ It verifies itself, and fixes itself

The answer to “can we trust code an AI wrote?” is built into the product. The agent **verifies twice** before presenting a result, and if something is wrong it goes back and fixes it before the user sees it.



1ST · MACHINE VERIFICATION

Type check

Errors caught at compile time are filtered out first. On failure the agent receives the error message and fixes it again.

2ND · VISUAL VERIFICATION

Vision self-check

At the end of a turn the screen is captured and the AI reviews it with its own eyes — broken layouts and overlapping elements cannot be seen from the code alone.

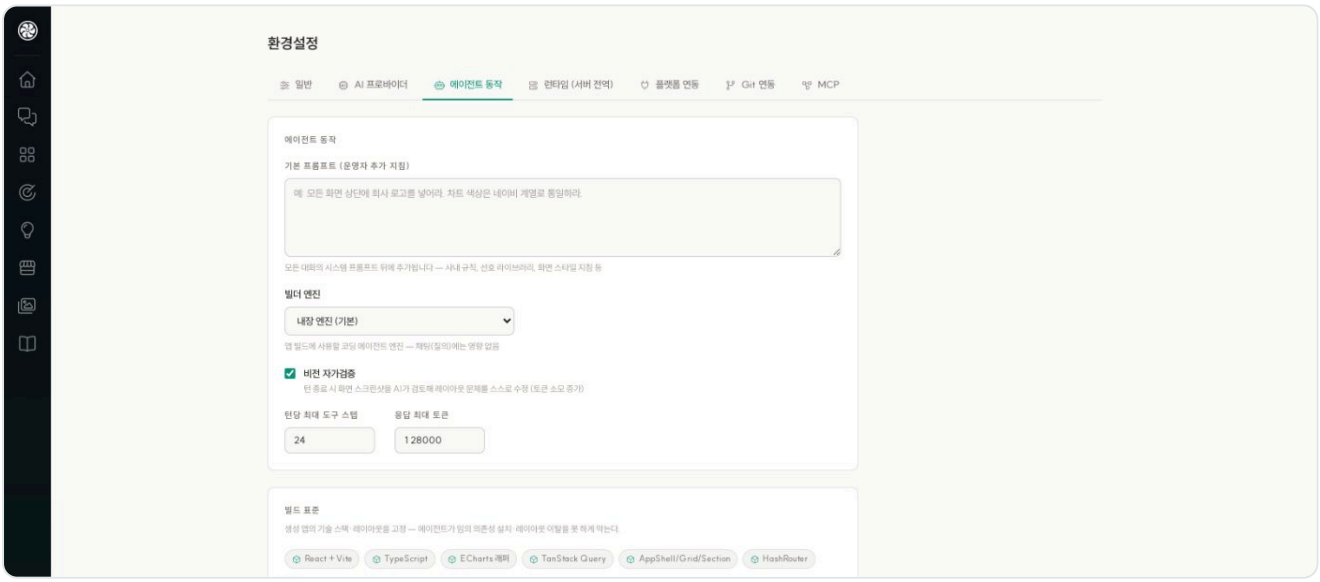


Figure 4. Administrators set self-verification on/off, the builder engine, the maximum tool steps per turn and company-wide guidance (preferred libraries, screen style rules and so on) as organisation policy.

05 · Key strengths

④ Veterans' know-how accumulates as an asset

Every floor has undocumented rules — “ignore the first 30 minutes after start-up on this machine”, “always look at pressure together with temperature”. **Floor skills** store this know-how so the agent applies it to every app built afterwards.

Record

Rules that come up in conversation are saved as skills. No more repeating yourself.

Apply

When building a new app the agent consults the relevant skills automatically.

Propagate

Export as a skill pack and transplant it as-is to other plants and subsidiaries.

Know-how stays with the organisation when people move on. Generation quality rises over time by design.

⑤ It looks like your own product

White-labelling is a standard feature. Upload your logo and product display name and they appear at once on the login screen, sidebar and gallery.

Branding

Logo upload and product display name. Applied without a restart.

Languages

Korean and English out of the box. Used as-is for shared roll-outs to overseas subsidiaries.

Source hand-over

Generated apps are pushed automatically to your Git server, one repository per app.

Themes

Light and dark modes, chosen to suit the lighting on the floor.



Figure 5. User management — three roles: admin, builder and viewer. Accounts can be delegated to the platform login or issued as Studio-only accounts for the floor.

05 · Key strengths

⑥ It goes into plants with no internet

Many manufacturing sites block external networks. PlantPulse Studio is not a cloud service but **a product installed in the customer's infrastructure**, with fully air-gapped installation as a supported procedure.

