PioDeploy

User Guide & Getting-Started Manual

Software deployment & policy management for Windows fleets

Covers: requesting access · subscribing · signing in · setting up clients & projects · installing the agent · deploying software · browser policies · viewing compliance · administration

Version 1.0 · TechPio · https://piodeploy.com

Contents

1. Introduction

PioDeploy is a centralised software-deployment and policy-management platform for Windows fleets, built for MSPs and IT teams. From one web portal you can install, update, patch, pin, and remove software, and lock down browser settings, across every managed machine — silently, with real-time compliance.

How it works, in one picture

The portal	A website (e.g. `https://piodeploy.com`) where you define what your machines should look like — packages, policies, browser rules — and see compliance.
The agent	A tiny Windows service installed once per machine. It runs as **SYSTEM**, checks in over HTTPS, and carries out deployments **silently** — the end user sees nothing.
Desired state	You set a rule once ("Chrome always latest", "remove AnyDesk"). Machines that drift are corrected automatically.

> **Key idea:** you never touch machines one by one. You describe the desired state in the portal; the agents make it so and keep it that way — invisibly, without interrupting anyone's work.

2. Getting started

2.1 Requesting access

PioDeploy is a back-office tool, so there is **no public self-registration** — accounts are provisioned for you. To get started:

1. Go to `https://piodeploy.com` and click **Get started** (or **Request access**).
2. Fill in the form — name, work email, company, and approximate number of machines — and submit.
3. The PioDeploy team receives your request and sets up your tenant and first admin login, then emails you the details.

> **Note:** if you run your own PioDeploy instance, your administrator creates accounts from **Users → Add user** instead. Either way, you receive a login email and password.

2.2 Choosing a plan & subscribing (buying)

Pricing is **per machine under management**, billed monthly, on a graduated schedule that gets cheaper at scale:

MACHINES	PRICE EACH / MONTH
First 20	$0.80
Next 480 (21–500)	$0.40
500 and above	$0.20

To subscribe:

1. Open the **Pricing** page on the website.
2. Drag the **slider** (or type a number) to your machine count — the monthly total and per-machine average update live. Example: 250 machines = **$108/month**.
3. Click **Subscribe**. You are taken to a secure **Stripe** checkout page.
4. Enter your card details and confirm. Stripe supports 135+ currencies for international cards.
5. You land on a confirmation page; a receipt is emailed to you and your subscription appears under **Admin → Billing**.

> **Test cards:** if the instance is in Stripe test mode, use card `4242 4242 4242 4242`, any future expiry and any CVC. No real money is charged.

Enterprise / unlimited / custom volumes: use the **Contact** page to reach sales for a tailored quote.

2.3 Signing in

1. Go to `https://piodeploy.com/login`.
2. Enter the email and password from your welcome email.
3. (Recommended) enable **two-factor authentication** from your profile menu for extra security.

> **Forgot your password?** Click "Forgot your password?" on the login page to receive a reset link by email.

3. The portal at a glance

The left sidebar is your map. What you see depends on your role.

SECTION	WHAT IT'S FOR
Dashboard	Fleet health at a glance — online/offline, pending & failed deployments, compliance.
Clients	Your customer organisations (each customer is a client).
Projects	A group of machines under a client. Agents enrol into a project.
Computers	Every enrolled machine, its hardware/software inventory and health.
Packages	The approved software catalogue (winget / Chocolatey / MSI / EXE …).
Deployments	The live queue and history of install/update/remove jobs.
Policies	Desired-state software rules (install / update / pin / remove).
Browser Policies	Lock down incognito, guest mode, password saving, dev tools.
Reports	Compliance, deployment activity and fleet health — with CSV export.
Users / Roles	Team accounts and what each role may do.
Notifications	Email/webhook alerts on failures, offline agents, drift.
Website / Billing / Settings	Edit the marketing site, manage payments, tune platform defaults.

4. Setting up your fleet

Structure is **Client** → **Project** → **Machines**. Do this once before enrolling machines.

