IntentGate Deployment Runbook

Pro v2 — Platform & Security Operations Guide

NetGnarus · IntentGate

May 2026

Contents

Overview

IntentGate is a self-hosted authorization gateway for AI agents. It sits between your agents and the tool servers they call (databases, internal APIs, SaaS, the file system, anything reachable over HTTP/MCP) and answers one question per call:

> *Should this agent be allowed to perform this action right now, given who's asking, what it's trying to do, and what your policy says?*

Every authorization decision is signed, audited, and cryptographically linked into a tamper-evident chain. Operators are gated by SSO + RBAC + step-up MFA. Privileged operations require just-in-time elevation with a documented reason, a different admin's approval, and automatic expiry. Compliance evidence — SOC 2, ISO 27001, GDPR Article 30, AI Act Article 12 — exports in one click.

This document is the operational runbook for platform engineering and security operations teams deploying and running IntentGate inside your environment.

The four-check pipeline

Each call hitting the gateway runs through four checks in order. The first failure short-circuits the rest; success forwards the call to the upstream tool server.

1. **Capability** — the caller presents a signed capability token. Did NetGnarus issue it? Is it within TTL? Has it been revoked? Does its scope cover the requested tool?
2. **Intent** — what does the agent actually want to do? An optional extractor service classifies the prompt into a one-line intent summary, evaluated against the policy.
3. **Policy** — a Rego (OPA) policy you write decides allow / block / escalate-to-human, with a human-readable reason and an optional `requires_step_up` advisory.
4. **Budget** — per-agent / per-tenant rate and cost ceilings. Refusing the 101st call this hour from an agent budgeted for 100.

A decision (allow, block, or escalate) becomes an audit event with the policy reason, the agent identity, the token JTI, the tenant, and the operator elevation id if applicable. The event is hashed into the per-tenant chain and fanned out to your SIEM and notification channels.

Open core boundary

Component	License	Purpose
intentgate-gateway	Apache 2.0	The authorization service

Component	License	Purpose
intentgate-extractor	Apache 2.0	Optional intent classifier
intentgate-sdk-python	Apache 2.0	Agent-side SDK
intentgate-sdk-typescript	Apache 2.0	Agent-side SDK (Node)
intentgate-helm	Apache 2.0	Kubernetes packaging
intentgate-console	Apache 2.0	Basic operator UI (token lifecycle)
intentgate-console-pro	**Commercial**	**OIDC SSO, RBAC, SCIM, TOTP, notifications, JIT elevation, audit verify, audit export, AI-assisted policy authoring**

The commercial Pro console is a drop-in replacement for the OSS console — same env vars, plus a license key and your IdP credentials.

Components & repositories

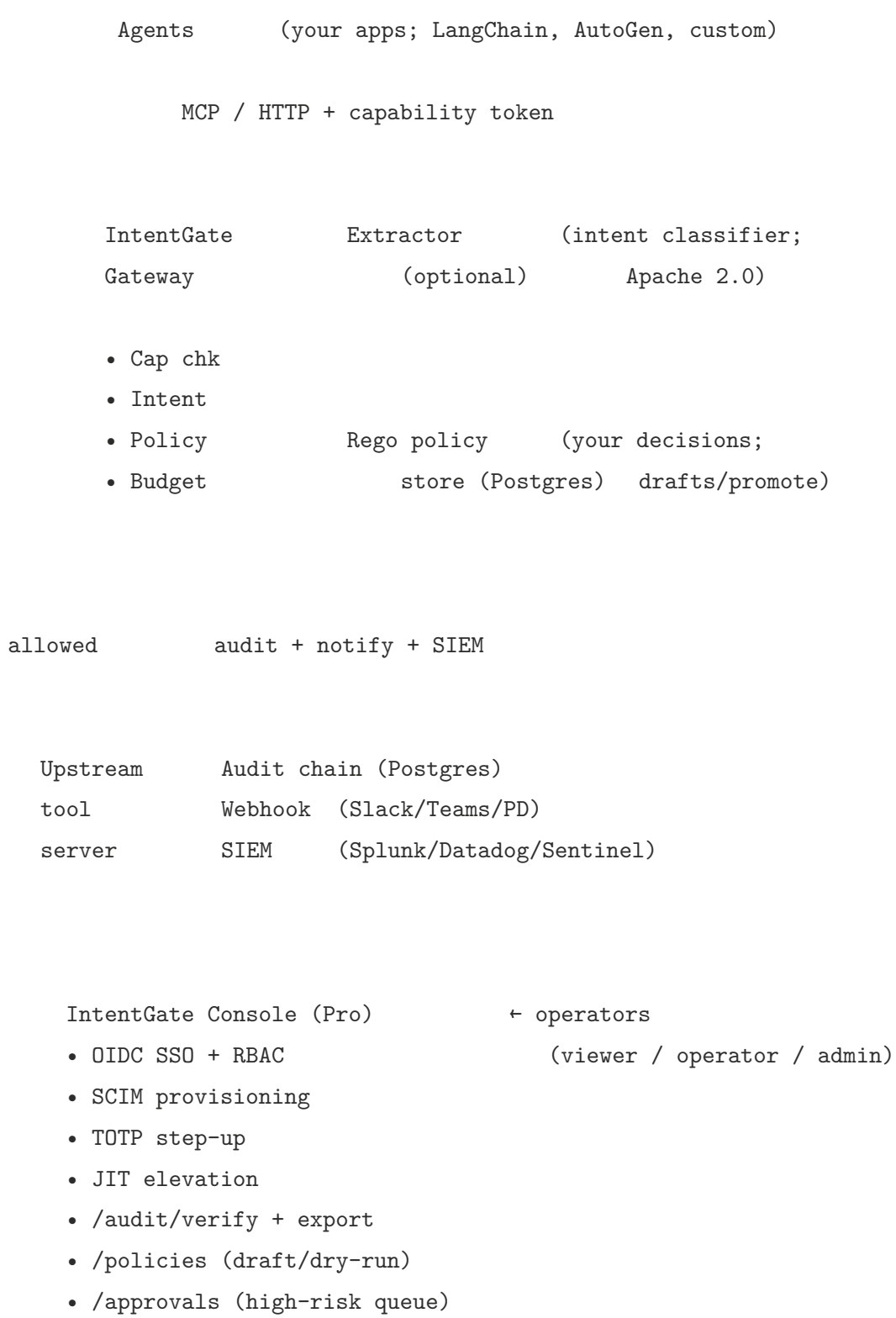
Every component is shipped as a separate repo on GitHub. Platform engineers can read, fork, audit, and contribute to the OSS components; the Pro console source is available to commercial customers under a source-available license.

