

## — THE COMPLETE GUIDE

# The Agent Promotion Playbook.

Why the hard part was never writing the automation, where a workflow actually earns trust, and how to move from a good idea in a doc to a standing agent your team relies on without a second thought.

12 chapters

3 frameworks

4 stages

## — THREE FRAMEWORKS YOU CAN REUSE

**1 The Automation Inventory**

What's actually worth turning into an agent

**2 The Trust Ladder**

Four stages, and what unlocks each one

**3 The Promotion Checklist**

What has to be true before a workflow becomes an agent

## — WHO THIS IS FOR

- Platform & SRE leads tired of rebuilding the same fix
- IT operations teams drowning in repeat tickets
- FinOps and cloud teams who can see waste but can't act on it

Governed AI agents for enterprise IT operations — built from what your team already knows.

## — CONTENTS

## What this playbook covers

This is for the people who keep re-solving the same operational problem by hand: platform and SRE leads, IT operations managers, and the FinOps and cloud teams who can see the waste but have never had a governed way to act on it. It assumes your team already has the expertise. The question is how to get that expertise running without you in the room.

|                                                                                              |     |                                                                                             |     |
|----------------------------------------------------------------------------------------------|-----|---------------------------------------------------------------------------------------------|-----|
| <b>01 Why the fix keeps living in one head</b><br>The knowledge was never the bottleneck     | p03 | <b>02 Where the effort in building an agent goes</b><br>It's not the code, and it never was | p04 |
| <b>03 Framework 1 — the Automation Inventory</b><br>Finding what's actually worth automating | p05 | <b>04 The objective is the whole spec</b><br>Why a sentence beats a design document         | p06 |
| <b>05 Composing operational logic</b><br>Triggers, skills, checkpoints, actions              | p07 | <b>06 Boundaries before behaviour</b><br>Why governance comes first, not last               | p08 |
| <b>07 Framework 2 — the Trust Ladder</b><br>Four stages, and what unlocks each one           | p09 | <b>08 Approval gates that don't slow anyone down</b><br>Designing the one step that matters | p10 |
| <b>09 Promotion is a decision, not a milestone</b><br>When a workflow is actually ready      | p11 | <b>10 Framework 3 — the Promotion Checklist</b><br>What has to be true first                | p12 |
| <b>11 Starting from a specialist</b><br>Six foundations, and when to use them                | p13 | <b>12 Getting started</b><br>Your first agent in a week, not a quarter                      | p14 |

## — HOW TO READ THIS

Chapters 1–2 make the case that the hard part of automation was never the code — it's deciding what's worth automating and describing it well. Chapters 3–6 cover building the thing. Chapters 7–10 cover the part most guides skip: how a workflow earns the right to run unattended. Chapters 11–12 get you moving this week.

## — THREE FRAMEWORKS YOU CAN REUSE

**The Automation Inventory**

Six questions that separate a good first agent from a distraction.

**The Trust Ladder**

Four stages from a described objective to a standing agent nobody questions.

**The Promotion Checklist**

What has to be true — not assumed — before a workflow runs unattended.

## — CHAPTER ONE

# Why the fix keeps living in one head

Every team has a version of this: a problem that comes up often enough to be annoying, that one person always fixes the same way, and that never quite becomes anyone else's job to automate.

It's not that nobody thought of automating it. It's that writing a script for it felt like overkill for a fifteen-minute task, and building a proper tool felt like a project nobody had a sprint for. So it stays a habit — something Priya does on Tuesdays, or something the on-call person does if they remember the trick from last time. The knowledge is fine. The distribution of that knowledge is the problem.

*The gap was never expertise. It was that turning expertise into something repeatable felt like more work than just doing the thing again by hand.*

## What actually makes a fix hard to hand off

Look closely at the tasks that stay manual, and a pattern shows up. They're not technically difficult — most of them are a handful of steps a competent engineer could describe out loud in under a minute. What's missing is a cheap way to turn that description into something that runs the same way without them, and a way to bound it so handing it off doesn't feel like handing over the keys.

