FREE BUILD GUIDE

Build your first AI employee

A step-by-step guide to one agent that runs on a schedule while you sleep, proves what it did, refuses to do it twice, and stops when you tell it to.

WHAT YOU WILL HAVE AT THE END

- One agent, live in the cloud, running on its own schedule
- Proof of every run it made, including the runs that failed
- A stop button, and a canary that keeps it away from customers

NINE SESSIONS · 24 PAGES · NO PRIOR EXPERIENCE ASSUMED

Roman Bediner · romanbediner.com/resources/agentic-ai-employees

HOW TO USE THIS GUIDE

Nine sessions. One at a time.

Open this beside a build assistant such as Claude, and work one session per sitting. Each one ends with something you can check with your own eyes. Do not read ahead and do not batch them: the order is what keeps the agent safe.

0	**The interview**	Answer ten plain questions. Nothing gets built.	p.4
1–2	**Plan and paperwork**	Agree what it does before any code exists.	p.6
3	**Accounts and keys**	Your turn. The only step you do alone.	p.8
4	**The empty run**	Deploy something that does nothing, on a schedule.	p.9
5	**The four controls**	Claim, ceiling, record, stop button.	p.13
6–7	**The real job, then break it**	Confined output, then deliberate failure.	p.18
8–9	**Gate it and promote it**	Make the rules mechanical, then widen the audience.	p.20

IF YOU BUILD ONLY FOUR THINGS, BUILD THESE

01 A claim written before it acts, so it cannot run twice
02 A declared limit on how far and how loud one run can go
03 A record of every run, and one separate job that reads them
04 A stop button that parks this agent without a deploy

FIRST, THE WORD ITSELF

What “autonomous” actually means

Most people who say they built an agent built a chat window they still have to open. An autonomous agent is different in four specific, checkable ways. If any one of these is missing, you have an assistant, not an employee.

01

It starts without you

A schedule or an event fires it. Nobody opens a tab, nobody types a prompt, nobody remembers.

02

It runs somewhere else

On a cloud machine that is awake when your laptop is shut. If it only runs on your computer, it stops when you close it.

03

It leaves proof

Every run writes a record: what it read, what it did, what it delivered, and what went wrong. You can answer “what happened at 6am?”

04

It can be stopped

One switch parks it without a deploy, and the difference between parked, broken and healthy is visible.

	A CHAT ASSISTANT	AN AUTONOMOUS AGENT
Starts	You open it and type	A cron schedule fires it at 06:00
Runs on	Your laptop, while you watch	A cloud function, while you sleep
Evidence	Scrollback you will lose	A run record you can query
Failure	You notice, eventually	It alerts a named person, same day
Stopping	Close the tab	A switch that parks it and says so

The rest of this guide is how you get each of those four properties, in the order that makes them safe. The single most common failure is building the clever part first and the evidence last, which produces an agent nobody can trust and nobody can debug.

SESSION ZERO · 20 MINUTES

0 Let it interview you

You do not need to know what to build in technical terms. Download the Agent Builder Starter Prompt, paste the whole thing into your build assistant, and answer its questions in plain English. It fills in the technical contract itself.

DO THIS FIRST

1. Open romanbediner.com/resources/agentic-ai-employees
2. Download 'Agent Builder Starter Prompt' (a .md text file)
3. Open Claude, ChatGPT, or your build assistant of choice
4. Paste the ENTIRE file as your first message
5. Answer the questions it asks, one at a time

YOU SHOULD SEE

- A reply under six lines that introduces itself and asks ONE question.
- No summary of the prompt, no lecture, no list of everything it plans to do.
- If it dumps a wall of text or asks five questions at once, say: "Re-read YOUR FIRST REPLY and try again."

WHAT IT WILL ASK YOU

01 What repetitive job do you want off your plate, and who does it today?

02 Walk me through how they do it now, step by step.

03 Where does the information come from? Name the system.

04 What does the finished thing look like, and where does it need to land?

05 Who checks it before it counts as done?

06 How often should it run, and what breaks if it ran twice one morning?

07 Who is allowed to see the output? Could it ever reach a customer?

08 What must it never do, even when that would seem helpful?

09 If it misbehaves overnight, who notices and what can they do?

10 What do you want to call it?

THERE ARE NO WRONG ANSWERS HERE

"I don't know" is a valid answer to any of these. It will tell you what it would assume instead, and mark that row ASSUMED so you can see it later.