Component	Language	Repository	Role
intentgate-gateway	Go 1.25	github.com/NetGnarus/intentgate-gateway	The authorization service. Single static binary. Stateless past Postgres. Ships the `igctl` CLI for ops.

Component	Language	Repository	Role
intentgate-extractor	Python 3.11+	github.com/NetGnarus/intentgate-extractor	Optional intent classifier sidecar. HTTP/gRPC service. Pluggable backend (stub for tests, Anthropic Haiku for prod).
intentgate-sdk-python	Python 3.10+	github.com/NetGnarus/intentgate-sdk-python	Agent-side library. Token attenuation, capability minting, MCP shim.
intentgate-sdk-typescript	TypeScript / Node 20+	github.com/NetGnarus/intentgate-sdk-typescript	Same surface as the Python SDK, for Node-based agents.
intentgate-helm	YAML	github.com/NetGnarus/intentgate-helm	Kubernetes packaging. Gateway, extractor, Postgres, optional console, optional Caddy/Ingress.
intentgate-console	TypeScript / Next.js 16	github.com/NetGnarus/intentgate-console	Basic OSS operator UI — token lifecycle and audit browse. No SSO, single shared admin user.
intentgate-console-pro	TypeScript / Next.js 16	github.com/NetGnarus/intentgate-console-pro	Commercial operator console. OIDC SSO, RBAC, SCIM, TOTP, notifications, JIT elevation, audit verify, audit export, AI-assisted policy authoring.

Everything except `intentgate-console-pro` is Apache 2.0 with a NetGnarus governance commitment never to relicense. The Pro console source is shipped to commercial customers; they can read and modify it for internal use under the commercial license.

Architecture



The gateway is the only component on the agent's hot path. The console talks to the gateway's

admin API; it does not sit between agents and tools. SIEM forwarders, webhook receivers, and audit persistence run as fan-outs from the gateway's audit pipeline and never block the authorization decision.

Prerequisites

Compute

A Kubernetes cluster (EKS, GKE, AKS, OpenShift, Rancher, k3s) is the recommended platform. Docker Compose is supported for development and proof-of-concept. For HA production, plan two or more gateway replicas behind an internal load balancer; the gateway is stateless past the database.

Recommended sizing for the pilot (single tenant, 1,000 RPS):

Workload	CPU request	Memory request	Replicas
gateway	250 m	256 Mi	2
extractor (optional)	500 m	512 Mi	1
console-pro	250 m	256 Mi	1

The gateway has been benchmarked at 2,000 RPS at p99 = 1.5 ms on a single replica on a developer laptop. Scale horizontally for headroom.

Database

A managed PostgreSQL instance (RDS, Cloud SQL, Azure DB, Aiven, Supabase) running 14 or newer is required for production. The gateway persists revocations, audit events, the per-tenant chain heads, the approvals queue, and (optionally) Rego policy drafts. The Pro console additionally persists SCIM records, elevation requests, and encrypted notification-channel configs.

A single database is sufficient; the schemas don't overlap. Two databases (one for the gateway, one for the console) is acceptable if your operations team prefers per-service isolation.

Schema migrations apply automatically at startup. They're idempotent — restarting against an already-migrated database is a no-op.

Network

The gateway exposes one inbound HTTP port (default 8080) for agent traffic and admin API. TLS termination should happen at your ingress controller (nginx-ingress, Traefik, AWS ALB) or at the gateway directly when running outside Kubernetes. The console exposes one inbound HTTP port (default 3000).

DNS: pick stable hostnames for both — agents pin to the gateway hostname inside their secret store, and operators bookmark the console. Common patterns are `intentgate.example.com` for

the gateway and `intentgate-console.example.com` for the console.

Outbound from the gateway: your upstream tool server, your Postgres, optional SIEM HEC/HTTP endpoints, optional webhook URL (if using the Pro console's notification fan-out, this points at the console's webhook receiver inside your network).

Identity (Pro only)

The Pro console authenticates operators against any OIDC-compliant identity provider you already run — Okta, Microsoft Entra, Auth0, Google Workspace, Keycloak, Ping, AWS Cognito. The console reads the standard OIDC profile plus one configurable JWT claim (default `groups`) to resolve operator role.

For automatic deprovisioning, the console accepts SCIM 2.0 push from your IdP. SCIM is optional — without it, the OIDC sign-in flow remains the authoritative gate and an off-boarded user simply stops being able to sign in.

Observability (recommended)

A Prometheus scraper (or compatible: Grafana Agent, Datadog Agent, OpenTelemetry Collector) is recommended for the gateway's `/metrics` endpoint. The gateway emits per-check decision counts, per-stage latency histograms, upstream-call duration, and revocation lookup metrics — disciplined cardinality (no per-tool, per-agent, or per-JTI labels) so the metrics scale with deployments, not with traffic.

OpenTelemetry tracing is supported via the standard `OTEL_EXPORTER_OTLP_ENDPOINT` env var. Tail-based sampling at your OTel Collector is recommended; the gateway honors `OTEL_TRACES_SAMPLER` for head-based sampling.