## — FOUR SIGNS A TASK IS STUCK IN ONE PERSON'S HEAD

- It gets asked about in the same Slack channel every few weeks.
- The runbook for it exists, but everyone still pings the one person anyway.
- It's never made it onto anyone's roadmap because it's not big enough to justify a project.
- The person who does it fastest is also the person everyone needs for the actual emergencies.

## — WHAT THIS CHAPTER SETS UP

**Chapter 2**

Where the actual effort in building an agent goes — and why it's rarely where people expect.

**Framework 1**

A way to find which of your team's habits are actually worth turning into an agent.

**The rest of the book**

How to describe, compose, govern and promote — without a platform migration.

## — A PREVIEW OF THE INVENTORY

**How often does it come up?**

Weekly or more, or it's probably not worth the setup.

**Can it be said in two sentences?**

If it needs a design doc, it isn't ready to describe yet.

**Is the risky step narrow?**

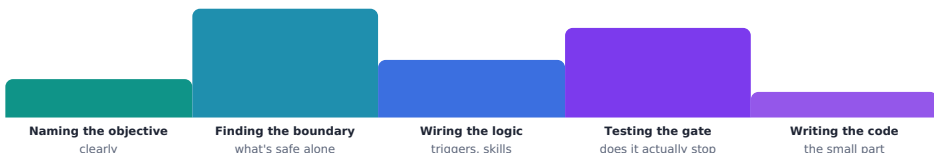
One nameable action, not the whole task — that's your gate.

## — CHAPTER TWO

## Where the effort in building an agent goes

People expect the hard part to be technical — wiring up an API, handling an edge case, getting a model to behave. In practice, most of the effort sits somewhere else entirely.

## — FIGURE — WHERE THE HOURS ACTUALLY GO ON A FIRST AGENT



Writing the code that executes an action is usually the smallest bar on this chart. Deciding what the agent may do without asking is the tallest.

### Why the boundary takes longer than the logic

Wiring a trigger to a skill is mechanical — most teams have it working in an afternoon. Deciding whether that skill should run unattended in production, and what happens if it's wrong, is a conversation that touches security, the service owner, and whoever answers for the outcome. That conversation is worth having. It's just not a coding problem, and treating it like one is why so many automation efforts stall exactly here.

## — WHAT THIS IMPLIES FOR HOW YOU BUILD

Write the objective and the boundary before you write a single trigger.

Budget time for the governance conversation — it's not overhead, it's the actual work.

A slow first agent because the boundary took a week is a good sign, not a failure.

*Teams that skip the boundary conversation don't actually save the time. They spend it later, after something the agent did surprises someone who wasn't in the room when it was built.*

## — WHERE CHAPTER 3 PICKS UP

The Automation Inventory turns "does this feel worth automating" into six concrete questions — so the naming-the-objective step in this chart gets shorter every time you run it.

## — CHAPTER THREE · FRAMEWORK ONE

# The Automation Inventory

Not every recurring task deserves an agent. Before building anything, it's worth running your team's habits through six honest questions.

| QUESTION                                      | WHAT A STRONG ANSWER LOOKS LIKE                                   |
|-----------------------------------------------|-------------------------------------------------------------------|
| How often does this actually come up?         | Weekly or more — rare enough tasks rarely justify the setup.      |
| Can someone explain the fix in two sentences? | Yes, cleanly — if it takes a design doc, it's not ready yet.      |
| Is the risky part narrow and nameable?        | One or two steps, not the whole task — that's your approval gate. |
| Would getting it wrong be survivable?         | Recoverable, or already caught by something downstream.           |
| Does the fix change often?                    | Rarely — a fix that changes weekly will fight being automated.    |
| Who owns saying yes to the risky step?        | A specific role, already identifiable, not 'whoever's around'.    |

## Running the inventory

Do this with a whiteboard and twenty minutes, not a spreadsheet and a quarter. List every task that's come up more than twice in the last month across your team's Slack, tickets and stand-ups. Score each one against the six questions above. The ones that score well on all six are your first candidates — not because they're the most impactful, but because they're the ones you can actually get right on the first try.

