Agentic AI Foundation

AGNTCon + MCPCon North America

Where the **agentic stack** is **being built**.

Stop Writing Agents, Declare Them

Declarative Agent Architecture in Production

Chris Knuteson · A Place for Mom

A Place for Mom

Helping families connect to care

Senior care referral since 2000 · free to families · 20,000+ communities and home care providers

Grace – Self-Service Chat Agent

Our AI home care advisor, in production since November 2025

2,526

agent runs

chat-agent runs · 1,767 sessions · 404 users

19,362

model calls

generations, incl. speech and embeddings

17

prompt packs

on the production label, resolved at runtime

Modest volume. Real families.

Grace production telemetry (Langfuse), 30 complete days, Sep 19 – Oct 18 2026 UTC.

Unpredictable

How do you know what each agent is running?

1 Unpredictable → 2 Spec → 3 Proven deliverable

The version number went backwards

Code + Langfuse + serving = ??

17 promoted · 0 skipped · 0 failed

version: 47 → version: 41

"Is it expected that the version number is going down?"

OpenAI SDK + at-runtime invocation + Langfuse + code fallback.
Nobody could say from the PR which agent production was actually running.

Each invocation depended on three sources

What a turn depended on

Git the app commit

Langfuse wherever the production label pointed, re-read every 60 s

Bundle base + profile packs merged in memory

```
// Fetch by label with caching
const label =
  options?.label ?? this.config.prompts.defaultLabel;
const cacheTtlSeconds = options?.cacheTtlSeconds ?? 60;

langfusePrompt = await this.langfuseClient.prompt.get(
  packId, { label, cacheTtlSeconds },
);
```

chat-persona-base today: 43 Langfuse versions; its own version field says 48.
Two counters, one agent.

To explain any one conversation we reconstructed the combination after the fact. Nothing named the agent.

Spec

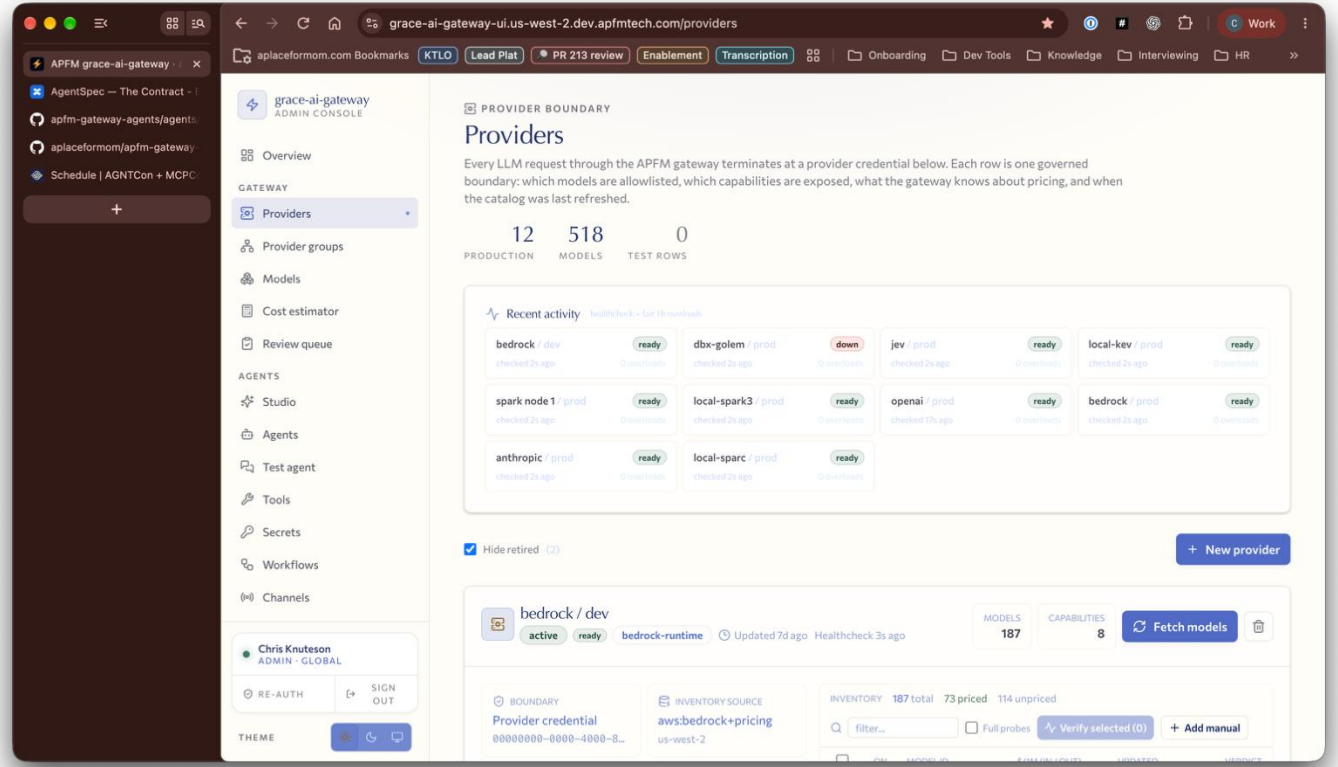
Declare the agent as one versioned document.

