Kneo Agent Platform

Reference

Version 0.4.0

2026-05-22

CLI usage and command reference, HTTP API contract, environment-variable surface, examples walkthrough, and project configuration — one printable reference.

Table of contents

CLI usage

Source: `docs/user/cli.md`

This page shows the day-to-day CLI patterns. The full subcommand and flag reference is generated from the Typer app and committed at `cli_reference.md`; refresh it with:

```
python docs/script/generate_reference_docs.py
```

For first-time setup, see `quickstart.md`.

Common commands

```
kneo config init --name research-agent-demo
kneo config show
kneo spec validate examples/research_agent.yaml
kneo spec lint examples/research_agent.yaml
kneo spec compile examples/research_agent.yaml
kneo spec migrate legacy_agent.yaml --output migrated_agent.yaml
kneo spec policy-report examples/research_agent.yaml --json
kneo spec validate examples/research_agent.yaml --env prod
kneo spec bundle sign examples/research_agent.yaml \
  --output bundles/research_agent.json \
  --approved-by release-manager --env prod
kneo spec bundle verify bundles/research_agent.json
kneo run examples/research_agent.yaml --input "Analyze Nvidia AI business"
kneo runs get <run_id>
kneo runs cancel <run_id>
kneo runs trace <run_id>
kneo runs checkpoints <run_id>
kneo runs replay <run_id>
kneo runs checkpoint-diff <run_id> --from-sequence 1 --to-sequence 3
kneo human get <continuation_id>
kneo human resume <continuation_id> --request-id <request_id> --approve
```

Talking to a service

For one-off service-backed calls, set environment variables:

```
export KNEO_SERV_API_KEY=<api-key>
kneo run examples/research_agent.yaml \
```

```
  --service-url http://127.0.0.1:8000 \
  --input "hello"
```

Profiles

For multiple service environments, store named profiles instead of re-typing URLs and tokens:

```
kneo config profile set local \
  --service-url http://127.0.0.1:8000 --api-key <api-key>
kneo config profile set staging \
  --service-url https://staging.example.com --api-key <api-key> --no-activate
kneo config profile list
kneo config profile use staging

kneo spec validate examples/research_agent.yaml --profile staging
kneo run examples/research_agent.yaml --profile staging --input "hello"
kneo runs get <run_id> --profile staging
kneo human list --profile staging
```

Profiles live at `~/.kneo_serv/profiles.json` with owner-only file permissions. Set `KNEO_SERV_PROFILES_PATH` to use a different location (handy in tests). The CLI reports whether a profile is authenticated but never prints stored API keys.

Service-connection precedence is:

1. `--service-url` (plus the API key from `--profile`, when given).
2. An explicit `--profile`.
3. `service.default_url` from `.kneo/config.yaml`.
4. The currently active CLI profile.
5. Local in-process execution.

Retry, timeout, and idempotency

Service-backed CLI commands honor retry and timeout settings:

```
export KNEO_SERV_CLIENT_TIMEOUT=120
export KNEO_SERV_CLIENT_RETRIES=2
export KNEO_SERV_CLIENT_RETRY_BACKOFF_SECONDS=0.25
```

Transient failures (`408`, `429`, `5xx`, plus network errors) are retried. Authentication, authorization, validation, and not-found failures fail fast with a clear message.

For retry-safe `POST`s, set a stable idempotency key:

```
export KNEO_SERV_IDEMPOTENCY_KEY=<stable-client-generated-key>
```

The service replays the original response for duplicate `POST /runs` or human-task resume requests that carry the same key and payload.

Spec migration

Use `kneo spec migrate` to convert older or unversioned YAML specs to the current `version: v1` shape:

```
kneo spec migrate legacy_agent.yaml --output migrated_agent.yaml
kneo spec migrate migrated_agent.yaml --check --json
```

Migration currently supports unversioned specs, `v0`, and `v1`. Unsupported future versions fail fast rather than being guessed.

Spec linting

`kneo spec lint` runs the same validator pipeline as `kneo spec validate` but **filters output to warnings and errors only** (info-severity diagnostics are dropped) and **exits non-zero if any are found**. Use it as a CI gate to catch deprecated fields, unsafe imports, shorthand tool selection without explicit permissions, network/shell/ filesystem-write capabilities without an allow-list, and missing human approvals on privileged surfaces — without having to filter `spec validate` output by hand.

```
# CI gate — fails the build if the spec carries any warnings or errors
kneo spec lint examples/research_agent.yaml
echo "exit=$?"

# Machine-readable output with summary counts
kneo spec lint examples/research_agent.yaml --json
```

The JSON envelope is:

```
{
  "clean": false,
  "warning_count": 3,
  "error_count": 0,
  "diagnostics": [
    {"severity": "warning", "code": "W_TOOL_POLICY_UNRESTRICTED", "path": "...", "message":
"...", "suggestion": "..."}
```

```
  ]
}
```

Lint is intentionally stricter than `validate` (which prints diagnostics but does not exit non-zero on warnings). For the full diagnostic list including info-severity items, use `kneo spec validate` instead.

Policy reports

`kneo spec policy-report` emits a structured governance report covering memory, tool permissions, guardrails, MCP imports, and human-review requirements:

```
kneo spec policy-report examples/research_agent.yaml --json
kneo spec policy-report examples/research_agent.yaml \
  --profile staging --json
```

The report includes validation diagnostics, so CI can block on errors while still surfacing warnings such as unrestricted network tools or missing human approval steps.

When project config declares `environments.<name>.policy_enforcement`, spec commands and `run --env <name>` enforce the selected environment after overlays and defaults are applied.

Spec bundles

`kneo spec bundle sign` produces an approved deployment bundle with a canonical spec digest and an HMAC signature:

```
export KNEO_SERV_SPEC_SIGNING_KEY=<deployment-signing-key>
kneo spec bundle sign examples/research_agent.yaml \
  --output bundles/research_agent.prod.json \
  --approved-by release-manager \
  --env prod
kneo spec bundle verify bundles/research_agent.prod.json
```

Verification fails if the bundle contents are edited after signing, or if a different signing key is used.

Kneo Serv CLI Reference

Source: *docs/user/cli_reference.md*

Generated by `python docs/script/generate_reference_docs.py`.

`kneo`

```
 Usage: kneo [OPTIONS] COMMAND [ARGS]...

Kneo Agent Platform CLI

Options
--install-completion          Install completion for the current shell.
--show-completion             Show completion for the current shell, to copy
                              it or customize the installation.
--help                        Show this message and exit.

Commands
config   Manage project config
spec     Validate and compile specs
run      Run agent/workflow specs
runs     Inspect runs
human    Manage human-in-the-loop tasks
service  Run or manage the Kneo service
```