*The first agent you build should be boring. Save the impressive one for after you've seen how the approval step actually behaves under real conditions.*

## — SCORING IT HONESTLY

**Five or six yeses**

A strong first candidate — small, well-understood, low-drama.

**Three or four yeses**

Worth composing, but keep the human step wide until it earns trust.

**Two or fewer**

Not yet. Either the task or the team's understanding of it needs to mature first.

— CHAPTER FOUR

# The objective is the whole spec

The instinct when building something new is to write it down properly first — a spec, a diagram, a plan. For a first agent, that instinct slows you down.

A good objective reads like something you'd actually say out loud to a colleague: investigate a Kubernetes crash loop. Rotate aging IAM keys with approval. Find and clean up unused cloud disks. None of these sentences describe a system. They describe an outcome. That's exactly the right altitude to start from — the system gets drafted around the objective, not the other way around.

— WHAT MAKES AN OBJECTIVE USABLE AS A STARTING POINT

- Named as an outcome, not a mechanism — 'clean up unused disks', not 'run a Lambda every Tuesday'.
- Specific enough to draft from — 'aging IAM keys', not 'security stuff'.
- Honest about the risky part, if there is one — 'with approval' is doing real work in that sentence.
- Short enough to say in one breath — if it needs a paragraph, split it into two objectives.

*A vague objective produces a vague draft. A precise one produces something you can react to — and reacting to a draft is much easier than staring at a blank canvas.*

## From objective to draft, in practice

```
describe "detect anomalous cloud spend"
# drafting workflow from objective...
  trigger: daily · billing export refreshed
  skills: compare spend to 30-day baseline, tag correlation
  ai step: rank anomalies by size and likely cause
  human step: none for detection · approval for any cleanup action
✓ draft ready · edit any block before you run it
```

— WEAK OBJECTIVE, REWRITTEN

| TOO VAGUE TO DRAFT FROM               | SPECIFIC ENOUGH TO START               |
|---------------------------------------|----------------------------------------|
| "Improve our cloud costs"             | "Find and clean up unused cloud disks" |
| "Handle security stuff automatically" | "Rotate aging IAM keys with approval"  |
| "Fix incidents faster"                | "Investigate a Kubernetes crash loop"  |

## — CHAPTER FIVE

# Composing operational logic

Once the objective is clear, composing is mostly assembly — six kinds of building block, arranged to match how the work actually happens rather than how a diagram says it should.

Most drafts get the shape roughly right on the first pass. What usually needs adjusting is the exceptions — the case where the disk isn't actually unused, the deploy that's mid-rollout when the crash loop starts, the ticket that doesn't match any known category. That's the real work of composing: not building the happy path, but deciding what happens when reality doesn't cooperate with it.

**Triggers****What starts the work**

A schedule, a webhook, an ITSM event, or someone just asking directly.

**AI capabilities****Where judgement gets applied**

Reasoning grounded in the runbooks and history your org already trusts.

**Human steps****Where a person still decides**

Placed exactly where judgement matters — and nowhere else.

**Skills****What it already knows how to do**

Reusable moves — fetch logs, compare a baseline, check a tag — shared across agents.

**Flow controls****What happens off the happy path**

Branches, retries, and the exception that would otherwise break a plain script.

**Actions****What actually changes**

The real, executed, logged change — not a suggestion waiting on someone to act.

*A workflow that only handles the happy path isn't half-finished automation. It's a demo wearing a production label.*

## — QUESTIONS WORTH ASKING ABOUT EACH BLOCK

- What triggers this too often, and does that need its own filter?
- Which skill result would you personally want to double-check?
- What's the most annoying exception this has hit in the last month?
- If the human step disappeared, what's the worst thing that could happen?