1 Unpredictable → 2 Spec → 3 Proven deliverable

Where the **agentic stack** is **being built**.

SPEC

Building block: AI Gateway



Where the **agentic stack** is **being built**.

Make the agent a release asset

AgentSpec one versioned document that declares one agent: identity, instructions, skills, tools, model, runtime, budget, policy.

Names the agent, its model and runtime

Carries instructions and 16 skill references

Requests capabilities by name; grants stay with the platform

```
"id": "review.apfm-agentic-review",
"runtime": "bedrock-agentcore-harness",
"model": "anthropic/claude-opus-4-7",
"harness": { "max_iterations": 80, "max_tokens": 16000 },
"spec": {
 "instructions": "You are the APFM Agentic PR Reviewer …",
 "skills": [ "apfm-review-orchestration", … 16 refs ],
 "tools": [ "review.call-all-dimensions",
          "connection.github-review-readonly" ],
 "surface": "annotate"
}
```

Actual reviewer source, selected fields. The compiler turns instructions into the harness system prompt.

AgentSpec is one typed contract

Typed

the compiler rejects an invalid agent before it ships

Versioned

every field is tagged v1, v2 or deferred, so the contract can grow

Indirect

names a gateway key and tool refs; the platform resolves models and grants

```
export const AgentSpec = z.object({
  id:      z.string(),  // "grace.intake"
  version:  z.string(),  // v1, v2 …
  runtime:  z.enum(["in-process", "bedrock-agent-core"]),
  llm:     { gatewayKey },  // model + fallback at gateway
  prompt:   { registryId, label, fragments },
  tools:    { refs: [{ id, requireApproval }] },
  triggers: z.array(TriggerSpec).min(1),
  outputSchema, surface,  // validated output
  scope:    { practice, workload },
  budget:   { maxTurns: 10, perTurnTimeoutMs: 15000 },
  workspace, context, capabilities,  // v2
  memory,  // deferred
});
```

AgentSpec schema, top-level fields. Full definition in the appendix.

One manifest declares the reviewer

Root the entry point and its handoff tool

Children three specialists, each its own AgentSpec

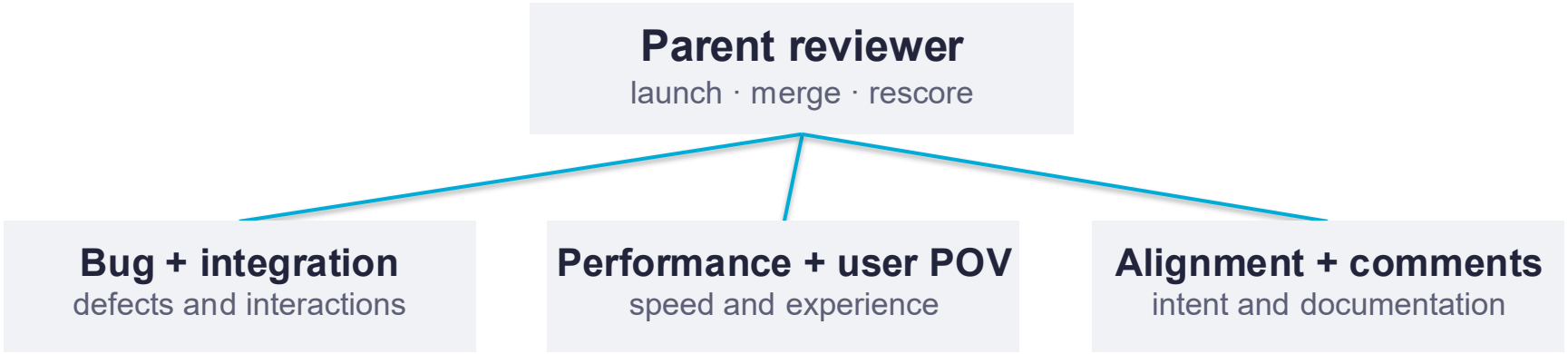
Skills versioned bodies the closure depends on

```
"root_agent": "review.apfm-agentic-review",
"agents": [
 { "id": "review.child.bug-integration",
   "path": "child-bug-integration/agent.json" },
 { "id": "review.child.perf-userpov", … },
 { "id": "review.child.alignment-comments", … }
],
"skills": [ … 16 registered skill entries … ]
```

A behavior change to one specialist is a diff against this manifest. Reviewers see the blast radius before it ships.

