

Become the AI champ in your org: stop project stall

Building the context and operating foundations for enterprise AI, with the architecture, code and operating model to go from pilot to production.

For AI engineering leads and programme heads in enterprises **Reading time** about 30 minutes

Includes 9 figures, 10 code samples, an open-source toolkit map, a 90-day plan

Key findings

- Adoption is no longer the constraint. 88% of organisations use AI in at least one function; about a third are scaling it, and 39% see enterprise-level EBIT impact [1].
- Agents fail in production for three structural reasons: fragmented business meaning, no view of current state, and intelligence disconnected from authority. The model is rarely the first thing to fix.
- A context layer closes those gaps when it is treated as a logical, operated capability over existing systems, with owners, freshness service levels and versioned change.
- Gartner expects more than 40% of agentic AI projects to be cancelled by the end of 2027 on cost, value and risk grounds [3]. Selecting one consequential workflow and designing for production from day one is the main defence.

Recommendations

- Start with one workflow where the prediction exists and the resolution is manual. Baseline it before building.
- Write the context requirement first: what the agent must know, from which source, how fresh, and what happens when it is missing.
- Reuse the warehouse, metric definitions and policy repositories. Add mappings, state, retrieval and controls. Do not build a second source of truth.
- Enforce rules outside the model: read rights at the source, policy in a policy engine, write rights at the execution system, a person on every consequential change until the evidence says otherwise.
- Measure the workflow against its baseline on a five-part scorecard and expand autonomy only on evidence.

0 Executive summary

Enterprise AI has moved beyond experimentation, but its operating model has not caught up. Most organisations use AI somewhere; far fewer run it across the business, and fewer still can show the financial result.

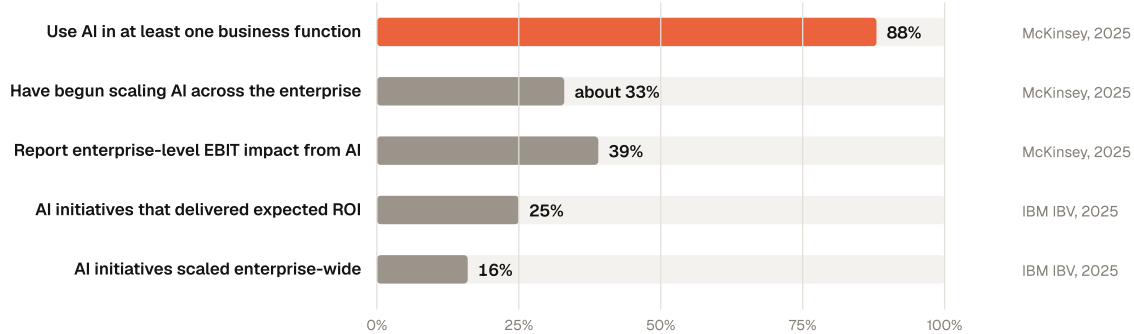


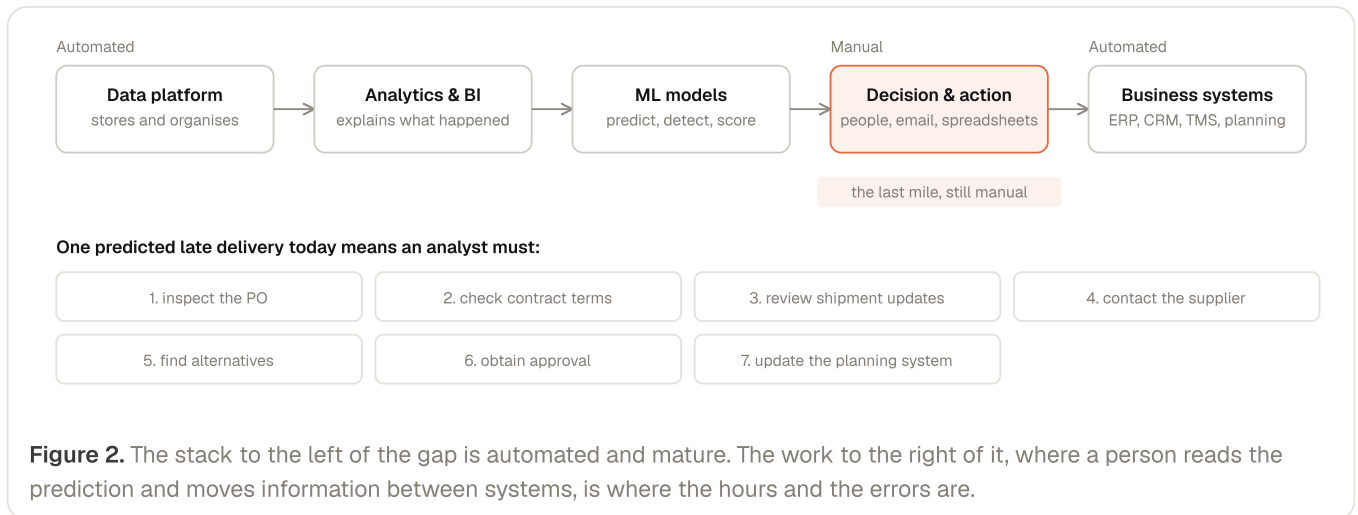
Figure 1. Use is widespread; scale and value are not. The first three bars are shares of surveyed organisations; the last two are shares of AI initiatives, as reported by CEOs. McKinsey gives the second figure as about one-third. McKinsey, *The State of AI 2025* (1,993 respondents) [1]; IBM Institute for Business Value, *CEO study 2025* (2,000 CEOs) [2].

The bottleneck is often not model capability. Most organisations already own warehouses, analytics, business intelligence, machine-learning models and workflow automation. What remains is the last mile: turning information and predictions into decisions, coordinated actions and completed work across business systems.