SESSION 0 OUTPUT

Read it back before it builds

When the interview ends it fills in the contract and shows it to you. Every row is labelled DECIDED (you said it) or ASSUMED (it worked it out). Read the ASSUMED rows. That is the whole point of this page: you see what you chose and what was chosen for you.

FIELD	WHAT IT WILL BUILD	SOURCE
Name	Scout, Inbound Inquiry Analyst	DECIDED
The job	Summarise yesterday's inbound inquiries so none sit unanswered	DECIDED
Source	One read-only query on the shared inbound mailbox	DECIDED
Output	One draft brief: who wrote in, what they asked, suggested next step	DECIDED
Runs	Weekdays 08:00 local, held at 08:00 across both DST states	DECIDED
Who checks	Head of Sales approves, rejects, or revises	DECIDED
Never does	Reply to an inquirer. Edit the CRM. Spend money.	DECIDED
Tools not built	No send-to-external tool. No CRM write tool. No payment key.	ASSUMED
Audience	team - the owner and one internal channel, never a customer	ASSUMED
Max actions	1 delivery per run, because the job produces exactly one brief	ASSUMED
Runs once per	local business day, claimed before it sends anything	ASSUMED
Stop button	An env flag the operator sets with no deploy	ASSUMED

THE ONE THING TO CHECK HERE

Every ASSUMED row is a decision made on your behalf. If one is wrong, say so now: changing 'audience' after it has already emailed a customer is an incident, not an edit.

SAY THIS TO MOVE ON

"Confirmed. Audience should be team-only, and never a customer. Start Session 1."

Nothing gets built until you say something like that.

SESSION 1 · 20 MINUTES

1 Get the plan in plain English

No code yet. You are checking that it understood the job, and finding out what you need to go and get before anything can run.

SAY THIS

```
Say:  "Give me Stage 1. Plain English only, no code."

Ask it to end with the full list of accounts and access I
need to create, because that is the next session and I
want to do it in one sitting.
```

YOU SHOULD SEE

- What it will build, in language you could repeat to a colleague.
- A rough running cost per month, and what drives it.
- A numbered list of accounts, keys and permissions YOU must create.
- No files written yet, and no code shown.

RED FLAGS AT THIS STAGE

- It starts writing code. Stop it: "Stage 1 is the plan. No code."
- It plans two jobs, or a second agent. Cut it back to one.
- It cannot say what it needs from you. It has not thought it through.
- It skips the cost question. Ask directly, before you build.

YOUR STOPPING RULE FOR THE WHOLE GUIDE

If you cannot explain, out loud, what this agent does and who reads its output, do not continue to Session 2. Every control after this point assumes that sentence exists.

SESSION 2 · 15 MINUTES

2 Four documents, written before the code

These are not bureaucracy. They are the only reason a change six weeks from now can be judged right or wrong. Ask for all four plus docs/launch-gate.md, then read the PRD.

`docs/PRD.md`

What it does, for whom, and what it must never do.

Read it and ask: Could a stranger build the right thing from this?

`docs/architecture.md`

Trigger, data path, state, tools, authority boundaries.

Read it and ask: Where does each piece of information live?

`docs/runbook.md`

How to operate it, read an alert, and park it.

Read it and ask: Could someone else fix it at 7am without you?

`.agent/personality.md`

Its voice, its limits, and what it must escalate.

Read it and ask: Would you recognise its writing as its own?

YOU SHOULD SEE

- Four files that exist on disk, not four descriptions of files.
- A PRD you can read in two minutes and disagree with.
- Your ASSUMED rows from Session 0 written down as real decisions.

SAY THIS IF THE PRD READS LIKE MARKETING

"Rewrite the PRD so a stranger could build the right thing from it and a reviewer could reject a change that breaks it. Be specific about what it must never do."

SESSION 3 · 30 MINUTES · YOUR TURN

3 Accounts, keys and permissions

This is the one session the assistant cannot do for you. It will give you numbered steps for each system. Do them one at a time and confirm each before asking for the next.

SAY THIS

```
Say:  "Stage 3. One system at a time. Give me numbered steps
       with the exact page and button names. Wait for me to
       confirm before moving to the next system."
```

THE THREE RULES THAT KEEP THIS SAFE

Never paste a key into the chat
Put it in the platform's secret store. If it asks you to paste one, say no and ask where it belongs instead.

Give it the narrowest access that works
Read-only on one folder or one mailbox, not your whole account. You can widen it later; you cannot un-leak it.

Make it its own identity
Its own service account or bot, never your personal login. If its credential is missing it must fail loudly, never fall back to acting as you.

