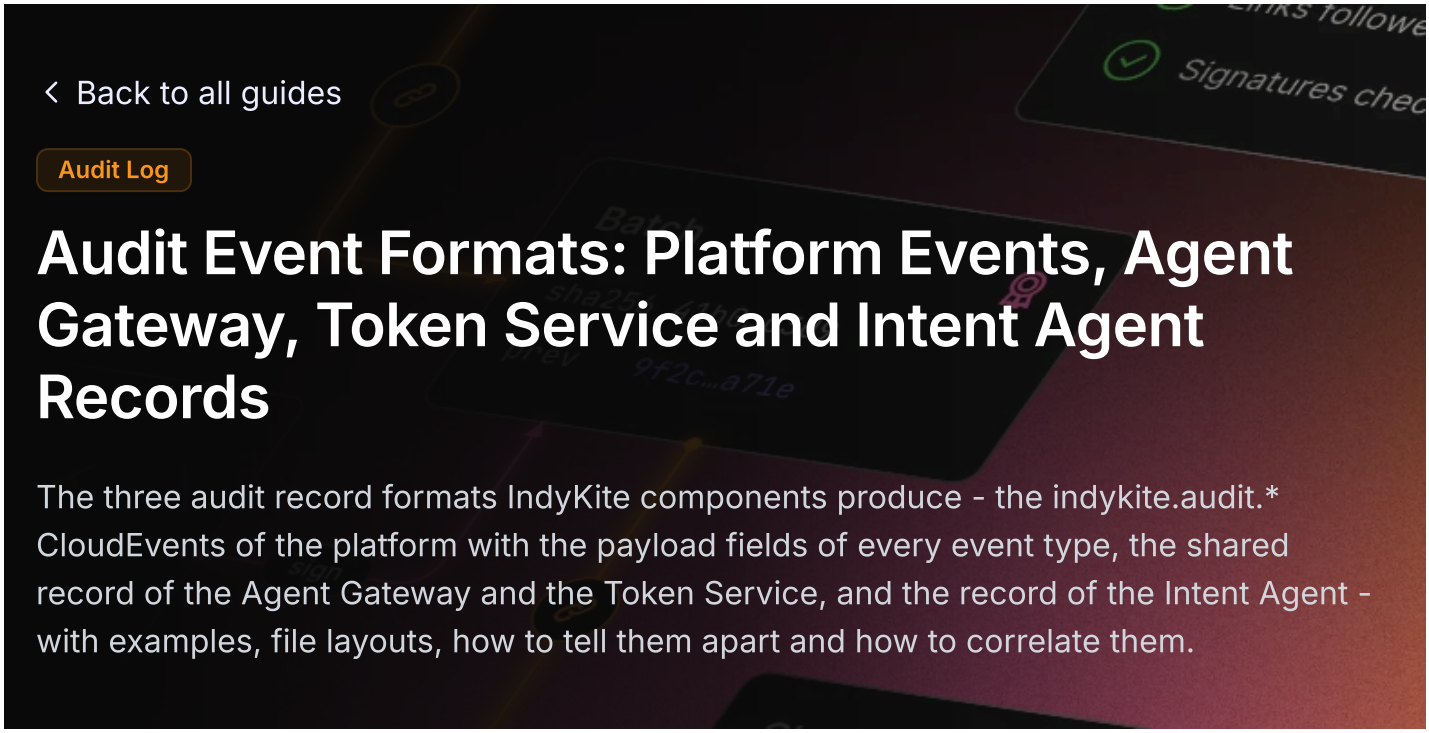
Source: https://developer.indykite.com/guides/guide-audit-formats · PDF generated 2026-11-25; the page is the current version.



Back to all guides

Audit Log

Audit Event Formats: Platform Events, Agent Gateway, Token Service and Intent Agent Records

The three audit record formats IndyKite components produce - the indykite.audit.* CloudEvents of the platform with the payload fields of every event type, the shared record of the Agent Gateway and the Token Service, and the record of the Intent Agent - with examples, file layouts, how to tell them apart and how to correlate them.

Which audit formats exist?

IndyKite components write three audit record formats. They share no schema: each is read with its own field list.