Closing that gap takes more than another agent framework. It takes a maintained foundation of business context: what the information means, how business entities relate, what is happening now, which rules apply and which actions are authorised. This note sets out how to build and operate that capability, with a reference architecture, working code for its key seams, a worked example in supply operations, the controls and scorecard that make it governable, and a 90-day plan. It is written for the engineer or programme head who will be accountable for the result, and it treats the context layer as real ongoing work rather than a cost-free addition to the stack.

1 The opportunity: from informing decisions to completing work

Data platforms make information available. Analytics explains what happened and why. Models forecast outcomes, detect anomalies and estimate risk. The response to an insight, however, usually remains manual.



A model may flag a supplier delivery as at risk. An operations analyst must then inspect the purchase order, check contract terms, review shipment updates, contact the supplier, identify alternatives, obtain approval and update the planning system. The prediction is automated; the resolution is not.

AI agents can close this gap. They interpret variable inputs, retrieve relevant information, reason across several steps, use tools and coordinate parts of a workflow. They earn their place where work involves exceptions, unstructured information and decisions spanning several systems.

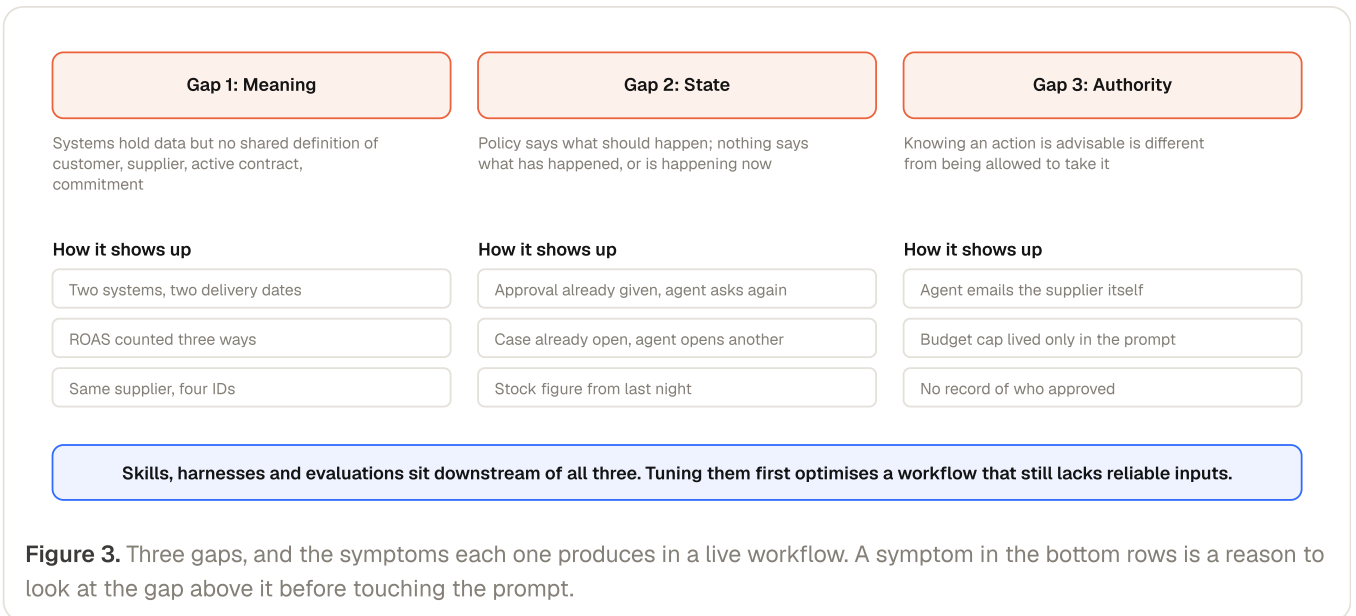
Where an agent is the right tool

Shape of the work	Better handled by	Why
Stable, repetitive, fully specified process	Conventional automation (RPA, scheduled jobs, workflow engine)	Deterministic, cheap, auditable; an agent adds variance without adding value
Find or summarise information for a person	Search or an assistant	No action is taken, so no authority or state management is needed
Exceptions across several systems, with judgement and follow-up	An agent, with bounded authority	Variable inputs, multi-step reasoning, tool use; this is the last-mile work
High-stakes, irreversible decisions	A person, with an agent preparing the evidence	The cost of a wrong action outweighs the cost of waiting for approval

Gartner recommends choosing agentic approaches where they deliver clear value rather than applying them indiscriminately [3]. The opportunity is to connect the existing data and analytics stack to the operational work that still depends on people moving information between systems, interpreting exceptions and initiating follow-up. It is not to replace that stack.

2 Why promising agents fail in production

The same three structural gaps appear in almost every stalled agent programme. Each one produces symptoms that look like model error and are not.



Business meaning is fragmented

IBM's 2025 CEO study found that 50% of surveyed CEOs believe rapid technology investment has left their organisations with disconnected, piecemeal technology [2]. Systems may contain the necessary data without sharing a definition of a customer, a supplier, an active contract or a current commitment. An agent reading two systems gets two answers and no rule for choosing between them.

The agent lacks the current state of the work

A policy document describes what should happen. It does not establish whether a particular approval has been given, whether a record changed an hour ago, or whether a colleague is already handling the case. A model produces a plausible response from stale or incomplete information, and the response reads as confidently as a correct one.

Intelligence is disconnected from authority and execution

Knowing that an action is advisable is different from being authorised to take it. Production workflows need decision rights, controlled access, approval routes, exception handling and an auditable record. A prompt that says "never exceed the budget cap" is advice to the model; it is not a control.

Where skills, harnesses and evaluations fit

Skills describe procedures, harnesses structure execution, and evaluations test performance against cases. All three are necessary engineering tools. They become distractions when a team optimises them before establishing reliable business inputs, workflow ownership and measurable outcomes. Section 5 shows evaluations used in their right place: testing the whole workflow, including stale data, conflicts and unauthorised requests.

Gartner warned in June 2025 that more than 40% of agentic AI projects could be cancelled by the end of 2027 because of escalating costs, unclear business value or inadequate risk controls [3]. That is a forecast, not a measured cancellation rate, but it names the three failure modes this note is organised around: cost without a baseline, value without a chosen workflow, and risk without enforced controls.

