RITHMO GUIDE

Agent Ready

Ten things that should be true before an AI agent takes consequential action on your behalf

August 2026 / About 16 minutes

Most agent deployments that go wrong do not go wrong at the model. They go wrong upstream, in the context the agent is handed before it acts, and in the controls around what it is allowed to do once it has acted on that context.

This guide is the checklist. Ten items, in three tiers.

Tier 1 is control: who the agent is, what it can do, what you can see, and how you stop it. Tier 2 is grounding on authored facts: the documents, records, and schemas your company already keeps. Tier 3 is grounding on resolved facts: what the company has actually decided.

Consequential is the operative word. A read-only research agent does not need this posture. An agent that writes, sends, spends, commits, or speaks to a customer does.

The first six are solvable with tooling most companies already own, and the tooling is getting better quickly. This guide gives specific instructions for them and names no vendors. The last four are where the standard agent stack starts to run out. It can govern the agent well. It usually cannot tell the agent what the company has actually decided.

Each item has three parts: what it means, a test you can run this week, and what to do if you fail the test.

Nothing here is an argument against deploying agents. It is an argument for knowing which of the ten you have before one of them touches a customer.

TIER 1

Control

Who the agent is, what it can do, what you can see, and how you stop it.

01

Identity

What it means. The agent authenticates as its own principal. Not a shared service account. Not a borrowed token belonging to whichever engineer set it up.

This is the item most often skipped, because borrowing a human's credentials is the fastest path to a working prototype, and prototypes become production. Two things break when an agent runs on a human's identity. It inherits every permission that human holds, which is almost always far more than the task requires. And every action it takes reads in the audit log as that person, which means you lose the ability to tell agent activity from human activity at exactly the moment you need to.

> TEST
>
> Pick one action an agent took last week. Open your identity provider logs. Can you tell it was the agent and not a person? Then, without looking anything up, list every system that agent can currently reach. If either answer takes more than two minutes, this item fails.

WHAT TO DO

- One principal per agent, per environment. The staging agent and the production agent are different identities.
- Use workload identity or OAuth client credentials, not static keys pasted into environment variables. Static keys can be rotated, but in practice they often lack automatic rotation, are trivially copied, and live far longer than anyone intended.
- Scope to the minimum resource set the task requires, not the minimum role your IdP offers. "Read access to the CRM" is a role. "Read access to open opportunities in two regions" is a scope.
- Short credential lifetimes with automatic rotation.
- Put agent principals in their own group with their own conditional access policy, separate from staff.

> **What good looks like.** You can revoke one agent's access completely, in under a minute, without affecting a single human's ability to work.

02

Action boundaries

What it means. Before the agent runs, you have decided which classes of action it may perform unattended, and enforced that in the tool layer rather than the prompt.

A useful classification, in ascending order of consequence:

Class	Example	Default posture
Read	Query a record, fetch a document	Unattended
Reversible write	Update an internal field, create a draft	Unattended with ceiling
Irreversible write	Delete, close, merge, submit	Confirm
External-facing	Send an email, post to a customer channel, file with a regulator	Confirm
Financial	Issue a credit, place an order, move money	Confirm, dual control above a threshold

Volume matters as much as class. An agent that can send one email can send ten thousand. Boundaries without ceilings are not boundaries.

> TEST
>
> Ask the person who owns the agent: what is the largest irreversible thing this could do in the next hour if its input were wrong? If nobody can answer in a sentence, that is the finding. You do not need to run anything else.

WHAT TO DO

- Allowlist tools by action class. An agent scoped to read should not have a write tool registered at all, rather than having one it is instructed not to use.
- Enforce confirmation in the tool wrapper, not in the system prompt. Prompt-level constraints degrade under unusual input.
- Set hard ceilings per hour and per run: records touched, messages sent, dollars committed.

- Build a kill switch that is a configuration flag, not a code deploy, and make sure a non-engineer can flip it.

03

Observability

What it means. You can reconstruct not only what the agent did, but why. That requires logging the decision path: the input, the context it retrieved, every tool call with its arguments, the output, and a trace identifier tying all of it together.

Most teams have logs. Far fewer have observability. The difference shows up during an incident, when the question is not “what happened” but “what did it think was true.”

> TEST
>
> Pick one agent action from last week. Reconstruct why it took that action. Time yourself. If it takes more than fifteen minutes, or requires an engineer to query a database by hand, this item fails.