Three ways to install

Method	Description	Requires
One-line install	One install script sets up Docker, configures the environment and starts everything	Outbound access
Air-gapped install	All images carried in as files; checksum and signature verified automatically on import	Fully closed network
Internal registry	Pushed to your internal image registry and managed through your standard deployment procedure	An internal registry

Supply-chain trust

- Every container image shipped carries a **digital signature**.
- Air-gapped bundles are verified by **checksum + signature** before import — installation stops on failure.
- Every release also leaves an **immovable snapshot version**, so deployment can be pinned to a specific build.

DATA SOVEREIGNTY
Process data never leaves the customer's infrastructure. Studio delegates data to the existing plant system and neither stores nor transmits it separately. The AI model, too, can be either an external commercial API or an in-house hosted model (OpenAI-compatible gateway), so a configuration in which not even prompts leave the site is possible.

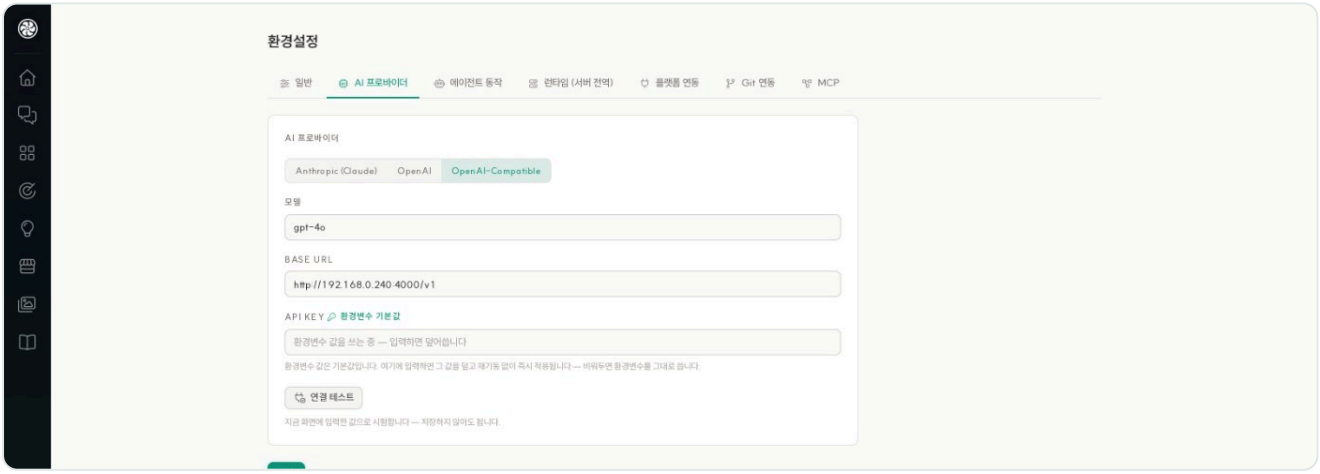


Figure 6. AI provider settings — choose a commercial API or an in-house hosted model (OpenAI-compatible) and test the connection before saving.

06 · Architecture

Topology — what runs where

Studio is a **thin orchestrator** that owns no data. Authentication and live data are delegated to the existing plant system; Studio only “builds and runs apps”.