Format	Written by	Shape	Where you read it
Platform events, `indykite.audit.*`	The Config API, Capture, Token Introspect, AuthZEN, ContX IQ and the graph's change data capture	A CloudEvents 1.0 message whose JSON body carries a typed `data` payload	Pushed to your broker by **Outbound Events**; read back, signed and chained, from the **Audit Log API**
Agent record	The **Agent Gateway** and the **Token Service**, one shared format	A flat JSON object with camelCase keys	Your webhook, or json, csv or txt files the service rotates
Intent Agent record	The **Intent Agent**	A flat JSON object with snake_case keys	Your webhook, or json, csv or txt files the service rotates

Only the platform events can be tamper-proof: in a project with an **Audit Signing configuration** every delivered event is also hashed, signed and chained, and the Audit Log API serves that chain. Agent, Token Service and Intent Agent records are written by services you run, straight to the sink you configure, and are not chained by the platform; keep them in a store with the integrity guarantees you need. The MCP server has no audit output of its own: changes to its configuration appear as `indykite.audit.config.*` events, and calls that pass through an Agent Gateway in `mcp` mode appear as agent records.

Platform events

The envelope

Each event is a CloudEvents 1.0 message. The attributes are the same whichever provider delivers it; what changes is where the provider puts them: Kafka record headers prefixed `ce_` (`ce_specversion`, `ce_type`, `ce_id`, and so on, with the body as the record value), Azure Service Bus application properties prefixed `cloudEvents:`, Azure Event Grid in its CloudEvents schema, and Google Cloud Pub/Sub attributes as defined by the CloudEvents Pub/Sub protocol binding. The body repeats every attribute you need to interpret the event, so a consumer can ignore the transport attributes and read the JSON alone; the Audit Log API serves the body only.

Attribute	Value
`specversion`	`1.0`
`type`	The event type, the same value as `eventType` in the body, for example `indykite.audit.capture.upsert.node`.
`source`	`https://audit.indykite.com/v1`. The suffix is the version of the body format; every event is `v1` today.
`id`	Unique per event. Delivery is at least once: deduplicate on it.
`time`	When the event happened, RFC 3339 UTC. The same instant as `time` in the body.
`datacontenttype`	`application/json`
`customerid`, `appspaceid`, `requestid`	Extensions repeating the organization, the project and the request identifier of the body, in lower case as the CloudEvents specification requires of extension names. `appspaceid` names the project whose sinks receive the event.
`originalsource`	Extension naming the part of the platform that recorded the event. Informational, like `eventSource` in the body.
`configtype`	Extension on `indykite.audit.config.*` events only: the configuration type the event is about, the same value as `data.type`.

The body

The body is one JSON object with the same top-level keys for every event type. The **Audit Log guide** shows a complete Capture upsert event inside a signed batch; the fields are:

Field	Meaning
`eventType`	One of the `indykite.audit.*` names in the catalogue below.
`time`	RFC 3339 with nanoseconds, UTC.
`customerId`, `appSpaceId`	The organization and the project the event happened in; `appSpaceId` is the `project_id` of the request.

`initiator`	Who made the request, as a flat object of string identifiers whose keys tell the kind of caller apart. An Application Agent carries `applicationId`, `appAgentId` and `appAgentCredentialsId`; a Service Account carries `serviceAccountId` and `serviceAccountCredentialsId`; an end user resolved by Token Introspect carries `digitalTwinId`, `viaApplicationId` and `sessionId`. Every form also carries `customerId` and `appSpaceId`. Empty identifiers are left out.
`requestId`	The identifier of the request that caused the event, the same value the API returned to the caller. Always present, empty when there was none.
`data`	The payload, typed by its `@type` member: `type.googleapis.com/indykite.auditsink.v1beta1.<Message>`. The catalogue below gives the message and its fields for each event type.
`context`	String key-value pairs some events attach, for example `soft_delete` on a configuration delete. Present only when there is at least one pair.
`eventSource`	Which part of the platform recorded the event. Informational.

Inside `data` the field names are the lowerCamelCase form of the message fields, and every field is present even when empty: a string as `""`, a number as `0`, a boolean as `false`, a list as `[]`, a nested message that was not set as `null`. Enumerations are written by name (`OPERATION_TYPE_UPDATE`), durations as a decimal number of seconds with the `s` suffix (`"0.012s"`) and timestamps in RFC 3339. Typed values, such as node properties, use the `{"stringValue": ...}`, `{"integerValue": ...}` family of the Capture API.