`kneo config`

```
 Usage: kneo config [OPTIONS] COMMAND [ARGS]...

Manage project config

Options
--help          Show this message and exit.

Commands
show         Show project config.
init         Create a .kneo/config.yaml project config.
resolve      Resolve project context, including environment and overlays.
secrets      Show configured secret references without exposing values.
render-spec  Render default spec plus environment overlays into one YAML
             file.
profile      Manage service connection profiles
```

kneo config show

```
Usage: kneo config show [OPTIONS]

Show project config.

Options
--json
--help          Show this message and exit.
```

kneo config init

```
Usage: kneo config init [OPTIONS]

Create a .kneo/config.yaml project config.

Options
--name         TEXT  [default: kneo-serv-project]
--force
--help               Show this message and exit.
```

kneo config resolve

```
Usage: kneo config resolve [OPTIONS]

Resolve project context, including environment and overlays.

Options
--env         TEXT
--spec        PATH
--json
--help              Show this message and exit.
```

kneo config secrets

```
Usage: kneo config secrets [OPTIONS]

Show configured secret references without exposing values.

Options
--env         TEXT
```

```
--json
--help              Show this message and exit.
```

kneo config render-spec

```
 Usage: kneo config render-spec [OPTIONS]

Render default spec plus environment overlays into one YAML file.

Options
*  --output  -o      PATH  [required]
   --env             TEXT
   --spec            PATH
   --help                  Show this message and exit.
```

kneo config profile

```
 Usage: kneo config profile [OPTIONS] COMMAND [ARGS]...

Manage service connection profiles

Options
--help          Show this message and exit.

Commands
set     Create or update a named service profile.
use     Set the active service profile.
list    List configured service profiles without exposing API keys.
show    Show one service profile without exposing its API key.
delete  Delete a service profile.
```

kneo config profile set

```
 Usage: kneo config profile set [OPTIONS] NAME

Create or update a named service profile.

Arguments
*    name      TEXT  [required]

Options
*  --service-url                    TEXT  [required]
```

```
  --api-key                       TEXT
  --activate      --no-activate         [default: activate]
  --json
  --help                                Show this message and exit.
```

kneo config profile use

```
Usage: kneo config profile use [OPTIONS] NAME

Set the active service profile.

Arguments
*    name      TEXT  [required]

Options
--json
--help          Show this message and exit.
```

kneo config profile list

```
Usage: kneo config profile list [OPTIONS]

List configured service profiles without exposing API keys.

Options
--json
--help          Show this message and exit.
```

kneo config profile show

```
Usage: kneo config profile show [OPTIONS]

Show one service profile without exposing its API key.

Options
--name        TEXT
--json
--help              Show this message and exit.
```

kneo config profile delete

```
Usage: kneo config profile delete [OPTIONS] NAME

Delete a service profile.

Arguments
*    name      TEXT  [required]

Options
--help          Show this message and exit.
```

kneo spec

```
Usage: kneo spec [OPTIONS] COMMAND [ARGS]...

Validate and compile specs

Options
--help          Show this message and exit.

Commands
validate
lint           Surface validator warnings and errors. Exits 1 if any are
               found.
compile
resolve
migrate        Migrate an older YAML spec to the current v1 shape.
policy-report  Generate a policy evaluation report for a spec.
bundle         Sign and verify approved spec bundles
```

kneo spec validate

```
Usage: kneo spec validate [OPTIONS] [SPEC_PATH]

Arguments
  spec_path      [SPEC_PATH]

Options
--env                TEXT
--json
--service-url        TEXT
--profile            TEXT  Service profile name
--help                     Show this message and exit.
```

kneo spec compile

```
Usage: kneo spec compile [OPTIONS] [SPEC_PATH]

Arguments
  spec_path      [SPEC_PATH]

Options
--env                TEXT
--json
--service-url        TEXT
--profile            TEXT  Service profile name
--help                     Show this message and exit.
```

kneo spec resolve

```
Usage: kneo spec resolve [OPTIONS] [SPEC_PATH]

Arguments
  spec_path      [SPEC_PATH]

Options
*  --output  -o      PATH  [required]
   --env             TEXT
   --help                  Show this message and exit.
```

kneo spec migrate

```
Usage: kneo spec migrate [OPTIONS] SPEC_PATH

Migrate an older YAML spec to the current v1 shape.

Arguments
*    spec_path      PATH  [required]

Options
--output  -o      PATH
--check
--json
--help                  Show this message and exit.
```

kneo spec policy-report

```
Usage: kneo spec policy-report [OPTIONS] [SPEC_PATH]

Generate a policy evaluation report for a spec.

Arguments
  spec_path      [SPEC_PATH]

Options
--env               TEXT
--json
--service-url       TEXT
--profile           TEXT  Service profile name
--help                    Show this message and exit.
```

kneo spec bundle

```
Usage: kneo spec bundle [OPTIONS] COMMAND [ARGS]...

Sign and verify approved spec bundles

Options
--help          Show this message and exit.

Commands
sign    Create an approved, signed spec bundle.
verify  Verify an approved spec bundle digest and signature.
```

kneo spec bundle sign

```
Usage: kneo spec bundle sign [OPTIONS] SPEC_PATH

Create an approved, signed spec bundle.

Arguments
*    spec_path      PATH  [required]

Options
*  --output       -o      PATH  [required]
*  --approved-by          TEXT  [required]
   --env                  TEXT
   --key-env              TEXT  [default: KNEO_SERV_SPEC_SIGNING_KEY]
```

```
   --json
   --help                      Show this message and exit.
```

kneo spec bundle verify

```
 Usage: kneo spec bundle verify [OPTIONS] BUNDLE_PATH

Verify an approved spec bundle digest and signature.

Arguments
*    bundle_path      PATH  [required]

Options
--key-env        TEXT  [default: KNEO_SERV_SPEC_SIGNING_KEY]
--json
--help                 Show this message and exit.
```

kneo run

```
 Usage: kneo run [OPTIONS] [SPEC_PATH] COMMAND [ARGS]...

Run agent/workflow specs

Arguments
  spec_path      [SPEC_PATH]  Path to Kneo YAML spec

Options
*  --input        -i      TEXT  Input text [required]
   --target               TEXT  agent or workflow [default: workflow]
   --env                  TEXT
   --json
   --service-url          TEXT
   --profile              TEXT  Service profile name
   --help                       Show this message and exit.
```

kneo runs

```
 Usage: kneo runs [OPTIONS] COMMAND [ARGS]...

Inspect runs

Options
```

```
--help          Show this message and exit.

Commands
get
trace
checkpoints
replay
checkpoint-diff
cancel
```

kneo runs get

```
 Usage: kneo runs get [OPTIONS] RUN_ID

Arguments
*    run_id      TEXT  [required]

Options
--json
--service-url        TEXT
--profile            TEXT  Service profile name
--help                     Show this message and exit.
```

kneo runs trace

```
 Usage: kneo runs trace [OPTIONS] RUN_ID

Arguments
*    run_id      TEXT  [required]

Options
--json
--service-url        TEXT
--profile            TEXT  Service profile name
--help                     Show this message and exit.
```

kneo runs checkpoints

```
 Usage: kneo runs checkpoints [OPTIONS] RUN_ID

Arguments
*    run_id      TEXT  [required]
```

```
Options
--json
--service-url        TEXT
--profile            TEXT  Service profile name
--help                     Show this message and exit.
```

kneo runs replay

```
 Usage: kneo runs replay [OPTIONS] RUN_ID

Arguments
*    run_id      TEXT  [required]

Options
--json
--service-url        TEXT
--profile            TEXT  Service profile name
--help                     Show this message and exit.
```

kneo runs checkpoint-diff

```
 Usage: kneo runs checkpoint-diff [OPTIONS] RUN_ID

Arguments
*    run_id      TEXT  [required]

Options
--from-sequence        INTEGER
--to-sequence          INTEGER
--json
--service-url          TEXT
--profile              TEXT     Service profile name
--help                          Show this message and exit.
```

kneo runs cancel

```
 Usage: kneo runs cancel [OPTIONS] RUN_ID

Arguments
*    run_id      TEXT  [required]
```

```
Options
--json
--service-url        TEXT
--profile            TEXT  Service profile name
--help                     Show this message and exit.
```

kneo human

```
 Usage: kneo human [OPTIONS] COMMAND [ARGS]...

Manage human-in-the-loop tasks

Options
--help          Show this message and exit.

Commands
get
list
resume
```

kneo human get

```
 Usage: kneo human get [OPTIONS] CONTINUATION_ID

Arguments
*    continuation_id      TEXT  [required]

Options
--json
--service-url        TEXT
--profile            TEXT  Service profile name
--help                     Show this message and exit.
```