YOU SHOULD SEE

- Each key stored in the cloud platform's secret manager, not in a file and not in chat.
- A short list in the runbook naming who owns each key and when it gets rotated.
- The key values themselves written down nowhere, including the runbook.

IF YOU GET STUCK HERE, YOU ARE NOT BEHIND

This session defeats more first builds than any other. If a permission screen does not match the steps you were given, paste what you actually see and ask it to re-derive the steps from that. Do not guess and do not grant broader access to move on.

SESSION 4 · 30 MINUTES

4 Deploy something that does nothing

The most skipped step, and the one that decides whether the rest of this works. You are going to put an agent in the cloud that reads nothing and produces nothing, purely to prove the schedule fires, the identity works, and the run leaves a record.

SAY THIS

```
Say:  "Stage 4. Deploy a scheduled job that does no real work.
       It must: fire on the schedule, write one run record, and
       stop. No sources, no model call, no output. Then tell me
       how to trigger it manually so I don't wait until tomorrow."
```

YOU SHOULD SEE

- A URL or dashboard where the job is registered, with its schedule visible.
- A run record you can open with your own eyes: a run ID, a start time, an outcome.
- A second run record after you trigger it manually.

WHY THIS PAGE EXISTS

Almost everyone builds the clever part first and discovers three days later that the schedule never fired, or that the agent had no permission to write anything, or that nothing anywhere recorded that it ran. Those are infrastructure problems wearing an AI costume. Separate them now, while there is nothing else to blame.