3 The context layer: reference architecture

A context layer assembles and maintains the business information, definitions, current state and rules that AI systems need to perform specific work.

It is a logical capability. It can use a vector store, a knowledge base, the warehouse or a document repository for storage and computation, but none of those is the layer by itself, and it does not have to be one product or one database. What makes it a context layer is the set of mappings, retrieval interfaces, state management and controls introduced on top. This follows the direction Anthropic describes as context engineering: context is curated and maintained throughout an agent's execution, not written once into a prompt [4].

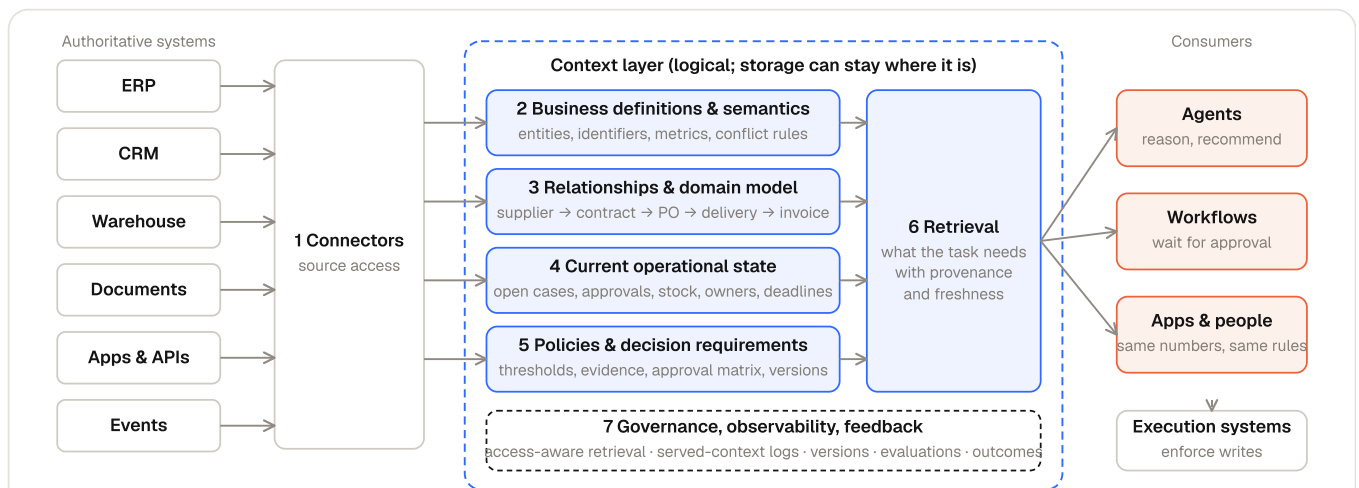


Figure 4. Reference architecture. Sources stay authoritative; the layer adds meaning, relationships, state, policy and governed retrieval; agents, workflows and apps read the same context; execution systems keep their own write controls.

The seven components, and what to reuse for each

Component	What it holds	Reuse from your stack	Usually new work
1 Connectors and source access	Interfaces to ERP, CRM, warehouse, documents, apps, APIs and event streams	Existing APIs, CDC and ingestion pipelines, the warehouse itself	Live reads for fast-changing state; identity pass-through so reads carry the user's rights
2 Business definitions and semantics	Canonical entities, reconciled identifiers, metric definitions, conflict-resolution rules	dbt models, the semantic layer, MDM golden records	Exposing definitions to agents; one meaning per term across systems; precedence rules when sources disagree
3 Relationships and domain model	Links between entities, transactions, documents and processes	Warehouse foreign keys, the data catalog's lineage graph	Cross-system links the warehouse never needed, such as campaign → product → stock, or supplier → contract → open exceptions
4 Current operational state	Open cases, approval status, inventory or capacity, task ownership, recent events, deadlines	Operational systems, event streams, case-management tools	Freshness rules, timestamps, stale and conflict detection, a small state store for cross-system facts
5 Policies and decision requirements	Procedures, thresholds, exception rules, required evidence, approval matrices, escalation paths	The policy repository, approval matrices in finance and procurement	Critical rules as checkable conditions, versioned with owner and effective date; a context requirement per task
6 Context retrieval and delivery	APIs, queries or tools that assemble only what the task and user need, with provenance and freshness	Query engines, search	The assembly logic, the task specs, an agent-facing interface such as an MCP tool
7 Governance, observability, feedback	Access-aware retrieval, logs of what was served, versions, quality checks, evaluation data, outcomes	IAM, the observability stack, the audit platform	A per-decision record that links inputs, versions, approval, action and outcome

Component 5 in practice: the context requirement

The single most useful artefact in the whole programme is a short, owned specification of what an agent must know before it may act on a task. It turns a vague request ("resolve late deliveries") into checkable inputs, each with a source, a freshness limit and a rule for what happens when it is missing.

Code 1. A context requirement for one task, in YAML. Business-owned, versioned, tested like code.

```
# context requirement for one task: what the agent must know before it may act.
# Owned by the business; versioned; tested like code.
task: resolve_late_delivery
version: 3
owner: supply-ops@example.com

inputs:
  - name: shipment
    source: tms.shipments           # authoritative system for delivery dates
    key: shipment_id
    max_age: 15m                   # older than this → stale → escalate
    required: true
  - name: purchase_order
    source: erp.purchase_orders
    join: shipment.po_number -> po_number
    max_age: 1h
    required: true
  - name: contract_terms
    source: clm.contracts
    join: purchase_order.supplier_id -> supplier_id
    fields: [sla_days, penalty_clause, alternate_supplier_allowed]
    max_age: 24h
    required: true
  - name: production_impact
    source: mes.schedule
    join: purchase_order.material_id -> material_id
    max_age: 30m
    required: true
  - name: approved_alternates
    source: srm.alternate_suppliers
    required: false                # absence is a warning, not a stop

definitions: [late_delivery, production_at_risk] # resolved from the semantic layer, never restated here
policies: [supplier-change-approval, spend-threshold]

conflicts:
  delivery_date: prefer tms.shipments over erp.purchase_orders

on_missing: escalate              # never guess
on_stale: escalate
```

Component 6 in practice: what retrieval returns

Retrieval returns facts with their provenance and age, the definitions and policy versions in force, and a record of any conflicts it resolved. It does not return large volumes of text for the model to sift. If a required input is missing or stale, it returns an escalation instead of a best effort.