Event type catalogue

The complete list of event types a route can filter, with the `@type` payload each carries and the payload's fields. Filter keys, such as `captureLabel` and `label0`, are described in the **Outbound Events guide**.

Capture

Event type	Payload
`indykite.audit.capture.upsert.node`	`UpsertData` with `node`: `id` (the platform identifier), `externalId`, `type`, `isIdentity`, `properties`, each `{type, value, externalValue}`. One event per node of the request.
`indykite.audit.capture.upsert.relationship`	`UpsertData` with `relationship`: `type`, `source` and `target` as `{externalId, type}`, `properties`. One event per relationship.
`indykite.audit.capture.delete.node`	`DeleteData` with `node`: `{externalId, type}`.
`indykite.audit.capture.delete.relationship`	`DeleteData` with `relationship`: `type`, `source`, `target`, `properties`.
`indykite.audit.capture.delete.node.property`	`DeleteData` with `nodeProperty`: `match` as `{externalId, type}` and `propertyType`.

`indykite.audit.capture.delete.relationship.property`	`DeleteData` with `relationshipProperty`: `type`, `source`, `target`, `propertyType`.
`indykite.audit.capture.delete.node.tag`	`DeleteData` with `nodeTags`: `match` and `tags`, a list of strings.
`indykite.audit.capture.delete.node.property.metadata`	`DeleteData` with `nodePropertyMetadata`: `match`, `propertyType`, `metadataType`.

Configuration

All seven carry `type`, the configuration type as an enumeration name such as `CONFIG_TYPE_TOKEN_INTROSPECT`, and `containersPath` as `{customerId, applicationSpaceId}`. A configuration object itself appears as a `ConfigDetail`: `name`, `displayName`, `description`, `status`, `tags`, and one member named after the configuration kind holding the stored object, with secrets reduced to their `kid`, `expireTime`, `keyFormat` and `originalKid`. Events about organization-level objects, such as Service Accounts, have no project and are not delivered to project sinks.

Event type	Payload
`indykite.audit.config.create`	`CreatedConfig`: `id`, `type`, `location` as `{id, type}`, `containersPath`, `detail`.
`indykite.audit.config.read`	`ReadConfig`: either `id` or `name` as `{locationId, locationType, name}`, plus `type` and `containersPath`. The object itself is not repeated.
`indykite.audit.config.update`	`UpdatedConfig`: `id`, `type`, `containersPath`, `before` and `after`.
`indykite.audit.config.delete`	`DeletedConfig`: `id`, `type`, `containersPath`, `before`, the state right before deletion. `context.soft_delete` says whether the object was soft-deleted.
`indykite.audit.config.list`	`ListConfig`: `locationId`, `locationType`, `containersPath`, `listed` as a list of `{type, ids}`, the identifiers the caller saw.
`indykite.audit.config.permission.assign`	`AssignConfigPermissions`: `targetIdentifier` and `targetType` (who receives the role), `role`, `customerId`, `objectId` and `objectType` (what the role is on).
`indykite.audit.config.permission.revoke`	`RevokeConfigPermissions`, the same fields.

Credentials

Event type	Payload

`indykite.audit.credentials.token.introspected`	`TokenIntrospected`: `resolvedDigitalTwinId` (the graph node the token was mapped to, empty when none), `configId` (the Token Introspect configuration that matched), `issuer`, `subject` (the `sub` claim or the configured `sub_claim`), `jti`, `tokenHash` (SHA3-256 of the token, base64url without padding). The token itself is never recorded.

Authorization

The five AuthZEN payloads share a skeleton: `request` as `{body, token}`, where `body` is the request body as the caller sent it, a JSON string, and `token` is `{subjectId}`, the subject of the user token sent with the request, empty when there was none; `response`, the response body as a JSON string; `results`, one entry per policy that took part, each naming the `policy` as `{id, name}`; `errorMessage`, empty on success; and `evaluationTime`.