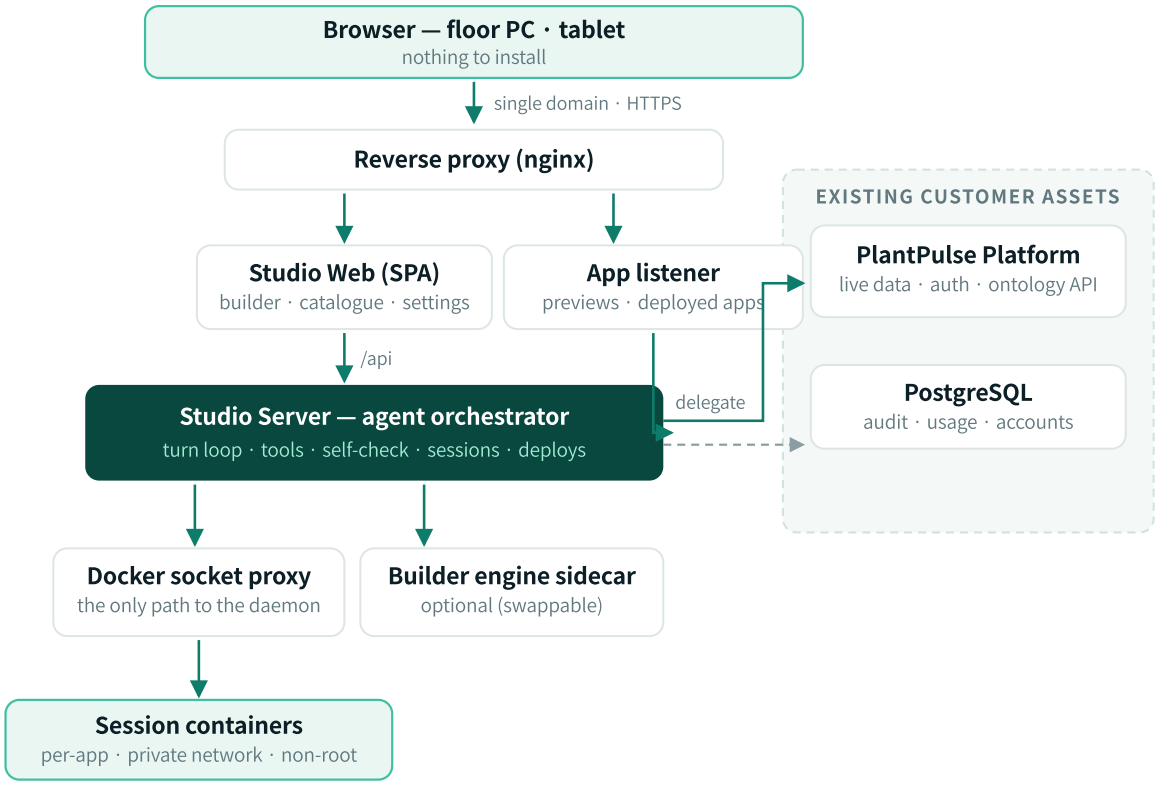


Figure 7. Request flow and placement. The dashed line is the boundary with existing customer assets — Studio creates no new data store inside it.

> **The operating stack is four containers.** Studio server · web · builder engine sidecar · Docker socket proxy. All ship with health checks, memory/CPU limits and log rotation configured.

The path one request takes

1. **Browser → reverse proxy** — users know a single address. Studio screens and generated apps are served under paths of the same domain, so there is no second set of certificates or firewall rules to manage.
2. **Studio server runs the turn** — receives the request, calls the AI provider, takes the tool results and decides the next action. Progress streams to the screen in real time.
3. **Live data comes from the platform** — when data is needed the existing plant API is called. Studio relays the response and does not store it.
4. **The result lands in the session container** — edited files are picked up immediately by the container's dev server, and that screen comes back as the preview.

07 · Architecture

Technology stack

Nothing exotic or proprietary. It is built on a **standard stack** that your own development team can take over and maintain.

Layer	Composition
Front end	Vite · React 19 · TypeScript · Tailwind CSS — charts (ECharts) · diagrams (Mermaid) · QR codes
Back end	Node.js LTS · Express · TypeScript — input and environment validation (Zod) · WebSocket · PostgreSQL driver
Security middleware	Security headers (helmet) · rate limiting (login / LLM / all APIs) · CORS denied by default
AI integration	Official Anthropic SDK + OpenAI-compatible interface — connects to an in-house gateway
Platform SDK	TypeScript client **generated automatically** from the OpenAPI specification. CI detects drift when the specification changes
Data	PostgreSQL (shared with the existing plant instance). Every feature also works with file storage and no database
Runtime	Docker · Docker Compose. Generated apps run in a separate container per session
Quality gates	Static analysis · format check · type check · automated tests — all must pass before merge
Supply chain	Private image registry · digital image signing · signature verification of air-gapped bundles

Three design principles

Thin orchestrator

Owns no data. Authentication and live data are delegated to the existing platform.

App isolation

Apps the agent builds run in a separate container and network per session.

Swappable engine

The coding engine can be changed at runtime. Not tied to any particular AI implementation.

Versions and releases

The product version is a **monthly calendar version** in `YYYY.MM` form. Each release also leaves an immovable snapshot version, so the customer can pin deployment to **a specific verified point in time** rather than “this month's latest”.

08 · Architecture

Builder engine and semantic layer

The coding engine is swappable

In AI the landscape changes every six months. To avoid tying the product to one implementation, the coding engine can be **swapped by runtime configuration**.

Engine	Characteristics	Suits
Built-in engine	The product's own tool-use loop. No extra container needed	Light installs · minimal resources
Open-source sidecar	A proven open-source coding agent run as a sidecar. Strong at context management over long tasks	Complex apps · long tasks

Whichever engine is used, the self-verification logic above it (type check · visual check) applies identically. The engine is abstracted as “the part that executes one turn”.

How the semantic layer works

1. **Snapshot cache** — the platform's asset model, tags, aliases and hierarchy are normalised and cached. Expiry does not cut service; the previous values are served while refreshing.
2. **Deterministic resolution** — turning “line 3” and “yesterday” into identifiers and time ranges is done **by rules, not by the AI**. The same input always gives the same result.
3. **Pre-injection + search** — a summary of the asset structure is injected at the start of a turn, and the agent queries further with its search tool when it needs more.
4. **Platform tools adopted** — the agent calls the platform's own analytics tools directly: health roll-ups, alarm diagnosis, tag history, relationship graph and more.