Code 2. A retrieval response. Every fact carries its source and age; the definitions and policies carry versions.

```
{
  "task": "resolve_late_delivery",
  "task_version": 3,
  "assembled_at": "2026-10-20T06:41:12Z",
  "complete": true,
  "facts": {
    "shipment": {
      "id": "SHP-88213", "eta": "2026-10-27", "promised": "2026-10-22",
```

```

    "source": "tms.shipments", "as_of": "2026-10-20T06:35:00Z", "age_s": 372
  },
  "contract_terms": {
    "sla_days": 2, "alternate_supplier_allowed": true,
    "source": "clm.contracts", "as_of": "2026-10-19T22:00:00Z", "age_s": 31272
  },
  "production_impact": {
    "line": "L2", "stockout_on": "2026-10-24", "units_at_risk": 1800,
    "source": "mes.schedule", "as_of": "2026-10-20T06:25:32Z", "age_s": 940
  }
},
"definitions": {
  "late_delivery": "eta > promised + contract_terms.sla_days",
  "semantic_version": "2026.10.2"
},
"policies": [
  {"id": "supplier-change-approval", "version": 7, "effective_from": "2026-09-01"}
],
"conflicts_resolved": [
  {"field": "delivery_date", "chosen": "tms.shipments", "rejected": "erp.purchase_orders"}
],
"warnings": []
}

```

Code 3. The assembly function behind it. Missing or stale required inputs raise; the caller escalates.

```

from datetime import datetime, timezone

class ContextError(Exception):
    """Raised when required context is missing or stale. The caller escalates; it never guesses."""

def assemble(spec: dict, entity_key: str, now: datetime | None = None) -> dict:
    now = now or datetime.now(timezone.utc)
    facts, warnings = {}, []
    for inp in spec["inputs"]:
        rec = SOURCES[inp["source"]].fetch(entity_key, inp) # live read; nothing is copied first
        if rec is None:
            if inp.get("required"):
                raise ContextError(f"missing:{inp['name']}")
            warnings.append(f"optional input absent: {inp['name']}")
            continue
        age = now - rec.as_of
        if "max_age" in inp and age > parse_duration(inp["max_age"]):
            raise ContextError(f"stale:{inp['name']} age={int(age.total_seconds())}s")
        facts[inp["name"]] = {**rec.data, "source": inp["source"],
                                "as_of": rec.as_of.isoformat(), "age_s": int(age.total_seconds())}
    facts, resolved = resolve_conflicts(facts, spec.get("conflicts", {})) # agreed precedence, recorded
    return {"task": spec["task"], "task_version": spec["version"],
            "assembled_at": now.isoformat(), "complete": True,
            "facts": facts, "conflicts_resolved": resolved, "warnings": warnings,
            "definitions": SEMANTIC.resolve(spec["definitions"]),
            "policies": POLICY.versions(spec["policies"])}

```

Code 4. The same assembly exposed to agents as a Model Context Protocol tool [6]. Read rights are checked per source against the calling user, and every call is logged.

```

from mcp.server.fastmcp import FastMCP # modelcontextprotocol/python-sdk

mcp = FastMCP("context-layer")

@mcp.tool()
def get_context(task: str, entity_key: str, principal: str) -> dict:
    """Assemble the verified context a task needs. Returns facts with source and freshness,
    or an escalation. Read permissions come from the source systems, not from the model."""
    spec = load_spec(task)
    authz.check_read(principal, spec) # the user's rights, applied per source
    try:

```

```

    ctx = assemble(spec, entity_key)
except ContextError as e:
    audit.record("context.escalated", principal=principal, task=task, reason=str(e))
    return {"complete": False, "escalate": str(e)}
audit.record("context.served", principal=principal, task=task, ctx_hash=digest(ctx))
return ctx

```

Component 2 in practice: reuse the definitions you have

If finance already defines late delivery, margin or ROAS in the semantic layer, the agent uses that definition. Restating it in a prompt or a separate store creates a second, drifting source of truth. The context requirement in Code 1 lists definitions by name; retrieval resolves them from where they live.

Code 5. The metric the agent uses is the one the business already owns, here in dbt's MetricFlow format. The context layer references `late_delivery_rate`; it does not re-derive it.

```

# reuse, don't restate: the metric the agent uses is the one finance already owns (dbt / MetricFlow)
semantic_models:
  - name: deliveries
    model: ref('fct_deliveries')
    entities:
      - { name: shipment, type: primary, expr: shipment_id }
      - { name: supplier, type: foreign, expr: supplier_id }
    dimensions:
      - { name: promised_date, type: time, type_params: { time_granularity: day } }
    measures:
      - name: deliveries
        agg: count
        expr: shipment_id
      - name: late_deliveries
        agg: sum
        expr: case when actual_date > promised_date + sla_days then 1 else 0 end

metrics:
  - name: late_delivery_rate
    type: ratio
    type_params: { numerator: late_deliveries, denominator: deliveries }

```

This layer must be operated, not merely built

Source schemas change. APIs fail. Definitions evolve. Policies are revised. New entities appear. An approval matrix changes before its cached copy does. Without ownership and maintenance the context layer becomes one more stale silo, with the added risk that agents act on it.

