Agent Framework

Production AI Agent Checklist

Use this checklist to review model access, context, memory, tools, orchestration, review, latency, cost, observability, safety, and auditability before trusting an agent in production.

01-04 Build: Design and Development

The structural choices that make an agent portable, grounded, useful, and maintainable before it ever reaches production traffic.

05-08 Run: Production Operations

The operating controls that keep agentic systems responsive, affordable, recoverable, and reviewable while real users depend on them.

09-11 Govern: Risk and Quality

The quality, safety, and audit practices that make an agent trustworthy enough to improve rather than merely impress.

01 Model Serving Layer

BUILD

Make model choice a runtime decision

Put every model provider behind a stable serving contract so the product can route, fail over, compare, and upgrade models without rewriting feature code.

BUILD THIS

- [ ] A typed request and response contract for chat, structured output, tool calls, streaming, and usage data.
- [ ] Provider adapters that normalise errors, retry semantics, model capabilities, and token accounting.
- [ ] A routing policy that chooses models by task class, budget, latency target, context length, and fallback priority.
- [ ] A capability registry for JSON mode, tool calling, vision, reasoning effort, context size, and cache support.

WATCH FOR

- [ ] Prompt code importing provider SDKs directly.
- [ ] Fallbacks that return a different schema or silently drop tool support.
- [ ] Model upgrades shipped without a baseline eval or latency comparison.
- [ ] Cost reports that cannot explain which feature, tenant, or task spent the tokens.

PROOF IT WORKS

- [ ] A model can be swapped by configuration for at least one task path.
- [ ] Traces record provider, model, routing reason, retry count, latency, and token usage.
- [ ] Failover has been tested against provider timeout, rate limit, schema failure, and safety refusal cases.

IMPLEMENTATION CHECKLIST

- [ ] Define the application contract first, then map each provider into it.
- [ ] Keep prompts and routing rules versioned together so output changes can be traced.
- [ ] Log every model selection as a decision, not just as metadata.
- [ ] Run upgrade evals before changing the default model for a production task.

02 Context Management

BUILD

Treat context like a production data pipeline

Design the full context pipeline: instructions, task state, retrieved evidence, user history, tool results, compression, ordering, and expiry.

BUILD THIS

- [ ] Separate lanes for policy, task instructions, user state, retrieved knowledge, tool results, and scratch artifacts.
- [ ] Ranking, pinning, trimming, summarisation, and expiry rules for every lane.
- [ ] Prompt and context bundle versions that can be compared in evals.
- [ ] Source provenance so the agent and reviewer can tell durable truth from generated text.

WATCH FOR

- [ ] Stuffing whole files or whole histories into every request.
- [ ] Retrieved chunks with no source, timestamp, or reason for inclusion.
- [ ] Tool results trusted as instructions instead of treated as untrusted data.
- [ ] Context changes that cannot be replayed because prompts live only in code comments or dashboards.

PROOF IT WORKS

- [ ] Traces show exactly which context blocks entered the request and why.
- [ ] A long-running session can shed stale context without losing current task state.
- [ ] A regression test catches when a prompt or retrieval change harms a known task.

IMPLEMENTATION CHECKLIST

- [ ] Model the context window as a budget with named allocations.
- [ ] Keep instructions at the boundary, data in data lanes, and tool output quoted or tagged.
- [ ] Prefer pointers to bulky source material until the agent actually needs the content.
- [ ] Put context assembly under tests the same way you test application logic.

03 Memory Management

BUILD

Give agents memory they can trust

Give the agent deliberate short-term and long-term memory with retention rules, retrieval policy, ownership, and stale-memory controls.

BUILD THIS

- [ ] A short-term state store for the active task, separate from long-term facts.
- [ ] Memory records with owner, scope, source, created date, last validated date, and confidence.
- [ ] Retrieval rules that load only memories relevant to the current task and user.
- [ ] Correction, expiry, and deletion flows when memory is wrong or no longer allowed.