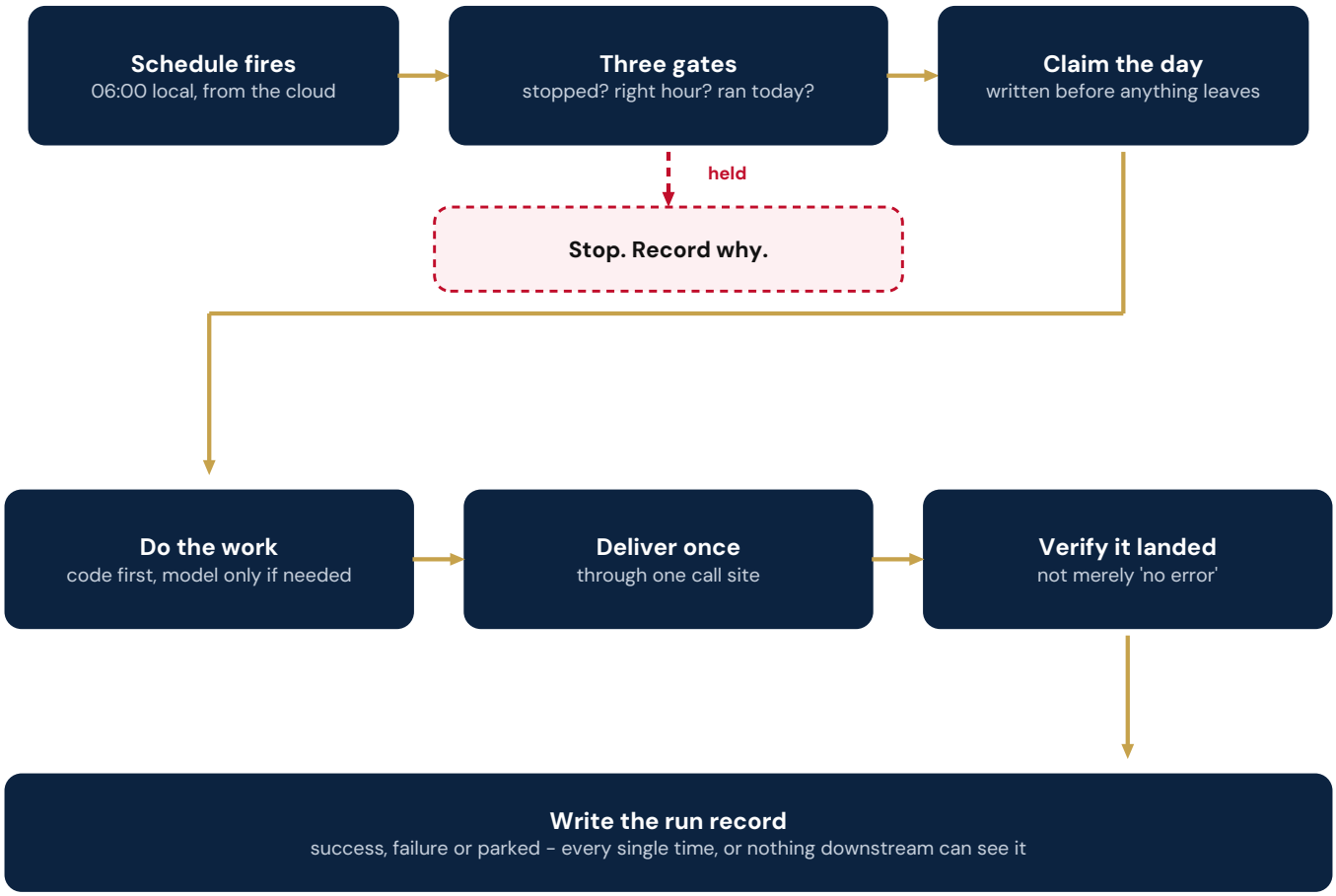
DO NOT CONTINUE UNTIL YOU HAVE SEEN A RUN RECORD

Not a log line that scrolled past. A durable record you can go back and open tomorrow. If you cannot find one, the agent has no memory of its own life and every later chapter of this guide is unenforceable.

THE SHAPE OF EVERY RUN

What happens, in order, every time

Three gates run before the agent is allowed to touch anything real. The order is the safety property, not a style preference.




```
the three gates run BEFORE the source is read. move one of them later and you have built an agent that
can act twice, act while parked, or act an hour early.
```

the run continues · the run stops, and says why · a step that always happens

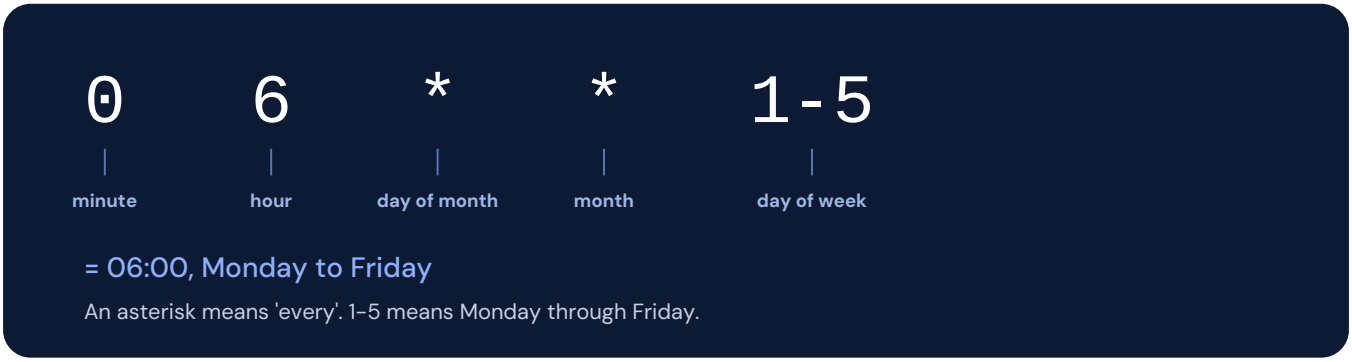
TWO BOXES PEOPLE LEAVE OUT, AND REGRET

Verify it landed: a model that finished without erroring has not delivered anything, so check the thing exists where it was meant to go. And write the record even when the run failed – especially then. A failed run that leaves no trace is indistinguishable from a run that never happened.

MAKING IT RUN ITSELF

Cron: the five numbers that start it

A cron expression is how you tell a cloud platform when to run something. It is five fields separated by spaces. This is the entire syntax.



YOU WANT	YOU WRITE	READS AS
Every weekday at 6am	0 6 * * 1-5	minute 0, hour 6, Mon-Fri
Every day at 6am	0 6 * * *	minute 0, hour 6, every day
Every hour, on the hour	0 * * * *	minute 0 of every hour
Every 15 minutes	*/15 * * * *	every 15th minute
First of the month, 9am	0 9 1 * *	day 1, hour 9

ALMOST EVERY CLOUD SCHEDULER RUNS ON UTC, NOT YOUR TIME

That single fact is the most common reason a first agent quietly runs at the wrong hour for half the year. The next page is entirely about it. Do not set a schedule until you have read it.

WHERE THE SCHEDULE ACTUALLY LIVES

- In your cloud platform's configuration file, committed to version control - never typed into a dashboard and forgotten. A schedule nobody can find is a schedule nobody can fix.
- Named in one place your assistant can list back to you, so you can ask "what is scheduled to run, and when?" and get a complete answer.
- With a manual trigger alongside it, so you never have to wait until tomorrow to test a change.