kneo human list

```
 Usage: kneo human list [OPTIONS]

Options
--json
--service-url        TEXT
```

```
--profile            TEXT  Service profile name
--help                     Show this message and exit.
```

kneo human resume

```
 Usage: kneo human resume [OPTIONS] CONTINUATION_ID

Arguments
*    continuation_id      TEXT  [required]

Options
*  --request-id         TEXT  [required]
   --approve
   --reject
   --edit               TEXT
   --provide            TEXT
   --select             TEXT
   --json
   --service-url        TEXT
   --profile            TEXT  Service profile name
   --help                     Show this message and exit.
```

kneo service

```
 Usage: kneo service [OPTIONS] COMMAND [ARGS]...

Run or manage the Kneo service

Options
--help          Show this message and exit.

Commands
serve  Run the Kneo FastAPI service.
```

kneo service serve

```
 Usage: kneo service serve [OPTIONS]

Run the Kneo FastAPI service.

Options
--host          TEXT     [default: 127.0.0.1]
```

```
  --port          INTEGER  [default: 8000]
  --reload
  --help                   Show this message and exit.
```

Platform service API

Source: `docs/user/service_api.md`

The HTTP contract exposed by `kneo service serve`. The generated OpenAPI schema is committed at `openapi.json`; refresh it with `python docs/script/generate_reference_docs.py`.

This page covers versioning, auth, redaction, governance, and the request / response shapes for each route group. The Worked examples section near the bottom has copy-pasteable `curl` invocations for the most common endpoints.

Versioning

The stable public HTTP API is exposed under `/v1`. Existing unversioned routes remain available for local development and backwards compatibility, but new service clients should prefer `/v1`.

Examples:

```
GET /v1/healthz
POST /v1/runs
GET /v1/runs/{run_id}
POST /v1/human-tasks/{continuation_id}/resume
```

Authentication

Authentication is disabled by default for local development. Enable it by setting API keys before starting the service:

```
export KNEO_SERV_AUTH_ENABLED=true
export KNEO_SERV_API_KEYS='operator:operator-token:operator;reviewer:reviewer-token:reviewer'
export KNEO_SERV_ADMIN_API_KEY='admin-token'
```

Clients authenticate with either header:

```
Authorization: Bearer <api-key>
X-Kneo-Api-Key: <api-key>
```

Built-in roles:

- `admin`: all scopes
- `operator`: `runs:read`, `runs:write`, `specs:read`, `human:read`, `audit:read`, `credentials:read`, `policies:read`, `policies:write`

- `reviewer`: `runs:read`, `human:read`, `human:write`, `audit:read`
- `service`: `runs:read`, `runs:write`, `specs:read`, `human:read`, `human:write`, `audit:read`, `audit:write`, `credentials:read`, `policies:read`
- `viewer`: `runs:read`, `specs:read`, `human:read`, `audit:read`

`KNEO_SERV_API_KEYS` accepts semicolon-separated entries:

```
name:key:role_or_scope[,role_or_scope]
```

Example with explicit scopes:

```
export KNEO_SERV_API_KEYS='ci:ci-token:service;runs-reader:read-token:runs:read'
```

Redaction

Service responses, traces, checkpoints, and CLI JSON output are redacted before they are returned or persisted as checkpoints. Redaction covers common secret keys and inline values such as passwords, tokens, API keys, authorization headers, emails, and SSNs.

Spec governance diagnostics

Spec validation includes static governance diagnostics before deployment:

- Unsafe tool or function implementation imports such as direct `os`, `subprocess`, `shutil`, `socket`, `importlib`, or `builtins` primitives are reported as errors.
- Shorthand tool selection or missing tool permission policies are reported as warnings.
- Network tools without `allowed_domains`, shell-capable tools, and filesystem write access are reported as warnings.
- Specs that expose privileged tools or unsafe imports without a human workflow approval step receive a `W_HUMAN_APPROVAL_MISSING` warning.

These diagnostics are returned by `kneo spec validate`, `POST /specs/validate`, and strict compiler flows.

`POST /specs/policy-report` returns a structured policy report covering memory configuration, tool permissions, declared MCP imports, guardrail stages, human reviewers, and human approval requirements. Use it in deployment gates when a spec needs a machine-readable policy summary before signing or promotion.

`GET /runs/{run_id}/policy-report` returns the same shape but operates on the spec the run was started with — the service reads it out of the run's stored metadata, so operators auditing a deployed

run don't need to ship the bundle to the service themselves. Same `specs:read` scope as the spec-bundle route.

```
curl -H "Authorization: Bearer $KNEO_API_KEY" \
  https://kneo.example.com/v1/runs/run-7c2f.../policy-report
```

Returns `404` if the run id is unknown, `400` if the run carries no spec metadata (older runs from a pre-0.3.0 store), and `200` with `{"valid": <bool>, "report": {...}}` otherwise. Each call records a `spec.policy_reported` audit event scoped to the run id with `metadata.source = "run"`, so spec-bundle calls and run-keyed calls are distinguishable in the audit log.

Project-based CLI flows can enforce different gates per environment through `environments.<name>.policy_enforcement`. Enforcement runs after overlays and defaults are applied, so `dev`, `staging`, and `prod` can require progressively stricter tool permissions, human review, guardrails, or blocked diagnostic codes.

Redaction is a safety layer, not a replacement for secret management. Provider keys and credentials should still be supplied through deployment secret stores or environment variables rather than embedded in specs or request payloads.

Workflow specs

YAML specs can target SDK-backed workflow families while preserving service validation, tracing, cancellation, and run-result metadata:

- `sequential`: ordered `steps`.
- `graph`: keyed `nodes`, conditional `edges`, and a `start` node.
- `concurrent`: fan-out `participants` executed by the SDK concurrent workflow.
- `handoff`: `participants` plus a `selector`; `sequence` and `round_robin` selectors are supported.
- `group-chat` or `group_chat`: `participants` repeated for `rounds`.

Orchestration workflow participants use the same step shape as sequential workflow steps:

```
workflow:
  type: handoff
  name: review-handoff
  participants:
    - id: researcher
      kind: agent
      ref: research_agent
    - id: reviewer
      kind: agent
      ref: review_agent
```

```
  selector:
    type: sequence
    sequence: [researcher, reviewer]
```

Participant ids must be unique, participant refs must resolve to declared components, handoff selector entries must reference participant ids, and group-chat `rounds` must be at least `1`.

Secret management

`kneo_serv` resolves provider keys, MCP credentials, service tokens, and runtime-specific values through named environment-variable references. Project config stores only env-var names, never raw secret values:

```
secrets:
  provider_env:
    openai: OPENAI_API_KEY
  extra_env:
    mcp_default: MCP_API_KEY
```

The default provider mappings include `openai` / `openai-agents`, `anthropic`, `google`, and `google-adk`. The CLI can show a redacted inventory for deployment checks:

```
kneo config secrets --json
```

Native provider startup can fail fast when a required provider secret is missing:

```
export KNEO_SERV_REQUIRE_PROVIDER_SECRETS=true
```

Service API keys remain in `KNEO_SERV_API_KEY`, `KNEO_SERV_API_KEYS`, and `KNEO_SERV_ADMIN_API_KEY`; the secret inventory reports whether they are present without exposing values.

The service exposes the same redacted inventory for operators:

```
GET /v1/security/credentials?providers=openai&include_service_tokens=false
```

This endpoint requires `credentials:read`. The response reports configured provider, extra, and service-token references with `present` flags and redacted values:

```
{
  "inventory": {
    "providers": {
```

```
      "openai": {
        "name": "provider:openai",
        "env_var": "OPENAI_API_KEY",
        "present": true,
        "value": "[REDACTED]"
      }
    },
    "extra": {}
  }
}
```

Every successful credential inventory request records a `credential.inventory_accessed` audit event. Audit metadata includes counts and which reference names were present; raw secret values are never included.