WATCH FOR

- [ ] Storing every conversation turn as long-term truth.
- [ ] Loading stale preferences after the user has changed their mind.
- [ ] Sharing private agent memory across users, tenants, or scopes.
- [ ] Letting generated summaries replace primary source records.

PROOF IT WORKS

- [ ] The agent can explain which memory influenced an answer.
- [ ] A changed preference overrides the older memory in the next relevant task.
- [ ] Private, tenant, and shared memories have separate access tests.

IMPLEMENTATION CHECKLIST

- [ ] Classify memories as session, project, user, organisation, or global before storing them.
- [ ] Attach memory to primary sources where possible rather than storing unsupported claims.
- [ ] Cap memory retrieval by task relevance and attention budget.
- [ ] Build a review surface for editing or retiring important memories.

04 Integrations and MCP

BUILD

Connect agents through controlled tools

Expose external tools, APIs, files, and data sources through a consistent integration layer that is discoverable, permissioned, and observable.

BUILD THIS

- [ ] Tool definitions with tight schemas, clear descriptions, and least-privilege credentials.
- [ ] A registry that separates read, write, destructive, and externally visible actions.
- [ ] MCP servers or adapter services that wrap real APIs without leaking implementation details.
- [ ] Tool result formatting that returns focused evidence instead of raw dumps.

WATCH FOR

- [ ] Tools that accept free-form strings where structured inputs would work.
- [ ] One broad tool that can do everything instead of narrow tools with clear intent.
- [ ] Credentials available to the agent when the task does not need them.
- [ ] Tool results that are too large, too vague, or impossible to audit.

PROOF IT WORKS

- [ ] Every tool call is logged with caller, input summary, output summary, latency, and permission decision.
- [ ] Dangerous tools require approval or a constrained sandbox.
- [ ] A failing integration returns a typed error the agent can recover from.

IMPLEMENTATION CHECKLIST

- [ ] Start with read-only tools, then add write tools behind explicit approval gates.
- [ ] Validate tool inputs and outputs at the integration boundary.
- [ ] Return IDs, links, and short summaries so reviewers can inspect primary sources.
- [ ] Keep MCP as a protocol choice, not the only design principle.

05 Orchestration

RUN

Coordinate agents with reliable workflows

Coordinate models, tools, workflows, agents, queues, retries, checkpoints, and recovery paths for complex tasks.

BUILD THIS

- [ ] A task graph or state machine that names each step, owner, timeout, retry policy, and completion condition.
- [ ] Durable checkpoints so long-running work can resume without rerunning successful steps.
- [ ] Clear boundaries between workflow steps, agent loops, subagents, and deterministic validators.
- [ ] Cancellation, compensation, and escalation paths for partial failure.

WATCH FOR

- [ ] One general agent asked to manage every step without state or deadlines.
- [ ] Retries that repeat non-idempotent actions.
- [ ] Subagents with overlapping responsibilities and no conflict resolution.
- [ ] Failures hidden inside natural language summaries instead of surfaced as states.

PROOF IT WORKS

- [ ] A task can be paused, resumed, cancelled, and inspected from durable state.
- [ ] Each step has an owner, input contract, output contract, and timeout.
- [ ] A known tool failure triggers a recoverable path instead of an endless loop.

IMPLEMENTATION CHECKLIST

- [ ] Draw the workflow first, then decide where agent judgement is actually needed.
- [ ] Make side effects idempotent or guarded by explicit operation IDs.
- [ ] Persist enough state to explain progress without reading raw logs.
- [ ] Use specialist agents only when specialisation reduces context or risk.

06 Human in the Loop

RUN

Put human review where it changes outcomes

Design approval, review, escalation, and override paths for decisions where autonomy is too risky or too ambiguous.

BUILD THIS

- [ ] Risk tiers for actions: safe read, reversible write, external send, destructive change, financial or legal impact.
- [ ] Approval payloads that show intent, diff, evidence, blast radius, and rollback plan.
- [ ] Resume logic so the agent continues from the approved decision instead of starting over.
- [ ] Escalation paths for uncertainty, repeated failure, policy conflict, and user disagreement.