Event type	Payload
`indykite.audit.authorization.evaluation`	`Evaluation`: each result carries `isAllowed` and `advices`, a list of `{values}` maps.
`indykite.audit.authorization.evaluations`	`Evaluations`: instead of a flat `results` list, an `evaluations` list with one entry per evaluated triple, each holding `request` as `{subjectType, subjectExternalId, resourceType, resourceExternalId, action}` and its `results`, each `{policy, isAllowed, reason, advices}`.
`indykite.audit.authorization.searchsubject`	`SearchSubject`: each result carries `externalIds`, the subjects the policy returned.
`indykite.audit.authorization.searchresource`	`SearchResource`: each result carries `externalIds`, the resources the policy returned.
`indykite.audit.authorization.searchaction`	`SearchAction`: each result carries `actions`.

ContX IQ

Event type	Payload
`indykite.audit.ciq.execute`	`ExecuteKnowledgeQuery`: `request` as `{query, policy, inputParams}`, where `query` is `{id, status, query}` and `policy` is `{id, status, policy}`, each with the stored definition as a JSON string, and `inputParams` the typed input values of the call; `authorized`, whether the policy let the query run. The rows the query returned are not recorded.

Change data capture

Delivered only when the event sink has `include_cdc_events` set. One event per changed node or relationship.

Event type	Payload

`indykite.audit.cdc.node.create`, `indykite.audit.cdc.node.update`, `indykite.audit.cdc.node.delete`, `indykite.audit.cdc.relationship.create`, `indykite.audit.cdc.relationship.update`, `indykite.audit.cdc.relationship.delete`	`CDCEvent`: `txId` and `seq` (the transaction and the position of the change in it), `operation` (`OPERATION_TYPE_CREATE`, `_UPDATE`, `_DELETE`), `eventType` (`EVENT_TYPE_NODE` or `EVENT_TYPE_RELATIONSHIP`), `elementId`; for a node, `node` as `{labels, labelKeys}` where `labelKeys` maps each label to the keys that identify the node; for a relationship, `relationship` as `{type, start, startElementId, end, endElementId, keys}`; `before` and `after` as `{properties, labels, richProperties}`, where `properties` maps each attribute to its JSON-encoded value and `richProperties` lists IndyKite properties as `{type, value, source, metadata}`; `txCommitTime` and `txStartTime`.

Agent Gateway and Token Service records

The two services write one and the same record, so a webhook or a file directory can receive both and the `service` field tells them apart. One record is written per decision: the gateway writes it before forwarding, so a forward that fails afterwards still shows `AUTHORIZED`; the Token Service writes one per token it looked at and one for the exchange itself.


```
{
  "decision": "AUTHORIZED",
  "reason": "subject User/millicent may reach Agent/indykiteagent-2 through the
delegation chain",
  "subject": "millicent",
  "actor": "indykiteagent-2",
  "action": "ADMISSION",
  "service": "retriever-iag",
  "timestamp": "2026-09-30T09:12:00.123456789Z",
  "traceID": "4bf92f3577b34da6a3ce929d0e0e4736",
  "cached": {
    "admission": false
  },
  "actorsChain": [
    "indykiteagent",
    "indykiteagent-2"
  ]
}
```


Field	Meaning

`decision`	What became of the request, read together with `action`. Gateway: `AUTHORIZED` or `NOT_AUTHORIZED` on its `ADMISSION`, `DISCOVERY` and `INVOCATION` records, `NOT_AUTHORIZED` on a `noop` record, and on its Intent Agent calls `INTENT_FOUND` or `INTENT_FAILED`. Token Service: `TOKEN_EXCHANGED`, `EXCHANGE_REFUSED`, `INTROSPECTED`, `INTROSPECTED_INACTIVE`, `INTROSPECTION_REFUSED`. Both: `ERROR`, when the service or something it depends on failed, so that an outage is never counted as a denial. Issuing or describing a token is never a grant of access, which is why those decisions are not spelled `AUTHORIZED`.
`reason`	Free text, the same message the caller received where there is one. The gateway's reason strings are listed in the **Agent Gateway guide**, the Token Service's in the **Token Service guide**. The workflow a decision was about appears only inside this text.
`subject`	The end user the request acts for.
`actor`, `actorsChain`	The immediate caller, and the full delegation chain from the token's `act` claim, oldest first; `actor` is its last entry.
`action`	Gateway: the question the record answers. `ADMISSION`, may the chain reach the protected agent; `DISCOVERY`, which capabilities of a kind the caller may see, written for a filtered MCP list or extended card; `INVOCATION`, may the caller use every capability the request names; `RESOLVED_EVALUATIONS`, does the platform allow the data an MCP call touches, with the evaluations sent and the answer for each; `INTENT`, the record of each Intent Agent call, one more per message after the request's own; `noop`, the request was refused before any question was asked. Token Service: `TOKEN_EXCHANGE` or `TOKEN_INTROSPECT`.
`durationMs`, `prompt`	Gateway `INTENT` records only: the duration of the Intent Agent call in milliseconds, and the prompt when the gateway's `audit.include_prompt` is on. Absent from the JSON of every other record.
`cached`, `capabilities`, `permittedSet`, `discovery`	Gateway decision records only, and only in JSON (webhook and `json` files; the csv and txt layouts do not carry them): whether the answer came from the gateway's cache, by kind; the identifiers an invocation named; the set a discovery was filtered to; and `{serverCalled, listed, kept}` for a discovery. Field by field in the **Agent Gateway guide**.
`service`	The `service.name` of the instance that wrote the record; how you tell the hops of a chain apart.
`timestamp`	RFC 3339 UTC with up to nanosecond precision, trailing zeros dropped.
`traceID`	The trace identifier also found in the service's JSON logs; the records of one exchange share it.
`tokenID`	Token Service records only: the `jti` of the token issued or described. Absent from gateway JSON; an empty column in csv and txt.

