How to Build a Customer Success AI Assistant

This Configuration Guide serves as an accompaniment to this video.

If you want to set up a customer success agent that helps your team walk into every renewal conversation prepared, with a complete account brief in minutes instead of hours, this guide walks you through how to think through and build that use case in your own organization.

Why Connect AI? Account intelligence doesn't live in one system—customer data is spread across your CRM, ticketing system, and call platform, which your AI assistant can't query directly. CData Connect AI is the data layer that sits between the AI and your systems: it manages the connections, controls access, and gives the AI a clean, queryable interface to your data.

Throughout this guide, we'll link to Connect AI's documentation for feature setup steps and focus here on the decisions that are specific to this use case: which data matters, where it lives in your organization, and how the pieces fit together.

Step 1: Identify Your Data Points

Before connecting anything, map out the data your assistant will need and where it lives in your organization. This step shapes everything that follows, because the quality of the assistant's briefs depends entirely on the data behind them. Below is the complete inventory of data points from our build. Use it as your starting checklist: for each item, identify which of your systems holds it, what the object and fields are called, and whether the data is actually populated and current.

The data inventory from our build

- **Account fundamentals:** Account name, ARR / contract value, renewal date, risk level, health score
- **Products and licenses:** What the customer owns, quantities, license status, term end dates
- **Usage telemetry:** Raw product usage by account and period — job runs, records processed, query counts, last telemetry event
- **Usage trends:** Rolling multi-month usage snapshots per product, so the assistant can spot declines without computing them from raw data
- **Support cases:** Open and pending cases, priority, age, and any reference linking a case to an engineering ticket
- **Engineering ticket status:** Live status, assignee, priority, fix version, and latest comment for tickets referenced by support cases
- **Contacts and reachability:** Key contacts, roles, last outreach, last response, bounced email flags
- **Engagement activity:** Logged calls, emails, meetings, and internal notes

- **CSM notes:** The Customer Success Manager (CSM)'s own next steps, insights, and health assessment — distinct from the activity log
- **Pre-sales context:** The Solutions Engineer's technical evaluation notes — what was evaluated, the data environment, blockers, and positioning from the pre-sales cycle
- **Call intelligence:** Call briefs, key points, next steps, and transcripts from recorded customer conversations

Custom Objects in Connect AI

Standard CRM objects like Account, Contact, Case, Opportunity, Asset, Task, and Event are available out of the box. Depending on your connector, Connect AI may also support custom objects — in Salesforce, for example, custom objects like usage telemetry, CSM notes, and pre-sales context are queryable through the same interface without any extra configuration beyond connecting the source. Check the documentation for your specific connector to confirm custom object support.

Where does this live in your organization?

In our build, these data points lived across three systems: our CRM (Salesforce), our ticketing system (Jira), and our call intelligence platform (Gong). Your setup will likely mirror this — a CRM, a ticketing system, and a call recording platform if you use one — though the specific systems, object names, and field names will be unique to your organization.

As you map each data point, record the exact object and field API names — Step 4 will ask for them. Two things worth checking while you're in there:

- **Data that exists in name only:** Some of the highest-value signals, like CSM notes, pre-sales context, and usage summaries, often have a home in your systems but are inconsistently filled in. An empty field can't inform a brief. If a data point matters for this use case, confirm it's being captured before you build around it.
- **Data your connections can't see:** Being in the system isn't the same as being visible to it. Permissions, sharing settings, and field-level security can all hide data from a connected app. In Salesforce, for example, custom fields return null to connected apps until field-level security is set to Visible for the relevant profiles. Verify visibility for each field on your list now; it's much easier to diagnose here than after everything is wired up.

Why this step comes first: *The hardest part of this build wasn't configuring the AI — it was deciding which fields matter for CS decisions, then making sure each one existed, was populated, and was visible. Everything downstream inherits the quality of this map.*

Step 2: Connect Your Systems

Once you've identified where your data lives, connect each system to Connect AI.

For connection steps, follow the documentation, which covers every supported source and stays current: Connecting data sources

Connect each of the systems you identified in Step 1. In our example, that meant three connections: Salesforce, Jira, and Gong. Each connected system appears as a named catalog that can be queried with standard SQL. The AI never needs to know each system's native API or query language.

Note: *The name you give each connection is referenced in your AI assistant's instructions in Step 4. Choose clear, stable names, since renaming a connection later means updating the assistant's configuration.*

Step 3: Scope a Workspace

Connecting your systems makes everything queryable; scoping makes it usable. In Connect AI, a Workspace lets you define exactly which tables and objects the assistant can see and query. Without it, the assistant explores full schemas on every request, which costs time and tokens and leaves more room for error.

Create a Workspace adding only the tables you identified in Step 1.

