DHARMA AI LABS.

GA / CUSTOMER ONBOARDING AND OPERATIONS

Complete customer guide.

Connect your agents. Evaluate real work.
Release reviewed improvements.

---

01 / CONNECT

Account / API / CLI

02 / EVALUATE

Evidence / Rubrics / Traces

03 / IMPROVE

GitHub / Skills / Rollback

GENERAL CUSTOMER EDITION / 2026.09.05

BEFORE YOU BEGIN

Dharma AI: Complete Customer Guide

GA customer documentation | 5 September 2026 | Edition 2026.09.05

An onboarding and operating runbook for your organization: connect an existing application, run managed agents, evaluate real work, investigate failures, and release reviewed improvements. This is the general customer edition. It contains no customer-specific configuration, private benchmark results, or commercial arrangements.

Start at the Dharma portal. Your selected organization's membership, credits, active revisions and endpoint capabilities determine what you can execute. Describing an optional integration here does not mean it is already provisioned in your account. This guide is not a universal latency guarantee or authorization to change production traffic.

The HTML, downloadable PDF and Markdown contain the same runbook. Code blocks on this page have copy controls. Use Markdown for a coding-agent handoff; use the authenticated portal's fresh instruction for enrollment. Public documentation never contains an enrollment grant or durable API key.

FIND YOUR NEXT STEP

Contents

01

CHAPTER

Choose your first outcome

Evaluate an application that already works: keep its current provider, prompts, credentials and fallback unchanged. Capture completed calls through an after-response observation worker (section 6), or design an explicitly launched evaluation campaign (section 7). Neither requires local CLI installation. Observation and a campaign are different operations: one records existing work; the other executes new test work.

Run a hosted production agent: select or create a managed agent, deploy a reviewed revision, submit a bounded request, and verify the terminal result and trace (section 5). Introduce traffic through your own reversible application feature flag.

Run in your own cloud: verify a GCP Vertex BYOK binding, then connect it to the intended logical agent (section 12). Direct Google API observation does not require this binding. BYOK changes who hosts and pays for execution, not the need for evidence, evaluation and release controls.

Connect a coding agent: use the organization-generated one-shot instruction to connect an approved repository, native skill and outbound relay (section 13). This enables local evidence, bounded tasks and signed skill delivery. Do not install the CLI merely to call a REST API from your application.

First useful session

1. Verify the organization and your role, then perform one authenticated read.

2. Choose one integration path and one representative, permission-cleared case.

3. For evaluations, validate and preflight the task package before any paid launch.

4. Approve the exact execution and budget; follow the durable ID to a terminal result.

5. Inspect evidence, failure flags and settled usage. Decide whether to refine a test, propose a remediation, or expand a small canary.

Access and a valid preflight are useful early milestones. They are not proof that a model run, provider installation or full remediation release has completed. Start with one to five cases; a large benchmark is not an onboarding prerequisite.

02 Account, team and credentials

CHAPTER

Owner setup

Register through Create an account, verify the requested email, and select or create your organization. Complete the activation steps shown by the portal. A trial or purchased balance must appear in the selected organization's credit view before relying on it. Provisioning is asynchronous: organization creation alone does not establish runtime readiness.

Provide the GitHub username that should receive access to the private control repository, or complete the GitHub account flow shown in onboarding. Verify the invitation and actual repository access. This repository holds governed agent configuration and remediation artifacts; it is not an automatic public mirror of your application's source.

Use the portal's organization selector, not a sample organization ID from documentation. Keep the organization ID, managed agent ID, logical repository-agent ID, workspace ID and endpoint ID distinct. They identify different resources and are not interchangeable.

Invite a teammate

An authorized organization owner/admin opens People, sends an invitation to the teammate's actual email and selects the minimum appropriate role. The teammate accepts the invitation with the same identity, signs in and selects the existing organization. They should not create a replacement organization to work around an invitation failure.

A member's ability to inspect resources, run work, manage credentials or approve a release depends on role and scopes. Ordinary membership is not organization-admin authority. If Developer API reports a developer-membership requirement, have the organization admin correct that membership; do not share another person's key.

Each teammate connects their own device through their own account instruction. Local provider credentials remain on that device. Revoking a member, a developer key and a device are separate operations.

Create separate application credentials

Open Developer API > Keys. Create least-privilege credentials for serving, evaluation and operations separately. A token is shown once: put it directly in your server's secret manager. Never paste it into a coding-agent prompt, client JavaScript, screenshot, URL or committed environment file.

Typical scopes include `agents:read`, `agents:run`, `evals:read`, `evals:run`, `traces:read`, `skills:read`, `skills:write` and `usage:read`. The exact operation contract and membership checks remain authoritative; adding a scope does not bypass an admin-only portal action.

Examples below assume your process already receives these values from configuration:

COMMAND

```
# Non-secret configuration; replace with your selected organization's values.
export DHARMA_API_ORIGIN="https://www.dharma-ai.io"
export DHARMA_ORG_ID="<organization-id>"
export DHARMA_AGENT_ID="<managed-agent-id>"
# DHARMA_API_TOKEN is injected by your server's secret manager.

curl --fail-with-body \
  "$DHARMA_API_ORIGIN/api/orgs/$DHARMA_ORG_ID/managed-agents" \
  -H "Authorization: Bearer $DHARMA_API_TOKEN"
```

Confirm that the read returns only the expected organization and agents. A 401 or 403 is an identity/scope problem to resolve, not a reason to use a broader person's credential. The existing browser-confirmed enrollment/bootstrap flow is used for the CLI; a developer key is not a substitute enrollment grant.

03 Portal workspace map

CHAPTER

All customer workflows use `/portal`. The book icon in the portal header opens this guide without changing your organization or current workspace. Existing modules remain available subject to the selected role and enabled capabilities.

- **Agent Fabric** > **Instructions:** current organization-specific setup prompt, CLI pin, repository setup and provider capability receipts.

- **Onboarding:** private control repository and GitHub access. **Devices:** machine enrollment, last-seen state and revocation. **Agents:** logical repository agents and their local, managed and BYOK endpoints.

- **Trajectories:** captured evidence and immutable revisions. **Analysis:** exact window membership and deterministic/semantic completion. **Failure Atlas:** recurring failures, affected agents and supporting evidence.

- **Remediations:** candidates, private GitHub PRs, held-out gates and release state. **Skills:** signed bundles, installation, activation and rollback. **Tasks & A2A:** dispatch, leases, acceptance, artifacts and handoffs. **Usage:** attributed consumption and settlement.