THE TRAP THAT GETS EVERYONE

Your 6am becomes 7am twice a year

Cloud schedulers fire on UTC. Your clock changes twice a year. UTC does not. So a job pinned to one UTC hour silently drifts by an hour every spring and autumn.

WHAT PEOPLE WRITE

```
0 11 * * 1-5
```

"11:00 UTC is 6am my time." True in winter. In summer it fires at 7am and nobody notices for months.

WHAT ACTUALLY WORKS

```
0 11 * * 1-5
0 10 * * 1-5
```

Register both. The job checks the local hour when it wakes and does nothing on the wrong one.

SAY THIS

```
Say:  "Register the schedule at BOTH the standard-time and
       daylight-time UTC hours, and have the job exit doing
       nothing when it wakes on the wrong one. Log which
       expression fired in the run record."
```

TEST THE SCHEDULE IN THREE MINUTES, NOT IN A DAY

- Ask for a manual trigger URL or command so you can fire it on demand.
- Temporarily set the schedule to every 5 minutes, watch two runs land, then set it back.
- Check the run record says which UTC expression fired, and what local hour it thought it was.
- Put a note in the runbook: the local time this must hold, and which DST state you assumed.

NEVER HARDCODE YOUR OFFSET

Do not write 'UTC-5' anywhere. Read the local time from a real timezone library at run time. An offset written into code is correct for about half the year, and wrong silently.

5

SESSION 5 · PART 1 OF 4

Make it impossible to run twice

Schedules fire twice more often than anyone expects: a retry, a redeploy, two cloud instances waking together. If your agent sends things, it will one day send them twice.

A REAL INCIDENT, AND IT IS ALWAYS THIS ONE

A publishing job fired at 09:15 and again at 09:33. Both were inside the same hour, so an hour-based guard let both through. The second run republished the entire first run's work: 122 messages into a channel real people read. Nothing errored. Every log was green.

SAY THIS

```
Say:  "Add a run claim for this job. Four requirements:
       1. the claim covers the whole business DAY, not the hour
       2. it lives in storage that survives the process restarting
       3. it is written BEFORE the first thing that leaves the system
       4. if the claim can't be read, DON'T run, and record the error"
```

WHY EACH OF THE FOUR MATTERS

- The day, not the hour: two runs in one hour is the case that actually happens.
- Storage that survives: an in-memory note cannot see a second copy of your job running elsewhere.
- Written first: a claim written at the end means a run that sent, then crashed, leaves no trace it ever ran.
- Fail closed: an unreadable claim means UNKNOWN, not 'hasn't run'. Refusing to run is loud; running twice is not.

YOU SHOULD SEE

- Trigger the job twice in a row by hand. The second run does nothing and says why.
- The run record for run two shows it found an existing claim.

SESSION 5 · PART 2 OF 4

5 Cap how far and how loud one run can go

A cost limit will not save you here. A hundred short messages into a customer channel is a serious incident that costs almost nothing. Distance and volume are their own axis.

SAY THIS

```
Say:  "Declare, in code, the widest audience this job may reach
       and the maximum number of external actions one run may
       perform. Check it when the run opens. An undeclared job
       must be REFUSED, never given a default. Going over the
       limit records an error and throws - it never silently stops."
```

AUDIENCE	MEANS	EXAMPLE
internal	Only you and your logs	A private ops channel nobody reads
team	Colleagues, your own inbox	The owner's direct messages
public	Customers or members	A channel your members read

ONE DOOR, NOT TWO

There must be exactly ONE place in the code that sends. If a second function can also send, your limit is not a limit. Ask for a check that fails the build when a second sending path appears. The 122-message incident happened because a second code path ignored a cap that was working perfectly in the first one.

YOU SHOULD SEE

- Ask it to set the limit to 1 and then try to send twice. The second attempt throws.
- The refusal appears in the run record as an error, not as a quiet success.
- The declared audience is written in code you can point at, not described in a prompt.

SESSION 5 · PART 3 OF 4

5 Observability: proof, and one watcher

If you cannot answer "what happened at 6am?" without guessing, you do not operate this agent yet. Two things fix that: a record from every run, and exactly one job whose whole purpose is to read those records.

SAY THIS

```
Say:  "Every run writes one record, including runs that fail
       and runs that are parked. Then build ONE separate job
       that reads those records on a schedule and is the only
       thing that alerts me. Individual runs must NOT alert."
```

THE SIX FIELDS THAT MATTER MOST

`run_id`
one id threaded through everything this run touched

`claim`
did it claim the day, and did that happen before it acted

`delivered`
proof the output actually landed, not just 'no error'

`outcome`
ok, failed, or parked – never blank

`actions`
how many external actions, against the declared ceiling

`next_action`
if something is wrong, the one thing a human should do

WHY RUNS MUST NOT SEND THEIR OWN ALERTS

A run that crashes can shout. A run that never started cannot. Crash-only alerting is blind to the exact failure you care about most: the 6am job that simply did not happen. Only a separate watcher reading the records can see an absence.