Notifications (Pro only)

For the Pro console's notification fan-out, you need at least one of:

- A Slack webhook URL with permission to post to your operations channel.
- A Microsoft Teams Incoming Webhook URL for the operations channel.
- A PagerDuty integration key for the on-call rotation that handles AI agent incidents.

You can configure multiple destinations and scope each per tenant.

Installation

Helm (recommended for production)

```
# 1. Generate secrets - once, outside the cluster, kept in your
#    secret manager (Vault, AWS Secrets Manager, sealed-secrets).
ADMIN_TOKEN=$(openssl rand -hex 32)
MASTER_KEY=$(openssl rand -hex 32)
LICENSE_KEY=<obtain-from-NetGnarus>

# 2. Create the namespace + a Secret with the connection string.
kubectl create namespace intentgate
kubectl -n intentgate create secret generic intentgate-db \
  --from-literal=url=
  'postgres://intentgate:<password>@db.internal/intentgate?sslmode=require'

# 3. Install the chart. Values are inline for clarity; in production
#    use a values.yaml + GitOps (Flux/Argo).
helm install intentgate \
  oci://ghcr.io/netgnarus/charts/intentgate \
  --namespace intentgate \
  --set gateway.image.tag=1.6.0 \
  --set gateway.adminToken.value=$ADMIN_TOKEN \
  --set gateway.masterKey.value=$MASTER_KEY \
  --set postgres.existingSecret=intentgate-db \
  --set audit.persist=true \
  --set metrics.enabled=true

# 4. Verify health.
kubectl -n intentgate port-forward svc/intentgate-gateway 8080:8080 &
curl http://localhost:8080/healthz
# {"ok":true,"version":"1.6.0"}
```

The chart deploys the gateway, the extractor, and the Postgres-backed audit/revocation/approvals stores. Ingress is left to the operator — see the chart's `values.yaml` for `ingress.enabled` and TLS configuration.

Docker Compose (proof-of-concept)

```yaml
# compose.yml - single-host, single-replica. Not for production.
services:
  postgres:
    image: postgres:16
    environment:
      POSTGRES_USER: intentgate
      POSTGRES_PASSWORD: dev
      POSTGRES_DB: intentgate
    ports: ["5432:5432"]

  gateway:
    image: ghcr.io/netgnarus/intentgate-gateway:1.6.0
    depends_on: [postgres]
    environment:
      INTENTGATE_ADMIN_TOKEN: dev-admin
      INTENTGATE_MASTER_KEY: dev-master-key
      INTENTGATE_POSTGRES_URL:
postgres://intentgate:dev@postgres/intentgate?sslmode=disable
      INTENTGATE_AUDIT_PERSIST: "true"
      INTENTGATE_METRICS_ENABLED: "true"
    ports: ["8080:8080", "9090:9090"]

  console-pro:
    image: ghcr.io/netgnarus/intentgate-console-pro:v0.5.0
    depends_on: [gateway]
    environment:
      INTENTGATE_GATEWAY_URL: http://gateway:8080
      INTENTGATE_ADMIN_TOKEN: dev-admin
      INTENTGATE_POSTGRES_URL: postgres://intentgate:dev@postgres/intentgate
      INTENTGATE_LICENSE_KEY: <obtain-from-NetGnarus>
      AUTH_PROVIDER: mock          # dev only; refuses to start in production
      AUTH_SECRET: dev-cookie-secret
    ports: ["3000:3000"]
```

```
docker compose up -d
open http://localhost:3000   # admin@local / dev
```

Pro console (separate workload)

The Pro console is a separate workload from the gateway. Deploy it alongside, or in a different namespace, depending on your network policy. It needs:

- Network access to the gateway's admin API (`INTENTGATE_GATEWAY_URL`).
- The same `INTENTGATE_ADMIN_TOKEN` the gateway is configured with (kept server-side; never sent to browsers).
- An `INTENTGATE_LICENSE_KEY` issued by NetGnarus.
- OIDC issuer/client/secret for SSO (`AUTH_OIDC_*`).
- Postgres for SCIM, elevation, and notification-channel persistence (may share the gateway's DB).
- `INTENTGATE_TOTP_ENCRYPTION_KEY` (32 random bytes, hex-encoded) — used to AES-256-GCM-encrypt TOTP secrets and notification webhook URLs at rest.
- `AUTH_SECRET` — HMAC secret for NextAuth session cookies (32 random bytes).

Day-1 Configuration

The install puts a running gateway in your environment. Day-1 turns it into an authorization control point.

Concept — how agent identity works in IntentGate

Before the first mint, the mental model:

An agent is identified by a signed capability token. There is no separate agent registry. The gateway has no `agents` table. There is no "register this agent first, then issue tokens" flow. The act of minting an agent's first token via `POST /v1/admin/mint` *is* the registration, from the gateway's perspective. Whoever holds the token is the agent named by its `subject` field, scoped to its `tenant`, restricted to its `tools` allow-list, capped at its `max_calls` budget, until its `expires_at` time.

This model has practical consequences for day-1 and day-2 operations:

- **Discovery — "what agents do we have?"** Query the audit log:

  ```
  SELECT DISTINCT tenant, subject
  FROM audit_events
  WHERE ts > now() - interval '90 days'
  ORDER BY tenant, subject;
  ```

  The audit log is the source of truth. The Pro console renders this query as a list under */tokens* and lets you click through to each agent's recent decisions.

- **Naming conventions.** The `subject` field is a free-form string — you choose. We recommend `service-purpose-version`, e.g. `agent-trade-bot-v2`, `support-triage-experimental`, `finance-copilot-prod`. Avoid GUIDs (illegible in audit logs) and avoid PII (audit logs are forwarded to your SIEM).