WHAT TO DO

- Emit one structured trace per run, with a stable trace ID carried through every downstream call.
- Retain enough immutable evidence to reconstruct the context the agent acted on, rather than a pointer to it. Sources change, and a pointer to a document that has since been edited tells you nothing about what the agent read. A content hash plus a redacted copy is usually sufficient and safer than storing the raw payload.
- Capture tool call arguments, with credentials, personal data, and regulated fields tokenized or redacted at the point of capture. The arguments are usually where the failure is visible, which is exactly why they are also where the sensitive data is.
- Treat the trace store as a system holding production data, with the same access controls, residency requirements, and retention limits as the sources it draws from. An observability layer that quietly accumulates unredacted customer data is a new liability, not a control.
- Set retention to cover your incident discovery window, which is longer than you think, and any regulatory obligation that applies to the underlying data.
- Sampling is acceptable for high-volume read traffic. Never sample irreversible or external-facing actions.

04

Rollback and containment

What it means. A named human can stop the agent, and there is a written undo path for each class of action it can perform, defined before launch rather than improvised during an incident.

> TEST
>
> Run a drill. Have someone who is not the agent's author stop it mid-run, reverse every reversible action it took, and walk the written remediation path for the irreversible ones. Time it. Note every step that required asking someone a question.

WHAT TO DO

- Name an on-call owner. Not a team. A person, with a rotation.
- Write the undo path per action class. Some actions have none, and discovering which ones during a drill is far better than discovering it during an incident.
- Build the kill switch at two levels: application flag, and credential revocation. Application-level shutoff requires the application to be healthy, which is not a safe assumption at the moment you need it.
- Know the blast radius in numbers: how many records, how many customers, in what window, at current throughput.

TIER 2

Grounding on authored facts

The documents, records, and schemas your company already keeps.

05

Authored facts

What it means. Authored facts are the ones that arrive with their own artifact. A contract, a policy document, a price list, a database schema, a ticket, a signed order form. Somebody wrote them down, and the writing down is what made them official.

This half of context is well served. Retrieval, search indexes, and direct queries against systems of record handle it. Most of what is published as "AI readiness" advice is about this tier, and the advice is largely correct.

> TEST
>
> Ask the agent a question whose authored answer changed thirty days ago. Then ask one whose answer changed three days ago. Then ask one where two documents in your environment disagree, which is more common than most teams expect.

WHAT TO DO

- Query the system of record live wherever one exists. Retrieval over a CRM should hit the CRM, not a vector snapshot of it taken last Tuesday. Hybrid retrieval over authoritative content is fine, and often better. The question to ask of any index is not whether it uses embeddings, but how stale it is and whether it can tell you where an answer came from.
- Resolve duplicates before indexing. Two versions of the same policy in two folders will both be retrieved, and the index has no opinion about which one is current.
- Attach provenance to every retrieved chunk: source system, document identifier, last modified timestamp. An answer without provenance cannot be checked.
- Exclude drafts, deprecated folders, and personal drives explicitly. Anything reachable will eventually be retrieved.

06

Freshness

What it means. Every index has a lag between the source changing and the agent seeing the change. Readiness means knowing that number per source, rather than assuming it is small.

TEST

Build this table for your environment and fill in the measured columns, not the intended ones.

Source	Sync mechanism	Lag P50	Lag P95	Agent behavior during the gap
CRM	Live query	Seconds	Seconds	Correct
Policy library	Nightly batch	12 hours	26 hours	Acts on yesterday's policy
Contract store	Weekly crawl	3 days	8 days	Acts on last week's terms

Most teams have never measured P95 and are surprised by it. The failure mode is not the average case. It is the run that lands inside the tail.

WHAT TO DO

- Move high-stakes sources to event-driven invalidation. When the source changes, the index is told, rather than waiting for the next crawl.
- Set a staleness budget per source and fail closed when a source exceeds it. An agent that stops is cheaper than an agent that proceeds on stale terms.
- Expose the lag to the agent so it can qualify its own output, and to the operator so the number is visible rather than assumed.

TIER 3

Grounding on resolved facts

What the company has actually decided.

This is where readiness checklists usually stop, and where the exposure actually sits.

07

Resolved facts

What it means. Resolved facts are the ones that become true because people decided them. The pricing exception approved on a call. The launch date moved in a thread. The vendor chosen, the scope cut, the condition met.

They are structurally different from authored facts in one respect that matters enormously: the act of deciding produces no record of itself. An authored fact and its artifact are the same event. Someone writes the contract, and the writing is what makes it binding. A resolved fact becomes true the moment people agree, and nothing is created at that moment.