- **Evaluation Runs** > **Sessions:** choose a campaign and compare arms. **Rubrics:** inspect its current contract, gates and scores. **New Evaluation:** import, validate, preflight and launch a task package.

- **Evaluation Runs** > **Failure Atlas, Traces, Backtesting and Reports:** inspect failures, runtime evidence, candidate retests and report downloads. A campaign report is different from a disclosure-controlled evidence package.

- **Configuration** > **Environments:** tenant runtime readiness. **Configuration** > **Agents:** managed-agent drafts and immutable revisions. **Cloud BYOK:** customer-owned runtime verification. **Packs:** available task/evidence packages, not an arbitrary executable-plugin installer.

- **Control Room** / **Runtime Health:** operational summaries and health. **People:** team access. **Developer API:** scoped keys. **Settings** / **Tokens** / **Billing:** account, credit and available payment controls.

- **Control-agent drawer:** organization-scoped conversation, tools, approvals and resource links (section 14).

An empty historical list is not a failed run or a passed test. Check selected organization, filters, capability/readiness state and source timestamps. Partial/error summaries are not complete evidence. Follow resource IDs through the workflow rather than interpreting a dashboard total alone.

04 API fundamentals

CHAPTER

Customer requests go to `https://www.dharma-ai.io`. Ordinary routes use `Authorization: Bearer`; the Gemini-compatible ingress uses a Dharma key in `x-goog-api-key`. That header name does not mean you should send a Google credential to Dharma.

The two live contracts are Evaluation Runs OpenAPI and Agent Fabric OpenAPI. Use their complete request schemas, scopes and response types. A route family in prose is not permission to invent an endpoint or SDK method.

IDs, retries and completion

Create one durable attempt record in your application before submitting paid work. Use its stable idempotency key on routes that support or require one. Replaying the same intended operation should return the existing receipt. Changing the model, prompt or evidence under the same key is a conflict, not a model comparison.

Persist the response's run/campaign/event ID before waiting. A `202`, enqueue receipt, released decision or accepted task is not a completed execution. Read the original resource and ordered events until the documented terminal state. Use a bounded deadline, jitter and backoff; honor retry-delay headers when provided. Do not poll every few milliseconds or repeatedly launch work to discover whether the original finished.

After a timeout, reconcile the original durable ID first. A terminal failure remains that attempt's terminal result. A deliberate new attempt needs a new linked attempt record and budget authorization. Do not assume all create/deploy endpoints are idempotent just because the serving route is.

Clients and connectors

- CLI: @dharma-ai-labs/agent-fabric.

- TypeScript SDK: @dharma-ai-labs/agent-fabric-sdk.

- Python SDK: dharma-agent-fabric-sdk.

CLI and SDK versions are independent. Use the current production CLI pin supplied by the portal, and verify the SDK operation against its installed version. Raw HTTP works without installing the CLI on a serving host. MCP and the OpenAI plugin use the account-generated connection instructions and

the same organization authorization boundary. They do not grant extra scopes or bypass approvals. Start machine discovery at llms.txt.

05 CHAPTER Managed agents and production serving

Create and deploy your own agent

In Configuration > Agents, select a ready managed agent or create a separate draft for your workflow. Preserve a working production revision while experimenting. A managed agent is the execution definition; a logical Fabric agent groups repository identity and endpoints.

`POST /api/orgs/{orgId}/managed-agents` creates a draft and candidate revision. This illustrative body uses an explicitly selected model; confirm that model is available to your organization/provider before deployment:

JSON

```
{
  "name": "Workflow evidence assistant",
  "slug": "workflow-evidence-assistant",
  "runtimeProvider": "gcp_agent_platform",
  "modelChain": ["gemini-3.7-flash"],
  "runtimeConfig": {
    "systemInstruction": "Use only authorized evidence. Distinguish facts, estimates and unknowns. Act
  within the declared authority."
  },
  "policyBundle": {}
}
```

Add the reviewed output/evidence policy for your actual use case. The installed `systemInstruction` is limited to 20,000 characters; it is not the per-request `prompt`. Do not put hidden evaluator labels in either. Naming an application tool in a prompt does not register an HTTP callback. Use a registered adapter, or let your application retrieve bounded evidence through its existing authorized tools.

Read the returned `agent.id` and `revision.id`. Deploy the exact reviewed candidate with `POST /api/orgs/{orgId}/managed-agents/{agentId}/deploy` and body `{"revisionId":"<candidate-revision-id>"}`. Inspect agent detail, revisions and `active_revision_id` before serving. A creation response is not a deployment receipt. If a response is lost, read the list/detail before repeating a create operation.

`POST /api/orgs/{orgId}/managed-agents/{agentId}/revisions` creates a successor. Review, deploy and smoke-test that immutable revision before changing traffic. For customer-owned execution, select `gcp_vertex_byok` only after section 12's binding is verified.

Ordinary asynchronous run

Save a request body using your actual managed agent ID. The following is a synthetic text-only smoke, not a recorded result:

JSON

```
{
  "agentId": "<managed-agent-id>",
  "model": "gemini-3.7-flash",
  "prompt": "A supplied note says delivery is expected Friday but not confirmed. State what is known, what
   is unknown and the next authorized read-only action.",
  "attachments": [],
  "estimatedCredits": 1000,
  "maxRuntimeSeconds": 120
}
```

Use the schema's attachment format for real multimodal work. The model must belong to the agent's active allowed chain. Per-run model selection does not rewrite that revision. `estimatedCredits` is part of the request authorization/estimation contract, not a declaration of the final charge.

COMMAND

```
# After approval of this request and its budget:
curl --fail-with-body \
  "$DHARMA_API_ORIGIN/api/orgs/$DHARMA_ORG_ID/agent-runs" \
  -H "Authorization: Bearer $DHARMA_API_TOKEN" \
  -H 'Content-Type: application/json' \
  -H "Idempotency-Key: $ATTEMPT_ID" \
  --data-binary @request.json --output run-receipt.json

# RUN_ID comes from the receipt. These reads do not launch work.
curl --fail-with-body \
  "$DHARMA_API_ORIGIN/api/orgs/$DHARMA_ORG_ID/agent-runs/$RUN_ID" \
  -H "Authorization: Bearer $DHARMA_API_TOKEN" --output run.json
curl --fail-with-body \
  "$DHARMA_API_ORIGIN/api/orgs/$DHARMA_ORG_ID/agent-runs/$RUN_ID/events" \
  -H "Authorization: Bearer $DHARMA_API_TOKEN" --output events.json
```

Require terminal status, intended model/revision, schema-valid output, trace attribution and one settlement receipt. A successful HTTP status alone is not a quality evaluation.