4.1 Create a client

1. Sidebar → **Clients** → **New client**.
2. Enter the company name (e.g. *Acme Corporation*) and status **Active**. Save.

4.2 Create a project

1. Sidebar → **Projects** → **New project**.
2. Pick the client, name it (e.g. *Acme HQ Laptops*), save.
3. On creation you are shown two things **once**:

Agent API key	`pio_xxxxxxxxxxxxxxxxxxxx` — machines use this to authenticate. Copy and store it safely.
Agent download URL	`https://piodeploy.com/download/agent/<token>` — the installer script for this project.

> **Lost the API key?** Open the project and click **Rotate key** to generate a new one (the old one stops working). The download URL is always visible on the project's edit page.

4.3 The software catalogue

Packages lists the software you can deploy — pre-seeded with ~120 popular titles (Chrome, 7-Zip, Firefox, VLC, Notepad++, …), each with its winget/Chocolatey id. To add your own, click **New Package** and provide the name, type (winget / choco / MSI / EXE / MSIX / ZIP) and identifier or installer URL.

5. Installing the agent

5.1 What the agent is

The agent is a small **.NET Windows service** that runs as **LocalSystem**. It registers the machine, sends hardware/software inventory, and carries out deployments — all **silently in the background**. Nothing appears on the user's screen; no prompts, no windows, no interruption.

5.2 The Enrolment page — get your script

You do not write the rollout script yourself. Go to **Projects** and click the ⤓ **Enrol machines** icon on the project you want machines to join. Paste the project's **API key** into the box at the top and the scripts below fill themselves in, ready to copy.

> **About keys:** PioDeploy stores only a hash, so each key's value is shown **once** — at creation. A project can hold **several keys at once** (the API keys panel at the top of the Enrolment page): one per site, per RMM, per rollout wave. Creating a key never affects machines enrolled with another, and **revoking a key stops only the machines that were enrolled with it** — the rest of the fleet never notices. Lost a key's value? Just create a new one; nothing breaks. The ⚿ rotate action on the Projects list now *adds* a key rather than replacing the old one — retiring the old key is a separate, deliberate Revoke.

Pick the tab that matches how you already manage the estate:

TAB	USE IT WHEN
Group Policy (Active Directory)	Machines are domain-joined. The most common case.
Intune / Entra	Machines are cloud-managed through Microsoft Intune.
RMM / one-liner	You run NinjaOne, Datto, Atera, Action1, ConnectWise…
Single machine	One-off, by hand — testing or a special case.
Uninstall / remove agent	Take the agent off a machine by hand — decommissioning, or an agent too broken to reach remotely. Needs no API key.

5.3 Test on one machine first

Before pointing a policy at hundreds of machines, prove the script on one. Copy the **Single machine** command, open **PowerShell as Administrator** on a test PC, and run it. Within a minute the machine appears under **Computers**, online.

Then run the very same script a **second** time. It should finish in about a second and do nothing, reporting that the agent is already current. That is the behaviour that makes it safe to attach to a policy that fires at every boot — verify it before you roll out.

5.4 Zero-touch rollout with Group Policy

At scale you never touch machines individually. A **computer startup script** runs as **SYSTEM at boot, before anyone logs in** — exactly the account the agent needs, with nobody to interrupt.

1. Copy the script from the **Group Policy** tab and save it where every domain computer can read it, e.g. `\\yourdomain.local\NETLOGON\PioDeploy\Install-PioDeployAgent.ps1`. NETLOGON is readable by Domain Computers by default.
2. Open **Group Policy Management** (`gpmc.msc`).
3. Right-click the OU holding the target computers → **Create a GPO in this domain, and Link it here...** → name it *Deploy PioDeploy Agent*.
4. Right-click the new GPO → **Edit** → **Computer Configuration** → Policies → Windows Settings → **Scripts (Startup/Shutdown)** → **Startup** → **PowerShell Scripts** tab → **Add** → browse to the UNC path.
5. On the machines: `gpupdate /force`, then **reboot** — startup scripts only run at boot.