Plain JSON. What makes it a desired state is the contract the compiler enforces, not the file format.

One asset, four managed harnesses



AgentCore runs the fan-out, outside Grace's application runtime. Each box is its own managed harness with its own session, budget and receipt.

Grace is not out of capacity. We chose the boundary before child runs, sessions and lifecycle became Grace's operating overhead.

"In production": DEV-hosted agents on production engineering workloads, the PRs we merge. PROD is the promotion target.

Each release is named by its digest

Pull request compile and dependency-check the affected definitions. Never publishes.

Merge to main submit changed source as a DEV candidate, named by its bundle digest.

Promotion stage · run the sealed canary suite · activate · roll back. Explicit, auditable steps.

Oct 6, one agent, one chain

commit 2687ce60 → bundle **614017b6…** → agent **ops.incident-triage-live.release-614017b6…** → canary run succeeded 11 minutes later

The release name carries the source digest, so the receipt answers "what ran?" by itself. Live for incident-triage, not yet for the reviewer.

Proven deliverable

Prove which release ran, on real pull requests.

1 Unpredictable → 2 Spec → 3 Proven deliverable

Where the **agentic stack** is **being built**.

PROVEN

Promotion held; rollback aged out

Candidate	merge submits a digest-named release; nothing is live until promoted
Canary	a synthetic suite is sealed into the bundle and must pass before activation
Activation	one revision marker in the reply proves which release answered
Rollback	refused when the target was sealed on a retired platform release; the lesson we did not expect

Tradeoff learned: an agent release is pinned to the platform release it was sealed on. Rollback targets age out when the platform moves.

Exercised on ops.incident-triage-live, Oct 6–16 2026, DEV. The reviewer still lives in the GUI and is synced back to Git.

PROVEN

Trust the evidence, not the agent

1 · THE AGENT CLAIMS

“Clean”

The parent agent reports the review is done.

2 · THE PLATFORM CHECKS

Its own run records

- Coverage marked complete
- Every expected specialist succeeded
- Every attempt resolved
- Not cut down to one agent

3 · THEN IT POSTS

Review posts to GitHub

Trusted code posts it. Any check fails: nothing posts.

Proves

The configured panel ran and the result reached GitHub.

Not yet proved

That findings are right: no precision or recall measured, so auto-merge stays off.

PROVEN

1,928 reviews across 26 repos

Agentic AI Foundation
AGNTCon + MCPCon North America

26

repos

of 29 subscribed repos produced a completed review

1,928

reviews delivered

successful workflow runs

1,017

pull requests

distinct PR branches reviewed

3,850

runs

1,382 draft skips · 367 superseded · 173 failed

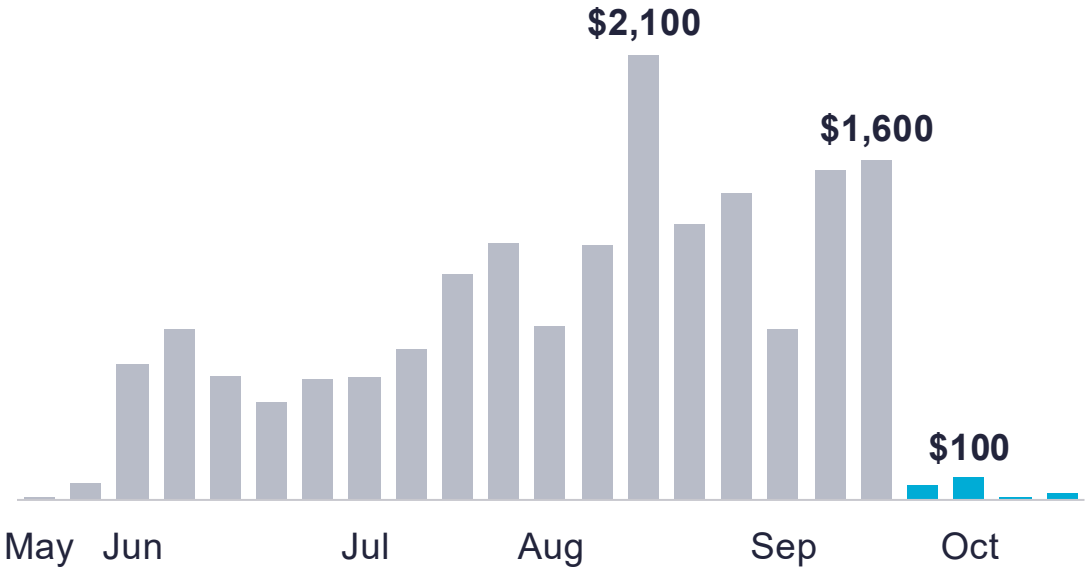
Window	Sep 19 – Oct 18 2026 UTC, 30 complete days, GitHub Actions run records
Engine	Gateway reviewer on every sampled run from Sep 25; legacy engine sampled on Sep 22
Not shown	finding accuracy, acceptance or cost: a run record measures none of these

