How to Unify ConnectWise and NinjaOne Data

Your PSA and RMM each hold half the picture. Here's how to join ConnectWise and NinjaOne data on a shared client key so you can report on health, value, and risk per account — without exporting CSVs.

The short answer

ConnectWise holds your business data — tickets, agreements, time, projects, billing — and NinjaOne holds your technical data — device health, patch compliance, backup status, alerts. Unifying them means joining those two datasets on a shared client identifier so that, for any account, you can see business performance and technical health on the same screen. The hard part isn't pulling from either API; both have capable REST APIs. The hard part is the entity mapping: ConnectWise's "company" and NinjaOne's "organization" are the same client wearing two different names and IDs, and nothing reconciles them for you out of the box. Get that mapping right and maintained, and every cross-system report you've ever wanted — QBRs, health scores, churn signals — becomes possible.

Catalyst OS does this join as the data foundation under every module. This guide explains how the integration actually works so you can evaluate it on the merits.

Why the two systems don't talk

ConnectWise Manage (PSA) and NinjaOne (RMM) are best-of-breed tools built for different jobs, by different companies, with different data models. ConnectWise's native integration with NinjaOne exists, but it's built for *operational* flow — create a ticket in ConnectWise when NinjaOne throws an alert, sync a device list, push asset data. That's valuable and you should use it. What it does *not* give you is a unified analytical layer: a single place where you can ask "for this client, what's their ticket trend, their patch compliance, their backup success, their last QBR, and their renewal date — together?"

That gap exists because the native integration syncs records between systems; it doesn't build a joined dataset you can report across. The two tools stay in their own lanes. To get business-and-technical-together reporting, something has to read both APIs, reconcile the entities, and hold the combined view. That something is the integration layer this guide is about.

The core problem: entity reconciliation

Everything in unifying these two systems comes down to one question: **how do you know that ConnectWise Company #4471 and NinjaOne Organization #882 are the same client?**

The two systems were set up independently. The names rarely match exactly — "Acme Corp" in ConnectWise might be "Acme Corporation" or "Acme" in NinjaOne. The IDs are unrelated. Sites, locations, and sub-organizations multiply the confusion: one ConnectWise company can map to several NinjaOne organizations, or vice versa. There is no shared key handed to you. Building one is the foundational work, and it's where naive integrations fall apart — they fuzzy-match on name, get it 85% right, and quietly mis-attribute the other 15%, which poisons every report built on top.

The reliable approach is an explicit, maintained mapping table: a deliberate, reviewed cross-reference of ConnectWise company IDs to NinjaOne organization IDs, established once with a human confirming the matches, then maintained as clients are added and offboarded. It's unglamorous and it's the whole game. Get the mapping right and everything downstream is clean; get it wrong and no amount of clever reporting saves you.

How the integration works, step by step

Step 1 — Authenticate to both APIs

ConnectWise Manage uses a REST API with public/private key pairs scoped to a member, plus a client ID for the integration. NinjaOne uses OAuth 2.0 with client credentials and scoped tokens. Both are well-documented and stable. You provision read-scoped credentials for each — you're reading data to report on it, not writing back — which also keeps the integration low-risk from a permissions standpoint.

Step 2 — Pull the entity lists from each side

From ConnectWise, pull the company list with their IDs, names, statuses, and agreement data. From NinjaOne, pull the organization list with their IDs and names. These two lists are the raw material for the mapping. Pull the full set, not a sample — the mapping has to cover every active client.

Step 3 — Build the mapping table

Match the two lists into a cross-reference. Start with exact and normalized name matches to handle the easy 80%, then surface the ambiguous remainder — the near-matches, the one-to-many sites, the renamed accounts — for human confirmation. The output is a stored table: `connectwise_company_id ↔ ninjaone_organization_id`, with the match confirmed rather than guessed. This table is the keystone. Treat it as a maintained asset, not a one-time script.

Step 4 — Pull the metrics from each system

With the mapping in place, pull the data that matters per account:

- **From ConnectWise:** ticket counts and trends, response and resolution times, time entries, agreement and billing data, project status, SLA performance.
- **From NinjaOne:** device inventory and health, patch compliance percentages, backup and antivirus status, alert volumes, agent coverage.