YOU SHOULD SEE

- Three states that never blur together: healthy, parked, and broken.
- Delete a run record on purpose and watch the watcher notice the gap.

HOW A PROBLEM REACHES A PERSON

Silence is the failure you cannot see

A run that crashes can shout. A run that never started cannot. Only something reading the records can notice an absence.


```
one alert per run, maximum. if three things are wrong, send the worst one and say how many others are waiting. a vague alarm at 3am is worse than a clean line in a log.
```

THE FAILURE THIS PAGE EXISTS FOR

A run can finish with no error, no crash and a clean success code, having written its answer as text that was then thrown away. Nothing alerts, because nothing failed. If a run owed a person a reply and delivered nothing, that is a failed run - record it as one and deliver the text yourself.

WHAT EVERY ALERT MUST CARRY

- One action. Not a status report with a problem buried in it - the one thing to do.
- One cause, labelled measured or inferred. An inferred cause is written as a hypothesis, so nobody acts on a guess believing it was a finding.
- The failed step, the run id, and what the system already tried before waking you.
- Never a request to do work the reader cannot do. An alert asking a non-engineer to read logs is a defect in the alert, not in the reader.

5

SESSION 5 · PART 4 OF 4

Build the stop button before you need it

One day it will misbehave. You want a switch that stops this one agent, right now, without a deploy, without a developer, and without taking anything else down.

SAY THIS

```
Say:  "Add a stop switch for THIS job that I can set without a
       deploy. When it is set, the job still wakes, records that
       it is PARKED, and returns success without doing anything.
       A manual or forced run must NOT bypass it."
```

STATE	WHAT IT MEANS	WHAT THE WATCHER SAYS
Healthy	Ran, did its job, left a record	Nothing. This is normal.
Parked	You stopped it on purpose	"News is parked" – not an alarm
Broken	Tried to run and could not	An alert with one action

THREE RULES THAT MAKE IT A REAL STOP BUTTON

It must stop the JOB, not just the sending – a switch that only silences output hides a runaway while it keeps burning money and changing data. It must be per-agent, because your platform's global 'disable all schedules' is an outage for everything else. And a forced run must not defeat it, or it is not a stop button.

YOU SHOULD SEE

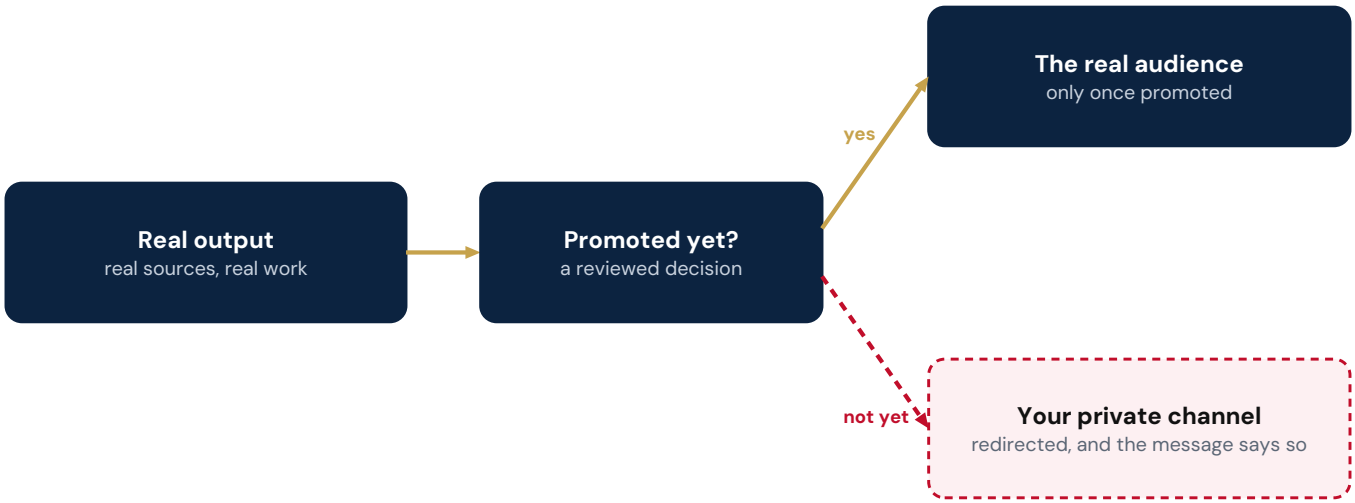
- Set the switch, trigger the job by hand, and watch it record PARKED and do nothing.
- The watcher reports it as parked, not as broken and not as healthy.
- Unset it, trigger again, and watch it work normally.