Environment policy management

Environment policy enforcement can be managed through the service when a deployment needs operator-controlled gates outside checked-in project config:

```
GET /v1/policies/environment
GET /v1/policies/environment/prod
PUT /v1/policies/environment/prod
```

Reads require `policies:read`; writes require `policies:write`. A policy update stores validated `EnvironmentPolicyEnforcement` settings in the run state store's `project_metadata` table/key-value area:

```
{
  "enabled": true,
  "fail_on_warnings": false,
  "blocked_diagnostic_codes": [],
  "require_human_review": true,
  "require_tool_permissions": true,
  "deny_unrestricted_tools": true,
  "require_guardrails": false
}
```

The response includes the current policy and, for updates, the previous policy when one existed. Each successful update records a `policy.changed` audit event with the policy surface, environment, previous/current redacted policy payloads, and changed field names.

Request limits

The service rejects oversized request bodies before route handling and applies strict request-model validation for inline payloads. Unknown request fields are rejected with `422`, and bodies above the configured transport limit return `413`.

Default limits:

- `KNEO_SERV_MAX_BODY_BYTES`: `1048576`
- `KNEO_SERV_MAX_INPUT_CHARS`: `20000`
- `KNEO_SERV_MAX_HUMAN_CONTENT_CHARS`: `20000`
- `KNEO_SERV_MAX_INLINE_SPEC_BYTES`: `262144`
- `KNEO_SERV_MAX_OVERRIDES_BYTES`: `65536`
- `KNEO_SERV_MAX_METADATA_BYTES`: `32768`
- `KNEO_SERV_MAX_LIST_ITEMS`: `100`
- `KNEO_SERV_MAX_PATH_CHARS`: `4096`

Structured logging

API requests emit redacted JSON log records on the `kneo_serv.service` logger. Each request record includes `event=http_request`, `request_id`, method, path, status code, duration, client IP when available, and route-supplied run, continuation, or trace IDs when known.

Clients can send `X-Request-ID`; otherwise the service generates one. The response always includes the effective `X-Request-ID`.

Configuration:

- `KNEO_SERV_REQUEST_LOGS`: defaults to `true`
- `KNEO_SERV_LOG_LEVEL`: defaults to `INFO`

SDK OpenTelemetry tracing

When SDK telemetry support is installed, set `KNEO_SERV_OTEL_ENABLED=true` to attach `kneo_agent.observability.OpenTelemetryMiddleware` to SDK-backed agents. The middleware uses the OpenTelemetry global tracer provider, so exporters and resources can be configured with standard `OTEL_*` environment variables in the deployment environment.

Service defaults keep potentially sensitive span attributes disabled:

- `KNEO_SERV_OTEL_RECORD_ARGUMENTS`: defaults to `false`
- `KNEO_SERV_OTEL_RECORD_RESULTS`: defaults to `false`

Enable those only for trusted deployments where tool arguments and results are safe to emit to telemetry backends.

Idempotency

`POST /runs`, `POST /specs/run`, and `POST /human-tasks/{continuation_id}/resume` support the `Idempotency-Key` header. When the same key is reused with the same request payload, the service returns the original response without creating a duplicate run or submitting a second human decision.

```
Idempotency-Key: <stable-client-generated-key>
```

Reusing a key with a different payload returns `409` with `idempotency_key_conflict`.

The CLI service client can send a key per call in code, or read one from:

```
export KNEO_SERV_IDEMPOTENCY_KEY=<stable-client-generated-key>
```

Human-task resume also takes a store-backed continuation lock. If another process is already resuming the same continuation, the service returns `409` with `resource_locked`.

Run cancellation

`POST /runs/{run_id}/cancel` marks a pending or running run as `cancelled`. Background execution receives a cooperative cancellation token through the SDK run config `extra` payload, so service workflows, agents, runtimes, and wrapped workflow steps check cancellation before and after unit-of-work boundaries. A cancelled run is not overwritten as completed if execution returns after cancellation was requested.

Provider calls that do not expose an interrupt primitive can only stop at the next cooperative boundary after the provider returns.

Retry, timeout, and backoff

Service-client retries are configured with `KNEO_SERV_CLIENT_*` variables. Provider/runtime and MCP calls use the same conservative policy shape:

```
 export KNEO_SERV_PROVIDER_RETRIES=2
export KNEO_SERV_PROVIDER_RETRY_BACKOFF_SECONDS=0.25
export KNEO_SERV_PROVIDER_TIMEOUT_SECONDS=120

export KNEO_SERV_MCP_RETRIES=2
export KNEO_SERV_MCP_RETRY_BACKOFF_SECONDS=0.25
export KNEO_SERV_MCP_TIMEOUT_SECONDS=30
```

Workflow steps can also set `on_error: retry`, `max_retries`, and `timeout_seconds` in YAML specs. Cancellation is never retried.

Health checks

This section is the API contract. For an on-call triage tree mapping each `/readyz` check to recovery actions, see `incident_response.md`.

- `GET /healthz`: lightweight API health.
- `GET /livez`: process liveness.
- `GET /readyz`: readiness for API wiring, run state store, continuation store, durable run queue, runtime registry, tool registry, and configured provider or MCP secret dependencies.

Provider and MCP dependency checks are opt-in so local development does not fail when no real upstream credentials are configured:

```
export KNEO_SERV_HEALTH_PROVIDERS=openai,anthropic
export KNEO_SERV_HEALTH_MCP_SECRETS=mcp_default
```

If a configured readiness dependency is missing or unhealthy, `/readyz` returns `503` with a structured `not_ready` detail payload.

Background worker queue

Async run creation enqueues run IDs into the configured run state store before worker execution. SQLite and file stores persist queue records with status, attempt count, lease owner, lease expiry, and error details; in-memory stores keep the same contract for tests and local ephemeral use.

Workers claim queued or expired leased records, execute the run through the same `PlatformManager.execute_run` path, and then mark the queue record `completed` or `failed`. On service startup the default manager starts a worker so previously queued records can be resumed.

Recovery and continuation

Workflow execution stores live execution context on run state and persists step/node completion and failure checkpoints. For interrupted non-human sequential workflows, the service can report the completed steps, failed step, resume input, and next step index:

```
GET /runs/{run_id}/recovery
```

When `replay_context.can_continue` is true, the run can continue from the last completed step boundary:

```
POST /runs/{run_id}/continue
```

Graph workflows expose replay context from node checkpoints, but automatic continuation is limited to sequential workflows until graph edge state is persisted at each routing decision.

Replay and checkpoint diff

Operators can inspect a compact replay timeline without reading full checkpoint payloads:

```
GET /runs/{run_id}/replay
```

The response includes checkpoint sequence, type, step/node IDs, status, current execution position, pending human request ID, error summary, and the same replay context used by `/runs/{run_id}/recovery`.

Checkpoint diffs compare checkpoint state and metadata. By default the latest two checkpoints are compared:

```
GET /runs/{run_id}/checkpoints/diff
GET /runs/{run_id}/checkpoints/diff?from_sequence=1&to_sequence=3
```

The diff response reports added, removed, and changed flattened paths. Values are redacted before returning.

Audit events

The service records redacted audit events in the configured run state store for successful spec operations, run creation, run cancellation, run continuation, spec-run execution, and human-in-the-loop decisions.

```
GET /audit-events
GET /audit-events?event_type=run.created
GET /audit-events?run_id=<run_id>
```

The audit list endpoint requires `audit:read` and returns events newest first. Each event includes `event_type`, `actor`, optional `run_id` and `continuation_id`, redacted metadata, and `created_at`.