> **Safe to run at every boot.** The generated script is version-aware, not just idempotent: it exits in milliseconds when the agent is already current, and **upgrades it when it is not**. So one GPO both enrols new machines and keeps the whole fleet on the current agent — link it once and leave it. A failure never blocks a boot; it is logged to `C:\Windows\Temp\PioDeploy-Enrollment.log` and retried at the next restart.

5.5 Intune and RMM

The **Intune** tab's script goes to *Devices* → *Scripts and remediations* → *Platform scripts*. Set **Run this script using the logged on credentials: No** — that is what makes it run as SYSTEM — and **Run script in 64 bit PowerShell host: Yes**. Assign it to the device group.

The **RMM** tab's script pastes straight into a "run script" job targeted at a device group; RMM agents already run as SYSTEM. Both carry the same version check, so re-running them is harmless.

> **Result:** every targeted machine enrols silently in the background, and from then on all deployments are invisible to users. This is how you reach thousands of endpoints with zero interruption.

5.6 Fixing or removing an agent remotely

Open **Computers** → **a machine**. In the page header, next to *Reassign project*:

BUTTON	WHAT HAPPENS
Reinstall agent	At its next check-in (within about a minute while online) the machine re-downloads the current agent bundle and replaces its own install — service, binaries, everything. Its settings, API key and identity are kept, so it comes back as the same computer. This is the remote fix for a broken or outdated agent that still checks in: nobody touches the machine.
Uninstall agent	At its next check-in the machine removes its own agent: the service is deleted and all agent files are wiped. Software that was installed through PioDeploy stays on the machine. The computer record and its full history remain in the portal until you delete them there.

While a command is queued the header shows a **pending** badge with a **Cancel** link — cancelling before the agent's next check-in withdraws it. Each command is delivered **exactly once**: if the machine never acts on it (for example it went offline first), nothing loops — just click again.

> **When the button can't help:** both buttons rely on the agent checking in. For a machine that is offline, or whose agent is too broken to reach the server at all, use the **Uninstall / remove agent** tab on the Enrolment page (run elevated, no key needed) — or re-run the enrolment script to repair it in place.

5.7 Retiring and deleting — the lifecycle rules

Deletion is deliberately hard to do by accident. Three rules hold for **everyone**, admins included:

RULE	WHY
A machine with a live agent can only be **retired** (the bin icon on the Computers list), never deleted.	Deleting the record would orphan a working agent — which would simply re-register. Retired machines sit under *Show deleted* with their history kept, and are revived automatically if their agent reports again.
Delete permanently (in the retired list) works only once the agent is confirmed gone.	"Gone" means: the machine never enrolled, or an *Uninstall agent* command was delivered and the machine has stayed silent since. A machine that checks in after the uninstall visibly still has its agent — the proof resets itself and deletion stays blocked.
A project cannot be deleted while **any** machine exists in it — active *or* retired.	Machines anchor jobs, history and compliance records. The path to deleting a project is: uninstall agents → permanently delete the machines → delete the project.

6. Deploying software (policies)

A **policy** is a desired-state rule applied to every machine in a project. The agent enforces it automatically on check-in and heals any drift.

Create a software policy

1. Sidebar → **Policies** → **New policy**.
2. Choose the **project** and **package** (e.g. Google Chrome).
3. Pick the **rule**:

RULE	MEANING
Install	Put it on machines that don't have it.
Auto update	Keep it current on machines that have it.
Force update	Reinstall the desired version even if current.
Remove	Uninstall wherever it's found.
Block	Forbidden software — removed whenever detected.

How enforcement decides — install once, then update. On every pass the policy first checks what the machine actually reports:

1. **Not installed** → one install job is queued (latest version, or the pinned one). Installs are paced to **one successful attempt per frequency window** — never repeated back-to-back.
2. **Already installed** → nothing is installed again. An **Auto update** policy takes over: it queues an update only when one is due, at most once per frequency window (daily/weekly/monthly), inside the maintenance window and ring schedule you set.
3. A machine whose agent cannot scan winget still counts a succeeded PioDeploy install as installed, so a blind scan can never cause repeat installs. Failed attempts back off before retrying; an in-flight job is never duplicated.