Owner	Accountable for	Typical cadence
Business owners	Definitions, policies, exceptions, decision rules, the context requirement for each task	Review on each policy change; quarterly audit of definitions
Data and integration owners	Source mappings, refreshes, data quality, lineage, failure recovery	Freshness SLOs monitored continuously; schema changes handled before they reach agents
AI engineering owners	Retrieval logic, context composition, evaluation suites, model and tool compatibility	Evaluation on every change to a spec, prompt, tool or model
Security and risk owners	Access enforcement, action boundaries, audit, retention, incident response	Access review per workflow launch; incident drills on unauthorised-action and stale-context cases

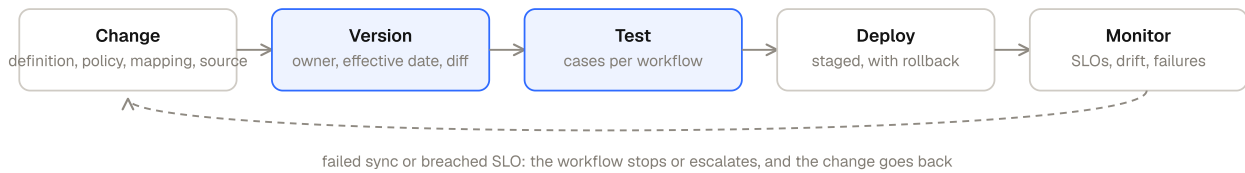


Figure 5. Context changes ship like code. A changed definition, policy or mapping is versioned, tested against the affected workflows' cases, deployed in stages and monitored against freshness service levels.

Class of fact	Example	Freshness limit	On breach
Live operational state	Shipment ETA, stock on hand, open approval	Minutes	Stop the task and escalate; never act on a stale value
Transactional records	Purchase order, invoice, case	Under an hour	Re-read from source; escalate if the source is unavailable
Reference and contract data	Contract terms, supplier master, approved alternates	A day	Proceed with a visible warning if the task allows; otherwise escalate
Definitions and policies	Metric logic, approval thresholds	Versioned, not time-based	A version change triggers regression tests before agents see it

Not all information needs to be copied into a new store. Frequently changing state is best retrieved live; stable definitions are managed centrally; authoritative policy documents can remain where they are. The architecture should minimise duplicated truth while making the required context dependable.

4 Worked example: supplier delivery exceptions

An organisation wants to cut the manual effort of resolving late supplier deliveries. Its analytics platform already predicts which shipments will miss their dates.

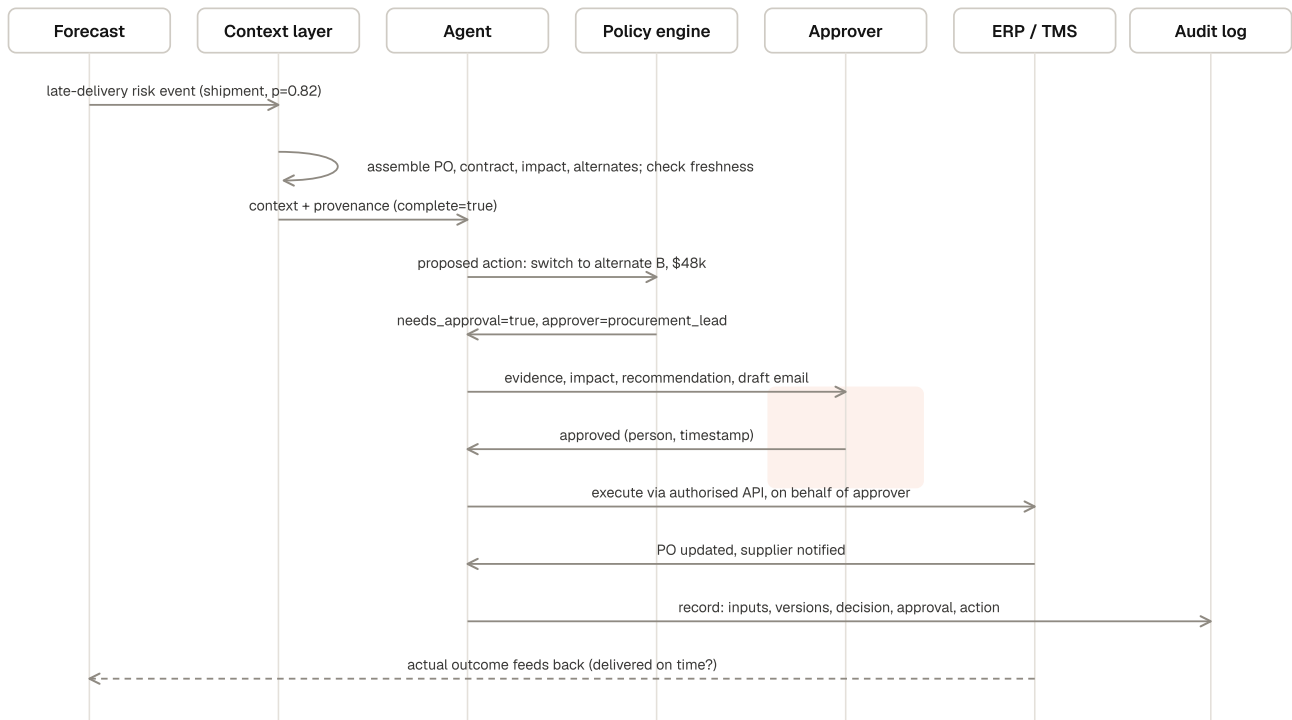


Figure 6. One case, end to end. The forecast raises the event; the context layer assembles and checks the evidence; the agent proposes; the policy engine decides whether a person is needed; the ERP enforces the write; the audit log records every step; the outcome feeds back.

The workflow as durable code

The recommendation and the enforcement are separate nodes. The agent reasons about the rule in `recommend`; the policy engine enforces it in `check_policy`; a person decides in `approve`; the ERP's own API enforces write rights in `execute`. The graph is checkpointed, so a case can wait hours for an approver without losing state.