SQLite migrations

SQLite state stores apply versioned migrations on startup. The migration table is `schema_migrations`, and the current schema covers run state, checkpoints, idempotency records, locks, durable run queue records, continuation records, audit event records, and project metadata records.

Existing unversioned SQLite databases are upgraded in place with `CREATE TABLE IF NOT EXISTS` and `CREATE INDEX IF NOT EXISTS` statements, so existing run payloads remain readable after migration.

Project metadata is used by service-managed environment policies. Upgrade coverage verifies that existing SQLite databases can create, persist, and reload policy metadata after migrations have applied.

Retention and pruning

`RetentionManager` provides an operator-callable pruning job for run state, checkpoints, completed or failed queue records, file-backed continuations, artifacts, and logs. It can be configured directly or through environment variables:

```
 export KNEO_SERV_RETENTION_RUNS_DAYS=30
export KNEO_SERV_RETENTION_CHECKPOINTS_DAYS=30
export KNEO_SERV_RETENTION_QUEUE_DAYS=14
export KNEO_SERV_RETENTION_CONTINUATIONS_DAYS=30
export KNEO_SERV_RETENTION_ARTIFACTS_DAYS=30
export KNEO_SERV_RETENTION_LOGS_DAYS=30
```

The platform manager exposes `prune_retention()` for embedded operators and future scheduled jobs.

Checkpoint payload limits

SQLite and file stores transparently compress large checkpoint payloads before writing them. If a checkpoint remains above the hard cap after compression, the store persists a bounded checkpoint preview that keeps run ID, checkpoint type, step/node IDs, timestamps, limited trace previews, and metadata describing the size reduction.

Defaults:

- `KNEO_SERV_CHECKPOINT_COMPRESS_BYTES`: `65536`
- `KNEO_SERV_CHECKPOINT_MAX_BYTES`: `1048576`
- `KNEO_SERV_CHECKPOINT_PREVIEW_CHARS`: `1200`

- `KNEO_SERV_CHECKPOINT_MAX_LIST_ITEMS`: `20`
- `KNEO_SERV_CHECKPOINT_MAX_DICT_ITEMS`: `50`

Backup and restore

This section documents the Python backup API. For the operator-facing production procedure (PostgreSQL `pg_dump`, off-site rotation, restore verification, DR checklist), see `backup_and_recovery.md`.

The default SQLite store can be backed up online with SQLite's backup API:

```
from kneo_serv.maintenance import backup_sqlite_database, restore_sqlite_database

backup_sqlite_database(".kneo/kneo_runs.sqlite", ".kneo/backups/kneo_runs.sqlite")
restore_sqlite_database(".kneo/backups/kneo_runs.sqlite", ".kneo/kneo_runs.restored.sqlite")
```

The smoke test covers run state and checkpoint restore from the copied database. File-backed continuations, artifacts, and logs should be included in deployment-level filesystem backups when those paths are used.

Runs

- POST /v1/runs
- GET /v1/runs
- GET /v1/runs/{run_id}
- POST /v1/runs/{run_id}/cancel
- GET /v1/runs/{run_id}/policy-report
- GET /v1/runs/{run_id}/recovery
- GET /v1/runs/{run_id}/replay
- POST /v1/runs/{run_id}/continue
- GET /v1/runs/{run_id}/checkpoints
- GET /v1/runs/{run_id}/checkpoints/diff
- GET /v1/runs/{run_id}/trace

Legacy aliases:

- GET /runs
- POST /runs
- GET /runs/{run_id}
- POST /runs/{run_id}/cancel
- GET /runs/{run_id}/recovery
- GET /runs/{run_id}/replay

- POST /runs/{run_id}/continue
- GET /runs/{run_id}/checkpoints
- GET /runs/{run_id}/checkpoints/diff
- GET /runs/{run_id}/trace

Human tasks

- GET /v1/human-tasks
- GET /v1/human-tasks/{continuation_id}
- POST /v1/human-tasks/{continuation_id}/resume

Legacy aliases:

- GET /human-tasks
- GET /human-tasks/{continuation_id}
- POST /human-tasks/{continuation_id}/resume

Specs

- POST /v1/specs/validate
- POST /v1/specs/compile
- POST /v1/specs/explain
- POST /v1/specs/run

Legacy aliases:

- POST /specs/validate
- POST /specs/compile
- POST /specs/explain
- POST /specs/run

Audit

- GET /v1/audit-events

Legacy alias:

- GET /audit-events

Security and policies

- GET /v1/security/credentials

- GET /v1/policies/environment
- GET /v1/policies/environment/{environment}
- PUT /v1/policies/environment/{environment}

Legacy aliases:

- GET /security/credentials
- GET /policies/environment
- GET /policies/environment/{environment}
- PUT /policies/environment/{environment}

Worked examples

Concrete `curl` invocations and abbreviated response shapes for the most common endpoints. The full schema is in `openapi.json`; these are illustrative.

All examples assume:

```
export BASE=http://127.0.0.1:8000
export KEY=operator-token   # an entry from KNEO_SERV_API_KEYS
```

Health

```
curl -sf "$BASE/livez"     # {"ok": true, "metadata": {}}
curl -sf "$BASE/readyz"    # 200 with checks: {} or 503 with not_ready details
```

`/livez` and `/readyz` are intentionally unauthenticated. See troubleshooting.md § 1.2 for the failure shape.

Create a run

Required scope: `runs:write`.

```
curl -sf -X POST "$BASE/v1/runs" \
  -H "Authorization: Bearer $KEY" \
  -H 'Content-Type: application/json' \
  -d '{
    "input": "Summarize Nvidia AI strategy",
    "spec_path": "examples/research_agent.yaml",
    "target": "workflow",
    "environment": "prod",
    "async_mode": false
  }' | jq
```

Synchronous response (run finished within the request):

```
{
  "run_id": "run_2026-05-10T12:34:56_a1b2c3d4",
  "status": "succeeded",
  "output_text": "Nvidia's AI strategy hinges on …",
  "human_intervention_required": false,
  "continuation_id": null,
  "metadata": {"workflow_kind": "sequential", "trace_event_count": 7}
}
```

If the workflow pauses on a human step:

```
{
  "run_id": "run_…",
  "status": "paused",
  "output_text": null,
  "human_intervention_required": true,
  "continuation_id": "cont_…",
  "metadata": {"pending_human_request": {"request_id": "req_…", "prompt": "Approve the
draft?"}}
}
```

For retry-safe submissions, send an `Idempotency-Key` header. Reusing the same key with the same body replays the original response; mismatched bodies return `409 idempotency_key_conflict`.

Get run state

Required scope: `runs:read`.

```
curl -sf "$BASE/v1/runs/run_…" \
  -H "Authorization: Bearer $KEY" | jq
```

```
{
  "run_id": "run_…",
  "status": "running",
  "agent_name": "research-copilot",
  "workflow_name": "research-pipeline",
  "workflow_kind": "sequential",
  "current_step_index": 1,
  "current_node_id": "analyze",
  "visited_steps": ["retrieve"],
  "visited_nodes": ["retrieve"],
  "trace_event_count": 4,
```

```
  "metadata": {"environment": "prod"}
}
```

For terminal status:

```
{
  "run_id": "run_…",
  "status": "succeeded",
  "output_text": "…",
  "visited_steps": ["retrieve", "analyze", "summarize"],
  "trace_event_count": 11
}
```

List runs (paginated)

```
curl -sf "$BASE/v1/runs?status=running&limit=20&sort_by=created_at&sort_order=desc" \
  -H "Authorization: Bearer $KEY" | jq
```

```
{
  "runs": [
    {"run_id": "run_…", "status": "running", "workflow_name": "research-pipeline",
"created_at": "2026-05-10T12:30:00Z"},
    {"run_id": "run_…", "status": "running", "workflow_name": "approval-workflow",
"created_at": "2026-05-10T12:28:11Z"}
  ],
  "count": 2,
  "total": 2,
  "limit": 20,
  "offset": 0,
  "sort_by": "created_at",
  "sort_order": "desc"
}
```

Cancel a run

```
curl -sf -X POST "$BASE/v1/runs/run_…/cancel" \
  -H "Authorization: Bearer $KEY"
```

The run transitions to `cancelled`; cancellation is cooperative — in-flight steps stop at unit-of-work boundaries. See troubleshooting.md § 5.2.