Records do sometimes appear afterward. An approval gets logged in a ticket, a stage moves in the CRM, someone writes a summary. Every one of those is a separate, discretionary act performed by a person who chose to do it, downstream of the decision and after the fact. That is why coverage is always partial and always uneven, and why the absence of a record tells you nothing about whether a decision was made.

This is why retrieval does not solve it. Retrieval will find the thread. The thread also contains the option that was argued for and abandoned, the objection that was raised and overruled, and the tentative version that was later revised. Retrieval has no view on which turn in that thread was the moment it became true, or whether it is still true now. An agent handed the thread has been handed the argument, not the outcome.

For years this was a human problem, and the human solution was to ask around. An agent can be told to ask. What it cannot do is recognize that there is something to ask about, because the thing it is missing left no trace of its own absence.

> **TEST**
>
> Name a decision your company made in the last two weeks that changed direction. Now name the system a newly deployed agent would read to learn it. Not the person who would know. The system.
>
> Most companies cannot answer, and the honest answer is that a new agent would not learn it at all.

WHAT TO DO WITHOUT A DEDICATED SYSTEM

This is genuinely harder than items 1 through 6, and it is worth being direct that the manual version decays. It still beats nothing:

- Maintain a single append-only decision record with a fixed schema: the decision stated as a sentence, the owner, the date it became true, its current status, what it supersedes, and a link to the source discussion.
- Never edit a record. Supersede it with a new one. Editing destroys the history an agent needs to reason about currency.
- Pick the two or three recurring meetings where direction actually changes and make written decision records a required output of each.

- Point the agent at the decision record, not at the meeting transcripts. Transcripts are the raw material, and handing raw material to an agent recreates the problem.

The failure mode of the manual approach is predictable. Coverage is partial, and partial coverage is worse than none for a system of record, because the agent cannot tell the difference between "no decision was made" and "the decision was made and nobody logged it." That distinction is item 9.

08

Supersession and latency

What it means. A decision reversed at nine in the morning can still look true in every downstream system until something catches the reversal. The interval between the moment people knew and the moment the agent knew is your exposure window.

Before agents, this window cost meetings. A few people acted on the old version for a day, someone noticed, it got corrected. With agents in the loop, the same window is traversed at machine speed and at whatever concurrency the agent is running.

> TEST
>
> Instrument this once, by hand. Take three decisions from the last quarter that reversed or materially changed. For each one, establish two timestamps: when the change was actually settled among the people in the room, and when the system an agent would read reflected it. The gap is the number.
>
> Then multiply by throughput. If your exposure window is six hours and your agent processes two hundred items an hour, twelve hundred items are the scale of the question. Use your own numbers. Nobody else's benchmark tells you anything about your environment.

WHAT TO DO

- Put status on every decision record, not merely existence. Current, superseded, proposed, confirmed, open. An agent reading a decision without a status is reading a rumor.
- Make supersession explicit and linked, so the chain from the current version back through what it replaced is traversable.
- Treat detection latency as an operational metric with an owner, alongside uptime.

09

Refusal

What it means. The agent can distinguish between "no decision exists on this" and "I found something nearby." It must be able to assert the first.

An agent that assembles a confident answer out of adjacent context is more dangerous than one that stops, because the confident answer is indistinguishable from a correct one at the point of use. Stopping is visible. Being wrong quietly is not.

> **TEST**
>
> Ask the agent about something in your business that has genuinely never been decided. A pricing question nobody has settled. A process nobody has defined. Watch what comes back. If it produces a plausible, confident answer built out of related material, it will do exactly that in production, on the questions that matter.

WHAT TO DO

- Make refusal a first-class return value from the grounding layer, not an instruction in the prompt. Prompt-level instructions to say "I don't know" degrade under pressure from a confidently phrased question.
- Set the threshold so that a partial match returns nothing rather than the nearest thing. Every miss should be a safe miss.
- Track the refusal rate as a metric. A grounding layer that never refuses is not being honest with you, and the rate itself tells you where coverage is thin.
- Route refusals to a human, with the question attached. A refusal that goes nowhere is a dead end. A refusal that gets a person to resolve the decision is how coverage improves.

10

Authorization trail

What it means. After the fact, you can show that the agent acted on direction that was current, owned, and authorized at the moment it acted.

This is not item 3 restated. Observability shows what the agent did and what it believed. The authorization trail shows it was entitled to believe it: who decided the thing it acted on, when, and that the decision was still current at execution time.

The reason this now matters outside of engineering is that organizations are increasingly expected to show that AI-driven actions were governed, traceable, and attributable to a human authority. Standards bodies have converged on the same idea from different directions, and it is the dimension of agent governance that current tooling covers least well. Being in command of your agents is a claim that has to be evidenced rather than asserted.