## — WHERE CHAPTER 6 PICKS UP

Every block above assumes the boundary is already decided — which action classes may run alone, and which need a person. That decision, and why it belongs first, is the whole next chapter.

## — CHAPTER SIX

## Boundaries before behaviour

It's tempting to build the whole workflow first and add the guardrails once it's working. Every team that's tried it the other way says the same thing: decide the boundary first, and composing gets easier, not harder.

Deciding the boundary early forces a useful discipline. You can't wire up an action you haven't yet decided is allowed to run unattended, so the design naturally separates into what the agent can just do and what it has to ask about. That separation is the entire governance model — everything else is detail.

## — THE FOUR QUESTIONS A BOUNDARY ANSWERS

| QUESTION                       | ANSWER BECOMES             |
|--------------------------------|----------------------------|
| What may this agent touch?     | Its access scope           |
| What may it do without asking? | Its autonomous action list |
| What must it ask about first?  | Its approval gate          |
| What happens when it's unsure? | Its escalation path        |

```
agent: cost-cleanup  environment: non-production
  scan_idle_resources: auto (read-only, no side effects)
  delete_untagged_disk: auto (cost ceiling: $500/run)
  delete_tagged_production: approval (role: infra-lead)
  on_uncertain_tag: escalate, do not guess
```

### Boundaries scale better than trust

"We trust the team that built it" is not a boundary — it's a hope that nothing changes. A written scope, on the other hand, survives the builder moving teams, the agent being reused elsewhere, and the inevitable day someone asks exactly what it's allowed to do. Write it down once, and that question stops being a fire drill.

## — SIZING AN ACCESS SCOPE

**Too wide**

Can reach systems its objective never mentions — a sign the scope was copied from an existing role.

**About right**

Can reach exactly what the objective requires, nothing adjacent, nothing 'just in case'.

**Too narrow**

Keeps hitting permission errors on its normal path — usually fixed once, not widened blindly.

## — CHAPTER SEVEN · FRAMEWORK TWO

# The Trust Ladder

Four stages between a described objective and a standing agent nobody questions. Each rung is unlocked by evidence, not by time passing or confidence rising on its own.

- 1 Described.** An objective exists in plain language. Nothing has run yet.  
**Unlocked by:** Nothing — this is the starting line.
- 2 Composed.** The workflow runs, watched closely, every action still gated.  
**Unlocked by:** A working draft with no track record yet.
- 3 Governed.** Boundaries are fixed and tested; reversible actions run unattended.  
**Unlocked by:** Enough runs to trust the boundary holds under real conditions.
- 4 Promoted.** A standing agent with objective, thresholds and cost ceiling attached.  
**Unlocked by:** A stable success rate and a known cost per run.

## Why the jump from 2 to 3 is the one that matters

Getting a workflow to run once, watched closely, is mostly a technical exercise. Getting a team comfortable letting it run without someone watching is a trust exercise — and trust here is earned specifically, not generally. A team that trusts an agent to clean up disks does not automatically trust it to touch IAM. The ladder resets, sensibly, at the boundary of what's actually been tested.

### — WHERE IS THIS AGENT, REALLY?

- Has anyone actually watched it run, or is 'it works' secondhand? (rung 1 vs 2)
- Does it have a cost ceiling and a named owner, or just a good track record? (rung 4)
- Have the reversible actions run unattended enough times to trust the pattern? (rung 3)
- If it moved to touch a new system tomorrow, would that reset its trust level? (it should)

*Most teams overestimate where they are on this ladder. Confidence rises faster than evidence does — the ladder is here to keep those two things honest with each other.*

## — CHAPTER EIGHT

## Approval gates that don't slow anyone down

A badly placed approval gate is worse than none at all — it trains people to click approve without reading, which is the opposite of what a gate is for.

The fix isn't fewer gates. It's better ones. A gate that arrives with the evidence already assembled — what triggered this, what the agent found, what it wants to do and why — asks the approver to confirm a conclusion, not start an investigation from a blank ticket. That difference is what keeps a gate meaningful instead of becoming a reflex.