Rollback & exact-version pinning. Rolling back — or pinning to an exact earlier version — means reinstalling a specific published build. Only **winget** and **Chocolatey** packages can do this: they fetch any version from their repository on demand. **MSI, EXE, ZIP, MSIX** and **portable** packages carry a single installer, so they cannot roll back — a **Force update** on those simply reinstalls the current version. PioDeploy only offers Rollback (and exact/maximum version pinning) for packages that support it.

PACKAGE TYPE	INSTALL / UPDATE	REMOVE (UNINSTALL)	ROLLBACK / PIN EXACT VERSION
winget	Yes	Yes	Yes*
Chocolatey	Yes	Yes	Yes*
MSI	Yes	Yes	No — reinstalls current
EXE / ZIP / MSIX / portable	Yes	No	No — reinstalls current

*Rollback also needs the target version to still be published by the source. Some vendors (Chrome especially) publish only the current version, so rollback isn't possible even on winget — the version dropdown lists what can actually be installed.

4 Optionally set a **version** (Latest / Exact e.g. `24.09` / Minimum / Freeze), **priority**, **maintenance window** (e.g. Sat 02:00–05:00), **frequency**, and **deployment rings** (Pilot first, Production after N days).

5 Choose the **mode**: **Enforce** (queues jobs) or **Audit only** (reports compliance, changes nothing). Save.

> **Tip:** open a policy to see its compliance drill-down — every machine's status (Protected / Pending / Failed / Scheduled) with reasons, plus per-machine **Exclude** for exceptions like a CEO laptop.

One-off deploy to a single machine

Open **Computers** → **a machine**, use **Quick deploy** to queue an install/update/remove immediately without a policy. The action list adapts to the package you pick — Rollback appears only for winget/Chocolatey, and Remove is hidden for types that can't be uninstalled — and the button states the outcome (e.g. "Upgrade 138.0 → 141.0", "Up to date") before you click.

When PioDeploy has version history for a winget/Chocolatey package on that machine, a one-click ↩ **Roll back to *X*** button appears next to the status. It queues a rollback to the exact version the machine ran before its most recent change — no version to look up by hand.

Bulk deploy to a whole project

From **Deployments** → **Bulk deploy**, push one package to every machine in a project at once. Pick the project, an optional **ring** (e.g. Pilot first, Production later), the package, and the action (Install / Auto update / Remove). Each machine is queued through the same checks as a single deploy: those already up to date are skipped, anything already queued is not duplicated, and you get a one-line summary such as "*Queued on 42 machines · 8 already up to date*". Rollback stays a per-machine action, since each machine's previous version differs.

7. Browser policies

Lock down browsers like enterprise Group Policy — without a domain. Supported: Chrome, Edge, Firefox, Brave (Opera reports "unsupported").

1. Sidebar → **Browser Policies** → **New policy**.

2. Name it, pick the project, and choose the restriction. The catalogue offers **30+ policies grouped by category** — selecting one shows its description, which browsers support it, and whether a restart is needed:

CATEGORY	EXAMPLES
Password Management	Password saving, leak detection
Private Browsing	Incognito / InPrivate, guest mode
Autofill	Addresses, credit cards
Sync & Sign-in	Browser sync, browser sign-in
Browser Security	Developer tools, QUIC, screen capture
Downloads & Cookies	Block all downloads, third-party cookies, session-only cookies
Site Permissions	Notifications, pop-ups, location, camera, microphone, clipboard
Homepage & Startup	Forced homepage URL, forced new-tab URL
Extensions	Block all installs, force-install extensions by ID
Lockdown	Printing, browsing history, bookmark editing
AI & Extras	Gemini / Copilot / Leo, shopping assistant, new-tab feed, Rewards, games
Advanced	Translate, WebUSB, WebBluetooth, WebSerial

3. Choose browsers (or **All**) and set **Active**. Save.

The agent applies the corresponding registry values (Chrome/Edge/Brave) or `policies.json` (Firefox) silently. Browsers already open show "pending restart" until reopened. Deleting the policy rolls the setting back automatically.