That is Session 5 complete. Your agent now has all four floor controls.

SESSION 6 · 45 MINUTES

6 Now build the real job - on a leash

Only now does it do actual work. Its first real outputs go somewhere narrow, where a mistake costs nothing, until it has earned a wider audience.




```
absent from the promoted list means confined. a new agent is held back by forgetting, never exposed by forgetting.
```

WHAT EARNS PROMOTION

- Three consecutive clean runs. Not one - one run proves the happy path, and the happy path is not where new agents fail.
- At least one clean run of every different job it owns. Three clean daily runs say nothing about the weekly one.
- Clean means: healthy record, nothing sent outside the narrow target, no alert raised, and the claim written before it acted.
- A run that errored is not clean and resets the count to zero. It does not merely fail to count.

REDIRECT, NEVER REFUSE

A refusal proves your guard works and proves nothing about the agent. You want to watch it do the real job - real sources, real output, real claim - with only the audience narrowed.

SESSION 7 · 40 MINUTES

7 Now break it on purpose

Evaluation and QA for an agent is mostly failure testing. Work down this list one row at a time and watch each result with your own eyes. This is the session that turns a demo into something you can leave running.

BREAK THIS	IT MUST DO THIS
Nothing to report	A correct empty result, clearly different from a failure
Source unreachable	Says unreadable. Never reports an empty day
Source times out	Retries within its stated limit, then fails loudly
Fired twice	Second run does nothing and names the existing claim
Claim store unreadable	Refuses to run and records the error
Permission removed	Fails closed, names what access it needed
Its own key missing	Throws. Never acts using your identity instead
Over the action limit	Records an error and throws, never silently truncates
Wrong audience	Refuses to send and says why
Produced nothing	Delivers its text anyway, records the rescue, escalates
Stop switch set	Records PARKED, returns success, changes nothing
Output rejected	Keeps the feedback, does not overwrite what was approved

NOTICE HOW FEW OF THESE ARE HAPPY-PATH TESTS

That ratio is the point. Every one of these rows exists because it went wrong for somebody. Ask your assistant to write them as automated tests so they run again on every future change, not once tonight.

Do not skip the row you are most confident about. That is usually the one.

8

SESSION 8 · PART 1 OF 2

Bounded self-healing

An agent can fix some of its own problems. It must never be able to ship its own changes. The line between those two is the whole subject.

IT MAY DO THIS ALONE

- Retry a safe, repeatable failure a few times
- Re-read state it is allowed to read
- Collect diagnostics and open a ticket
- Draft a fix for a human to review

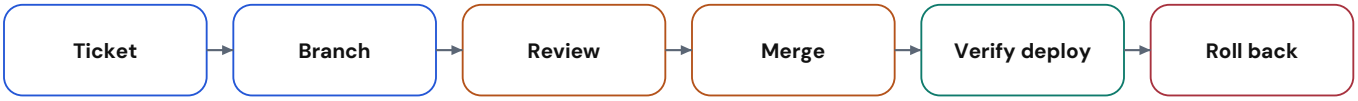
IT MUST NEVER DO THIS

- Deploy, or merge its own change
- Change permissions or rotate a secret
- Spend money, or contact a new audience
- Act as a human being

DO NOT RESTRICT A DANGEROUS POWER IN WRITING. REMOVE THE TOOL.

An agent that must not merge its own code should not have a merge tool at all. An agent that must not spend money should hold no payment credential. A rule written in a prompt is a request to a system that guesses. A missing tool cannot be talked around. Ask which of your prohibitions are enforced by absence, and which are only instructions.

WHEN IT DOES PROPOSE A CODE CHANGE



A Staff Engineer review – a person, or an independent reviewer that is not the author – sits between Branch and Merge. Verify the deploy, not the merge: a merge is not a deploy, and a deploy is not a running change.

8

SESSION 8 · PART 2 OF 2

Make the rules mechanical

Everything in this guide is advice until something refuses when it is broken. This is the session that turns your launch gate from a document into a check.