## — WHAT A GOOD APPROVAL REQUEST INCLUDES

- The trigger — what started this, in plain language.
- The finding — what the agent investigated and concluded.
- The proposed action — exactly what will happen if approved.
- The blast radius — what's affected, and whether it's reversible.

*Route the gate to a role, not a person. An approval that waits on someone's vacation schedule teaches everyone to route around it — and once people start routing around a gate, it stops being a gate.*

### One gate, not ten

Resist the urge to gate every step "just in case." A workflow with one clear gate at the genuinely risky step gets used. A workflow with five gates scattered across low-risk steps gets rubber-stamped, and the one gate that mattered gets the same tired click as the other four.

## — WHAT THE APPROVAL ITSELF LOOKS LIKE

```
approval requested · agent: iam-key-rotation
  trigger: key age exceeded 90-day policy
  finding: 1 production key eligible, no active sessions detected
  proposed action: rotate key, revoke old after 24h grace window
  blast radius: one service account, reversible within grace window
→ awaiting: security-lead
```

## — CHAPTER NINE

## Promotion is a decision, not a milestone

It's easy to treat promotion as something that happens automatically once a workflow has run a while. Treating it as a deliberate decision, made against evidence, is what keeps the whole ladder honest.

A workflow that has run ten times without incident has told you something, but not everything. It hasn't told you what happens at hour three of an outage, when the trigger fires differently than it ever has before. Promotion isn't a reward for good behaviour — it's a specific claim: this objective, these thresholds, this cost ceiling, this safety rule, together, as a package other teams can rely on without re-deriving it.

## — WHAT PROMOTION ACTUALLY ATTACHES TO A WORKFLOW

- A named objective and owner — not 'whoever built it'.
- A performance threshold — success rate, latency, or both.
- A cost ceiling per run, tracked against actual spend.
- A safety requirement that travels with every future run.

*If you can't say what would make you un-promote it, you haven't actually decided what promotion means for this agent yet.*

## — TOO EARLY VS. OVERDUE

**PROMOTED TOO EARLY**

- Success rate based on a handful of runs
- Cost per run still an estimate
- Approval gate never actually triggered
- No one outside the builder has reviewed it

**PROMOTION OVERDUE**

- Running reliably for months, still manually re-approved each time
- Team fully trusts it but has never written that down
- Clear owner in practice, no owner on paper
- Everyone treats it as standard, no one has promoted it

## — CHAPTER TEN · FRAMEWORK THREE

# The Promotion Checklist

Ten questions worth answering honestly before a workflow becomes a standing agent. The point isn't to pass all ten on day one — it's to know exactly which ones you haven't earned yet.

1

Has it run enough times to show a stable success rate, not a lucky streak?

2

Is the cost per run known, from real runs, not estimated?

3

Has the approval gate actually been exercised — has someone said no at least once?

4

Would someone other than the builder understand what it does from its config alone?

5

Is there a named owner who answers for its behaviour?

6

Does it degrade safely — escalate, not guess — when something unexpected happens?

7

Is every run logged in a way that would satisfy an audit, not just a debugging session?

8

Has its access scope been reviewed by someone outside the team that built it?

9

Is there a threshold that would trigger a review, not just a vague sense of unease?

10

Would you be comfortable explaining this agent's behaviour to your compliance team, unprompted?

## — USING THE CHECKLIST HONESTLY

**Eight or more yes**

Promote it. The evidence is there, and waiting only adds risk.

**Five to seven**

Keep it at governed a little longer — note exactly which answers are missing.

**Four or fewer**

It's still earning trust. That's fine — most agents spend a while here.

## — CHAPTER ELEVEN

# Starting from a specialist

Six foundations exist because six categories of operational pain show up in almost every enterprise estate, regardless of industry. Starting from one saves the part of composing that's rarely specific to your org anyway.

**Network Specialist**

Use when the pain is diagnosis speed — latency, routing, connectivity issues that need topology context to interpret.