- **Multi-tenant scoping.** The same `subject` value can exist in multiple tenants and is treated as a distinct agent in each. The pair `(tenant, subject)` is the unique key. Off-tenant calls fail capability check immediately.

- **Disablement / revocation.** Add the token's JTI (returned by mint, included in every audit row) to the gateway's revocation list. The revocation table is checked on every capability evaluation; revoked JTIs fail in microseconds.

- **Delegation.** Any token-holder can attenuate the token — using the Python or TypeScript SDK — to mint a narrower child token with the same subject (or a new subject) and a subset of the parent's tools, budget, and TTL. Children cannot widen what the parent grants. The

audit chain records both the child's JTI and the chain root's JTI for traceability.

- **Protect the admin token.** Whoever holds the `INTENTGATE_ADMIN_TOKEN` bearer can mint capability tokens for any subject in any tenant. Treat it like a root credential: secret-manager-backed, rotated when an operator leaves, never embedded in code, never reachable from the public internet in production. The admin API should be exposed only on a private network or behind your operator console (Pro).

Why no registry at the gateway

The most common follow-up question: *"Why doesn't the gateway have an agents table? Surely we want to register agents before they can act."*

The runtime authorization layer is best when it's stateless on the hot path. Every `tools/call` verifies the token's HMAC + caveats and runs the four checks — sub-millisecond, no database lookup for identity. Adding an "is this agent registered?" check on every call would:

1. **Add latency** — a registry lookup on the hot path. For a deployment doing 10k authorizations/sec, that's an extra 10k Postgres queries per second the gateway is forced to make.
2. **Introduce a sync-drift failure mode** — what wins when the token says active but the registry says revoked? When the token's tools list has `send_email` but the registry says the agent's allowed tools are only `read_invoice`? The registry-vs-token contradiction has no clean answer; every implementation has corner cases that bite at the worst time.
3. **Break the capability model** — the token *is* the authority. If the gateway second-guesses the token against a separate authority list, you've reinvented RBAC with a token bolted on, and lost the property that makes capability-based authorization clean: *whoever holds it can do exactly what it authorizes; nothing else does.*

So the gateway, by design, never reaches into a registry. The token is sufficient.

Where agent registration does belong: in Console-Pro, as a governance layer above the gateway. The Pro tier ships an agents table that captures display name, owner, purpose, risk classification, and lifecycle state (draft → pending review → active → retired), with optional approval workflows before a token is minted. The gateway hot path is unchanged; the console adds the operator experience that enterprise teams expect — agent inventory, ownership, approval flows, compliance export.

If you're running open-core only (no console-pro), you don't have a registry. Your inventory is the `SELECT DISTINCT (tenant, subject) FROM audit_events` query above. That's enough for many production deployments — particularly ones where agent provisioning is automated through your own platform and the gateway is purely the enforcement point.

With that model in mind:

Step 1 — Mint a capability token for your first agent

```
curl -sX POST http://intentgate.internal:8080/v1/admin/mint \
  -H "Authorization: Bearer $ADMIN_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "subject":      "agent-trade-bot",
    "tenant":       "default",
    "ttl_seconds":  86400,
    "tools":        ["read_invoice", "send_email_internal"],
    "max_calls":    1000
  }'
```

The response carries `token`, `jti`, `subject`, and `expires_at`. **The token string is shown ONCE.** Copy it into your agent's secret store immediately (Vault, AWS Secrets Manager, K8s Secret). The gateway does not persist the token; it only stores the JTI for revocation lookup.

Distribute the token to your agent's environment and point the agent's MCP server URL at the gateway:

```
INTENTGATE_TOKEN=eyJhbGc...   # in the agent's secret store
MCP_SERVER_URL=https://intentgate.internal/v1/mcp
```

Step 2 — Wire the upstream tool server

The gateway forwards allowed calls to whatever `INTENTGATE_UPSTREAM_URL` points at. This is typically your existing MCP tool registry, an internal API, or a third-party service that already speaks MCP.

```
INTENTGATE_UPSTREAM_URL=http://internal-tools.svc/mcp
INTENTGATE_UPSTREAM_TIMEOUT_MS=10000
```

A 5xx from the upstream surfaces in the audit log as `upstream_status` so SOC analysts can distinguish authorization blocks from operational failures.

Step 3 — Write the first Rego policy

The default policy denies everything. In the Pro console, navigate to `/policies` and author a baseline. Minimum viable policy:

```
package intentgate.policy
```

```
default decision := {
  "allow":  false,
  "reason": "denied by default"
}

# Allow the read_invoice tool unconditionally.
decision := {"allow": true, "reason": "read-only invoice access"} if {
  input.tool == "read_invoice"
}

# Allow internal email under a budget, escalate above it.
decision := {"allow": true, "reason": "internal email under daily quota"} if {
  input.tool == "send_email_internal"
  input.agent.calls_today_for_tool < 50
}

decision := {
  "allow":            false,
  "escalate":         true,
  "reason":           "internal email over daily quota - review",
  "requires_step_up": true
} if {
  input.tool == "send_email_internal"
  input.agent.calls_today_for_tool >= 50
}
```

Use the **dry-run** button on `/policies` to replay the candidate against the last 24 hours of traffic before promoting. The diff panel shows every decision that would change (allow→block, allow→escalate, etc.). Promote to active only when the diff matches your intent.

Step 4 — Configure SSO (Pro)

On the Pro console, set:

```
AUTH_PROVIDER=oidc
AUTH_OIDC_ISSUER=https://your-idp.example.com
AUTH_OIDC_CLIENT_ID=<from-IdP>
AUTH_OIDC_CLIENT_SECRET=<from-IdP>
AUTH_ROLE_CLAIM=groups
```

```
AUTH_ROLE_MAPPING=intentgate-admins:admin,intentgate-ops:operator,intentgate-viewers:viewer
```

Roles resolve from the JWT claim at sign-in. Operators with no mapped group are denied by default. Set `INTENTGATE_DEFAULT_ROLE=viewer` if you want unmapped sign-ins to land as read-only — usually you don't.