SAY THIS

```
Say:  "Put the launch gate in ONE file and write a script that
       runs in CI. It fails the build when the gate line is
       missing, or any item is unconfirmed. 'N/A' alone fails;
       'N/A - <reason>' passes. Then break it on purpose so I
       can watch the build go red."
```

```
Launch-Gate:  schedule OK · claim OK · ceiling OK · record OK ·
              watcher OK · stop-button OK · canary OK
```

YOU SHOULD SEE

- A build that goes red when you delete one word from that line.
- A gate kept in exactly one file, referenced everywhere else, so copies cannot drift.

AN ADVISORY GATE IS NOT A GATE

A checklist a person confirms is a promise. The same checklist run by CI is a control. Things ship with items written down as unmet, by people who meant well, because nothing refused. This one script is the difference between this guide being applied and being admired.

This is also the step most people skip, and it is about twenty lines of code.

9

SESSION 9 · 20 MINUTES

Widen the audience, on evidence

The last session. You decide, from the record rather than from a feeling, whether this agent is ready to reach the audience you originally intended.

SAY THIS

```
Say:  "Show me, from the run records: how many consecutive
       clean runs, whether every job shape has had a clean run,
       every alert raised, and what I corrected. Recommend
       promote or wait. The decision is mine."
```

LOOK AT	READY WHEN
Consecutive clean runs	Three or more, with no error resetting the count
Every job shape	Each distinct job has had at least one clean run
Corrections you made	Falling, and each one captured as a test
Alerts raised	Each one was real, actionable, and you knew what to do
Cost per useful result	Known, and you are willing to pay it
The 6am question	You can explain the last good run and the last bad one

PROMOTION IS A CHANGE, NOT A MOOD

Make it an explicit, reviewed edit with a date and a reason, the same as any other change. If it goes wrong you want to know exactly when the audience widened, and why somebody thought it was ready.

Then stop. Do not add a second agent this week.

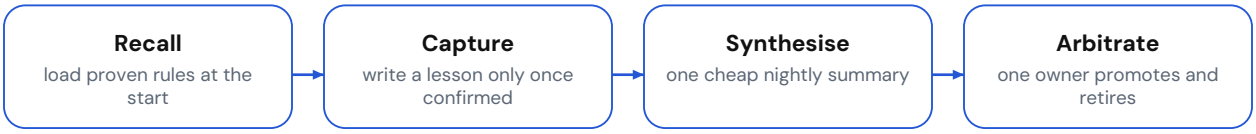
Run this one for a fortnight. The corrections you make in that time are worth more than anything you would build instead.

ONCE IT IS RUNNING

Teaching it, and what to spend on thinking

Two things you will want in month two: an agent that stops repeating last month's mistake, and a bill that does not surprise you.

SHARED MEMORY: THE HIVEMIND PATTERN



- A lesson starts unproven. Confirmation nudges it up; a correction pulls it down harder, because being wrong should cost more than being right earns.
- Unused lessons fade. Above a threshold, a lesson becomes a rule the agent is given every run.
- A memory with no retirement path is a leak: it only grows, and it is read in full on every single run. Wire the fade and the review queue on day one, and check they are actually called.
- One person owns promoting and retiring. Never the agent itself.

MODEL ROUTING AND CACHING, IN TWO LINES

- Model routing: use plain code for anything with a right answer, the cheapest model for sorting and formatting, and an expensive one only for judgement. Record which route each run took.
- Caching: keep the unchanging instructions byte-for-byte identical between runs so they can be cached, and put anything that changes per run - times, counts, today's data - outside that block. One live number in the wrong place silently disables the saving and nothing errors.

NEITHER OF THESE BELONGS IN WEEK ONE

Get one agent running, watched, and stoppable first. Memory and cost tuning are what you do to an agent that already works.

WHERE TO GO NEXT

You have one. Here is when to build two.

A second agent should remove a constraint the first one cannot. Before you start, you should be able to answer all four of these about it, in one sentence each.

The job	What repeated result can the first agent not safely own?
The handoff	What does it receive, what does it return, and who owns a mismatch?
The memory	What may it read, what may it write, what must it never see?
The authority	What can it do alone, and what must it only ever propose?

SHARE THE PLUMBING, NOT THE AGENT

Keep one run ledger, one watcher, one alerting contract and one memory store across every agent you add. Duplicating them per agent is the exact moment a fleet stops being observable, and you will not notice on the day it happens.

THE COMPANION TO THIS GUIDE

Agent Builder Starter Prompt

Paste it into your build assistant and it runs the interview on page 4, fills in your contract, then builds with you through the nine sessions in this guide.

`romanbediner.com/resources/agentic-ai-employees`