Over the webhook each record is one JSON request. In files, `json` writes one object per line; `csv` writes a header row per file and the columns `decision, reason, subject, actor, actorsChain, action, service, timestamp, traceID, tokenID, prompt, capabilityName, skillID, reasonCode, skillIDs, evaluations, durationMs` in that order, the chain comma-joined, lists as JSON arrays, each later column appended at the end so that files written

before it existed still read; of the seven after `tokenID` the gateway fills `durationMs` and, with `include_prompt`, `prompt` on its `INTENT` records, and the rest, `capabilityName`, `skillID`, `reasonCode`, `skillIDs` and `evaluations`, belong to the `IBAC_DISCOVERY` and `IBAC_USAGE` records described in the **IBAC engine guide** and stay empty on every other record. `txt` writes one line per record, values unquoted except `prompt`, `capabilityName` and `skillID`:

```
[2026-09-30T09:12:00.123456789Z] decision=AUTHORIZED reason=subject
User/millicent may reach Agent/indykiteagent-2 through the delegation chain
subject=millicent actor=indykiteagent-2 actorsChain=indykiteagent,indykiteagent-
2 action=ADMISSION service=retriever-iag
traceID=4bf92f3577b34da6a3ce929d0e0e4736 tokenID= prompt="" capabilityName=""
skillID="" reasonCode= skillIDs= evaluations= durationMs=0
```

Files are named `audit—<service.name>-<nanoseconds>.<format>` and rotated by time, size or both. Both services take the same `audit` configuration section, `delivery: file` or `delivery: webhook`; the webhook authentication and queue keys are `api_key` (`apikey`), `api_key_header` (`apikeyheader`), `mtls_certificate_file_path` (`mtlscertificatepath`), `mtls_private_key_path` (`mtlsprivatekeypath`) and `buffer_size` (`buffersize`), the name in parentheses being the one the current releases read, as the **Agent Gateway guide** details. Without the section nothing is recorded.

Intent Agent records

The Intent Agent writes one record per request, in its own format with snake_case keys. The field order below is also the column order of csv and txt files and the key order of the JSON.

```
{
  "timestamp": "2026-09-07T10:15:04.123456Z",
  "request_id": "req-1",
  "subject_id": "user-1",
  "subject_type": "Customer",
  "requesting_agent_id": "",
  "context_id": "ctx-1",
  "prompt": "rent kitt",
  "evaluations": [
    {"action": {"name": "RENT"}, "resource": {"type": "Car", "id": "kitt"}}
  ],
  "schema_version": "abc123",
  "policy_version": "",
  "llm_provider": "gemini",
  "llm_model": "gemini-3.5-flash-lite",
  "latency_ms": 42,
  "validation_retries": 0,
```

```
  "policy_fetch_failed": false,
  "outcome": "success",
  "message": ""
}
```