Assignment & inheritance

Each policy is assigned to a scope: **All machines**, a **Client** (every project it has), a **Project**, a **Device group** (a hand-picked set of machines that can cut across clients — manage them under **Fleet** → **Device Groups**), or a **Single computer**. When policies of the same type overlap on one machine, the most specific wins:

Computer › Device group › Project › Client › All machines

So a company-wide "block incognito" can coexist with a developer group that explicitly allows it — the group rule wins on those machines and the company rule everywhere else. Per-machine **exclusions** still override everything.

> **Quick walkthrough — your first device-group policy.**
> 1. **Fleet** → **Device Groups** → type a name (e.g. *Finance workstations*) → **Create group**.
> 2. Click **Manage** on the group → pick machines from the dropdown → **Add** (they can come from any client or project).
> 3. **Browser Policies** → **New policy** → **Assign to: Device group** → pick your group → choose a restriction (e.g. *File downloads*) → **Create policy**.
> 4. The policies list now shows *Group: Finance workstations* with a per-machine compliance count, and **Browser Policies** → **Compliance** tracks it fleet-wide. Group members receive it on their next agent check-in; everyone else is untouched.

> **Value policies need agent 1.4.0+.** Policies that carry a value — a forced homepage/new-tab URL or a force-installed extension list — use new operation types the agent learned in **1.4.0**. Older agents report them as "unsupported" until they self-update on their next check-in. For extensions, enter the 32-letter ID from the Chrome Web Store URL (one per line); forced extensions install silently and cannot be removed by users.

Policy templates

From **Browser Policies** → **Templates**, apply a whole bundle in one click. Seven built-in templates ship with the platform — **High Security**, **Standard Office**, **Developer**, **Kiosk**, **Finance**, **Healthcare** and **Education** — each creating its policies on the chosen project (types the project already has are skipped, never overwritten). You can also **save any project's current policies as a custom template** and reuse it across clients.

Import & export

Every template card has an **Export** link, and picking a source project in "Save a project as a template" offers **Export JSON** — both download a portable `.piodeploy-policies.json` file. **Import** that file on the Templates page (of any PioDeploy instance) and it becomes a custom template, values and browser scopes intact; policy types the receiving catalogue doesn't know are skipped and counted, never silently lost.

Compliance dashboard

Browser Policies → **Compliance** answers "are we protected, and where not?" in one page: fleet-wide totals (protected / failing / pending / unsupported), a per-policy breakdown with a compliance bar and last-report time, an **"only policies with problems"** filter, and a **machines needing attention** list showing exactly which computer, browser and policy is failing, with the agent's own error detail. Client-role users see only their own fleet.

> **Example:** "Block incognito — all browsers" on *Acme HQ Laptops* disables private browsing across every Chromium browser and Firefox on every machine in that project.

8. Viewing your fleet

Dashboard

Online vs offline machines, pending & failed deployments, software counts, and a browser-policy compliance widget.

Computers

Click any machine to see hardware, health checks (disk, secure boot, last check-in), its deployment history and browser-policy status.

Installed software

A patch-status table per machine: **Application** (with publisher and source/managed/PioDeploy chips), **Installed (current)**, **Latest** (what the machine's own package manager offers — "—" when nothing newer is published), **Update required** (Yes/No badge) and **Status**. The view opens on **Deployed by PioDeploy** — what you are accountable for — with **In catalogue**, **Update available** and **All software** one click away.

The **Status** column is actionable: an outdated app that matches a catalogue package shows **Update now** →, which queues an update job on that machine immediately (clicking twice while it runs never duplicates). Outdated apps outside the catalogue read *Update available* — add them as a package to manage their updates. Everything current reads *Up to date*.