Step 5 — Join on the shared key and store the unified view

Using the mapping table, join the ConnectWise and NinjaOne metrics onto a single per-account record. Now "Acme" has one row that carries both its business performance and its technical health. Store this unified view and refresh it on a schedule so it stays current without anyone exporting anything. This joined dataset is the thing the native integration never gives you — and the thing every downstream report needs.

Step 6 — Maintain the mapping as the book changes

New clients get added to both systems; old ones get offboarded; companies get renamed or merged. The mapping table has to keep up, or your reports slowly drift out of sync with reality. A maintained integration re-checks for unmapped entities on each run and flags them for confirmation. This ongoing reconciliation is the difference between an integration that works for one demo and one that works for two years.

What unifying the data unlocks

Once ConnectWise and NinjaOne sit on a shared key, the reports that were previously a half-day of manual CSV work become automatic:

- **Quarterly business reviews** that show business value (tickets, SLAs, projects) and technical health (patch, backup, uptime) for each client, together, generated on a schedule.
- **Churn early warning** that reads engagement signals from ConnectWise alongside health and usage signals from NinjaOne — the cross-system pattern that predicts a client drifting toward the door.
- **Per-client profitability and capacity** views that put agreement value next to the device load and ticket effort actually consumed.

In other words, the integration isn't the point — it's the foundation. Every operational and revenue report an MSP wants is blocked on this join being done correctly and maintained.

Build vs. install

	DIY scripts	Native ConnectWise–NinjaOne sync	Operating-layer integration
Operational sync (tickets, alerts)	Possible	Yes — its strength	Yes
Unified analytical view per client	If you build it	No	Yes
Entity mapping maintained over time	Your job	N/A	Handled and monitored
Survives an API change	You fix it	Vendor-maintained	Handled for you
Powers QBRs, churn, profitability reporting	If you build all of it	No	Yes — it's the foundation

The native integration is the right tool for keeping the two systems operationally in sync — use it. It is not built to give you the joined analytical dataset that cross-system reporting needs. You can build that dataset yourself if you have the engineering capacity to write and maintain the API clients and the mapping logic. Most owner-led MSPs would rather their technical people stay billable. Catalyst OS performs and maintains this join as the data foundation beneath every module, delivered as a managed service — so the unified view is just there, and the reports built on it run whether you're paying attention or not.

Frequently asked questions

Can ConnectWise and NinjaOne data be unified?

Yes. Both systems expose stable REST APIs, so their data can be pulled and joined on a shared client identifier. The work is in entity reconciliation — building and maintaining a mapping that confirms which ConnectWise company corresponds to which NinjaOne organization — because the two systems use unrelated names and IDs. Once that mapping is reliable, business data from ConnectWise and technical data from NinjaOne can sit on one per-account record.

Doesn't the native ConnectWise–NinjaOne integration already do this?

The native integration handles operational sync — creating tickets from alerts, syncing device and asset data between the two systems. It does not build a unified analytical dataset you can report across. For QBRs, health scores, and churn signals that combine business and technical data per client, you need a layer that reads both APIs, reconciles the entities, and holds the joined view. The native sync and the analytical layer solve different problems.

What's the hardest part of integrating PSA and RMM data?

Entity reconciliation — knowing that a given company in the PSA is the same client as a given organization in the RMM. Names rarely match exactly, IDs are unrelated, and one company can map to several organizations. A maintained, human-confirmed mapping table is the reliable solution; fuzzy name-matching alone quietly mis-attributes a meaningful share of accounts and corrupts every report built on top.

Do I need write access to both systems?

No. Unifying the data for reporting only requires read-scoped credentials on both APIs. Read-only access keeps the integration low-risk and is sufficient for QBRs, churn prediction, and profitability reporting, which consume data rather than change it.

What can I build once the data is unified?

Automated QBRs that combine business value and technical health per client, churn early warning that reads engagement and usage signals together, and per-client profitability and capacity views. The unified dataset is the foundation; most cross-system MSP reporting is blocked until this join is done correctly.

Catalyst OS is the business-layer operating system for an MSP — the layer the major MSP vendors have no incentive to build. The PSA-to-RMM join is the data foundation beneath every module. The first conversation is a 30-minute listen, not a pitch.