Gemini-compatible ingress

For an explicitly approved provider-path migration, retain supported `contents`, `systemInstruction`, temperature and thinking-level fields. Save only the JSON body as `gemini-request.json`, never the old Google authorization header. Resolve template variables before submission.

COMMAND

```
curl --fail-with-body \
  "$DHARMA_API_ORIGIN/api/v1beta/models/$DHARMA_MODEL:generateContent" \
  -H "x-goog-api-key: $DHARMA_API_TOKEN" \
  -H "x-dharma-agent-id: $DHARMA_AGENT_ID" \
  -H "x-dharma-idempotency-key: $ATTEMPT_ID" \
  -H 'Content-Type: application/json' \
  --data-binary @gemini-request.json \
  --dump-header response-headers.txt --output response.json
```

Set `DHARMA_MODEL` from the active revision's allowed models; do not assume the newest provider model is enabled everywhere. Parse `candidates[].content.parts[].text` and `usageMetadata`, then validate your own output contract. Preserve `x-dharma-run-id`, `x-dharma-trace-id` and `x-dharma-settled-credits` when returned. Response headers/files can contain confidential identifiers; do not publish them blindly.

The compatible managed ingress accepts up to eight JPEG, PNG or WebP images within a 3 MiB aggregate decoded inline-image budget. Preserve evidence and image order; reject or explicitly resize with an agreed policy, never silently drop images. Assistant chat and evaluation task packages have different limits. Keep provider fallback and traffic controls in your application until a matching quality and capacity canary passes.

06 Observe your existing provider

CHAPTER

This is the lowest-change path for an application already calling Gemini. Keep its serving request, provider keys, output parser and fallback unchanged. Store a completed-call outbox record with stable case/attempt/provider request IDs. Return the provider response to the user; a separate durable worker sends the observation to Dharma. An unawaited serverless promise is not a durable queue.

Use the current Gemini observation helper. The helper's starter revision is v3; its wire schema remains `dharma.evaluation-sidecar.gemini/v1`. It submits one event, not the eventual judgment and not a full durable outbox implementation.

TYPESCRIPT

```
// In your trusted outbox worker, using a real completed call:
const receipt = await enqueueCompletedGeminiObservation({
  apiOrigin: "https://www.dharma-ai.io",
  organizationId: configuredOrganizationId,
  organizationToken: evaluationSecret,
  model: completedCall.model,
  providerRequestId: completedCall.providerRequestId,
  traceId: completedCall.traceId,
  agentRevisionId: completedCall.promptRevision,
  geminiRequest: normalizedRequest,
  geminiResponse: normalizedResponse,
  contentPolicy: "metadata_only"
});
// Persist receipt/eventId; retry the same outbox record, not a new ID.
```

This excerpt assumes your existing storage, secret management, normalization and imported helper. It must not be pasted into a browser or treated as invented completed-provider output.

Evidence and image boundaries

Normalize only the supported subset: request roles and text/`inlineData`/`fileData` parts, optional system-instruction text, temperature and thinking level; response candidate text/finish reason and actual returned token counts. Do not fabricate missing usage, send provider-only dumps or include authorization headers.

Sidecar limits are eight images, 3 MiB aggregate decoded inline-image bytes, 256 KiB text/metadata and 4,456,448 bytes for the complete serialized request. Inline images must be JPEG, PNG or WebP. HTTPS references include a SHA-256 and size, with a 20 MiB per-reference limit; they are provenance references, not automatically fetched semantic evidence. Eight references do not authorize a 160 MiB inline payload.

`metadata_only` is the default. It discards image bytes and does not authorize semantic judging of raw content. For visual semantic judging, obtain disclosure/retention approval, use `customer_authorized_content`, retain the actual validated inline images and set `semanticMaxCredits` explicitly from 1 to 50,000. Reference-only images cannot be semantically judged by this route and fail explicitly when semantic judging is requested. Do not make private images public merely to obtain a URL.

Omitting `semanticMaxCredits` keeps the helper deterministic-only. Setting it opts into separate paid judging; the value is a maximum authorization, not an expected fee. An example value such as 500 is not a price quote.

Submission and verification

The helper calls `POST /api/orgs/{orgId}/evaluation-sidecar/events` with Bearer authentication and `Idempotency-Key` equal to the stable provider request ID. Read `GET /api/orgs/{orgId}/evaluation-sidecar/events/{eventId}` for deterministic and semantic state. Enqueue success does not mean the semantic result exists.

Test one real completed observation and replay its ID. Require the same/duplicate receipt and no second settled charge. For image-dependent work, verify approved image hashes and terminal semantic state. Test a representative full-image request and an over-limit rejection. Simulate Dharma unavailable: the original provider response must still succeed, with observation safely queued or quarantined for retry.

Quarantine invalid/oversized observations instead of truncating them. Bound retries and worker concurrency; a retry must not generate a new event identity. Retain both failed and successful evidence under your agreed policy.

07
CHAPTER

Design and run evaluations

What you can configure today

A supported task rubric contains visible scenario evidence, scorer-only hidden truth, required state fields, allowed/blocked actions, actual registered-tool expectations, standard integrity hard gates and a pass threshold. Configure these for your use case; no private evaluator source is required to begin.

A new arbitrary domain formula or semantic axis is different. Managed campaign preflight currently rejects non-empty `customerDomainRubric.dimensions`. JSON describing a custom formula does not install an executable evaluator. You can keep a domain scorer in your own stack and join its results to exported outputs by stable task ID. Native execution of new custom dimensions requires a separately supported implementation, not a prompt workaround.

The **Rubrics** tab displays the selected campaign's contract, gates and scores. Configure tasks through **New Evaluation** and its task-package import. A control-agent draft is text, not an executable rubric or a saved general-purpose scorer.

Create a task package

Download the supported task starter and task-package schema. Replace the synthetic scenario, hidden truth, package identifier and task IDs with authorized cases. Retain a stable version/hash for every comparison.

Use `scenario` only for evidence the evaluated agent may see. Put expert labels and negative-control truth in `hidden_ground_truth`. A reference string does not fetch a document/image automatically: verify the selected runtime actually receives the intended evidence. Do not put answer keys in the prompt or installed agent policy.