A run record proves a review was produced and posted. It does not prove the review was right.

PROVEN

API bill fell 96% on the Gateway

Weekly cost, apfm-agentic-review Anthropic key



Before $1,300 a week in the 3 weeks to Sep 27

After $59 a week in the 3 weeks from Sep 28

Excludes Bedrock and AgentCore spend after the move

Source: Anthropic API bills, Claude Code cost on the apfm-agentic-review key, May 18 – Oct 21 2026, weeks start Monday; week of Oct 19 partial.

Two systems, one question: what ran?

	Grace (chat, prod)	Gateway reviewer (DEV)
Definitions	17 prompt packs on the production label; the agent is assembled in-process	4 AgentSpecs + 16 skills in one manifest, compiled to harness config
Usage (30 d)	2,526 agent runs · 11,122 traces · 19,362 model calls	1,928 reviews delivered · 1,283 parent + 4,099 child runs (14 d receipts)
Identity	app commit + label at run time; versions flattened into a string	sealed revision + session id + coverage receipt
Custody	Langfuse label, re-read every 60 s	GUI-managed, synced to Git on Oct 15; Git binding next
Execution	one NestJS process, OpenAI Agents SDK	AgentCore harness sessions, parent + 3 children

Different workloads: a chat turn and a ten-minute review. Latency and cost are not compared on purpose.

PROVEN

Two of four layers port as-is

AgentSpec + release identity	ids, instructions, skill refs, manifest, bundle digest	**Portable. Plain documents and digests.**
APFM policy and promotion	compile, authorization, candidate → canary → activate → rollback	**Ours. Written against the Gateway; the mechanism travels, the code does not.**
AgentCore execution	harness runtimes, versions and endpoints, microVM sessions, InvokeHarness	**Provider-specific. An adapter re-implements compile, skill mounting, session and child-run evidence, credentials.**
Consumers and delivery	GitHub router, coverage gate, posting, approval policy	**Ours, runtime-agnostic by construction: it only sees receipts.**

JSON is not portability. Still coupled: harness semantics (iterations, stop reasons), the child-handoff protocol, log and metric shapes.

Lessons for another platform team

1. **Name the release in the thing that runs**
   a digest in the agent name turns a log line into an identity receipt
2. **Separate submit, activate and roll back**
   and expect rollback targets to age with the platform release
3. **Get coverage from the control plane**
   a model saying “done” is not evidence; child-run records are
4. **Keep write authority out of the agent**
   the runtime returns text; trusted code posts, approves, merges
5. **Treat exports as snapshots**
   custody is a handoff you enforce, or the GUI quietly wins

Everything above came from running agents and getting the release question wrong first.

Where the **agentic stack** is **being built**.

NEXT STEPS

What comes next

Now	incident-triage on Git custody with sealed canaries and a rollback proof; the reviewer serving 26 repos from DEV, GUI-managed
Next	hand the four review agents to Git custody; run the reviewer on a release-named endpoint instead of DEFAULT
Then	linting on top of the canonical compiler — no second schema; registry references resolved into the release
Proposal	promote the reviewer to PROD through the same candidate → canary → activate path. Not started.

Sequenced by blast radius: prove the path on an ops agent, then move the one everyone depends on.