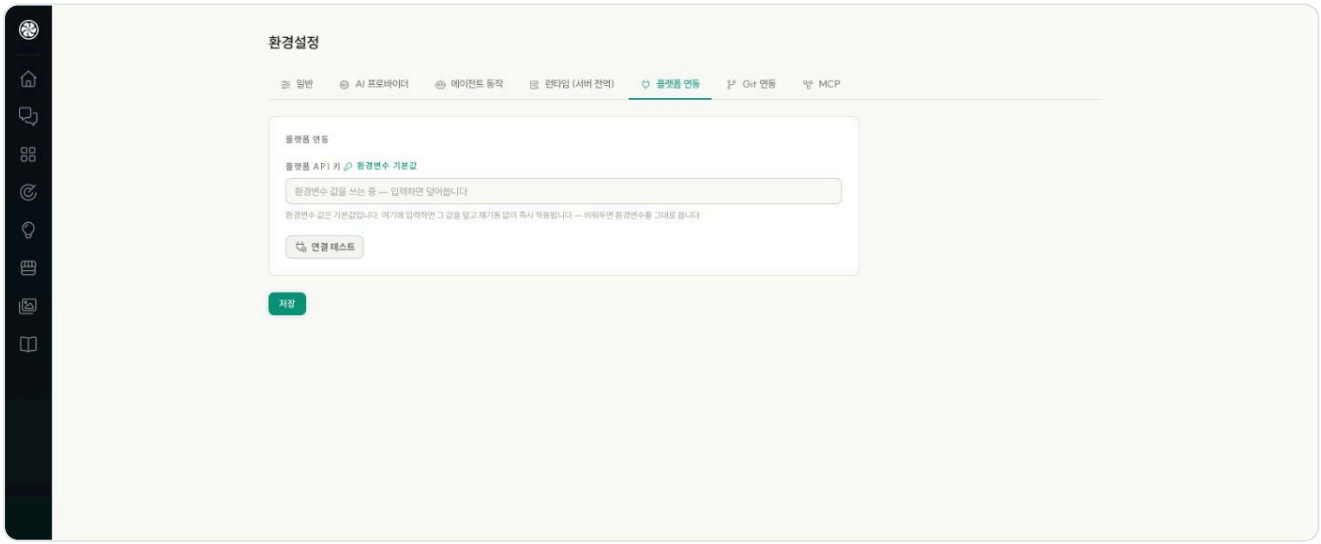


Figure 8. Platform integration settings — values injected via environment variables are the defaults; overriding them on screen applies immediately without a restart. A connection test is provided before saving.

09 · How it is used

Five steps in practice

Every screen below was **captured from a running instance**. None of them are mock-ups.



Step 1 · Log in

Floor staff connect from a browser with nothing to install. Accounts can be delegated to existing plant accounts, or Studio-only accounts can be issued for the floor.