Code 6. The workflow as a durable graph (LangGraph). A missing or stale context stops it at the first node. The approval step pauses the run until a named person answers.

```
from langgraph.graph import StateGraph, START, END      # langchain-ai/langgraph
from langgraph.types import interrupt
from langgraph.checkpoint.postgres import PostgresSaver

def gather(state):
    return {"ctx": get_context("resolve_late_delivery", state["shipment_id"], state["principal"])}

def recommend(state):
    if not state["ctx"]["complete"]:                    # missing or stale → stop here
        return {"outcome": "escalated", "reason": state["ctx"]["escalate"]}
    return {"plan": llm.plan(state["ctx"])}             # reasons about the rule

def check_policy(state):
    return {"decision": opa.decide("supplier/change", state["plan"]["args"])} # enforces the rule

def approve(state):
    if not state["decision"]["needs_approval"]:
        return {"approved": True, "approver": "policy:supplier.change@7"}
    answer = interrupt({"plan": state["plan"], "approver": state["decision"]["approver"]}) # waits, durably
    return {"approved": answer["approved"], "approver": answer["by"]}

def execute(state):
    if not state["approved"]:
        return {"outcome": "rejected"}
    erp.switch_supplier(*state["plan"]["args"], on_behalf_of=state["approver"]) # ERP enforces write rights
    return {"outcome": "executed"}

g = StateGraph(dict)
for node in (gather, recommend, check_policy, approve, execute):
    g.add_node(node.__name__, node)
g.add_edge(START, "gather"); g.add_edge("gather", "recommend"); g.add_edge("recommend", "check_policy")
g.add_edge("check_policy", "approve"); g.add_edge("approve", "execute"); g.add_edge("execute", END)
app = g.compile(checkpointer=PostgresSaver(conn))      # state survives the wait for a person
```

Code 7. The policy the workflow enforces, as Open Policy Agent Rego. Thresholds live in versioned policy data, owned by procurement, not in a prompt.

```
# policy as code (Open Policy Agent). The agent may reason about this rule;
# the workflow enforces it here, against trusted data, before anything executes.
package supplier.change

default allow := false
default needs_approval := true

# switching to an approved alternate, under the auto threshold: allowed without a person
allow if {
    input.action == "switch_supplier"
    input.new_supplier in data.approved_alternates[input.material_id]
    input.order_value_usd <= data.thresholds.auto_switch_usd
}

needs_approval if not allow

approver := "procurement_lead" if {
    input.order_value_usd > data.thresholds.auto_switch_usd
    input.order_value_usd <= data.thresholds.director_usd
```

```

}
approver := "procurement_director" if {
    input.order_value_usd > data.thresholds.director_usd
}

```

The record that makes a decision reviewable

Every case produces one record linking the trigger, the exact context used (by hash and source timestamps), the recommendation, the policy result, the approval, the action and the eventual outcome. This is what an auditor, a risk owner or an engineer debugging a bad decision reads first.

Code 8. One audit record. Anyone can reconstruct what the agent knew, what it proposed, who approved, what changed and what happened next.

```

{
  "case_id": "LD-2026-10-20-0417",
  "workflow": "resolve_late_delivery", "workflow_version": "3.2.0",
  "trigger": {"source": "forecast.delivery_risk", "event_id": "evt_71ac", "p_late": 0.82},
  "context": {
    "hash": "sha256:9f1c...", "assembled_at": "2026-10-20T06:41:12Z",
    "sources": ["tms.shipments@06:35Z", "erp.purchase_orders@06:10Z",
      "clm.contracts@22:00Z", "mes.schedule@06:25Z"],
    "semantic_version": "2026.10.2"
  },
  "recommendation": {
    "action": "switch_supplier", "to": "SUP-0192", "order_value_usd": 48000,
    "model": "<model id>", "prompt_version": "r14"
  },
  "policy": {
    "id": "supplier-change-approval", "version": 7,
    "result": "needs_approval", "approver_role": "procurement_lead"
  },
  "approval": {"by": "a.rao@example.com", "at": "2026-10-20T07:03:40Z", "decision": "approved"},
  "action": {
    "system": "erp", "call": "PurchaseOrder.switchSupplier",
    "status": 200, "at": "2026-10-20T07:03:52Z"
  },
  "outcome": {"delivered_on": "2026-10-23", "on_time": true, "recorded_at": "2026-10-23T15:20:00Z"}
}

```

Stages, with exit criteria

Stage	What happens	Exit criterion
Establish the baseline	Map the current process, exception categories, systems, handoffs and decisions. Measure time to resolution, analyst effort, avoidable downtime and the rate of incorrect interventions.	A written baseline the business owner signs
Inventory existing capabilities	Reuse warehouse definitions, forecasting outputs, supplier records, contract repositories and workflow APIs. List only the missing relationships, live status, policies and decision rights.	The context requirement (Code 1) drafted and owned
Assemble the operational context	Connect the predicted delay to the correct purchase order, supplier, contract, shipment status, production schedule and approved alternates. Record timestamps; resolve conflicts by agreed precedence.	Retrieval returns complete, fresh context for 95% of live cases; the rest escalate cleanly
Deploy with bounded authority	The agent gathers evidence, explains impact, recommends and drafts the communication. A qualified employee approves supplier or plan changes. Approved actions run through authorised systems.	Zero unauthorised actions; approvals and overrides logged; error rate at or below the human baseline
Learn from outcomes	Compare recommendations with expert decisions, inspect errors, find recurring data gaps. Test late updates, missing contracts, conflicting commitments, duplicate cases and unauthorised access before widening scope.	Evaluation suite green on all failure classes; scorecard reviewed against baseline

The same pattern applies to invoice exceptions, service-desk escalations, claims review, compliance investigations and any operation where information is fragmented and the last mile is manual.

5 Governance, reliability and the KPIs that matter

Governance is a property of the complete workflow, not a prompt instruction. Each control is enforced by the component best placed to enforce it.