> **Only the machine knows.** The portal knows your catalogue and what is installed; it has no idea a new Chrome shipped this morning. The agent asks winget and reports back — which is why a package can be **Compliant** and still show an update: your policy asked for it to be installed, and it is. Whether to update is your call, or a job for an **Update** policy.

Software status

Why each policy is — or is not — acting on this machine, in words: “Installed (141.0)”, “Waiting for maintenance window”, “Cadence ring eligible in 2 hours”, “Installed 138.0, policy wants Minimum 141.0”, or “audit only, so nothing will be queued”. This answers “so why isn't it installed?” when there is no job to point at.

Client compliance reports

Every client can get a **branded compliance PDF** — fleet overview, software- and browser-policy compliance, 30-day deployment activity and a machine list. Download it any time from the **PDF** link on the Clients list; client-portal users have a **Compliance report** button on their own dashboard. Tick **Monthly** on a client row to email the PDF to that client's portal users automatically on the 1st of each month (off by default).

Deployment log

Every attempt and the reason it ended that way, with the agent's own output behind a toggle. The log is **collapsed by default** — the header shows the attempt count; click **Show log** to expand. Note that “Succeeded” is not always “installed something”: **Already installed — nothing was changed** means

winget found the package present and did nothing, which is a success worth telling apart from real work.

Deployments

The live job queue and history — filter by status/action, retry failed jobs, cancel pending ones. Repeats of the same task on the same machine fold into one row with a `×N` badge; tick **Show full history** for every attempt.

Reports (with CSV export)

REPORT	SHOWS
Policy compliance	Every policy's % compliant with drill-down.
Deployment activity	Jobs over a date range, success rate, failures.
Fleet health	Every machine: ring, agent state, last seen, disk pressure.

Each report has an **Export CSV** button (for users with the export permission).

9. Notifications

Get alerted when things need attention. Sidebar → **Notifications** → **Add channel**.

CHANNEL	USE
Email	Any address, via the configured mail server.
Webhook	Posts JSON with a Slack/Discord-compatible message — paste a Slack incoming-webhook URL as-is.

Subscribe each channel to events: **deployment failed**, **new computer enrolled**, **agent offline**, **daily compliance drift digest**, **browser policy failed**. Use **Send test** to verify.

10. Administration

Users & roles

Users → **Add user** creates a team account with a role. Roles (Super Admin, Admin, Manager, Technician, Client, Viewer) are permission sets you can edit on the **Roles** matrix — tick a box to grant a permission to a role. **Client** users are scoped to their own client's data only. The user list shows a **2FA** badge per account, so you can see at a glance who has two-factor authentication enabled.

Two-factor authentication

Every user can enable authenticator-app 2FA (with recovery codes) on their **Profile** page. To make it mandatory, set **Settings** → **Security** → **Require two-factor authentication** to **Staff** (everyone except client-portal users) or **Everyone**. Users without 2FA are then routed to their profile to enrol before they can continue — nobody is signed out, and the profile always stays reachable so enrolment can't deadlock.

Settings

Company name, agent online threshold, offline-alert delay, default job retries, policy backoff, activity-log retention, and 2FA enforcement — all editable, no redeploy.

Email (SMTP)

Administration → **Email** is where outgoing mail is configured — in the portal, not over SSH. Choose your provider (Hostinger, Gmail, Microsoft 365, Postmark, SendGrid, Brevo, Mailgun, SES, or a custom relay) and the server details fill themselves in; you supply the username, password and sender. Then **Send test email**, because saving proves nothing and sending does. A failure shows your provider's own words plus what to do about them.

> **The most common failure:** the **From address** must be the mailbox you authenticated as, or another address verified with your provider. Authenticating as `info@yourdomain.com` and sending as `hello@yourdomain.com` is rejected — `553 5.7.1 Sender address rejected`. Your SMTP password is encrypted before it is stored and is never shown again; leave the field blank to keep the stored one.