Validate a spec

Required scope: `specs:read`.

```
curl -sf -X POST "$BASE/v1/specs/validate" \
  -H "Authorization: Bearer $KEY" \
  -H 'Content-Type: application/json' \
  -d '{"spec_path": "examples/research_agent.yaml", "environment": "prod"}' | jq
```

```
{
  "valid": true,
  "diagnostics": [],
  "report": {
    "agent_name": "research-copilot",
    "workflow_name": "research-pipeline"
  }
}
```

For an invalid spec, `valid` is `false` and `diagnostics` is populated:

```
{
  "valid": false,
  "diagnostics": [
    {
      "severity": "error",
      "code": "E_UNKNOWN_TOOL",
      "message": "Tool 'web_search' is not registered.",
      "path": "agent.tools[0]"
    }
  ]
}
```

List human tasks

Required scope: `human:read`.

```
curl -sf "$BASE/v1/human-tasks?run_id=run_…" \
  -H "Authorization: Bearer $KEY" | jq
```

```
{
  "tasks": [
    {
      "continuation_id": "cont_…",
      "run_id": "run_…",
      "request": {
        "request_id": "req_…",
```

```
        "prompt": "Approve the draft?",
        "deadline_epoch": 1715432400
      }
    }
  ],
  "count": 1,
  "total": 1,
  "limit": 100,
  "offset": 0
}
```

Resume a human task

Required scope: `human:write`. Pair with `Idempotency-Key` for safe retries.

```
curl -sf -X POST "$BASE/v1/human-tasks/cont_…/resume" \
  -H "Authorization: Bearer $KEY" \
  -H "Idempotency-Key: $(uuidgen)" \
  -H 'Content-Type: application/json' \
  -d '{
    "request_id": "req_…",
    "decision": "approved",
    "content": "Looks good. Ship it."
  }' | jq
```

```
{
  "run_id": "run_…",
  "status": "succeeded",
  "output_text": "Published. https://…",
  "human_intervention_required": false,
  "continuation_id": null,
  "metadata": {}
}
```

`decision` is one of `approved`, `rejected`, `edited`, `selected`, `provided`. See human_in_the_loop.md.

List audit events

Required scope: `audit:read`. Audit payloads are redacted; secret and PII patterns never appear.

```
curl -sf "$BASE/v1/audit-events?event_type=human.decision" \
  -H "Authorization: Bearer $KEY" | jq
```

```
{
  "events": [
    {
      "id": "evt_…",
      "event_type": "human.decision",
      "actor": "reviewer",
      "occurred_at": "2026-05-10T12:35:01Z",
      "payload": {
        "request_id": "req_…",
        "decision": "approved",
        "selected_option": null,
        "result_status": "succeeded",
        "content_present": true
      }
    }
  ],
  "count": 1
}
```

Inspect credential references

Required scope: `credentials:read`. Returns *presence* metadata only; secret values never appear.

```
curl -sf "$BASE/v1/security/credentials" \
  -H "Authorization: Bearer $KEY" | jq
```

```
{
  "inventory": {
    "providers": [
      {"name": "openai", "env_var": "OPENAI_API_KEY", "present": true, "value":
"***REDACTED***"},
      {"name": "anthropic", "env_var": "ANTHROPIC_API_KEY", "present": false, "value": null}
    ],
    "mcp": [],
    "service": [
      {"name": "operator", "env_var": "KNEO_SERV_API_KEYS", "present": true, "value":
"***REDACTED***"}
    ]
  }
}
```

Each access records a `credential.inventory_accessed` audit event.

Read or update environment policy

Read requires `policies:read`; write requires `policies:write`.

```
curl -sf "$BASE/v1/policies/environment/prod" \
  -H "Authorization: Bearer $KEY" | jq
```

```
{
  "environment": "prod",
  "policy": {
    "enabled": true,
    "fail_on_warnings": false,
    "blocked_diagnostic_codes": ["E_UNSAFE_TOOL_IMPORT"],
    "require_human_review": false,
    "require_tool_permissions": true,
    "deny_unrestricted_tools": true,
    "require_guardrails": false
  }
}
```

```
curl -sf -X PUT "$BASE/v1/policies/environment/prod" \
  -H "Authorization: Bearer $KEY" \
  -H 'Content-Type: application/json' \
  -d '{
    "enabled": true,
    "require_tool_permissions": true,
    "deny_unrestricted_tools": true,
    "blocked_diagnostic_codes": ["E_UNSAFE_TOOL_IMPORT", "E_UNSAFE_FUNCTION_IMPORT"]
  }' | jq
```

The response includes `previous_policy` so you can audit what changed. Each write records a `policy.changed` audit event.

Error response shape

All error paths use the same envelope:

```
{
  "error": "forbidden",
  "message": "Missing required scope: runs:write",
  "required_scope": "runs:write"
}
```

Common error codes: `unauthorized` (401), `forbidden` (403), `invalid_request` (400), `not_found` (404), `idempotency_key_conflict` (409), `payload_too_large` (413), `not_ready` (503). Errors map through `service/errors.py`.

Pagination, filtering, and sorting

List-style endpoints return the original collection field plus pagination metadata:

```
{
  "count": 25,
  "total": 91,
  "limit": 25,
  "offset": 50,
  "sort_by": "updated_at",
  "sort_order": "desc"
}
```

Supported query parameters:

- `GET /v1/runs`: `status`, `limit`, `offset`, `sort_by`, `sort_order`
- `GET /v1/runs/{run_id}/checkpoints`: `type`, `limit`, `offset`, `sort_by`, `sort_order`
- `GET /v1/runs/{run_id}/trace`: `event_type`, `limit`, `offset`, `sort_by`, `sort_order`
- `GET /v1/human-tasks`: `run_id`, `workflow_kind`, `limit`, `offset`, `sort_by`, `sort_order`

`sort_order` is `asc` or `desc`; `limit` is capped at `1000`.

Environment variables

Source: *docs/user/environment.md*

Every environment variable read by `kneo-serv`, grouped by surface area. Defaults shown are the values used when the variable is unset.

For deployment templates that exercise these in context, see `tutorial_postgres_deployment.md` and `deploy/*.env.example`. The full HTTP contract that uses these knobs lives in `service_api.md`.

Project

Variable	Default	Purpose
`KNEO_PROJECT_CONFIG`	auto-discovery	Explicit path to `.kneo/config.yaml`.
`KNEO_ENV`	`dev`	Default project environment when `--env` is not provided.
`KNEO_SERV_SPEC_SIGNING_KEY`	unset	HMAC signing key used by `kneo spec bundle sign` and `verify`.

Service auth

Variable	Default	Purpose
`KNEO_SERV_AUTH_ENABLED`	enabled when API keys are configured	Require API keys on protected HTTP routes.
`KNEO_SERV_API_KEYS`	empty	Semicolon-separated `name:key:role_or_scope[,role_or_scope]` entries.
`KNEO_SERV_ADMIN_API_KEY`	empty	Admin API key with all scopes.
`KNEO_SERV_API_KEY`	empty	Client API key used by `ServiceClient` and one-off CLI calls.