Visit `/settings/access` after the first SSO sign-in to confirm your role and the resolver source.

Step 5 — Enable SCIM (Pro)

Push SCIM from your IdP to:

```
POST https://intentgate-console.example.com/scim/v2/...
Authorization: Bearer <SCIM_BEARER_TOKEN>
```

Set `INTENTGATE_SCIM_BEARER_TOKEN` on the console to match. The IdP becomes the authoritative source for user existence and active-state; off-boarding flows automatically.

SCIM records can carry custom `role` and `tenant` fields that override the OIDC group resolver — useful when one user needs a different role than their group implies.

Step 6 — Wire notifications (Pro)

In the console, navigate to `/settings/notifications` and add channels. The console encrypts URLs and integration keys at rest with `INTENTGATE_TOTP_ENCRYPTION_KEY` (AES-256-GCM).

For each channel, click **Send test** to fire a synthetic event end-to-end. Click **Preview** to inspect the exact per-kind JSON payload (Slack Block Kit, Teams Adaptive Card, PagerDuty Events v2) before saving.

Step 7 — Wire SIEM forwarding

The gateway can fan audit events to Splunk HEC, Datadog Logs, or Microsoft Sentinel side-by-side. Configure any combination:

```
INTENTGATE_SIEM_SPLUNK_HEC_URL=https://splunk.example:8088/services/collector
INTENTGATE_SIEM_SPLUNK_TOKEN=<HEC-token>
INTENTGATE_SIEM_DATADOG_API_KEY=<dd-api-key>
INTENTGATE_SIEM_SENTINEL_WORKSPACE_ID=...
INTENTGATE_SIEM_SENTINEL_SHARED_KEY=...
```

The forwarders share the gateway's audit fan-out: a slow or failing SIEM endpoint produces a counter increment and a logged warning, never a block on the authorization decision.

Day-2 Operations

Token lifecycle

Mint — issue a new token via `POST /v1/admin/mint` (Pro console: `/tokens` page, "Mint" button). Tokens are signed under the master key, scope is encoded as caveats, and the encoded form is shown once.

Revoke — `POST /v1/admin/revoke` with `{"jti":"<token-id>","reason":"<context>"}`. Idempotent. The revocation is recorded in the audit log with operator identity and flows to every gateway replica through the shared Postgres revocation store.

Attenuate — agents can produce constrained sub-tokens client-side from a root capability using the SDK's `attenuate()` (`pip install intentgate` or `npm install @netgnarus/intentgate`). The sub-token is signed by the same chain; the gateway verifies the full caveat list at check time. Useful for sandboxing untrusted sub-agents in a multi-agent system.

Policy operations

Draft — author Rego in the `/policies` editor or via `POST /v1/admin/policies/drafts`. Drafts are unbounded in number; name them descriptively.

Dry-run — replay a candidate draft against historical audit events with `POST /v1/admin/policies/dry-run` (or the editor button). The summary surfaces `would_allow / would_block / would_escalate` counts and a transition matrix (`allow → block`, etc.). Sample divergent rows highlight specific calls that change behavior.

Promote — `POST /v1/admin/policies/active` (or the editor's "Promote to active" button). Routes through TOTP step-up in the Pro console. The previous active becomes the rollback target.

Rollback — `POST /v1/admin/policies/rollback` (or the editor's "Rollback" button). Step-up gated. Returns the active pointer to the prior draft instantly; no redeploy.

Operator onboarding

1. IdP admin adds the user to the appropriate group (`intentgate-admins`, `intentgate-ops`, `intentgate-viewers`).
2. User signs into the console via SSO; role resolves from the JWT group claim.
3. User enrolls TOTP at `/settings/security` (required for any operation that triggers step-up).
4. (Optional) Admin assigns elevation approvers — same role gating; any admin can approve.

JIT admin elevation

The flow:

1. Operator clicks **Request elevation** at `/settings/elevation`. Fills in reason + duration (5 min / 15 / 30 / 1 h / 4 h).
2. Slack/Teams/PagerDuty webhook fires to designated approvers.
3. A different admin opens `/elevations`, reviews the reason, clicks **Approve**, types their TOTP code in the step-up modal.
4. The requester's session role auto-upgrades to admin for the window.
5. Topbar shows a green countdown badge ticking down.
6. Every admin call carries the elevation id through to the gateway; audit rows are linkable back to the approval.
7. On expiry the role reverts automatically.

Self-approval is blocked at the Server Action layer. The audit row's combination of `(decided_by, elevation_id)` is what an auditor verifies against the approval record.

Approving a high-risk tool call

The gateway escalates calls when the Rego policy returns `escalate: true`. The call hangs in `/approvals` waiting for an operator's decision (default timeout: 10 minutes; configurable via `INTENTGATE_APPROVAL_TIMEOUT`).

For each pending row, an operator sees: agent, tool, intent summary, the raw arguments, the policy reason, and (when the policy flagged it) a **Step-up required** badge.

- **Step-up required** — click Approve, type TOTP, the verdict fires. Audit row carries `requires_step_up: true` and `decided_by`.
- **No step-up** — click Approve directly. Single-click decision.
- **Reject** — single-click, optional note. The agent receives a block response with the operator's note.

If no operator decides within the timeout, the row transitions to `timeout` and the agent receives a block response with reason `"escalation timed out"`.

Audit chain verification

Routine practice: run `/audit/verify` (Pro console) or `GET /v1/admin/audit/verify` (raw) once per week or before producing an evidence pack. Pick a window — typically the reporting period — and click **Verify chain**.