WATCH FOR

- [ ] Review prompts that ask for approval without showing what will actually happen.
- [ ] Humans approving too many low-risk steps and missing the important ones.
- [ ] Agents continuing after a rejection without understanding the reason.
- [ ] Approval decisions missing from the audit trail.

PROOF IT WORKS

- [ ] Dangerous actions cannot execute without the required approval record.
- [ ] A rejected action changes the next agent step, not just the UI state.
- [ ] Reviewers can understand the request without reading raw traces.

IMPLEMENTATION CHECKLIST

- [ ] Define risk tiers before building the approval UI.
- [ ] Use diffs, previews, and linked evidence rather than generic confirmation text.
- [ ] Capture who approved, what they approved, when, and against which artifact version.
- [ ] Measure approval burden so guardrails do not become noise.

07 Rate Limits and Latency

RUN

Keep agents fast when providers slow down

Keep agents responsive under provider limits, slow tools, concurrent users, long context, retries, and multi-step workflows.

BUILD THIS

- [ ] Timeouts and deadlines for model calls, tools, workflow steps, and whole tasks.
- [ ] Queues, concurrency caps, exponential backoff, and provider-specific rate limit handling.
- [ ] Streaming, progress events, partial results, and background completion for slow paths.
- [ ] Caches for stable retrieval, prompt prefixes, tool results, and deterministic computations.

WATCH FOR

- [ ] Retry storms after a provider starts throttling.
- [ ] Parallel agents that multiply rate limit pressure without improving user-perceived speed.
- [ ] Long context windows used as the default path for small tasks.
- [ ] A UI that looks frozen while a long-running task is still healthy.

PROOF IT WORKS

- [ ] Load tests include provider throttling, slow tools, and retry behaviour.
- [ ] The product has a user-visible state for queued, running, waiting for review, failed, and complete.
- [ ] Metrics separate model latency, tool latency, queue time, and orchestration overhead.

IMPLEMENTATION CHECKLIST

- [ ] Set a latency budget per task class before choosing models and tools.
- [ ] Prefer fast routing for simple tasks and escalation for hard tasks.
- [ ] Use backpressure and queue depth alerts instead of letting requests pile up invisibly.
- [ ] Cache only when correctness, freshness, and invalidation are explicit.

08 Cost Control

RUN

Control LLM spend before it surprises you

Track and shape LLM spend by task, model, tenant, user, feature, token type, retry path, and business value.

BUILD THIS

- [ ] Usage accounting for input tokens, output tokens, cache tokens, tool calls, and retries.
- [ ] Budgets by environment, tenant, user, feature, workflow, and background job.
- [ ] Model routing rules that reserve expensive models for tasks that justify them.
- [ ] Kill switches, soft limits, degradation paths, and alerts before spend becomes an incident.

WATCH FOR

- [ ] Only tracking total provider invoice cost.
- [ ] Long agent loops with no max step count or budget ceiling.
- [ ] Background jobs using frontier models because the default client does.
- [ ] Caching decisions made without measuring freshness or cache hit quality.

PROOF IT WORKS

- [ ] A trace can show the cost of a single task and the reason for each expensive call.
- [ ] Budget limits are tested for both foreground and background work.
- [ ] Cost dashboards can rank spend by feature, tenant, and model.

IMPLEMENTATION CHECKLIST

- [ ] Attach cost metadata to every model request at the serving layer.
- [ ] Set per-task token and step budgets, then make overruns explicit states.
- [ ] Measure quality before and after routing a task to a cheaper model.
- [ ] Review high-spend traces regularly, not only when invoices arrive.

09 Evaluation and Observability

GOVERN

Make agent quality measurable and observable

Measure quality and expose system behaviour through evals, traces, labels, dashboards, alerts, and regression checks.