Step 2 · Describe the screen in one sentence

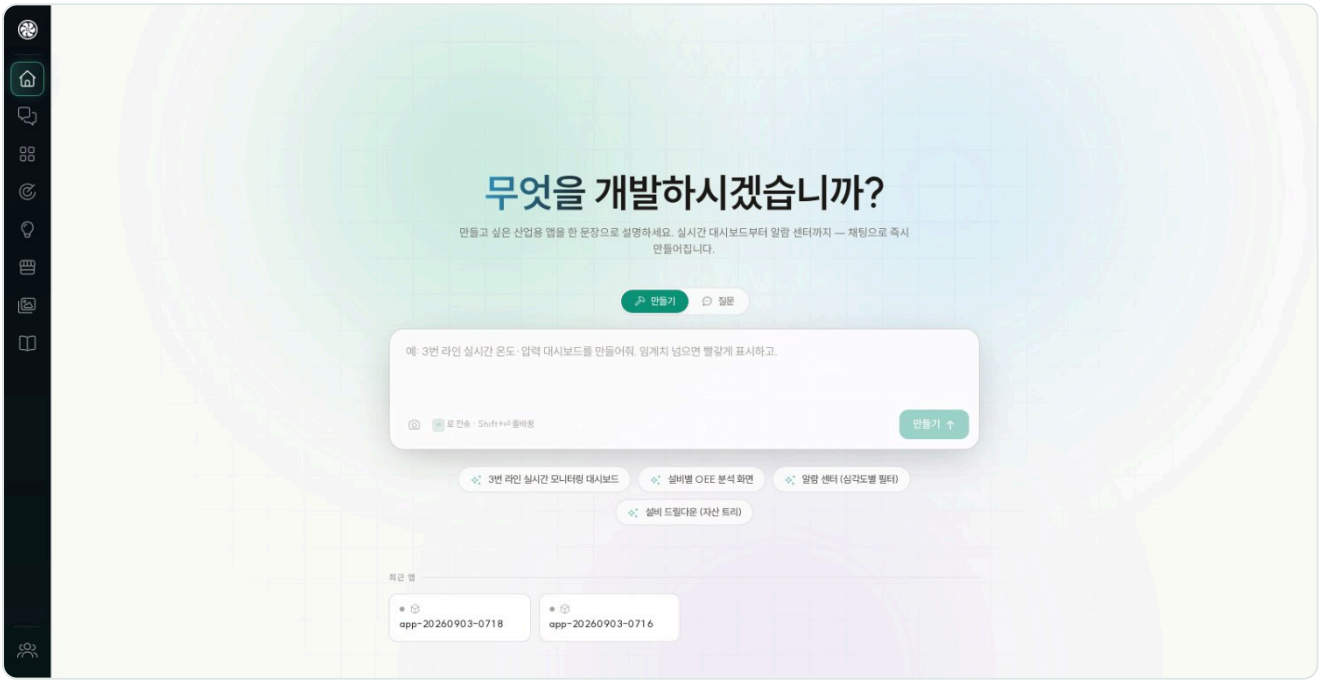


Figure 9. Start screen. Common requests are offered as example buttons, and there are two modes — **Build / Ask**. Ask mode only queries data and never touches code.

Apps built earlier are continued from the list below. The conversation history is restored with them, so “where we got to” is still there when you come back days later.

09 · How it is used

Step 3 · Watch it being built

On the left the agent explains what it is doing in plain language; on the right the **live preview** updates in real time. There is no waiting until it is finished.