A green panel with `verified: N, skipped: M` and a "chain last advanced N seconds ago" indi-

cator is the all-clear. A red panel with `broken_at: {id, ts, reason}` indicates tampering — proceed to the **Incident response: audit chain divergence** runbook in §7.

Audit export for evidence

For SOC 2, ISO 27001, GDPR Art. 30, AI Act Art. 12 evidence:

1. `/audit/verify` page → **Download audit log** card.
2. Pick **CSV** (spreadsheet-friendly) or **NDJSON** (lossless with nested fields).
3. Pick the window (matching the reporting period).
4. Click **Download**.

The file streams from the gateway's `/v1/admin/audit/export` endpoint with the active-tenant filter and operator-selected window. Cap: 200,000 rows per export. For larger sets, narrow the window or split by month.

Pair the export with a verification screenshot dated within the same reporting period.

Health & Observability

Liveness / readiness

```
GET /healthz
{"ok": true, "version": "1.6.0"}
```

The endpoint returns 200 when the process is up and Postgres is reachable. Use as both liveness and readiness in Kubernetes.

Prometheus metrics

Enable with `INTENTGATE_METRICS_ENABLED=true`. The endpoint serves on port 9090 by default; do not expose on the public ingress.

Key metrics:

- `intentgate_gateway_check_decisions_total{check,decision}` — per-check decision counts.
- `intentgate_gateway_check_latency_seconds{check}` — per-check latency histogram.
- `intentgate_gateway_upstream_latency_seconds` — upstream tool-server latency.
- `intentgate_gateway_upstream_status_total{status}` — upstream HTTP-status histogram.
- `intentgate_gateway_revocation_lookups_total{result}` — revocation-store hit/miss.
- `intentgate_gateway_audit_chain_inserts_total` — chain-advance counter.
- `intentgate_gateway_audit_chain_lock_wait_seconds` — contention indicator.

OpenTelemetry tracing

Set `OTEL_EXPORTER_OTLP_ENDPOINT` to your collector. The gateway emits one span per HTTP request with child spans per check, upstream call, and revocation lookup. Honors standard `OTEL_*` env vars (samplers, resource attributes).

Chain-head freshness

`/audit/verify` surfaces a "chain last advanced N seconds ago" indicator. Set up a Prometheus alert if you want a monitoring signal: scrape `intentgate_gateway_audit_chain_inserts_total` and alert when the rate is zero for longer than your expected idle window.

Logs

The gateway logs in JSON to stdout (`slog`-style). Key event types:

- `policy.decision` — every authorization decision.

- `audit.insert` — every chain-advance.
- `audit export hit row cap` — an export was truncated at the row cap.
- `elevation overlay lookup failed` — Pro console DB hiccup during JIT role resolution.
- `webhook send failed` — outbound notification failure (best-effort).

Set `LOG_LEVEL=debug` for SDK-level tracing during pilot. Default is `info`.

Troubleshooting

Symptom — Agent calls return 401

Cause A: token expired or revoked. Check `GET /v1/admin/revocations` for the JTI. If listed, mint a replacement.

Cause B: gateway can't decode the token. Likely the master key on the gateway has rotated since mint. Re-mint under the current key.

Cause C: tenant mismatch. A token minted for tenant `acme` will fail capability check when the gateway expects `globex` for that admin token. Check the token's tenant claim against the gateway's per-tenant admin config.

Symptom — `/audit/verify` shows `broken_at`

Cause A: a Postgres restore or backup-replay introduced rows out of order. Check the broken event's id; if it sits at a restore boundary, this is expected and the chain needs to be re-baselined.

Cause B: tampering. Pull the offending row via `GET /v1/admin/audit?id=<broken_id>` and compare the canonical hash. If the row body doesn't match its stored hash, treat as an incident: investigate database access, capture the verify output as evidence, and escalate to your security team.

Cause C: pre-feature rows. Rows from gateway < 1.6 have `hash=''` and surface as **skipped**, not as a break. If your `skipped` count is non-zero, that's expected for legacy data — the cryptographic chain begins where the hash column starts populating.

Symptom — Notifications not firing

Cause A: webhook secret mismatch. The gateway HMAC-signs every event; the console verifies in constant time. Set `INTENTGATE_WEBHOOK_SECRET` on the gateway and `INTENTGATE_WEBHOOK_RECEIVE_SECRET` on the console to the same value.

Cause B: channel disabled. `/settings/notifications` shows a per-channel toggle; verify it's on.

Cause C: per-channel HTTP failure. Click **Send test** on the channel row. The toast surfaces the per-channel error (4xx from Slack with an expired webhook URL, 401 from PagerDuty with a revoked integration key, etc.).

Cause D: tenant scoping. A channel scoped to tenant `acme` won't fire on tenant `globex` events. The "Scope" column in `/settings/notifications` shows the binding.

Symptom — Step-up modal rejects the valid code

Cause A: clock skew. TOTP allows ±1 30-second step. If your operator's phone clock has drifted more than ~45 seconds, codes fail. Sync the device clock.

Cause B: rate limited. Five failed attempts within 120 seconds triggers a 30-second lockout with a live countdown on the modal. Wait it out.

Cause C: never enrolled. The modal shows a deep-link to `/settings/security`. Enroll first, retry.

Cause D: recovery code already consumed. Recovery codes are one-time. If the operator used one previously, that code is dead — the codes pack issued at enrollment is the master list.

Symptom — Gateway returns 503 on `/v1/admin/audit`

Cause A: audit persistence disabled. Set `INTENTGATE_AUDIT_PERSIST=true` on the gateway and provide `INTENTGATE_POSTGRES_URL`. Restart. Past audit events emitted before persistence was enabled are not retroactively persisted.

Cause B: Postgres unreachable. Check the gateway logs for `audit query failed`. The store error is propagated; surface it from your logs to identify the connection failure.