For setup steps, follow the documentation: Creating and managing Workspaces

In our build, we used the following tables:

- **From the CRM:** Account, Contact, Asset, Opportunity, OpportunityLineItem, Case, Task, Event, plus our custom usage, CSM notes, and pre-sales context objects
- **From the ticketing system:** Issues and Comments
- **From the call intelligence platform:** Call transcripts / call records

Connect AI vs individual MCP

Connecting each system to Claude separately with individual MCP connectors is the obvious path. But it creates its own problems: no central governance, inconsistent results across team members, and no support for custom objects that native connectors can't reach

Connect AI solves all three:

- Acts as a single governed data layer
- Manages all connections centrally
- Exposes custom objects alongside standard ones through the same queryable interface.

You can point the assistant at this Workspace by name in Step 4 — its project instructions simply specify which Workspace (or connections) to query.

For IT and security stakeholders: *Connect AI adds a layer of governance over what your AI assistant can access. Permissions and access control let you define what a connected AI can see and query, and access can be reviewed, modified, or revoked in Connect AI without touching the assistant's configuration. All on top of the permissions already enforced by your source systems. Admins who want a hard, tool-level boundary—a fixed set of approved operations the assistant can perform, and nothing else—can define a Toolkit and provide the assistant its endpoint instead. In our testing, tight workspace scoping also reduced token consumption by roughly 97% compared to an unrestricted approach.*

Step 4: Configure The AI Assistant

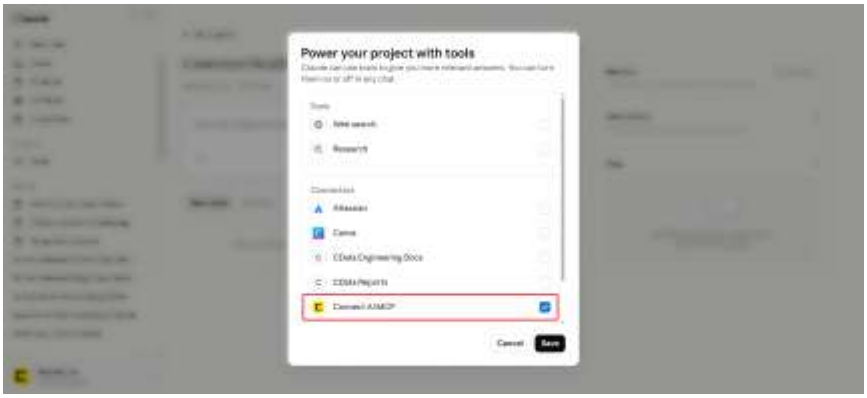
Without structured configuration, the AI would need to be told everything from scratch in every conversation—what counts as a risk signal, how to format the output, where to find the data. Most AI platforms offer a way to encode this once: a persistent assistant configuration with standing instructions and connected tools.

In our build, that was a Claude Project, and the steps below reflect that. But the same approach applies to whichever AI assistant your organization uses, as long as it supports MCP connections and persistent instructions. Paired with Connect AI for live, governed data access, this turns a general-purpose AI into a specialist that's consistent, accurate, and ready to use without setup per session.

Connecting the project to your data takes three steps:

1. **Add the CData Connect AI connector,** available in Claude's official Connectors Directory [Connect AI Claude Client Documentation], and authenticate with your Connect AI account.

2. **Create a new Claude Project** and enable Connect AI as a tool for the project.



3. **Specify the data scope in the project instructions** by naming the workspace from Step 3 (or the specific connections) the assistant should query. The assistant discovers and queries your data through the connector; the instructions tell it where to look.

The substance of this step is the **project instructions**. This is where you translate your Step 1 data map into assistant behavior. Ours define five things—each is a decision you'll make for your own build:

- **Identity and guardrails:** the assistant reports data as-is and applies analytical commentary only in clearly marked sections. It never fabricates: if something isn't in the data, it says so. A gap is information too.
- **Connection and schema map:** your exact connection names, catalog prefixes, and object/field API names from Step 1. This is the part that differs most from our setup, so document it precisely and the assistant will query the right fields every time.
- **Query sequence:** resolve the account name to a stable account ID first, then fan out to every related object in a defined order.
- **Signal logic:** your rules for what constitutes a risk signal. Ours: zero usage, imminent renewal with no logged activity, bounced contact emails, open bugs with no assignee, and discrepancies between CRM case status and live ticket status.
- **Output format:** a fixed structure for every brief. Ours renders a visual dashboard with charts, color-coded signals, a contact table, and three to five tagged action items.