> **TEST**
>
> Have someone play auditor. They pick one agent action from six months ago. You produce the decision it acted on, the person who owned that decision, and evidence it was current at that moment. Watch how far you get.

WHAT TO DO

- Capture the state of the grounding at execution time, immutably, alongside the action itself. Reconstructing it later from current state does not work, because current state has moved.
- Bind owner and timestamp to every decision the agent can act on, so entitlement is answerable per action rather than per policy.
- Keep the trail for as long as the underlying obligation runs, which is usually longer than your application logs are retained.
- Report at the level of the system, not the individual. What you need to demonstrate is that the process was in control. A list of which people were slow is not that, and building one changes how people behave around the record in ways that make the record worse.

The readiness table

Score each item honestly. Ready means you passed the test as written, not that you have a plan.

#	Item	Tier	Covered by standard tooling
1	**Identity**	Control	Yes, IdP and secrets management
2	**Action boundaries**	Control	Yes, tool layer design
3	**Observability**	Control	Yes, tracing and logging
4	**Rollback and containment**	Control	Yes, operational practice
5	**Authored facts**	Grounding	Yes, retrieval and live queries
6	**Freshness**	Grounding	Mechanisms yes, measurement rarely done
7	**Resolved facts**	Grounding	No authoritative system of record exists
8	**Supersession and latency**	Grounding	Mechanisms yes, but only once the decision exists as state
9	**Refusal**	Grounding	Agents can refuse, nothing tells them there is nothing to find
10	**Authorization trail**	Grounding	Agent-side audit yes, link back to the business decision no

The pattern in the right-hand column is the point of this guide. Items 1 through 6 are a resourcing question. Your team can close them with tools you already pay for, the vendor offerings are maturing fast, and most published advice on agent readiness will help.

Items 7 through 10 are a different kind of gap, and the improvement in items 1 through 6 makes it more visible rather than less. The agent stack is getting very good at establishing who the agent is, what it may do, and what it did. None of that answers what the company has actually decided, which is the input every one of those controls is applied to.

The reason is structural. The resolved half of business context creates no record of itself, so a system built to index records cannot hold it. A decision is too durable to live in an agent's run state and too unstructured to live in a context layer built for documents. It falls between them, and most stacks never notice.

Why these controls, and where they come from

This guide is a practitioner's checklist rather than a standards document, but the structure of it is not invented. Two reference points are worth reading directly.

NIST NCCoE, "Accelerating the Adoption of Software and Artificial Intelligence Agent Identity and Authorization" (concept paper, February 2026). Organizes agent controls across four dimensions: identification, authorization, auditing, and non-repudiation. It defines non-repudiation as accountability that ties an agent's actions back to the human authority that sanctioned them, which is item 10 of this guide stated in standards language. The paper's position is that existing identity standards should be extended to agents rather than replaced, and it flags multi-hop delegation as unresolved. Published alongside the broader AI Agent Standards Initiative launched by NIST's Center for AI Standards and Innovation in February 2026.

The EU AI Act. For systems falling in its high-risk category, it imposes record-keeping and human oversight obligations. Whether or not a given deployment is in scope, the direction of travel is the same one the NIST work points at: an organization should be able to show that an automated action was overseen, logged, and attributable.

On the commercial side, agent identity and governance tooling has matured quickly. Microsoft Agent 365, now generally available, assigns each agent its own directory identity with lifecycle management, access packages, policy templates, and unified audit logs. AWS and others are building in the same direction. This is genuinely good news for items 1 through 4, and it is the reason this guide does not claim those items are unsolved.

It is also the reason the last four stand out. The control plane is arriving. The thing being controlled, what the business has decided, is still missing.

ABOUT RITHMO

Rithmo is the resolved context layer in an AI stack. It keeps a live picture of where a business actually stands.

Every company runs on two kinds of places. The places where things get discussed and settled, and the places where things get written down. Those two are never in sync, and the distance between them is where direction goes stale. Rithmo continuously reconciles one against the other, so that what has been decided and what is recorded stay the same thing.

The result is a single current source of truth about the state of the business, with an owner and a date behind every part of it. Teams work from it instead of asking around. Agents work from it instead of guessing, and it tells them plainly when something has not been decided yet.

It runs inside the customer environment, on-premise or in their own cloud, or in Rithmo Cloud. No customer data is held externally.

The technical treatment of why resolved context is structurally different from authored context is published in full in The Resolved Half.

Size your own exposure window hello@rithmo.ai