Until this is set, nothing can send: deployment-failure alerts, offline warnings and website enquiries all go nowhere silently. `php artisan security:check` flags it.

Signups — self-service accounts from the pricing page

The pricing page's **Get started** runs a four-step public wizard: fleet size (live price) → account (email + password) → company → review & pay. With Stripe connected the visitor pays right there; without it, the application arrives marked for manual invoicing. Either way **nothing is created yet** — the application lands in **Administration** → **Signups** as exactly that: an application.

1. **Verify the payment.** The row says it plainly: Paid via Stripe (the webhook confirmed the money), *Awaiting Stripe payment*, or *Verify manually* (invoice/transfer — check your bank first).

2. **Approve.** One click creates the client, the owner login (**Client Owner** role — full control of their own company, nothing outside it), and emails them that they can sign in. The password is the one they chose at signup — you never see or send it.
3. Or **Reject** with an optional reason; nothing is created.

> **What the new customer can do:** the **Client Owner** role manages everything inside their own company — create projects, enrol machines, deploy software, see reports — inside their own tenant and nowhere else. They also get a **Team** page to add their own **Technician** and **Viewer** accounts (they cannot mint other owners or admins; that stays with you). To be alerted as applications arrive, subscribe a notification channel to "signup.received".

Recurring billing — what happens every month

A wizard signup that paid by card created a real **monthly Stripe subscription** — Stripe charges the card automatically each month; nobody invoices anyone by hand. PioDeploy mirrors that subscription per client:

WHERE	WHAT YOU SEE
Clients list (staff)	A subscription badge next to each client's status — *active*, *past due*, *canceled* — with amount and renewal date on hover.
Billing page (the client)	Their plan, monthly amount, machines enrolled vs. licensed, renewal date, a **Change plan size** control (self-serve resize at the graduated price — Stripe prorates the difference on the next invoice), and a **Manage billing** button that opens **Stripe's own portal** — update card, download invoices, cancel — so card details never touch PioDeploy.

Dunning — what happens when a charge fails: the client is emailed **immediately** ("your payment didn't go through", with a fix-payment link), you get a "billing.payment_failed" notification, and Stripe retries the card on its own. While it stays unpaid, the client receives a **reminder every 3 days** carrying a real countdown; when the **grace window** closes (default 14 days — set it in Admin → Billing settings), the client is **suspended automatically**: suspension email to them, notification to you, and a red notice on their Billing page. **Paying reactivates everything by itself** — restore email included, nobody needs to contact anyone. Two deliberate limits: suspension is a status and emails, *never* fleet breakage (agents keep running — cutting machines off over a card decline is an operator's call); and a suspension **you** applied by hand is never lifted by a payment — only billing's own suspensions auto-restore.

> **Setup checklist (once):** Stripe keys in the server .env (`STRIPE_KEY` , `STRIPE_SECRET`), billing enabled in Admin → Settings, and a webhook endpoint in the Stripe dashboard pointing at `https://your-domain/billing/webhook` (events: `checkout.session.completed` , `invoice.paid` , `invoice.payment_failed` , `customer.subscription.updated` , `customer.subscription.deleted`) with its signing secret as `STRIPE_WEBHOOK_SECRET` . The webhook is what marks signups Paid and keeps renewal status honest.

Enquiries

Administration → **Enquiries** lists every contact message and access request submitted on your website — who, their company, fleet size, and what they wrote. **Mark handled** moves one out of the open list; untick **Open only** to see the history.

> **Why it matters:** these are recorded whether or not an email went out, so a misconfigured mailer costs you a notification, never the lead itself. To be alerted as they arrive, add a channel under **Notifications** subscribed to “Contact / access-request submitted on the website”.

Website content & Billing

Website edits the public marketing copy (hero, pricing intro, contact). **Billing** shows your Stripe connection status, currency, and recent payments.

Impersonation ("Login as")

A Super Admin can click the → on a user row to log in as them (in a new tab) for support — every start/stop is audit-logged, and a banner shows you're impersonating. Click **Return to my account** to exit.