The query pattern

The assistant's queries follow one consistent pattern regardless of which systems you use. Generic names below — substitute your own catalogs, tables, and fields:

```
-- 1: Resolve account name to a stable ID (names are rarely clean)
SELECT [Id], [Name], [HealthScore], [RenewalDate], [RiskLevel]
FROM [YourCRM].[Schema].[Account]
WHERE [Name] LIKE '%{account_name}%' LIMIT 1

-- 2: Fan out to related objects using the resolved ID
SELECT [Id], [ProductName], [Status], [LicenseKey], [TermEndDate]
FROM [YourCRM].[Schema].[Asset]
WHERE [AccountId] = '{account_id}'
```

```
-- 3: Pull usage telemetry
SELECT [Period], [JobRuns], [RecordsProcessed], [LastTelemetryDate]
FROM [YourCRM].[Schema].[UsageData]
WHERE [AccountId] = '{account_id}'
ORDER BY [LastTelemetryDate] DESC LIMIT 12

-- 4: Pull open cases and their engineering ticket references
SELECT [CaseNumber], [Subject], [Status], [TicketReference]
FROM [YourCRM].[Schema].[Case]
WHERE [AccountId] = '{account_id}'
  AND [Status] IN ('Open', 'Pending')

-- 5: Cross-reference the ticketing system for live status
--    (switching data source: CRM -> ticketing system)
SELECT [Key], [Summary], [StatusName], [PriorityName],
       [AssigneeDisplayName], [FixVersions], [Updated]
FROM [YourTicketingSystem].[Schema].[Issues]
WHERE [Key] = '{ticket_reference_from_case}'

-- 6: Pull the latest comment on that ticket
SELECT [IssueKey], [Body], [Author], [Created]
FROM [YourTicketingSystem].[Schema].[Comments]
WHERE [IssueKey] = '{ticket_reference_from_case}'
ORDER BY [Created] DESC LIMIT 1

-- 7: Pull recent call intelligence for the account
--    (switching data source: ticketing -> call platform)
SELECT [Title], [CallStart], [CallDuration], [CallBrief],
       [CallKeyPoints], [CallNextSteps], [Transcript]
FROM [YourCallPlatform].[Schema].[CallTranscript]
WHERE [AccountId] = '{account_id}'
ORDER BY [CallStart] DESC LIMIT 5
```

Optional: Start with our skill template

Rather than writing project instructions from scratch, you can download the skill template we packaged from this build. It includes the query sequence, schema map structure, and output format. Fill in your connection names and field API names from Steps 1–3 and upload it to your Claude Project: account-intelligence.skill

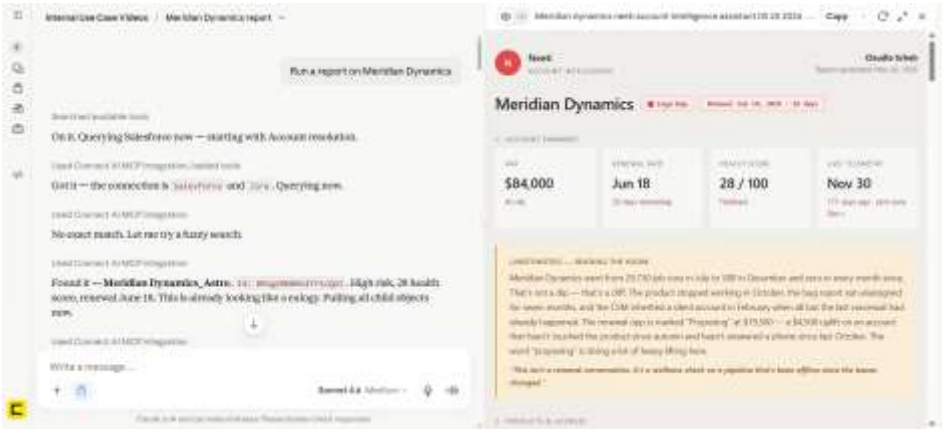
Note: *Requires Claude Pro, Max, Team, or Enterprise with code execution enabled. The schema map section will need the most adaptation as every organization's object and field names are different.*

Step 5: Test Against an Account You Know

The output format you defined in Step 4 is a deliberate design decision: separating raw data from analysis, rendering a visual structure rather than plain text, and always ending with specific

tagged action items. Testing against an account you know well is how you confirm that design is working — not just that the data is correct, but that the brief is actually usable in a pre-call context.

Type an account name into the project. The assistant resolves it, queries every connected source, and returns a full brief. Verify it against an account you know well:



- **Numbers, priorities, and dates** should match exactly what you'd find manually in each source system.
- **Gaps should be flagged, not skipped** — missing pre-sales notes, an empty CSM card, or unreachable contacts should surface as signals.
- **Null values you didn't expect** usually mean a field-visibility problem — check field-level security in your source system before debugging anything else.

The output is the same structure every time, which means CSMs can build a consistent pre-call routine around it, and CS leaders can read any account brief the same way, regardless of who ran it.

Things We Learned Building This

- **Resolve by ID, not by name.** Account names are rarely clean. Resolve a fuzzy name match to a stable ID first, then fan out.
- **Name the gaps.** The assistant flagging missing data — no SE notes, no logged outreach — turned out to be one of its most valuable outputs.
- **Discrepancy detection is a feature.** When the CRM says a case is pending a bug fix but the linked ticket is unassigned with no fix version, that cross-system gap is operationally important. Build the assistant to look for it.
- **Field-level security matters.** If the AI returns nulls for fields you know exist, check field visibility for connected apps before debugging anything else.

Get started with a *free trial* of CData Connect AI.