BUILD THIS

- [ ] Offline eval suites for known tasks, edge cases, regressions, and adversarial inputs.
- [ ] Online quality signals from user outcomes, reviewer labels, retries, edits, and abandonment.
- [ ] Traces that link model calls, context blocks, tool calls, workflow states, approvals, and artifacts.
- [ ] Dashboards for quality, latency, errors, spend, refusal rate, tool failure rate, and escalation rate.

WATCH FOR

- [ ] Demo examples used as evals without expected outputs or scoring rules.
- [ ] Logs that record final text but not context, tool calls, or model versions.
- [ ] Quality measured only by thumbs-up feedback from users who bothered to click.
- [ ] Alerts for infrastructure errors but not quality regressions.

PROOF IT WORKS

- [ ] A prompt, model, retrieval, or tool change runs against a baseline before release.
- [ ] A bad production answer can be traced to the exact context and tool evidence used.
- [ ] Human review labels feed back into eval cases or product metrics.

IMPLEMENTATION CHECKLIST

- [ ] Start with task-specific evals instead of one generic quality score.
- [ ] Instrument the full request path before adding more agent autonomy.
- [ ] Store traces with enough IDs to connect product events and backend logs.
- [ ] Promote real incidents into regression tests.

10 Safety and Guardrails

GOVERN

Keep agents inside clear safety boundaries

Constrain agent behaviour with input handling, output validation, permissions, sandboxing, policy checks, and circuit breakers.

BUILD THIS

- [ ] Threat models for prompt injection, data exfiltration, unsafe tool use, policy violations, and destructive actions.
- [ ] Input classification and output validation for structured data, generated content, and external messages.
- [ ] Permission boundaries by user, tenant, tool, environment, and action risk.
- [ ] Circuit breakers for repeated failures, abnormal spend, policy hits, and unsafe outputs.

WATCH FOR

- [ ] Relying on a system prompt to protect secrets or production systems.
- [ ] Tool descriptions that reveal capabilities the user should not access.
- [ ] Agents reading untrusted content and treating it as instructions.
- [ ] Safety checks that happen after the external side effect has already happened.

PROOF IT WORKS

- [ ] Prompt injection tests cannot make tools access forbidden data.
- [ ] Unsafe or policy-violating output is blocked before external delivery.
- [ ] A runaway loop, cost spike, or repeated tool failure trips a circuit breaker.

IMPLEMENTATION CHECKLIST

- [ ] Apply least privilege to every tool and credential.
- [ ] Run untrusted work in a sandbox or constrained environment.
- [ ] Validate structured output with schemas and reject invalid states.
- [ ] Keep high-risk actions behind approval and make rollback part of the action design.

11 Reproducibility and Audit

GOVERN

Make every agent decision replayable

Capture enough state to replay, explain, review, and defend important agent decisions after the fact.

BUILD THIS

- [ ] Decision records that link user request, task state, prompt version, context references, model version, and output artifact.
- [ ] Tool call logs with input summary, output summary, external IDs, errors, and side effects.
- [ ] Artifact versioning for generated files, messages, plans, approvals, and final results.
- [ ] Replay tooling for important paths, with clear limits where exact deterministic replay is impossible.

WATCH FOR

- [ ] Only storing the final answer.
- [ ] Logs that include sensitive data without redaction or retention rules.
- [ ] Changing prompts or tools without versioning the old behaviour.
- [ ] Audit records that cannot connect model output to the action users saw.

PROOF IT WORKS

- [ ] A production incident can be reconstructed from stored IDs and trace records.
- [ ] Reviewers can see which context, memory, and tool results influenced the final action.
- [ ] Prompt, model, tool, and approval versions are preserved for high-impact actions.

IMPLEMENTATION CHECKLIST

- [ ] Decide audit requirements per risk tier before launch.
- [ ] Store pointers and hashes for bulky artifacts when storing the full content is unsafe or costly.
- [ ] Redact secrets at capture time, not only in the viewer.
- [ ] Turn every serious failure into a replayable case or a documented limitation.