Example task-criteria excerpt, not a complete launch payload:

JSON

```
{
  "expected_state_gate": {
    "required_state_fields": [
      "intent", "evidence_used", "known_state",
      "unknown_or_missing_state", "allowed_next_actions",
      "blocked_actions", "decision_authority", "tool_results"
    ],
    "allowed_next_actions": ["respond", "request_clarification", "escalate"],
    "blocked_actions": ["claim_unverified_evidence_as_fact"]
  },
  "tool_expectations": {"should_call_or_reference": []},
  "scoring": {"pass_threshold": 85},
  "success_criteria": ["Distinguish a provisional estimate from a verified fact."]
}
```

An empty tool list requires no invocation. If a registered adapter is necessary, use its exact ID; merely mentioning the tool does not satisfy the evidence-bearing invocation check. A required state field is not itself a domain-accuracy test. Use hard-gate IDs from the starter, not invented names.

`pass_threshold` is a percentage: 85 means 0.85. The authoritative score uses the released standard Cognitive Integrity dimensions and gates. Free-text success criteria are non-gating observations unless tied to an implemented deterministic check. Editing incidental weights or success text does not redefine the authoritative formula or install a semantic judge.

Tasks have a 100,000-character serialized bound, and `scenario.user_message` is limited to 20,000 characters. Do not paste a multi-megabyte serving request into a task package. Do not silently truncate a workload and call it an exact reproduction.

Portal: preflight to terminal results

1. Select your organization, then Evaluation Runs > New Evaluation and the intended managed agent.

2. Choose the integration mode and comparison arms deliberately. Preserve existing production serving unless migration is explicitly approved.

3. Import the edited task package. Validate its structure, evidence separation, thresholds and standard gates.

4. Review preflight, requested agent/revision, arms, total trajectories and credit authorization. A paired one-task campaign produces two trajectories.

5. Approve that exact launch. Keep the returned campaign ID and open Sessions, then its results, Rubrics, Traces and report.

6. Require terminal execution and authoritative scoring. Missing results, failed judges or empty lists are not passing scores.

Campaigns execute new work, including when `integrationMode` is `observe_existing_runtime`; they are not passive log ingestion. A `direct_baseline` arm is not proof that your own production provider deployment was called. Compare actual endpoint/revision provenance.

API: build the launch body from the package

The import package and API campaign wrapper differ. This local transformation preserves its tasks and evaluation contract; it does not run a model:

COMMAND

```
jq --arg agent "$DHARMA_AGENT_ID" '{
  name: "Workflow smoke v1",
  agentId: $agent,
  integrationMode: "observe_existing_runtime",
  arms: ["direct_baseline", "stateful_dharma_runtime"],
  tasks: .tasks,
  evaluationContract: .evaluation_contract
}' task-package.json > campaign.json

curl --fail-with-body \
  "$DHARMA_API_ORIGIN/api/orgs/$DHARMA_ORG_ID/managed-evals/preflight" \
  -H "Authorization: Bearer $DHARMA_API_TOKEN" \
  -H 'Content-Type: application/json' \
  --data-binary @campaign.json --output preflight.json
```

Preflight validates without launching execution or debiting execution credits. Resolve reported errors before continuing. After explicit approval of the exact package and budget:

COMMAND

```
curl --fail-with-body \
  "$DHARMA_API_ORIGIN/api/orgs/$DHARMA_ORG_ID/managed-evals/campaigns" \
  -H "Authorization: Bearer $DHARMA_API_TOKEN" \
  -H 'Content-Type: application/json' \
  -H "Idempotency-Key: $CAMPAIGN_ATTEMPT_ID" \
  --data-binary @campaign.json --output campaign-receipt.json

curl --fail-with-body \
  "$DHARMA_API_ORIGIN/api/orgs/$DHARMA_ORG_ID/managed-evals/campaigns?campaignId=$CAMPAIGN_ID" \
  -H "Authorization: Bearer $DHARMA_API_TOKEN" --output campaign-results.json
```

There are at most 100 tasks per campaign; larger datasets need stable, explicit shards. Begin with one to five cases. Do not assume a campaign model field overrides the installed agent: check the current schema and executed model/revision receipts. The ordinary serving API's model-selection support does not imply every endpoint uses the same field.

Optional domain benchmark

Define each domain axis, evidence needed, normalization, applicability, missing-value handling, aggregation, threshold and scorer version. Calibrate against small permission-cleared fixtures. Run an existing external scorer on exported outputs if desired; transfer of private source is not required. Record unrequested/inapplicable/missing scores explicitly, never invent zeros or passes. Preserve the baseline contract when changing criteria for a later experiment.

08 Traces, scores and reports

CHAPTER

Keep the actual linkage chain: application case/attempt -> provider request or managed run -> trace/evaluation result -> Fabric trajectory revision -> analysis window -> failure family -> remediation target -> signed bundle and rollout. Not every observation automatically creates every downstream record.

A serving run can complete without an evaluation. A sidecar event has its own asynchronous scoring state. A managed campaign executes its own task arms. A Fabric window consumes eligible captured trajectory revisions. Distinguish these populations when measuring usage and quality.

Read these records together:

- Campaign package/version/hash, exact case membership, selected agent, arms and task count.

- Per-result task ID, arm, run/trace ID, executed endpoint/model/revision and terminal status.

- `metadata.verdict`: authoritative state, scorer/contract version, authoritative score, hard gates, failed gate IDs and customer criteria.

- Queue time, runtime/provider time and end-to-end time separately; token usage, settled credits and retries.

- Optional external scorer version and actual domain scores, joined by stable case ID.

Diagnostic bars or an incidental weighted score do not replace `metadata.verdict.authoritativeScore` and the authoritative hard-gate decision. Infrastructure errors, invalid output, missing evidence, deterministic gate failures and domain-quality errors need separate counts. A failed/absent semantic judge is incomplete, not a successful zero-cost score.

Use the selected campaign's **Download evaluation report** action. Where available, the authenticated Traces view exposes a separate evidence-package export. Review exported fields and disclosure policy before sharing. Preserve hashes and stable IDs. Filter an exact cohort rather than assuming lifetime organization activity represents only your app's current production usage.

For a fair before/after comparison, hold task membership, visible evidence, baseline scorer and intended model/prompt controls constant. Report sample count, failures, confidence limits where applicable and actual behavior change. Do not turn one successful example into a general quality or latency guarantee.

09 Analysis and Failure Atlas

CHAPTER

Continuous Agent Fabric analysis operates on exact 100-trajectory windows for eligible, enabled organizations. These are retained captured Fabric trajectories in the selected scope, not simply any 100 HTTP requests or uploaded log rows. Evidence policy, credit authorization and capability gates still apply; reaching the count does not guarantee immediate semantic completion.