Variable	Default	Purpose
`KNEO_SERV_IDEMPOTENCY_KEY`	empty	Stable idempotency key for retry-safe service `POST` calls.

Persistence

Variable	Default	Purpose
`KNEO_SERV_DATABASE_URL`	empty	PostgreSQL DSN. When set, service stores use PostgreSQL instead of SQLite plus file continuations. Requires `kneo-serv[postgres]` or `kneo-serv[deploy]`.
`KNEO_SERV_PROFILES_PATH`	`~/.kneo_serv/profiles.json`	CLI service-profile store location.

Service limits

Variable	Default	Purpose
`KNEO_SERV_MAX_BODY_BYTES`	`1048576`	Maximum HTTP request body size.
`KNEO_SERV_MAX_INPUT_CHARS`	`20000`	Maximum run input size.
`KNEO_SERV_MAX_HUMAN_CONTENT_CHARS`	`20000`	Maximum human response content size.
`KNEO_SERV_MAX_INLINE_SPEC_BYTES`	`262144`	Maximum inline spec payload size.
`KNEO_SERV_MAX_OVERRIDES_BYTES`	`65536`	Maximum spec override payload size.
`KNEO_SERV_MAX_METADATA_BYTES`	`32768`	Maximum request metadata payload size.
`KNEO_SERV_MAX_LIST_ITEMS`	`100`	Maximum requested list page size.

Variable	Default	Purpose
`KNEO_SERV_MAX_PATH_CHARS`	`4096`	Maximum path field size.

Runtime reliability

Variable	Default	Purpose
`KNEO_SERV_PROVIDER_RETRIES`	`0`	Provider/runtime retry count.
`KNEO_SERV_PROVIDER_RETRY_BACKOFF_SECONDS`	`0.25`	Provider/runtime retry backoff.
`KNEO_SERV_PROVIDER_TIMEOUT_SECONDS`	unset	Provider/runtime timeout.
`KNEO_SERV_MCP_RETRIES`	`0`	MCP tool retry count.
`KNEO_SERV_MCP_RETRY_BACKOFF_SECONDS`	`0.25`	MCP retry backoff.
`KNEO_SERV_MCP_TIMEOUT_SECONDS`	unset	MCP tool timeout.
`KNEO_SERV_REQUIRE_PROVIDER_SECRETS`	`false`	Fail native provider setup when provider secrets are absent.
`KNEO_SERV_RUN_PROVIDER_INTEGRATION`	`false`	Enable opt-in real provider integration tests. Requires provider credentials such as `OPENAI_API_KEY`.
`KNEO_SERV_PROVIDER_TEST_MODEL`	`gpt-4o-mini`	OpenAI model used by the opt-in provider integration smoke test.
`KNEO_SERV_RUN_POSTGRES_INTEGRATION`	`false`	Enable opt-in PostgreSQL persistence smoke tests. Requires `KNEO_SERV_DATABASE_URL` and the `postgres` extra.

CLI client

Variable	Default	Purpose
`KNEO_SERV_CLIENT_TIMEOUT`	`120`	HTTP client timeout in seconds.
`KNEO_SERV_CLIENT_RETRIES`	`2`	Service-client retry count for transient failures.
`KNEO_SERV_CLIENT_RETRY_BACKOFF_SECONDS`	`0.25`	Service-client retry backoff.

Observability

Variable	Default	Purpose
`KNEO_SERV_REQUEST_LOGS`	`true`	Enable structured request logs.
`KNEO_SERV_LOG_LEVEL`	`INFO`	Structured logging level.
`KNEO_SERV_OTEL_ENABLED`	`false`	Attach `kneo_agent.observability.OpenTelemetryMiddleware` to SDK-backed agents. Requires `kneo-serv[telemetry]` or `kneo-serv[deploy]`.
`KNEO_SERV_OTEL_RECORD_ARGUMENTS`	`false`	Record tool-call arguments in SDK OpenTelemetry spans. Keep disabled when arguments may contain PII or secrets.
`KNEO_SERV_OTEL_RECORD_RESULTS`	`false`	Record tool results in SDK OpenTelemetry spans. Keep disabled for large or sensitive results.
`KNEO_SERV_HEALTH_PROVIDERS`	empty	Comma-separated provider secret names to include in readiness checks.

Variable	Default	Purpose
`KNEO_SERV_HEALTH_MCP_SECRETS`	empty	Comma-separated MCP secret names to include in readiness checks.

Retention

Variable	Default	Purpose
`KNEO_SERV_RETENTION_RUNS_DAYS`	unset	Delete old runs with terminal statuses.
`KNEO_SERV_RETENTION_CHECKPOINTS_DAYS`	unset	Delete old checkpoints.
`KNEO_SERV_RETENTION_QUEUE_DAYS`	unset	Delete old completed or failed queue records.
`KNEO_SERV_RETENTION_CONTINUATIONS_DAYS`	unset	Delete old continuations.
`KNEO_SERV_RETENTION_ARTIFACTS_DAYS`	unset	Delete old artifact files.
`KNEO_SERV_RETENTION_LOGS_DAYS`	unset	Delete old log files.

Retention windows can also be set per-project in `.kneo/config.yaml` under a top-level `retention:` block — see project_config.md § Retention. Env-var precedence: an env var, if set, overrides the project-config value for that field. Use the env-var path as the operator escape hatch when the host needs to deviate from per-project defaults.

Checkpoint storage

Variable	Default	Purpose
`KNEO_SERV_CHECKPOINT_COMPRESS_BYTES`	`65536`	Compress checkpoint payloads at or above this size.
`KNEO_SERV_CHECKPOINT_MAX_BYTES`	`1048576`	Hard checkpoint payload limit after compression.

Variable	Default	Purpose
`KNEO_SERV_CHECKPOINT_PREVIEW_CHARS`	`1200`	Preview length for oversized checkpoint values.
`KNEO_SERV_CHECKPOINT_MAX_LIST_ITEMS`	`20`	Maximum list items retained in previews.
`KNEO_SERV_CHECKPOINT_MAX_DICT_ITEMS`	`50`	Maximum dict items retained in previews.

Examples

Source: *docs/user/examples.md*

The repository ships a set of runnable specs and supporting Python helpers under `examples/`. Use them to validate a local install, exercise the CLI, or as starting points for your own specs.

These specs are non-production placeholders — the `provider` / `model` fields point at common defaults and should be retargeted before any real use.