**Compute Specialist**

Use when the pain is waste — over-provisioned instances, container right-sizing, utilization nobody's tracking.

**Storage Specialist**

Use when the pain is drift — disk health, cleanup backlog, performance degradation nobody notices until it's acute.

**Database Specialist**

Use when the pain is slow decay — capacity and error trends that creep until they don't.

**FinOps Agent**

Use when the pain is untracked spend — anomalies and idle resources that show up in the bill before anyone flags them.

**ITSM Agent**

Use when the pain is triage volume — tickets that need classifying and routing faster than a queue can keep up.

*A specialist foundation isn't a shortcut around understanding your own estate. It's a shortcut around re-solving problems that a thousand other teams already solved the same way.*

## — MATCHING PAIN TO FOUNDATION

| IF YOUR TEAM KEEPS SAYING...                         | START FROM                         |
|------------------------------------------------------|------------------------------------|
| "Nobody can tell me why this is slow"                | Network or Database Specialist     |
| "We're paying for compute nobody's using"            | Compute Specialist or FinOps Agent |
| "Tickets are piling up faster than we can read them" | ITSM Agent                         |
| "We keep finding orphaned storage every audit"       | Storage Specialist                 |

## — CHAPTER TWELVE

## Getting started

None of this needs a quarter of planning. The version that actually gets built is small on purpose, and moves through the ladder on evidence, one rung at a time.

| STEP                          | WHAT TO DO                                              | WHAT IT PRODUCES                              |
|-------------------------------|---------------------------------------------------------|-----------------------------------------------|
| 1 · Run the inventory         | Twenty minutes, one whiteboard, your team's real habits | A short list of real candidates               |
| 2 · Pick the boring one       | The smallest, clearest, most repeated task on the list  | A first agent you can actually get right      |
| 3 · Describe it in a sentence | Say the objective the way you'd say it to a teammate    | A first draft in minutes, not days            |
| 4 · Compose the exceptions    | Add the branches for what usually goes wrong            | A workflow that survives contact with reality |
| 5 · Fix the boundary          | One clear gate on the risky step, access scoped tight   | Something your risk team can actually approve |
| 6 · Promote on evidence       | Only once the checklist genuinely passes                | A standing agent, not a fragile demo          |

## — FIVE THINGS WORTH TAKING AWAY

- The knowledge already exists. The gap was always distribution, not expertise.
- A precise objective is the whole spec — resist the urge to over-design upfront.
- The boundary takes longer than the logic, and that's time well spent.
- Trust is earned per boundary, not granted once and assumed forever.
- Promotion is a decision against evidence, not a reward for time passing.

**Ready to run the inventory?**

Twenty minutes, one whiteboard, and the playground is free to try against your first candidate.

## — IN CLOSING

# The knowledge was never the problem.

Every fix in this playbook already existed before an agent touched it. Someone on the team could already explain it clearly. What changed was giving that explanation somewhere to run — inside a boundary the whole team helped set, earning trust one rung at a time instead of all at once.

None of that requires a platform migration or a quarter of planning. It requires picking one boring, recurring task, describing it honestly, and being deliberate about the one step that still needs a person. Everything else in this playbook is detail in service of that.

**60%**

LESS MANUAL TRIAGE

**45%**

FASTER TO FIRST RUN

**3×**

MORE AGENTS PROMOTED

**\$18**

PER AUTOMATED RUN

## — THE THREE FRAMEWORKS, IN ONE PLACE

**The Automation Inventory**

Six questions that separate a good first agent from a distraction.

**The Trust Ladder**

Four stages, unlocked by evidence — described, composed, governed, promoted.

**The Promotion Checklist**

Ten honest questions before anything runs unattended.

## Put Agent Builder to work.

Describe an objective, watch it draft the workflow, and promote what works into a standing agent.

GET IN TOUCH

[contact@lumiops.ai](mailto:contact@lumiops.ai)

PLAYGROUND

Try it, no setup

LINKEDIN

[linkedin.com/lumiops](https://linkedin.com/lumiops)