Read Analysis before requesting more work. Completed windows can generate versioned rubric proposals, failure families and remediation candidates. Failure Atlas groups evidence-backed recurring problems. Proposals are not silently activated and do not rewrite the rubric that measured your baseline.

COMMAND

```
curl --fail-with-body \
  "$DHARMA_API_ORIGIN/api/v1/orgs/$DHARMA_ORG_ID/agent-fabric/evals" \
  -H "Authorization: Bearer $DHARMA_API_TOKEN" --output windows.json
```

For an explicitly approved paid analysis, `POST` to the same route with an idempotency key and a scoped request:

JSON

```
{
  "trajectoryTarget": 100,
  "scope": {
    "mode": "agents",
    "organizationAgentIds": ["<logical-organization-agent-UUID>"]
  }
}
```

Use the logical agent ID, not a managed serving ID. A `409 analysis_window_not_ready` means the scope lacks enough eligible trajectories. Do not fill it with unrelated cases. Retry an existing failed window using the supported `retryWindowId` request. Reprocessing retained semantic artifacts with `reprocessWindowId` is a separate operation; do not rerun completed model work unnecessarily. Inspect scheduled recovery status rather than assuming it succeeded.

Fabric evaluation contracts

Read `GET /api/v1/orgs/{orgId}/agent-fabric/evaluation-contracts` with `evals:read`. Inspect the proposed rubric, deterministic verifiers, semantic judge configuration, confidence threshold, logical agent and source window. This contract governs signed action decisions; it is not an arbitrary managed-campaign domain scorer.

An organization admin can transition an eligible proposal through the authenticated contract flow. Its body is:

JSON

```
{
  "contractId": "<existing-proposal-UUID>",
  "action": "activate",
  "confirmation": "ACTIVATE EVALUATION CONTRACT <same-proposal-UUID>"
}
```

The route also supports retire/reject with corresponding confirmation. It is a transition API, not a general create-rubric endpoint. Ordinary developer tokens do not substitute for the required admin session. Confirm the returned active contract and registered verifier IDs before using it for decisions.

10 Remediation, GitHub and skills

CHAPTER

Your organization's private control repository and permanent agent branches connect evaluated behavior to reviewed changes. Verify your own GitHub access and actual branch from onboarding. A branch can hold governed artifacts without mirroring all application source.

From a failure to a deployed change

1. Choose an evidence-backed Failure Atlas family and the exact remediation target. Identify the smallest plausible causal change.

2. Preserve source trajectories, baseline contract and pinned versions. Inspect the generated proposal and private PR; do not treat generated text as verified causality.

3. Review scope: instructions, policy, skill or application code. Reject unrelated changes, secrets and unauthorized data disclosures.

4. Run the candidate's required backtest and security/regression checks with distinct non-source held-out cases. The required candidate gate includes at least 20 distinct non-source tasks; retries of one case do not increase that count.

5. Obtain the required organization approval. The first release and high-risk remediation require the designated approval gates.

6. Release the signed bundle only to matching logical-agent endpoints, initially as an approved canary. Verify installation and activation separately.

7. Run a matched post-activation check. Require actual bundle/revision attribution, intended behavior, schema validity and usage receipts. Expand or roll back based on that evidence.

For a held-out campaign, use `held_out_backtest`, the actual `policyCandidateId` and a `single_held_out` comparison contract as specified by the API. Preserve original case membership and missing/failed results. A small smoke is appropriate before release; it does not replace a required held-out gate.

Read `/api/v1/orgs/{orgId}/agent-fabric/remediations` and `/{targetId}` for target state. Use the supported approval/rollout controls in Remediations and Skills. A PR merge, file upload or branch edit is not a runtime activation receipt.

What changes at runtime?

For a managed or verified BYOK endpoint, the reviewed bundle/revision must be acknowledged by the runtime, and a subsequent run must show that version. This is how a skill produced from evaluation becomes operating behavior; it is not enough for the file to exist in GitHub.

For an observation-only integration, Dharma does not rewrite your production Google call or load patches into it automatically. Apply an approved code/prompt change through your application's existing review and deployment process, then observe and compare the changed behavior.

Before rollout, retain the signed ancestor and a canary stop rule. If the candidate regresses, stop expansion, roll back the affected target through its signed path, verify the ancestor is active and run a small matched regression check. Keep failed activation and rollback receipts; never delete evidence to make a gate pass.

11 CHAPTER Signed decisions, tasks and A2A

Fabric decisions authorize a bounded proposed effect. They are not a replacement for your application's answer JSON and are not proof that an effect occurred.

- **release:** dispatch only the unchanged authorized task before expiry; require receiver and terminal receipts.

- **block:** do not retry the prohibited effect unchanged. Redesign it or pursue a separately governed policy change.

- **escalate:** obtain the required human/organization authority, then request a fresh bounded decision.

- **withhold:** repair missing/stale evidence or evaluator availability. Keep the current working revision active until a new valid decision exists.

Use `POST /api/v1/orgs/{orgId}/agent-fabric/decisions`, then the task route only if released. The target must be an authorized same-organization endpoint with the required capability, matching workspace, registered commands, retained pinned evidence and active evaluation contract.

This structural example describes a bounded LOCAL task. Every placeholder, path and command must match real registered authority; it is not ready to dispatch unchanged:

JSON

```
{
  "actionId": "<fresh-action-UUID>",
  "evaluationContractId": "<active-contract-UUID>",
  "task": {
    "taskId": "<fresh-task-UUID>",
    "targetEndpointId": "<active-local-endpoint-UUID>",
    "workspaceId": "<matching-workspace-UUID>",
    "taskType": "remediation_smoke",
    "instructions": "Apply the reviewed skill change and run its registered test.",
    "authority": {
      "readPaths": ["skills/workflow/**", "tests/**"],
      "writePaths": ["skills/workflow/**", "artifacts/**"],
      "commandIds": ["test.workflow"],
      "network": "deny",
      "allowlistedDomains": []
    },
    "acceptanceCommandIds": ["test.workflow"],
    "requiredArtifacts": ["artifacts/workflow-test.json"],
    "timeoutSeconds": 300,
    "leaseSeconds": 120,
    "expiresAt": "<short-future-UTC-expiry>"
  },
  "stateEnvelope": {
    "intent": "Correct the specified evidence-boundary failure.",
    "evidence_used": ["trajectory:<UUID>:revision:<revision>"],
    "known_state": {"findingId": "<actual-failure-family-id>"},
    "unknown_or_missing_state": [],
    "allowed_next_actions": ["apply_reviewed_skill_patch"],
    "blocked_actions": ["deploy_production", "modify_unrelated_source"],
    "decision_authority": "Only the approved skill and registered test.",
    "tool_results": [],
    "proposed_action": "apply_reviewed_skill_patch"
  },
  "evidenceReferences": [{
    "trajectoryId": "<actual-trajectory-UUID>",
    "revision": 1,
    "capsuleHash": "sha256:<64-lowercase-hex-characters>"
  }]
}
```

Populate unknowns and tool results honestly. Empty arrays in a shape example are not evidence that a real task has no unknowns. Unsupported commands, absent evidence or expired authority can legitimately cause rejection or withholding. Keep network denied unless a bounded exception is explicitly granted.