Spec files

`research_agent.yaml`

A single-agent research pipeline using a plan-act strategy with two tools (`web_search`, `webpage_reader`) and a sequential workflow that retrieves, analyzes, and summarizes.

```
kneo spec validate examples/research_agent.yaml
kneo spec compile examples/research_agent.yaml
kneo run --input "Analyze Nvidia AI business" --target workflow examples/research_agent.yaml
```

Three environment overlays show the overlay system in action:

- `research_agent.dev.yaml` — faster model, fewer iterations, tracing enabled.
- `research_agent.staging.yaml` — larger model, mid iterations, tracing enabled.
- `research_agent.prod.yaml` — conservative temperature, more iterations, step checkpointing, tracing.

```
kneo spec validate examples/research_agent.yaml --env prod
kneo spec bundle sign examples/research_agent.yaml \
  --output bundles/research_agent.prod.json --approved-by release-manager --env prod
```

`graph_review_workflow.yaml`

A graph workflow with conditional edges: retrieve → analyze → review → revise → finalize, where the review step routes to `revise` or `finalize` based on output. Demonstrates `GraphWorkflow`, conditional edges, and component agent references.

```
kneo spec compile examples/graph_review_workflow.yaml
```

`concurrent_review_workflow.yaml`

A concurrent workflow that fans out a single input to three reviewers (security, accessibility, performance) running in parallel. The platform collects each participant's response and returns the combined result. Demonstrates `ConcurrentWorkflow`, `participants:`-style declaration, and the fan-out / fan-in pattern.

```
kneo spec compile examples/concurrent_review_workflow.yaml
kneo run --input "Review the auth middleware refactor" \
  --target workflow examples/concurrent_review_workflow.yaml
```

`group_chat_workflow.yaml`

A group-chat workflow with three personas (proponent, skeptic, pragmatist) debating a design proposal over two rounds. Each round visits all participants in declaration order, so `rounds: 2` produces six total turns. Demonstrates `GroupChatWorkflow`, the `rounds:` knob, and ordered participant declaration for structured back-and-forth.

```
kneo spec compile examples/group_chat_workflow.yaml
kneo run --input "Should we adopt gRPC for service-to-service calls?" \
  --target workflow examples/group_chat_workflow.yaml
```

`human_approval_workflow.yaml`

Sequential workflow with a human-in-the-loop step (`kind: human`) between draft and publish. Use it to exercise the pause/resume API.

```
kneo run --input "hello" --target workflow --json examples/human_approval_workflow.yaml
# Output includes a continuation_id; resume with:
kneo human resume <continuation_id> --request-id <request_id> --approve
```

The deeper human-task documentation is in `design.md § 8.5` and the HTTP API's `/human-tasks/...` endpoints.

Timeout branches

The `approval-reviewer` block in this spec declares a 24-hour timeout with `on_timeout: escalate`. Two other literals are available; the platform dispatches per `human_in_the_loop.md § 9`:

`on_timeout`	Lifecycle	Audit event(s)	Continuation
`fail` (default)	Run transitions to `expired`	`human.expired`	deleted

`on_timeout`	Lifecycle	Audit event(s)	Continuation
`continue`	Synthesizes an auto-approved `HumanResponse` and resumes the workflow	`human.continued`; `human.continue_failed` on resume error	deleted on success or failure
`escalate`	Run stays `blocked`; `escalated_at` stamped on the continuation; subsequent prune calls skip it (escalation fires once)	`human.escalated`	preserved (operator reassigns + resumes via the normal `/continuations/{id}/resume` path)

Auto-routing of an escalated task to a different reviewer is up to the operator's external workflow — the platform marks + audits the task as escalated, it does not auto-reassign. Operators call `PlatformManager.prune_expired_human_tasks()` (cron, scheduled run, manual sweep — same pattern as `prune_retention()`; there is no built-in scheduler) to dispatch the timeout branch.

`run_with_timeout.py`

Worked walkthrough of the **run-level** timeout — distinct from the human-task timeout above. `start_run_from_spec(..., timeout_seconds=N)` schedules a run with a wall-clock deadline written to `RunState.deadline_at`; `prune_timed_out_runs()` is the operator-callable sweep that force-cancels every `running` or `blocked` run past its deadline, transitions the state to `timed_out`, deletes any associated continuation, and emits a `run.timed_out` audit event.

```
python examples/run_with_timeout.py
```

Whichever timeout fires first wins. The dispatch matrix between run-level and human-task deadlines is documented in `human_in_the_loop.md § 9` under *Run-level timeouts vs. human-task timeouts*.

`smoke_human_workflow.yaml`

Lightweight human-in-the-loop spec that uses the `dummy` provider so it runs without real provider credentials. Used by the deployment smoke script:

```
python scripts/deployment_smoke.py --base-url http://127.0.0.1:8000
```

See `deployment_smoke.md` for the full smoke sequence.

Project config

`project_config.yaml`

Reference `.kneo/config.yaml` content showing project metadata, service defaults, runtime defaults, and per-environment policy enforcement overlays. Copy into `.kneo/config.yaml` to bootstrap a new project, or use:

```
kneo config init --name research-agent-demo
kneo config show
```

The schema and overlay rules are in `project_config.md` .

Helper Python

`app_functions.py`

Stub implementations for the tools and helpers referenced by `research_agent.yaml` (`compress_history` , `web_search` , `webpage_reader` , `summarize`). Ship-quality replacements would call real services; these just return formatted strings so the agent loop has something to do.

`human_functions.py`

Stub `draft_report` and `publish_report` used by the human-approval workflow. Same pattern as `app_functions.py` .

Adapting an example

1. Copy a spec into your project, e.g. `cp examples/research_agent.yaml my_agent.yaml` .
2. Replace the `model.provider` / `model.name` with your provider, and add the corresponding env-var reference under your project secrets.
3. Replace the tools with real ones — either Python functions registered through `ToolRegistry` or MCP servers (see `extending.md`).
4. Validate against your target environment: `bash kneo spec validate my_agent.yaml --env prod`
5. Compile to confirm the workflow builds: `bash kneo spec compile my_agent.yaml`
6. Run locally before deploying: `bash kneo run --input "<prompt>" --target workflow my_agent.yaml`

For deployment to a service, see `deployment.md` .

Project `.kneo/` config

Source: `docs/user/project_config.md`

Each project keeps a `.kneo/` directory for local service configuration, generated artifacts, and logs. This page covers the contents and the overlay/policy story; for the runtime variables read from the environment (rather than from project config), see environment.md.

```
.kneo/
  config.yaml
  README.md
  artifacts/.gitkeep
  logs/.gitkeep
```

The `.kneo/config.yaml` demonstrates:

- default project name and owner
- service URL
- local state/artifact/log paths
- default spec
- environment overlays
- runtime defaults
- model defaults
- policy defaults
- environment-variable secret references
- retention windows (per-project)

Environment-specific policy enforcement can be configured under `environments.<name>.policy_enforcement`:

```
environments:
  dev:
    policy_enforcement:
      enabled: false
  staging:
    policy_enforcement:
      require_tool_permissions: true
      blocked_diagnostic_codes: [E_UNSAFE_TOOL_IMPORT, E_UNSAFE_FUNCTION_IMPORT]
  prod:
    policy_enforcement:
      require_tool_permissions: true
      deny_unrestricted_tools: true
      require_human_review: true
```

```
      require_guardrails: true
      blocked_diagnostic_codes: [E_UNSAFE_TOOL_IMPORT, E_UNSAFE_FUNCTION_IMPORT]
```

Policy enforcement runs after spec overlays and project defaults are applied. `kneo spec validate --env prod`, `kneo spec compile --env prod`, `kneo spec policy-report --env prod`, and `kneo run --env prod` all honor the resolved environment policy.

Retention

Retention windows for runs, checkpoints, queue records, continuations, artifacts, and log files live in a top-level `retention:` block. Each field is a count of days to keep; unset fields disable pruning for that category. Values must be zero or greater.

```
retention:
  runs_days: 30
  checkpoints_days: 14
  queue_days: 7
  continuations_days: 21
  artifacts_days: 60
  logs_days: 90
```

The same six retention fields can be overridden per-host via env vars (`KNEO_SERV_RETENTION_RUNS_DAYS` and friends; see environment.md § Retention). **Precedence is env var > project config > unset.** Set the project-config field for the per-project default; set the env var to deviate on a specific host (staging vs. prod, etc.) without editing the committed `.kneo/config.yaml`.

The retention values feed `kneo_serv.maintenance.retention.RetentionPolicy.from_project_and_env(config.retention)`, which the operator can pass to `PlatformManager.prune_retention(policy=...)` on whatever cadence makes sense for the deployment (cron, scheduled workflow, manual operator action).