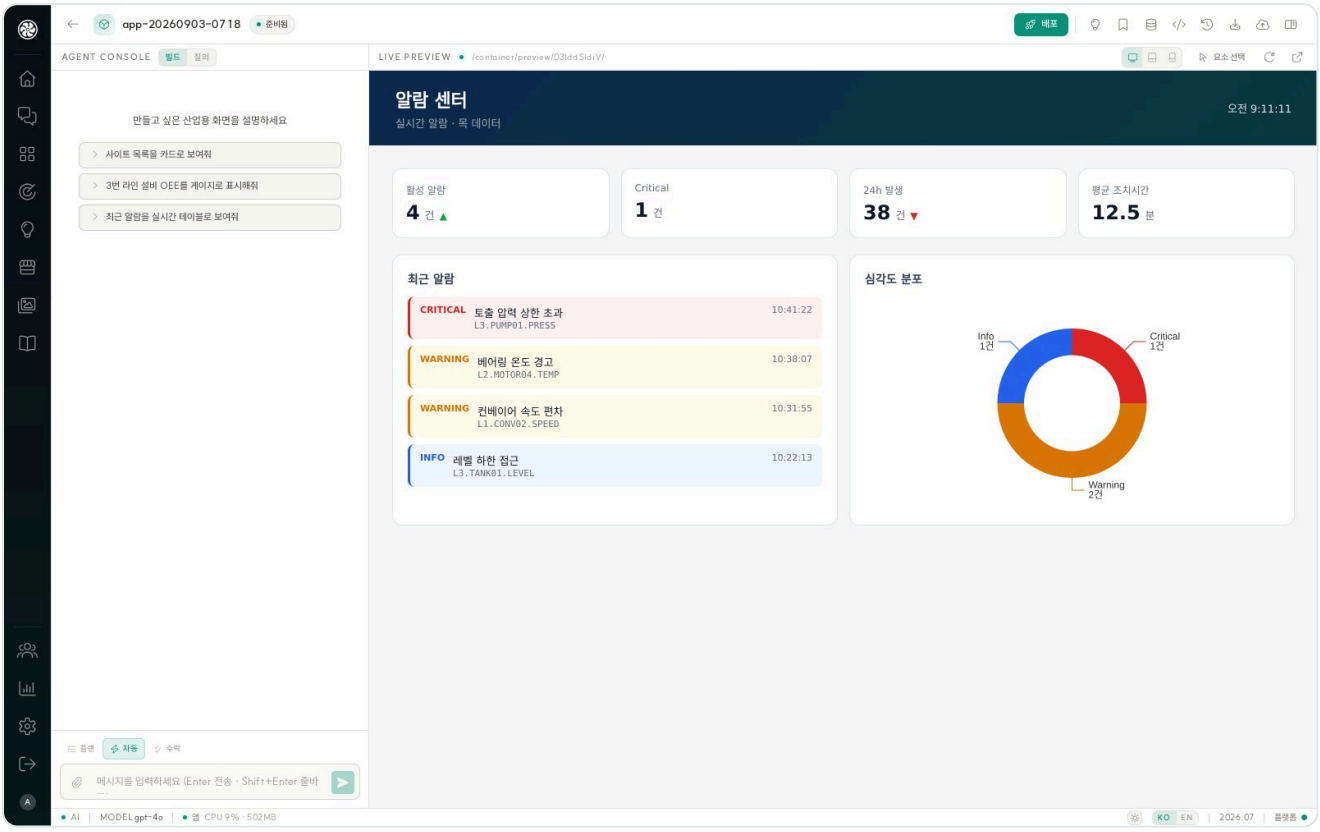


Figure 10. Builder screen. **Agent console** on the left, **live preview** on the right. The **Deploy** button at the top right records the current state as a version. The preview switches between desktop, tablet and mobile widths.

POINT AT THE SCREEN TO FIX IT

Element selection

Click the part of the preview you want changed and the agent edits that element. No need to describe “the chart in the upper middle”.

YOU CAN GO BACK

History and revert

An automatic commit is recorded every turn. If you don't like it, revert to the previous state and try again.

The agent explains what it did. Which data it looked up, why it built it that way and what is missing are all written down in plain language — for example, if a tag you need is not yet on the platform, it says so and proposes an alternative. This is designed so you never get a result where “nobody knows why it came out like this”.

09 · How it is used

Steps 4–5 · Deploy, share, keep watch

Step 4 · Deploy

One click pins the current state as a version and deploys it; earlier versions are retained for **rollback**. Deployed apps are shared by **QR code** or **embed** — one scan on a floor tablet opens the app.

Versioning

Every deployment leaves a version you can roll back to.

Access

Who can see each app is set per app.

Sharing

Choose QR, embed or gallery publishing.

Step 5 · Operations and visibility

Administrators see who used what and how much on one screen. AI usage is aggregated **per user, per project and per engine**, so cost can be forecast and allocated.