Submit the decision with its own idempotency key. Read the actual `decision.outcome`, reasons, expiry and signature. Only a released response may be used to build the task submission from the exact task judged:

COMMAND

```
jq -e '.decision.outcome == "release"' decision-response.json >/dev/null || exit 1
jq --arg id "$(jq -r '.decision.id' decision-response.json)" \
  '.task + {actionDecisionId: $id}' decision-request.json > task-dispatch.json

curl --fail-with-body \
  "$DHARMA_API_ORIGIN/api/v1/orgs/$DHARMA_ORG_ID/agent-fabric/tasks" \
  -H "Authorization: Bearer $DHARMA_API_TOKEN" \
  -H 'Content-Type: application/json' \
  -H "Idempotency-Key: $TASK_ATTEMPT_ID" \
  --data-binary @task-dispatch.json
```

Do not synthesize a signature, change the judged task, escalate paths or reuse expired authority. Keep the receiver relay healthy and inspect acceptance, lease, required artifacts, terminal status and effect-enforcement receipts. A released decision authorizes one exact attempt, not arbitrary future work.

For A2A use the supported task contract with `taskType=a2a_handoff`, actual source/target task and endpoint linkage, a state envelope and pinned evidence. Preserve intent, known/unknown state, allowed/blocked actions, decision authority and tool results. Conversation history is available from `/agent-fabric/conversations`. A2A is not unrestricted shell access or an arbitrary cross-organization network callback.

12 GCP Vertex and local BYOK

CHAPTER

GCP Vertex BYOK executes in your own GCP project through the verified runtime/federation path. It is distinct from logging a direct Gemini API call. Local BYOK retains provider credentials on an enrolled device. Neither is automatically configured by reading this guide.

GCP setup

1. An organization admin opens Cloud BYOK > GCP Vertex and enters the requested project, region, runtime and identity configuration in the authenticated form.

2. Apply the exact organization-specific workload-identity/IAM commands generated for that binding. Do not copy another organization's principal or introduce service-account JSON keys.

3. Run Verify connection. Require authenticated identity/invocation, project/tenant isolation, trace and usage attribution, and the required nonce-bound rollback check.

4. Keep the endpoint disabled if verification fails. Preserve the specific failure and repair only its cause.

5. Bind the verified endpoint to the intended logical repository agent. Inspect its endpoint list and execute a few approved cases before choosing it for a campaign or production traffic.

The status/configuration family is `GET/POST /api/v1/orgs/{orgId}/agent-fabric/byok/gcp`; configuration/verification requires the appropriate admin context. Browser-safe status is not permission to expose cloud identities or internal URLs in logs, prompts or public reports. Use short-lived federation, never send provider secrets to browser-to-runtime calls.

A syntactically accepted model name is not proof that your provider/project can serve it. Verify model availability on the selected runtime. Customer GCP pays model/runtime costs; Dharma evaluation, analysis and orchestration can still use Dharma credits.

Local endpoint

Enroll the device and approved repository (section 13). Its provider remains locally authenticated. The outbound relay carries signed tasks and approved evidence, not provider secrets. Managed, GCP BYOK and local endpoints can belong to the same logical agent while retaining separate capabilities and execution provenance.

For removal, stop new dispatch, revoke the relevant endpoint/device or WIF binding and verify new work is denied. Retain required audit records. Removing a local device does not rotate every upstream provider credential.

13 CHAPTER CLI and repository onboarding

The preferred entry is the fresh **Agent Fabric** > **Instructions** message inside your selected organization. It supplies the current production CLI pin, required Node version, isolated profile, enrollment/bootstrap steps and permitted scope. A static guide cannot safely embed an expiring, user-specific enrollment grant.

Zero-to-connected procedure

1. Open the coding harness in the intended approved repository. Copy the portal-generated instruction into that agent.

2. Let it perform the installation and browser-confirmed/device or short-lived bootstrap flow specified by the instruction. Do not separately run an unrelated installer or use another organization's prompt.

3. Approve legitimate native tool/install/directory questions when the harness requires them. Read an organization-change request carefully; it is not a normal step for a clean first connection.

4. Require actual organization, repository, workspace and endpoint IDs; native-skill verification; policy state; relay readiness; and a received trace/evidence receipt where supported.

5. Resume interrupted work from stored enrollment and durable status. Reissue an expired unused grant in the portal; do not resubmit a consumed grant or overwrite another organization's profile silently.

Use the same provider-neutral instruction for Claude Code, Codex, Hermes or another detected harness. Native authentication and permissions remain the harness's responsibility. Do not disable security controls globally or ask an agent to rewrite its own permissions to evade a denial. A grant cannot supply missing provider login or capabilities the harness does not implement.

Unknown harnesses may receive generic instructions and a signed manifest. Capture, task execution, installation, activation and rollback must be reported separately; unavailable lifecycle capabilities are not a successful installation merely because a file exists.

Commands after enrollment

The examples use `dharma` as shorthand for the exact pinned executable from your portal instruction. If onboarding uses `npm exec` with a pinned package, retain that invocation and the same organization-specific profile. Do not substitute a stale globally installed binary.

COMMAND

```
dharma --help
dharma repositories status --repo .
dharma providers list
dharma skills status --workspace .
dharma evidence preview --workspace .
```

For another repository within the user's approved scope:

COMMAND

```
dharma repositories discover --root <approved-root>
dharma repositories connect --repo <selected-repository> \
  --organization-id <organization-id> --policy-revision <current-policy-revision>
```

Do not scan a whole drive by default. The same normalized source repository reuses a logical agent; another machine/provider is an endpoint. Follow the installed command's help and generated policy path for `skills sync`, `skills verify`, `evidence capture`, `evidence sync` and `relay start`. Preview evidence before synchronization. Metadata-only authorization is not permission to upload raw sessions.