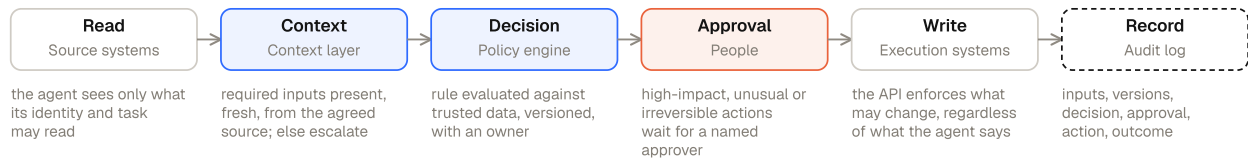


Figure 7. Six control points, each enforced where it belongs. The model is a participant in this chain, never the enforcer of it.

Source systems enforce read permissions; execution systems enforce write and action permissions. High-impact actions wait for approval, and missing, stale or conflicting context triggers escalation rather than fabricated certainty. Risk-based controls can draw on the [NIST AI Risk Management Framework and its Generative AI Profile](#) [7]. Microsoft Research's CI-Work benchmark shows why access alone is not enough: an agent can retrieve information that is useful and still inappropriate to share in that context [5].

Test the workflow, not the prose

Evaluation cases are written against the whole workflow and its failure classes. A green suite means the system escalated on stale data, chose the agreed source in a conflict, routed an over-threshold change to the right approver and refused an unauthorised caller.

Code 9a. Evaluation cases as data. Each names a situation and the outcome the workflow must produce.

```
# evals/resolve_late_delivery.yaml - the whole workflow is under test, not the model's prose
- id: stale-shipment-feed
  given: { shipment: { as_of: "-45m" } }
  expect: { outcome: escalated, reason: "stale:shipment" }

- id: conflicting-delivery-dates
  given: { tms.eta: "2026-10-27", erp.eta: "2026-10-23" }
  expect: { chosen_source: tms.shipments }

- id: over-threshold-switch
  given: { order_value_usd: 240000 }
  expect: { needs_approval: true, approver: procurement_director }

- id: unauthorised-principal
  given: { principal: "intern@example.com" }
  expect: { error: forbidden }

- id: duplicate-case
  given: { open_case_for_shipment: true }
  expect: { outcome: attached_to_existing }
```

Code 9b. The harness. The cases drive the real graph against faked sources, clock and policy data.

```
import pytest
from fixtures import World, load_cases

@pytest.mark.parametrize("case", load_cases("evals/resolve_late_delivery.yaml"), ids=lambda c: c["id"])
def test_workflow(case):
```

```
world = World(case["given"]) # fakes the sources, clock and policy data
result = app.invoke({"shipment_id": world.shipment_id, "principal": world.principal})
for key, expected in case["expect"].items():
    assert result[key] == expected, f"{case['id']}: {key}"
```

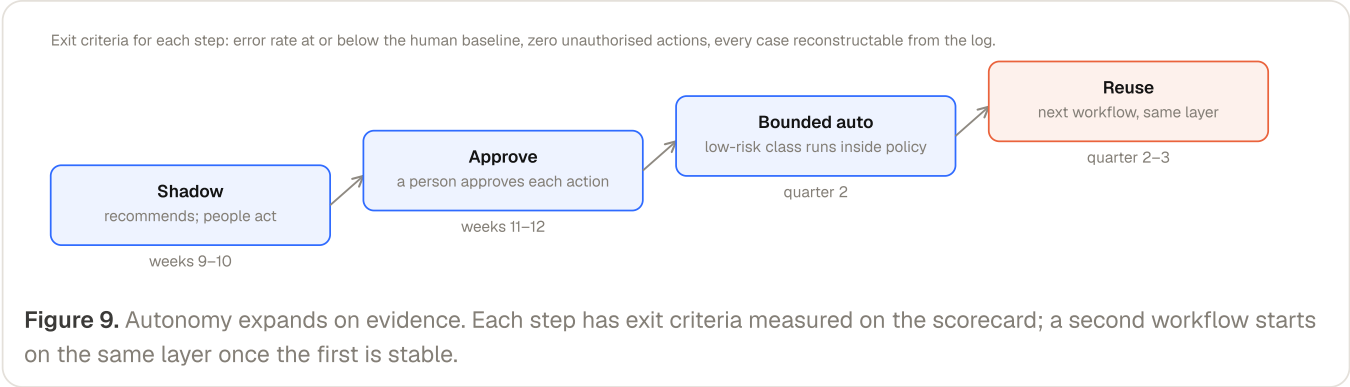
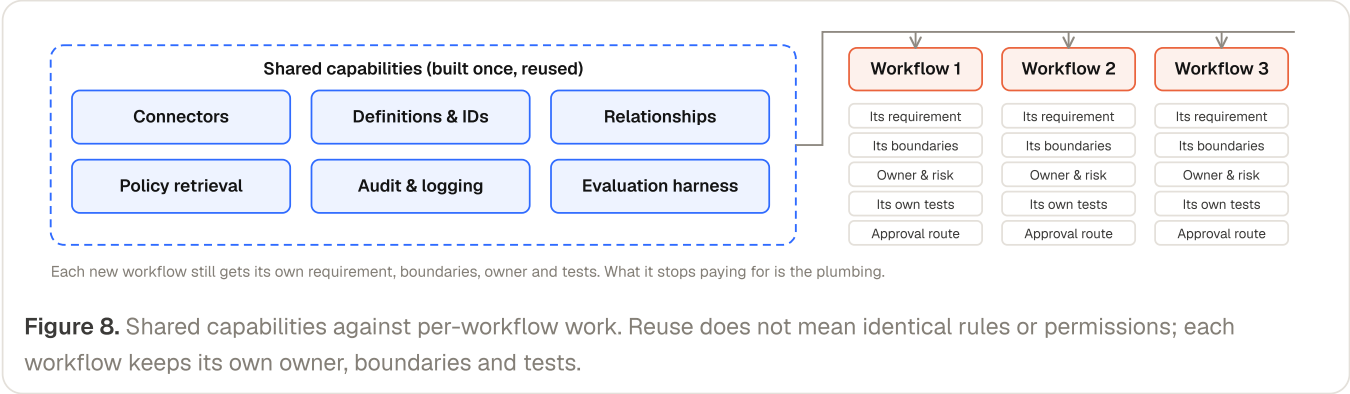
The scorecard