Symptom — Approval queue empty even after a flagged call

Cause A: approvals backend off. Set `INTENTGATE_APPROVAL_QUEUE=postgres` (or `memory` for single-replica) on the gateway. Restart.

Cause B: Rego policy returns `allow: false` instead of `escalate: true`. The block path bypasses the queue. Run a dry-run of the policy against the relevant call to confirm the decision shape.

Cause C: cross-tenant view. The signed-in operator is scoped to a tenant that doesn't include the escalating call. Switch tenant in the topbar (mode 1) or sign in with the appropriate tenant-bound account (mode 2).

Incident Response Runbooks

Audit chain divergence detected

When `/audit/verify` returns `broken_at`:

1. **Preserve evidence.** Take a full screenshot of the verify result panel. Export the audit window via the **Download audit log** card before any further action.
2. **Snapshot the database.** Stop writes to the audit store if possible (set gateway to read-only mode by removing `INTENTGATE_ADMIN_TOKEN` so admin paths return 401; the four-check pipeline keeps running). Take an out-of-band Postgres snapshot.
3. **Open an incident.** Severity-1 in your incident management system. The audit chain is a load-bearing security control; an unexpected break should be treated as a security incident until proven otherwise.
4. **Investigate.** Pull the offending row via `GET /v1/admin/audit?id=<broken_id>`. Diff the row body against any backup or replica. Check Postgres access logs over the window. Was there a restore? A direct UPDATE? A backup-replay across replica boundaries?
5. **Communicate.** If tampering is confirmed, follow your security disclosure process. Customers who depend on your IntentGate audit pack as evidence need to know which window is affected.
6. **Re-baseline (only after investigation completes).** A new chain head can be seeded by inserting a marker event noting the break and the investigation outcome. The previous chain remains queryable for forensics; verification of the new chain starts from the marker.

Admin token leaked

1. **Generate a new admin token.** `openssl rand -hex 32`.
2. **Update the gateway.** Roll the new token into `INTENTGATE_ADMIN_TOKEN`. Restart replicas one at a time (the gateway is stateless past the DB; rolling restart is safe).
3. **Update the console.** Roll the matching `INTENTGATE_ADMIN_TOKEN` env var on the Pro console.
4. **Update other admin consumers.** `igctl` shells, CI/CD pipelines, monitoring probes. The old token returns 401 immediately on the gateway after the restart.
5. **Audit the leak window.** `GET /v1/admin/audit?from=<leak_start>&to=<rotation_complete>` and review every admin event for unauthorized operations.
6. **Post-mortem.** Identify how the token leaked. Common causes: committed to a public repo, exposed in a screenshot, captured in a misconfigured logging pipeline.

Master key compromise

The master key signs every capability token. Compromise is catastrophic and rare.

1. **Rotate the master key on the gateway.** Generate new (`openssl rand -hex 32`) and set `INTENTGATE_MASTER_KEY`.
2. **Mint replacement tokens for every agent.** Every existing token becomes invalid the moment the new key takes effect.
3. **Audit the agent fleet.** Every agent's secret store must be updated before they retry. Plan for downtime proportional to your agent count.
4. **Investigate.** As above; master key compromise generally implies broader environment compromise.

Suspected tool exfiltration

An agent is calling a tool the operator suspects shouldn't be in scope:

1. **Revoke the agent's capability token.** `POST /v1/admin/revoke` with reason `"suspected exfiltration"`. Effective immediately on every gateway replica.
2. **Pull the agent's audit timeline.** `GET /v1/admin/audit?agent_id=<id>` over the last 7 days. Look for the call sequence — first occurrence, frequency, target tools.
3. **Trace upstream.** The audit row's `upstream_status` indicates whether the call actually reached the tool server. If yes, coordinate with the tool server's owner to assess data exposure.
4. **Patch the Rego policy.** Add a `block` rule for the exfiltration shape; promote with dry-run preview against the audit timeline to confirm the patch catches the pattern.
5. **Mint a replacement token with tighter caveats** if the agent is legitimate (the revocation prevents further calls under the suspect identity).

Reference

Gateway environment variables

`INTENTGATE_ADMIN_TOKEN` **Required.** Superadmin bearer token for `/v1/admin/*`.

`INTENTGATE_MASTER_KEY` **Required.** HMAC secret used to sign capability tokens.

`INTENTGATE_TENANT_ADMINS` Optional. `tenant1:token1,tenant2:token2` — per-tenant admin scopes.

`INTENTGATE_POSTGRES_URL` **Required for production.** DSN for audit, revocations, approvals, and policy drafts.

`INTENTGATE_AUDIT_PERSIST` Optional. `true` to enable Postgres audit. Default `false`.

`INTENTGATE_AUDIT_PERSIST_ARG_VALUES` Optional. `off` / `scalars` / `raw` — argument redaction mode. Default `off`.

`INTENTGATE_UPSTREAM_URL` Optional. Upstream tool server URL. Stub-allow when unset (dev only).

`INTENTGATE_UPSTREAM_TIMEOUT_MS` Optional. Upstream call timeout. Default 5000.

`INTENTGATE_APPROVAL_QUEUE` Optional. `off` / `memory` / `postgres`. Default `off`.

`INTENTGATE_APPROVAL_TIMEOUT` Optional. Seconds before an unanswered escalation times out. Default 600.

`INTENTGATE_POLICY_STORE` Optional. `off` / `memory` / `postgres`. Default `off` (uses file or embedded policy).

`INTENTGATE_METRICS_ENABLED` Optional. `true` to expose `/metrics`. Default `false`.

`INTENTGATE_WEBHOOK_URL` Optional. HMAC-signed event fan-out target.

`INTENTGATE_WEBHOOK_SECRET` Optional. HMAC secret for webhook signatures.