Connect a ready managed/BYOK runtime through Agent Fabric > Agents, or the confirmed `dharma agents bind-runtime` flow after reading its help. Keep managed IDs and logical endpoint IDs separate. Verify the resulting binding and which endpoint actually executes a task.

For unattended operation, verify one bounded read-only task and one authorized evidence sync. Keep leases, expiry, workspace containment, registered commands and no-network defaults intact. Monitor relay health and disk space. Clean only completed task worktrees after required evidence is retained; never delete an active workspace.

14 Organization control agent

CHAPTER

The account assistant is accessible from the portal's control-agent drawer. It uses the same authenticated organization context and server-side broker and is recorded in Fabric. Inspect its actual readiness and logical-agent registration. A chat response is not proof that an operation or first-time registration completed.

Read tools can inspect organization status, agents/endpoints, trajectories, failures, remediation/rollout state, usage, runtime runs and campaign results. Supported proposals include analysis, a runtime run, an evaluation and bounded tasks/A2A or release operations, subject to their actual scopes and gates.

Paid execution, writes, dispatch, approvals, rollout/rollback, membership, profile or payment operations need the applicable explicit confirmation and authorization. High-risk remediation and destructive actions retain organization-admin gates. The assistant cannot create an arbitrary new evaluator or callback tool merely because asked. Use actual portal controls when an operation is not in its tool catalog.

CLI access

COMMAND

```
dharma assistant chat --help
dharma assistant history --help
dharma assistant status --help
dharma assistant approve --help
dharma assistant reject --help
```

Use your enrolled profile and the installed version's arguments. Keep credentials out of prompts/history. The CLI's existence is not proof that all approvals are available to an ordinary developer key; sensitive actions can still require the Clerk portal session.

API operations

- `POST /api/v1/orgs/{orgId}/control-agent/sessions`: create a session.

- `GET /api/v1/orgs/{orgId}/control-agent/sessions`: list permitted history.

- `POST /api/v1/orgs/{orgId}/control-agent/sessions/{sessionId}/messages`: submit a message.

- `GET /api/v1/orgs/{orgId}/control-agent/sessions/{sessionId}/events`: retrieve durable progress/events.

- `POST /api/v1/orgs/{orgId}/control-agent/tool-calls/{toolCallId}/approve` or `/reject`: act under the required approval context.

Use OpenAPI for message/approval bodies, scope and authentication. Persist session and event IDs to recover an interrupted stream; do not resubmit the same paid instruction just because a browser disconnected. Inspect tool state, approval state and resulting resource links before reporting

completion. Assistant attachments currently have a separate limit of two images; do not apply the serving/sidecar eight-image limit to chat.

15 Reusable integration prompts

CHAPTER

These prompts are useful **after enrollment** or in the organization assistant. They are not credentials and cannot create authority. Supply approved IDs, task files and secret-manager variable names separately, never secret values. Use the portal's one-shot instruction for the installation itself.

Integrate an existing codebase without replacing serving

AGENT PROMPT

```
Use https://www.dharma-ai.io/docs/customer-onboarding and its linked OpenAPI contracts.
Work only in this repository and the organization already enrolled by the portal instruction.
Read the current provider integration, output schema, test harness and data policy first.
Keep production serving, its provider credentials, parser and fallback unchanged.
Use server-side secret references; never print credentials, raw sessions or customer images.
Propose a durable after-response observation outbox using the current Gemini helper.
Preserve actual request IDs, model/prompt revision, evidence and returned usage.
Add tests for duplicate delivery, image limits, outages and interrupted worker recovery.
Prepare one supported task package with separate visible evidence and hidden ground truth.
Use implemented standard gates and a clear threshold; do not invent custom scoring APIs.
Perform approved reads and preflight, then show the exact paid/write proposal and budget.
After required approval, continue through the terminal result and verify its resource receipts.
Report completed steps, actual IDs, failures and unknowns; do not stop at package installation.
Do not launch a large benchmark or alter production traffic without separate authorization.
```

Prepare and execute a bounded evaluation

AGENT PROMPT

```
Use the selected organization's intended agent and the supplied authorized cases.
Read https://www.dharma-ai.io/docs/customer-onboarding#7-design-and-run-evaluations.
Create a versioned task package from the current public template.
Keep expert labels out of agent-visible input. Preserve the existing production provider.
Explain standard gates, pass thresholds, comparison arms and expected trajectory count.
Validate and preflight without executing. Present the exact launch and credit authorization.
Wait for required approval, then launch only that package and follow its durable campaign ID.
Inspect terminal results, authoritative verdicts, trace/revision provenance and settled usage.
Separate missing judgments, infrastructure failures and task-quality failures.
Export the report and propose the smallest evidence-backed next action.
Do not claim a conversational rubric draft installed a new executable domain evaluator.
```

Investigate and remediate an observed failure

AGENT PROMPT

```
Use only the specified Failure Atlas family and approved repository scope.
Read https://www.dharma-ai.io/docs/customer-onboarding#10-remediation-github-and-skills.
Inspect actual source trajectories, candidate state and the private control-repository PR.
Preserve the original scoring contract and matched baseline; propose a minimal causal change.
Show the required non-source held-out cases, security/regression checks and approval gates.
Obtain explicit authority before paid runs, GitHub writes, dispatch or release operations.
Continue after approval to actual test and task completion, recording failures honestly.
Do not declare deployment from a merged PR: require signed installation and activation receipts.
Verify a subsequent run used the new bundle/revision and compare the intended behavior.
Keep a signed rollback ancestor and stop expansion on regression.
Report unknowns rather than inventing a quality uplift or overriding a withheld decision.
```

Read-only account review

AGENT PROMPT

```
For this organization, summarize recent campaigns, failed runtime runs and pending remediations.
Link actual resources, timestamps, model/revisions and settled credit receipts.
Separate quality failures, infrastructure failures and incomplete semantic judgments.
Explain missing prerequisites and the next smallest action. Do not execute or change anything.
```

Teammate handoff

Send this text alongside the actual organization invitation, not instead of it:

AGENT PROMPT

```
Accept the organization's invitation using the invited email, then sign in at
https://www.dharma-ai.io/portal and select the existing organization.
Do not create a replacement organization or reuse another member's credentials.
Read https://www.dharma-ai.io/docs/customer-onboarding.
For a local coding agent, open Agent Fabric > Instructions and copy your fresh instruction
into the agent running in the approved repository. Approve legitimate native prompts.
Verify the connected workspace, endpoint, skill, policy and received evidence before calling it ready.
If a permission or invitation fails, send the organization admin its exact reference, not a key.
```

16 Credits and spend controls

CHAPTER

Dharma credits are the customer billing unit, not model tokens. Use your organization's purchased entitlement, current price and settled receipts. This general guide deliberately does not publish provider costs, internal margins or a customer's negotiated conversion.