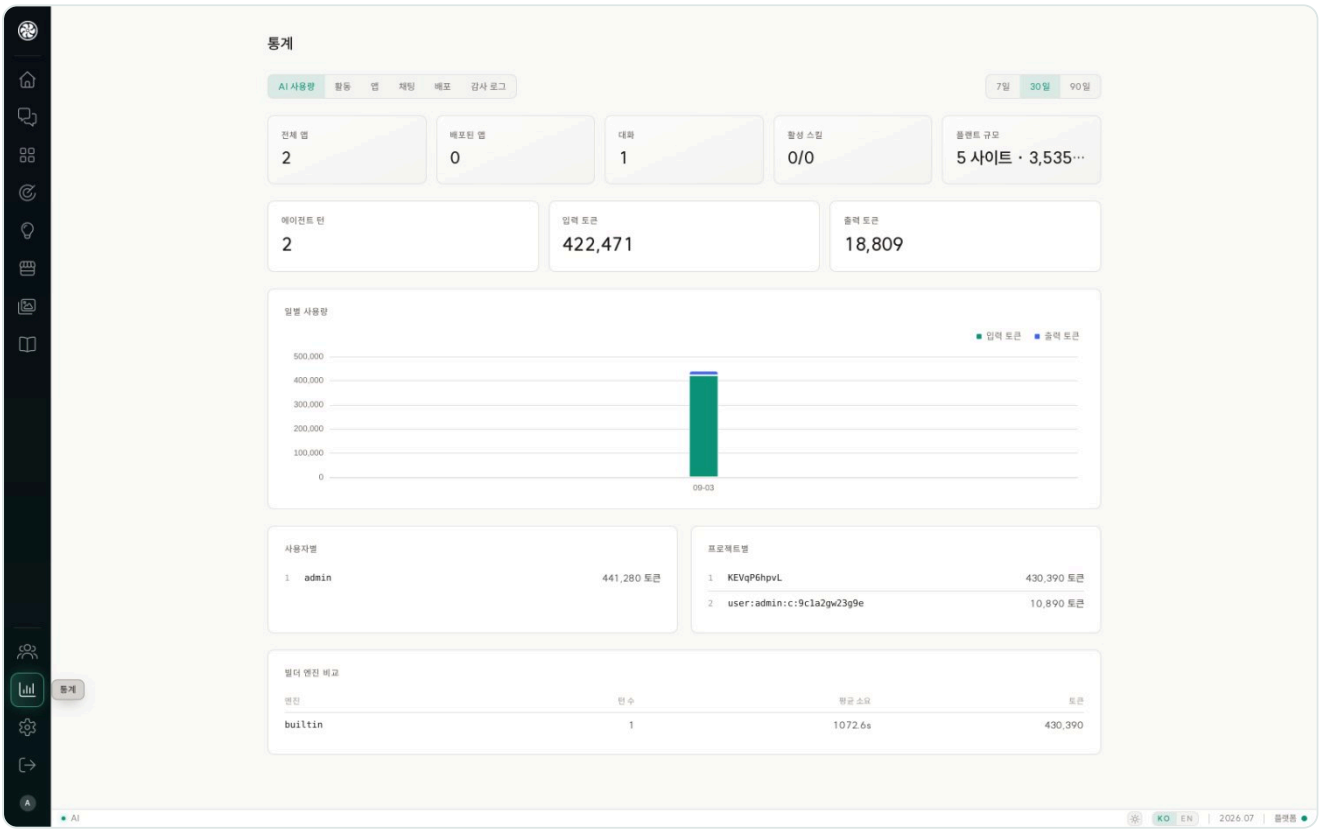


Figure 12. Statistics — apps, deployments and conversations, the size of the connected plant, AI token usage, per-engine cost comparison, audit log. *Figures are live measurements from the demo instance.*

Audit log

Who did what, when — paged lookup.

Usage management

Aggregated per user and project for cost allocation.

Self-healing

Lost containers recovered, idle sessions cleaned up.

10 · Security

Security · governance

“An AI writes code and that code runs” is a structure that needs control by its nature. PlantPulse Studio treats generated apps as **untrusted code** and is designed around isolation and audit.

Execution isolation

- **A separate container per app** — generated apps run in a session-specific container.
- **Non-root · resource limits** — memory, CPU and process-count limits apply. A runaway app cannot affect other sessions or the host.
- **No network between sessions** — session containers sit on a private network with inter-container traffic off. One session's app cannot look into another.
- **File-path isolation** — the agent's file tools cannot step outside their own workspace.
- **Platform query allow-list** — the agent's data queries are limited to permitted read paths.

Infrastructure access control

The Studio server **never touches the container daemon directly**. It goes only through a dedicated proxy, with **only the operations actually needed** allowed. Unneeded management functions are never exposed, and all daemon access converges on one point — one audit point.

Accounts · permissions · audit

Item	Content
Roles	Three levels: admin · builder · viewer. Roles exist only in the database; there is no path to grant permissions via environment variables or configuration files
Authentication	Delegated login with existing plant accounts, or floor-only accounts. Passwords are stored with a standard hash algorithm
Sessions	Token lifetimes are limited. Sessions survive a server restart
Key management	Platform access keys are held by the server, which queries on the user's behalf. Keys are injected into a generated app's data requests **only for the logged-in user**
Audit log	Key actions recorded and viewable in the admin screen
Secrets	Environment settings, workspaces and build artefacts are excluded from version control

Application-layer defences

Security headers · request limits on login, AI calls and all APIs · cross-origin requests denied by default · validation of outbound (webhook) targets. Container control commands are passed as arguments without a shell, leaving no command-injection path.

> **We recommend a security review before deployment.** The controls above are what the product provides by default; configuration to your security policy (network segregation level, identity integration, log collection scope) is agreed during deployment. A detailed security architecture document is available on request.

11 · Delivery

Deployment · delivery model

One line to install, or a file import

Case	Procedure	Time
Open network	Run the install script once → container runtime installed, environment configured, started	Tens of minutes
Closed network	Import the image bundle → checksum and signature verified automatically → load → start	Tens of minutes (plus your import procedure)
Upgrade	Swap in the new version's images and restart. Database schema applied automatically at start-up	Minutes

What operations needs

Backup

Backup scripts for the data root and the database are provided, with a tool to register a regular backup schedule.

Restore

The restore procedure from a backup is scripted.

Diagnostics

A diagnostics tool checks install state, connections and health checks in one pass.

Logs

Log rotation is configured on all four containers so disks never fill up.

Upgrade cadence and version pinning

Regular releases as monthly calendar versions (`YYYY.MM`). At the same time each release leaves an **immovable snapshot version**. Instead of following “latest” automatically, the customer **pins to a verified snapshot** and upgrades deliberately, only when needed.

SYSTEM REQUIREMENTS (RECOMMENDED)
One Linux server · a container runtime · an access path to the existing PlantPulse Platform · (optional) an in-house AI gateway or external AI API. Several session containers run at once, so **memory headroom** matters most for performance — exact specifications are sized from the expected number of concurrent users.

12 · References

References — cases and verification

Alongside deployment cases, we present the product's reliability as **verification machinery that is actually running**. Every check below is automated and repeated on every release.