11. Integration API

Automate PioDeploy from your own tools. Create a token from your **profile menu** → **API Tokens**, choosing abilities **read** and/or **deploy**.

```
# list computers
curl -H "Authorization: Bearer <token>" https://piodeploy.com/api/v1/computers

# queue a deployment
curl -X POST https://piodeploy.com/api/v1/deployments \
  -H "Authorization: Bearer <token>" \
  -d computer_id=12 -d package_id=7 -d action=install
```

Endpoints cover clients, projects, computers, packages, deployments and policies (read), and queuing deployments (deploy). Everything respects your role permissions and data scope.

12. Troubleshooting & FAQ

SYMPTOM	CAUSE / FIX
Machine doesn't appear under Computers	Agent not installed or wrong API key. Re-run the installer as Administrator with the correct `-ApiKey`.
Deployment failed: "Win32Exception … start process"	Agent couldn't find winget as SYSTEM. Ensure the agent is **v1.3.1+** (re-run the enrolment script to upgrade), then retry the job.
Installed software says **"0 managed"** though the machine clearly has managed apps	The agent's winget scan is failing, so no package ids reach the server and the catalogue cannot match anything. Upgrade the agent to **v1.3.1+**; the count fills in on the next check-in.
Install "Succeeded" but the app is nowhere on the machine	Some winget packages (Brave is the classic case) default to a **per-user** installer. The agent runs as SYSTEM, so the app landed in the SYSTEM account's profile — invisible to every real user, exit code 0. Agents **1.4.1+** force `--scope machine`: it installs for all users, and a package with no machine-wide installer now fails with a clear explanation instead of pretending. Re-run the job after the agent self-updates.
The same package installs over and over, always "Succeeded"	The server now stops this by itself: a machine whose agent can't scan winget counts a succeeded PioDeploy install as "present", and installs are paced to one per policy frequency window regardless. Upgrading the agent (which restores the winget scan) also restores exact version reporting.
No emails arrive at all	Mail is almost certainly not configured. **Administration** → **Email** → pick your provider → **Send test email**. It reports your provider's own error and what to do about it. Until it passes, nothing can send — but nothing is lost either: website submissions are still recorded under **Enquiries**.
Test email fails: `553 5.7.1 Sender address rejected`	Your login was accepted; the **From address** was not. It must be the mailbox you authenticated as, or another address verified with your provider.
Test email fails: `535 Authentication failed`	Several providers refuse your account password here. Gmail needs an App Password (with 2-Step Verification on), SendGrid's username is the literal word `apikey`, Postmark wants its Server API token in *both* boxes, Brevo needs an SMTP key.
"Update available" never appears	The version on offer is reported by the machine's own package manager, so the agent must be **v1.3.3+**. Re-run the enrolment script to upgrade. Binary (MSI/EXE) packages have no package manager to ask, so they never show one.
A rollback fails, or cannot be queued	A rollback needs a version pinned — "roll back to what?" — and that version must still be published. Some vendors (Chrome especially) only publish the current one, which makes rollback impossible however you ask. The version dropdown lists what can actually be installed.

Install script warns "no agent bundle published"	The server hasn't got the agent bundle. Your administrator publishes it once; then re-download the script.
Browser policy shows "pending restart"	The browser was open when the policy applied — it takes effect on next browser restart.
Client user sees "account isn't linked to a client"	The Client-role account has no client binding. Admin: Users → set its Client binding, or change the role.

Is it really silent?

Yes. The agent runs as a SYSTEM service in session 0; every installer runs quietly (winget `--silent`, MSI `/qn`, etc.) with no window. Users are never interrupted.

Do I need a domain?

No. The agent works on standalone and domain-joined machines alike. Push it via GPO, Intune, your RMM, or install manually.

Getting help

Use the **Contact** page on the website, or email your PioDeploy administrator. Include the machine hostname and the deployment job id when reporting a failure.

> **You're all set.** Define desired state once, push the agent silently, and let PioDeploy keep your fleet compliant — invisibly.