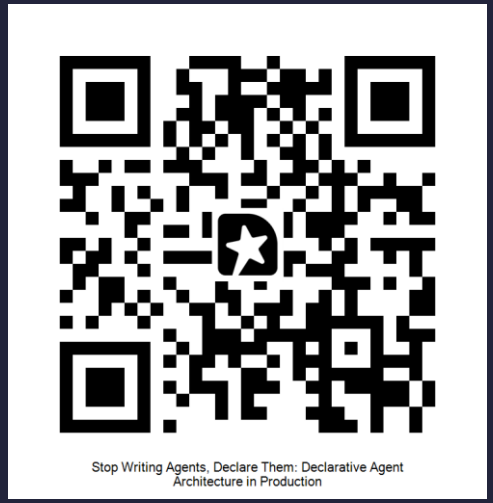
"Next" onward are proposals. The platform costs are real: compiler maintenance, contract evolution, running the promotion path.

Thank you

Agentic AI Foundation
AGNTCon + MCPCon North America

Ship the agent. Keep the receipt.

Name the release before you trust the run.



Scan to leave session feedback

Chris Knuteson · Director of Agentic, A Place for Mom

Questions

Versioning fixes in Grace

Jun 2	promotion PR proposed metadata v47 → v41; closed unmerged
Jun 16	semantic pack versions separated from Langfuse counters; downgrades and missing bumps rejected
Jul 1	repo v1 → production v5 bookkeeping; content-identical, no behavior change
Today	chat-persona-base: 43 Langfuse versions, pack version field 48 — the counters still differ

Pins and version guards fixed prompt movement. They never named the whole agent: instructions, skills, children, runtime, policy.

APPENDIX

Metric definitions

Grace usage	Langfuse, environment prod, Sep 19 – Oct 18 2026 UTC; agent run = root trace “Agent workflow”; model call = GENERATION observation
Reviews delivered	GitHub Actions, workflow “Agentic PR Review”, 29 subscribed repos, runs created Sep 19 – Oct 18 UTC; success = run conclusion success
Pull requests	distinct (repo, head branch) with a successful run — an upper bound on PRs
Parent / child runs	DEV control-plane log lines “channel invoke run … succeeded”, Oct 5–18 UTC (14-day retention)
Latency	“Run review agent in Grace” step duration, grace-ai-gateway only, 84 completed runs Oct 7–16
Not measured	finding precision/recall, acceptance, token cost, PROD traffic, per-run engine for all 3,850 runs

APPENDIX

Three receipts, one review

GitHub run	Oct 8 · agent step 17:51:29 – 17:56:34 UTC · review posted 17:56:38 · HTTP 200
Control plane	parent succeeded 17:56:32 UTC · coverage v2: 3 expected, 3 succeeded, 0 failed, 0 recovered
Policy artifact	result SHA-256 bound to repo, run and attempt · risk 4/10, floor 3/10 · approve=false

What this run does not prove: which source commit produced the reviewer that ran. That join exists for the Git-managed agent, not yet for this one.

grace-ai-gateway PR #781, invocation 7f60b245…; the one run whose policy artifact was preserved locally.

APPENDIX

AgentSpec schema, full (1 of 2)


```
import { z } from "zod";
export const AgentSpec = z.object({

  // ---- Identity ----
  id:  z.string().regex(/^[a-z][a-z0-9-]*(\.[a-z][a-z0-9-]*)+$/),  // v1
     // dotted namespace, e.g. "grace.intake"
  version:  z.string().regex(/^v\d+$/), // v1
  name:  z.string().min(1).max(64),  // v1
  description: z.string().max(2000).optional(),  // v1

  // ---- Type + Runtime (kagent-inspired) ----
  type:  z.enum(["Declarative", "BYO"]).default("Declarative"),  // v1 (BYO = deferred)
  runtime: z.enum(["in-process", "bedrock-agent-core"])  // v1 (agent-core = deferred)
         .default("in-process"),

  // ---- LLM policy (via gateway key) ----
  llm: z.object({  // v1
    gatewayKey: z.string().min(1),
      // gateway resolves: provider, model, fallback chain, PII hop,
      //  budget, rate, audit, killswitch
  }),

  // ---- Tools ----
  tools: z.object({  // v1 (req'd-approval = v2)
    refs: z.array(z.object({
      id:  z.string(),
      required:  z.boolean().default(false),
      requireApproval: z.boolean().default(false),  // v2 — HITL gate per tool
      allowedHeaders:  z.array(z.string()).default([]),  // v2 — header propagation allowlist
      inheritWorkspace: z.boolean().default(false),  // v2 — sub-agent tool refs only:
      // child receives the parent's workspace snapshot (immutable,
      // digest-verified) materialized into its own /workspace
    })).default([]),
  }).default({ refs: [] }),

  // ---- Prompt + composition (kagent-inspired) ----
  prompt: z.object({  // v1
    registryId: z.string(),
      // opaque id in the prompt registry (TBD: Langfuse or DIY)
    label:  z.enum(["dev", "qa", "production"]).default("production"),
    fragments:  z.array(z.object({  // v2 — composable fragments
      registryId: z.string(),
      alias:  z.string().optional(),
      // {{include "alias/key"}} sugar; falls back to registryId/key
    })).default([]),
    variables:  z.record(z.string()).optional(),
      // resolved at promote time: {{.AgentName}}, {{.ToolNames}}, etc.
  }),

  // ---- Triggers (when this agent fires) ----
  triggers: z.array(TriggerSpec).min(1), // v1

  // ---- Output (validated; drives surface routing) ----
  outputSchema: z.unknown(),  // v1
     // Zod schema serialised to JSON Schema at register time
  surface: z.enum(["inject", "replace", "surface", "annotate"]),  // v1
```