Verification run automatically in the release pipeline

Check	Content
Code quality gate	Static analysis · format check · type check · automated tests. **A single failure blocks the merge**
Install drill	A clean environment is spun up every time and **the exact install procedure the customer will run** is completed and checked — a package that does not install can never ship
Image content verification	Confirms the release image really contains what it declares and passes its health checks
Image signing	A digital signature is attached automatically to the digest after push to the registry
Air-gapped bundle check	The bundle's checksum and signature are verified automatically before install
API specification sync	Drift between the platform API specification and the client SDK is detected, preventing silent mismatches
Database migrations	Applied automatically at start-up, with a lock so simultaneous multi-instance start-up never applies them twice

Product documentation

User manual

An online manual for floor users, linked directly from the product screens. — **docs.plantpulse.io**

Technical documentation

Architecture, installation, deployment, backup and release procedures are delivered with the product.

Deployment cases

HEAVY INDUSTRY · POWER

Doosan Heavy Industries

Gas turbine digital twin project
Digital-twin screens for rotating equipment.

SHIPBUILDING · OFFSHORE

Hanwha Ocean

No-code equipment dashboard project
Floor staff building their own equipment dashboards without code.

Detailed scope and outcomes are provided individually through your sales contact.

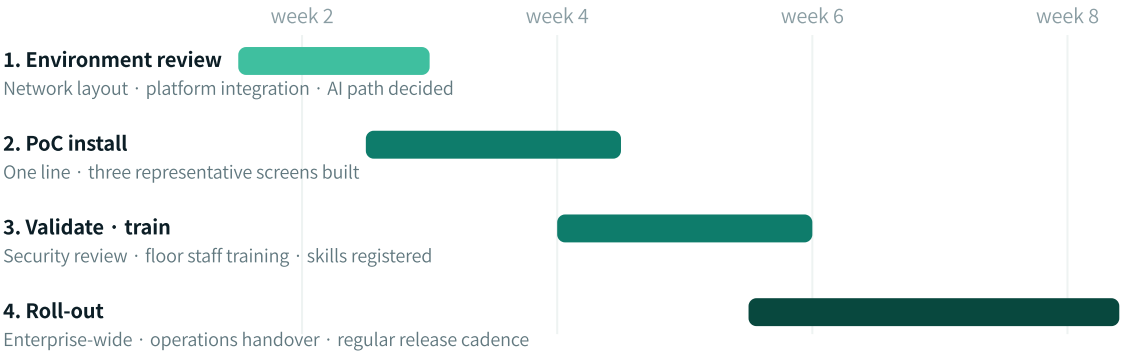
Operating record

Every screen in this proposal was captured from **an instance in continuous operation**. That instance is deployed automatically by the release pipeline on every release — showing that the product is **actually installed and running**, not merely documented.

13 · Deployment

Deployment plan and schedule

From PoC to full roll-out in **four stages**. The schedule below is the standard one and is adjusted to your environment and review process.



What to confirm in the PoC

- Can floor staff get the screen they want from one sentence **without training**
- Does it find our process's tags **from names alone**
- Do screens the existing dashboards could not build **actually get built**
- Does the air-gapped install procedure **complete within our security policy**

Decisions to make together during deployment

AI execution path

An external commercial API, or an in-house gateway / self-hosted model — decided by network policy.

Account integration

Delegated plant accounts or floor-only accounts as the main path, and how roles are distributed.

Server sizing

Memory and CPU sized from the expected number of concurrent users.

Backup · log policy

Backup interval, retention and log collection scope aligned to your policy.

NEXT STEP

30-minute demo · PoC environment discussion

The fastest way is to see “one sentence → screen” on a live instance. Share some of your tag conventions in advance and we will demonstrate **with your own data names**.

PlantPulse® Studio

Industrial AI app builder · On-premises / air-gapped

Product documentation docs.plantpulse.io · Document Rev. 2026.09

kopens

PlantPulse® **Studio**

INDUSTRIAL APP BUILDER

GET IN TOUCH

Start your industrial AI transformation today.

Consultation, demo requests, quotations — whatever you need, leave us a message. A specialist replies within one business day, and we can schedule a live demo on the real product right away.

PHONE

+82 1588-5268

EMAIL

webmaster@kopens.com

INQUIRIES · DEMO REQUESTS

kopens.com/contact

COMPANY

kopens.com

PRODUCT

plantpulse.io

DOCUMENTATION

docs.plantpulse.io

THE PLANTPULSE® FAMILY — DEPLOYED TOGETHER, ONE STACK FROM THE FLOOR TO AI

Edge	Platform	AI	Studio
Industrial protocol edge gateway	Enterprise IIoT / DataOps platform	Plant operations AI copilot	AI application builder

KOPENS, Inc.

© 2026 KOPENS. All rights reserved. PlantPulse® is a registered trademark of KOPENS, Inc.

Document Rev. 2026.09 · Figures and screens are as of the date of writing and may change without notice.

kopens®