`INTENTGATE_WEBHOOK_EVENTS` Optional. Comma-separated filter. Valid values: `deny`, `escalate`, `requires_step_up`, `approval_timeout`.

`INTENTGATE_SIEM_SPLUNK_HEC_URL` Optional. Splunk HEC endpoint for audit forwarding.

`INTENTGATE_SIEM_SPLUNK_TOKEN` Optional. Splunk HEC token.

`INTENTGATE_SIEM_DATADOG_API_KEY` Optional. Datadog Logs API key.

`INTENTGATE_SIEM_SENTINEL_WORKSPACE_ID` Optional. Microsoft Sentinel workspace.

`INTENTGATE_SIEM_SENTINEL_SHARED_KEY` Optional. Microsoft Sentinel shared key.

`OTEL_EXPORTER_OTLP_ENDPOINT` Optional. OTLP collector endpoint for tracing.

Console Pro environment variables

`INTENTGATE_GATEWAY_URL` **Required.** URL of the gateway's admin API.

`INTENTGATE_ADMIN_TOKEN` **Required.** Same superadmin token as the gateway.

`INTENTGATE_LICENSE_KEY` **Required.** Commercial license.

`INTENTGATE_POSTGRES_URL` **Required.** DSN for SCIM, elevations, notification channels.

`INTENTGATE_TOTP_ENCRYPTION_KEY` **Required.** 32 random bytes (hex) for AES-256-GCM at rest.

`INTENTGATE_WEBHOOK_RECEIVE_SECRET` **Required.** HMAC secret matching the gateway's webhook secret.

`INTENTGATE_SCIM_BEARER_TOKEN` Optional. SCIM push authentication. Set to enable SCIM.

`AUTH_SECRET` **Required.** NextAuth session-cookie HMAC.

`AUTH_PROVIDER` Optional. `oidc` (default in prod) / `mock` (dev only).

`AUTH_OIDC_ISSUER` **Required when** `AUTH_PROVIDER=oidc`. IdP discovery URL.

`AUTH_OIDC_CLIENT_ID` **Required when** `AUTH_PROVIDER=oidc`. OIDC client id.

`AUTH_OIDC_CLIENT_SECRET` **Required when** `AUTH_PROVIDER=oidc`. OIDC client secret.

`AUTH_OIDC_SCOPES` Optional. Default `openid profile email groups`.

`AUTH_ROLE_CLAIM` Optional. JWT claim with role. Default `groups`.

`AUTH_ROLE_MAPPING` Optional. `value:role` comma-separated mapping.

`AUTH_TENANT_CLAIM` Optional. JWT claim binding session to a tenant (mode 2).

`INTENTGATE_DEFAULT_ROLE` Optional. Fallback role when no resolver fires. Default deny.

Admin API endpoints

Method	Path	Purpose
GET	`/healthz`	Liveness / version probe.
POST	`/v1/mcp`	Agent-facing tool-call entry point.
POST	`/v1/admin/mint`	Issue a capability token.
POST	`/v1/admin/revoke`	Revoke a capability token by JTI.
GET	`/v1/admin/revocations`	List revocations.
GET	`/v1/admin/tenants`	Configured per-tenant admin scopes.
GET	`/v1/admin/audit`	Paged audit query with filters.
GET	`/v1/admin/audit/verify`	Tamper-evident chain verification.
GET	`/v1/admin/audit/export`	Streaming CSV / NDJSON export.

Method	Path	Purpose
GET	`/v1/admin/approvals`	Pending / decided approvals queue.
POST	`/v1/admin/approvals/{id}/decide`	Operator decision on a pending row.
GET	`/v1/admin/policies/active`	Current active policy pointer.
POST	`/v1/admin/policies/active`	Promote a draft to active.
POST	`/v1/admin/policies/rollback`	Roll back to the previous active.
DELETE	`/v1/admin/policies/active`	Clear the active pointer (falls back to default).
GET	`/v1/admin/policies/drafts`	List policy drafts.
POST	`/v1/admin/policies/drafts`	Save a new draft.
DELETE	`/v1/admin/policies/drafts/{id}`	Delete a draft.
POST	`/v1/admin/policies/dry-run`	Replay a candidate against history.
GET	`/v1/admin/integrations`	SIEM emitter status.
GET	`/metrics`	Prometheus metrics (when enabled).

Support & Escalation

Support channels

- **Documentation portal** — this runbook plus per-feature deep-dives, code samples, change logs.
- **Email** — `contact@netgnarus.com` for issue triage and feature questions. Acknowledged within one business day.
- **Slack Connect** — available with paid subscriptions. Same-channel chat with NetGnarus engineering during business hours.
- **GitHub issues** — public bug reports against the OSS components (`intentgate-gateway`, `intentgate-extractor`, SDKs, helm). Pro-console issues go through email or Slack Connect.

Severity matrix

Severity	Definition	Response target
Sev-1	Gateway down, audit chain divergence, master key compromise.	1 business hour to first response.
Sev-2	Pro console unreachable, OIDC sign-in broken, notifications stuck.	4 business hours.
Sev-3	Feature degraded but workaround available.	1 business day.
Sev-4	Documentation gap, enhancement request.	5 business days.

Security disclosure

Coordinated disclosure address: `security@netgnarus.com`. PGP fingerprint published on the documentation portal.

Embargo policy: 90 days from confirmed receipt or earlier if a patch is available and customer-deployed.

License

The Pro console is licensed commercially. The license key controls feature access; an expired key falls back to the OSS console behavior (full token lifecycle remains operational; OIDC SSO, SCIM,

JIT elevation, notification channels, audit verify, and policy authoring degrade to feature-disabled cards).

The OSS components remain Apache 2.0 in perpetuity. NetGnarus commits to never relicensing them; see the open core boundary commitment (when published).

IntentGate is a product of NetGnarus.
For evaluation enquiries, contact j.cordoba@netgnarus.com.