- Managed serving consumes Dharma credits according to actual settled execution.

- Direct-provider serving remains on your provider bill. Deterministic sidecar capture currently debits zero Dharma credits; optional semantic judging is separately authorized and metered. Check current entitlement/usage before relying on this for future planning.

- GCP BYOK model/runtime charges remain in your project; Dharma analysis, evaluation and orchestration can still consume credits.

- Local provider spend stays with your local provider. Managed analysis and control-plane services have separate Dharma usage records.

- Evaluation campaigns and the control agent are additional work, not free because a serving request already ran.

Budgeting and reconciliation

Preflight the exact campaign; review task count and arms before approving execution. For observations, set a semantic sampling rate and per-event credit ceiling deliberately. If a ceiling is 500 credits for each of 100 events, the maximum authorized total is 50,000 credits; that is not an expected bill and says nothing about a dollar price.

Read Usage and `GET /api/v1/orgs/{orgId}/agent-fabric/usage` with `usage:read`. Join each serving run, sidecar event, analysis or campaign resource to its actual settlement and any adjustment. Separate estimates/reservations, settled debits, refunds/adjustments and remaining balance. Do not infer spend from input tokens alone or mix different cohorts into one cost-per-result claim.

Retried settled operations should reuse their receipt. If an apparent duplicate or conflicting amount exists, stop repeat mutations and request reconciliation with the bounded resource IDs. Never delete test rows or create new retry IDs merely to make accounting look clean.

Work can stop/reject when credits, authorization or runtime entitlement are exhausted. Trial expiration, current allowance and payment controls are shown in the account; do not assume a public example renews a trial. Adding a card or enabling auto-recharge is a separate authenticated commercial action with explicit consent. Maintain application-level spend alerts and a stop rule even when an account still has credits.

17 Launch, recovery and offboarding

CHAPTER

First integration acceptance

- Correct membership and a least-privilege key can read the intended organization and agent.
- The selected runtime/endpoint is ready; model and revision match the intended workflow.
- Original serving remains independently controllable when observation is the chosen path.
- One real observation/run reaches terminal state with the expected trace and usage attribution.
- Replay reuses the receipt; observation outage does not break the original provider response.
- Relevant image/payload limits and over-limit errors are tested without silent evidence loss.
- One task package validates/preflights; its approved campaign reaches authoritative results.
- Optional CLI, BYOK and connectors are checked only when selected, not marked ready from this guide.

Before a larger launch

Agree the representative prompt/image mix, model/thinking setting, output contract, steady/burst rate, maximum concurrency, timeout, fallback and budget. Measure queue, provider/runtime and end-to-end latency separately. Report sample size, success rate, schema validity, missing results and p95 under that exact load.

A quota setting or low-load success does not establish sustained high-concurrency latency. Do not assume sub-30-second p95 for low-hundreds concurrent requests from a generic guide. Establish a workload-specific capacity allocation and acceptance gate before increasing traffic. Keep serving, observation, semantic judging and release controls independently reversible.

For governed improvements, require the actual failure/window linkage, preserved baseline, private reviewed PR, required held-out evidence, organization approval, signed matching-endpoint activation, post-activation behavior check and verified rollback ancestor. A published document is not a substitute for those receipts.

Failure and recovery

- **401/403:** repair the actual login, selected organization, accepted invitation, developer membership or scope. Do not bypass with another person's key.
- **400/413:** correct the explicit request or size problem. Preserve evidence; do not silently drop images or truncate a task.
- **409:** inspect the conflict/not-ready reason. Reconcile idempotency bodies or eligible trajectory count rather than forcing a new ID.
- **429:** honor retry delay, limit concurrency and use bounded backoff. Avoid a retry storm.

- **Timeout/502:** query the original durable run and events before paying for another attempt. A valid input followed by invalid model output is not automatically a client 4xx.

- **Judge outage:** preserve deterministic results and mark semantic judgment incomplete. Never substitute a fabricated score.

- **Installation/activation failure:** leave the current working revision active or the new endpoint disabled, repair the specific gate, and verify again.

- **Exhausted credits/expired authority:** stop new paid work and obtain the appropriate renewal/approval; never silently auto-charge or broaden authority.

For support, send organization ID, resource IDs, UTC time, endpoint/operation, status/error code, model/revision and a sanitized request-shape summary. Do not send secrets, private URLs, raw images, local absolute paths or other tenants' data by default. Use an agreed private channel if content disclosure is necessary.

Retention and departure

Review the evidence policy before raw-content synchronization. Export required reports, hashes, usage and release receipts before their retention expires. Raw content and audit metadata can have different retention requirements. Withdraw disclosure through its supported flow when appropriate.

For offboarding, stop new tasks, revoke membership, application keys, device/relay authority, GitHub access and WIF grants as applicable. Verify no new work is accepted under revoked authority. Keep required evidence under policy; do not unlink active billing projects or delete unrelated organization data merely to remove a user.

18 Reference and compatibility

CHAPTER

This edition is based on the released customer API contracts and supported public evaluation starter as of 5 September 2026. The docs-only release does not imply a new model benchmark, custom-evaluator implementation or a fresh test of every harness. Organization-specific instructions, capability state and current API contracts take precedence over a saved PDF when versions change.

- Portal and account
- Documentation index
- API and SDK
- Managed agents
- Evaluations
- BYOK
- Remediation
- CLI
- Control agent
- Security and billing
- Agent Fabric OpenAPI
- Evaluation Runs OpenAPI
- Supported task package
- Task-package schema
- Observation helper
- Machine-readable discovery

Compatibility checks when updating an integration

Record the installed CLI/SDK version, API schema, task package hash, scorer/contract version and active agent revision. Revalidate examples after changing any of these. Do not hardcode a documentation example's model into every customer agent. Do not reuse a downloaded grant, private identity or customer-specific default.

Keep a single current onboarding runbook for the team. Archive previous editions internally with their dates; send new colleagues this guide and their own invitation, not an old customer's instructions or an operator evidence archive.