APPENDIX

AgentSpec schema, full (2 of 2)


```
// ---- Scope (Practice / Workload / cross-scope handshake) ----
scope: z.object({  // v1
  practice: z.string(),  // e.g. "agentic-ai"
  workload: z.string(),  // e.g. "grace-chat"
  allowedReferrers: z.object({  // v2 — kagent's bidirectional handshake
    from:  z.enum(["Same", "All", "Selector"]).default("Same"),
    selector: z.record(z.string()).optional(),
  }).optional(),
}),

// ---- Budget (timeouts, turns) ----
budget: z.object({ // v1
  maxTurns:  z.number().int().positive().default(10),
  perTurnTimeoutMs: z.number().int().positive().default(15000),
}).default({}),

// ---- Context compaction (kagent ContextCompressionConfig) ----
context: z.object({  // v2
  compaction: z.object({
    compactionInterval:  z.number().int().positive().default(5),
    overlapSize:  z.number().int().nonnegative().default(2),
    tokenThreshold:  z.number().int().positive().optional(),
    eventRetentionSize:  z.number().int().positive().default(20),
    summarizer: z.object({
      gatewayKey: z.string().optional(),
    // separate key — cheap summariser model, its own budget
      prompt:  z.string().optional(),
    }).optional(),
  }).optional(),
}).optional(),

// ---- Workspace (v2 — artifact-shaped context; Harness/AgentCore runtime) ----
workspace: z.object({  // v2
    source: z.enum(["none", "s3-archive"]).default("none"),
      // s3-archive: an immutable, content-addressed snapshot
      // {bucket, key, versionId, sha256} that the platform materializes
      // into /workspace before the first model turn. Never a live clone,
      // never a credential — callers stage content; the runtime verifies
      // the digest and extracts. Complements token-shaped context: streams
      // carry signals, workspace snapshots carry state.
    maxBytes: z.number().int().positive().default(262144000),  // 250 MiB
  }).optional(),

  // ---- Capabilities (for inter-agent discovery — kagent's a2aConfig.skills) ----
  capabilities: z.array(z.object({  // v2
    name:  z.string(),
    description: z.string(),
    examples:  z.array(z.string()).optional(),
  })).default([]),

  // ---- Memory (deferred — AgentCore Memory or pgvector) ----
  memory: z.object({  // deferred — Phase 4
    gatewayKey:  z.string(),
    ttlDays:  z.number().int().positive().default(15),
    autoSaveEveryNTurns:  z.number().int().positive().default(5),
    popularityExtensionThreshold: z.number().int().positive().default(10),
    minScore:  z.number().min(0).max(1).default(0.3),
  }).optional(),
  metadata: z.record(z.unknown()).optional(),  // v1
});
export type AgentSpec = z.infer<typeof AgentSpec>;
```

Trusted delivery gate, in code


```
coverage.get("complete") is True
and set(expected) <= set(succeeded)
and _attempts_resolved(coverage)
and coverage.get(
    "narrowed_to_single_agent"
) is False
```

Proves the configured panel ran and the result reached GitHub

Not yet proved that findings are right: no precision or recall measured; auto-merge stays off

Trusted delivery gate · the control plane builds the coverage receipt from its own child-run evidence, never from the model

The model can say “Clean”. Only a complete receipt lets the review post to GitHub.

Reasoning and write authority have different custody: the runtime returns text; trusted code posts it.