Field	Meaning
`timestamp`	RFC 3339 UTC with microseconds.
`request_id`	The caller's `X-Request-ID` when it is non-blank and at most 40 characters, as the Agent Gateway accepts it; otherwise a generated UUID. A longer value is ignored, not truncated.
`subject_id`, `subject_type`	The end user the prompt is evaluated for, from the message metadata of the call.
`requesting_agent_id`	Reserved; always empty today.
`context_id`	The A2A context the message belongs to.
`prompt`	The natural-language prompt as received.
`evaluations`	The AuthZEN evaluations derived from the prompt, each `{action, resource}` and, where the prompt named one, `subject`; empty when none were derived.
`schema_version`	Not a version number: the SHA-256 of the project's data schema as read from `GET /data-schema/v1`, computed over the document with sorted keys, so it changes whenever the schema changes and is stable otherwise. Empty when no schema was available.
`policy_version`	Likewise the SHA-256 of the subject type's KBAC policies as read from `GET /access/v1/policies`, each policy serialised with sorted keys and the list sorted, so neither key order nor policy order affects it. Empty when there were no policies or no fetch.
`llm_provider`, `llm_model`	The provider, derived from the model name, and the model that produced the evaluations.
`latency_ms`	Wall-clock time of the request in milliseconds, authentication included.
`validation_retries`	How many times the model's answer was rejected and asked for again before this outcome.
`policy_fetch_failed`	Three states: `null` when no policy fetch was attempted, `false` when it succeeded, `true` when the live fetch failed and the last known-good policies were used. In csv and txt `null` is an empty cell.
`outcome`	`success` when evaluations were returned; `unmatched` when the prompt yielded no intent or a free-text reply; `validation_failed` when the model's answer never passed validation; `error` for an authentication rejection, a missing or empty schema, a subject type mismatch, a timeout or any other failure.
`message`	The text the caller received, exactly; empty when the caller received no text.

In files, `json` writes one object per line; `csv` writes the 17 field names as a header row per file, then one row per record with lists and objects as compact JSON, `null` as an empty cell and numbers and booleans as text; `txt` writes `[<timestamp>] request_id=<v> subject_id=`

`<v> ... message=<v>` with the same cell rendering and no quoting, so a space inside `prompt` or `message` is written as is. Files are named `audit—<INTENT_AGENT_NAME>-<nanoseconds>.<format>`, created with mode `0600` in a directory with mode `0750`, and never appended to a file the service did not create. Over the webhook each record is one JSON request with `Content-Type: application/json`; mTLS requires TLS 1.3.

Delivery is configured with the `AUDIT_*` variables listed in the **Intent Agent guide**. Unlike the gateway and the Token Service, auditing cannot be switched off: `AUDIT_DELIVERY` has no default and the service refuses to start without it. A failed delivery, to an unreachable webhook or a full disk, is logged and never fails the request; with `AUDIT_ASYNC=true` a full queue drops the record with the warning `audit: dropping payload, async queue is full`.

Telling the formats apart and correlating them

A receiver that takes all three on one endpoint can classify a record by its keys: a platform event has `eventType` and `data`; an agent record has `decision` and `actor`; an Intent Agent record has `outcome` and `prompt`. No key is shared by two formats.

To follow	Match
One request across the Agent Gateway and the Token Service	`traceID`: the gateway's record and the Token Service's introspection and exchange records of the same hop share it.
A delegation token from issue to use	`tokenID` of the Token Service's `TOKEN_EXCHANGED` record is the token's `jti`; when the platform later validates that token, its `indykite.audit.credentials.token.introspected` event carries the same value in `data.jti`.
An Intent Agent request to the client that sent it	`request_id`, when your client sets `X-Request-ID` on the call to the Intent Agent; the Agent Gateway does not set that header on its own calls, so pass it from your side if you need the link.
A platform API call to its events	`requestId` in the body and the `requestid` extension: every event of one request carries the value the API returned to the caller.
Who acted	Platform events name the credential in `initiator`; agent records name the user in `subject` and the agents in `actorsChain`; Intent Agent records name the user in `subject_id` and `subject_type`.

Next Steps

- Reading the signed chain, with a complete event inside a batch: **Audit Log guide**
- Routing the platform events to your broker and the filter keys per event type: **Outbound Events guide**
- The gateway's reason strings and its `audit` configuration: **Agent Gateway guide**; the Token Service's decisions: **Token Service guide**
- The Intent Agent's delivery variables: **Intent Agent guide**
- Reading gateway records live in a demo: **Agent Gateway MCP tutorial**, chapter 8