Dimension	Measures	Illustrative baseline → target
Business value	Resolution time, cost per case, rework, avoidable downtime, service levels	Median resolution 2 days → same day
Decision quality	Correct recommendations, material error rate, escalations	Error rate at or below the expert baseline; escalation rate falling as data gaps close
Operational reliability	Successful completion, stale-context incidents, integration failures, recovery time	Stale-context incidents trending to zero; recovery under one hour
Governance	Approval compliance, blocked unauthorised actions, audit completeness	100% of actions approved per policy; 100% of cases reconstructable
Economics and adoption	Human effort per case, cost per resolved outcome, usage, overrides, maintenance cost	Analyst minutes per case down; override rate down over the quarter

The right-hand column is illustrative, to show the shape of a target. Set real targets from your own baseline. Compare every measure against that baseline, including human-review costs and failure handling. A reduction in model tokens is useful only if it contributes to a better operating outcome.

6 Scale through reuse, not replication

Scaling should mean that the second and third workflows are easier because they reuse trusted connectors, definitions, entity relationships, policy retrieval, evaluation cases and audit mechanisms.



The strategic shift is from treating every agent as a standalone project to managing context, workflows and controls as shared organisational capabilities. That is the path from isolated experiments to a dependable operating model, one that converts the organisation's existing data and intelligence into completed, measurable work.

7 A 90-day plan

For one workflow, with one accountable owner. Dates assume the sources and APIs already exist.

01 Weeks 1–2: choose and baseline

Pick a workflow where a prediction or signal exists and the resolution is manual, with a measurable cost. Map it with the people who do it. Record the baseline: time, effort, error rate, downtime. Name the business owner and the risk owner.

02 Weeks 3–4: write the context requirement and inventory the stack

Draft Code 1 for the task. For each input, find the authoritative source, its access path and its freshness limit. List what exists and what is missing. Decide which rules become policy code and who owns them.

03 Weeks 5–8: build the seams

Connectors for the missing sources; retrieval with provenance and freshness (Code 3, 4); the first policies in a policy engine (Code 7); the audit record (Code 8). Reuse the semantic layer for definitions (Code 5). Write the evaluation cases (Code 9) before the agent is wired in.

04 Weeks 9–10: shadow mode

The agent assembles context and recommends; people act as before. Compare recommendations with decisions daily. Fix data gaps and conflicts as they appear. Run the evaluation suite on every change.

05 Weeks 11–12: approved execution

Turn on execution behind a named approver (Code 6). Review the scorecard against the baseline. Decide, on evidence, which low-risk class of action may run without a person in the next quarter, and which workflow comes next.

8 Open-source toolkit map

Mature projects that cover each component. Choose by fit with your stack; none of these is required, and several may already be in place.

Component	Project	Licence	Use it for
2 Definitions and semantics	dbt and MetricFlow	Apache-2.0	Modelled datasets and governed metric definitions agents can reference (Code 5)
2 Definitions and semantics	Cube	See repo	A semantic layer with APIs, when definitions must be served to many consumers
3 Relationships, lineage, ownership	DataHub , OpenMetadata	Apache-2.0	Entity relationships, ownership, glossary terms and lineage as a graph the layer can query
3 and 7 Lineage events	OpenLineage	Apache-2.0	A standard for emitting lineage from pipelines, so provenance in Code 2 has a source of truth
4 Freshness and quality	Great Expectations ; dbt source freshness	Apache-2.0	Checks that feed the freshness and quality rules in Code 1 and the SLO table

Component	Project	Licence	Use it for
5 Policy as code	Open Policy Agent	Apache-2.0	Approval thresholds and action rules evaluated outside the model (Code 7)
6 Agent-facing retrieval	MCP Python SDK	MIT	Exposing <code>get_context</code> and other governed tools to any MCP-capable agent (Code 4)
Workflow and approval	LangGraph , Temporal	MIT	Durable, checkpointed workflows that can wait for a person (Code 6)
Scheduled refresh	Apache Airflow	Apache-2.0	Refreshing reference data and running evaluation suites on a schedule
7 Observability and evaluation	OpenTelemetry ; Langfuse , Arize Phoenix , promptfoo	Apache-2.0; see repos; MIT	Traces that link context, tool calls and outcomes; running the evaluation cases in Code 9

Licences were read from each repository on 20 October 2026. Check them again before adoption, and check your organisation's policy on projects with mixed or source-available licences.

References

1. McKinsey & Company, *The State of AI in 2025: Agents, Innovation, and Transformation*, November 2025. Survey of 1,993 respondents across 105 nations. mckinsey.com
2. IBM Institute for Business Value, *IBM Study: CEOs Double Down on AI While Navigating Enterprise Hurdles*, May 2025. Survey of 2,000 CEOs across 33 countries and 24 industries. newsroom.ibm.com
3. Gartner, *Gartner Predicts Over 40% of Agentic AI Projects Will Be Canceled by End of 2027*, press release, 25 June 2025. gartner.com
4. Anthropic, *Effective Context Engineering for AI Agents*, 29 September 2025. anthropic.com
5. Microsoft Research, *CI-Work: Benchmarking Contextual Integrity in Enterprise LLM Agents*, ACL Industry Track 2026. microsoft.com/research
6. Model Context Protocol, *Specification*. modelcontextprotocol.io
7. NIST, *AI Risk Management Framework* and *Generative AI Profile* (NIST AI 600-1). nist.gov

Code samples are illustrative and abbreviated. Names such as `SOURCES`, `authz`, `audit`, `opa` and `exp` stand for your own adapters. The supplier, order and value figures are invented.



Meerkats builds the context layer that AI agents, workflows and apps run on: connections to your systems, business definitions kept current, rules and permissions enforced, and a record of every action. This note describes the approach in general terms; the design applies whether you build the layer yourself or run it on Meerkats. meerkats.ai