Sachin Gupta
Governance for Agentic Systems

TOOL CONTRACTS FOR AI AGENTS

Versioned, validated, and governed tool access.

A Java + Spring Boot pattern that forces AI agents to call tools through versioned contracts.

THE PROBLEM

The agent era arrived. The guardrails did not.

Agents are in production.

Not chatbots. They place orders, query secrets, ship code.

Tools touch real money and real PII.

Every call is a privileged action against a real API.

Most teams ship without a control plane.

Auth alone does not catch this. We need tool-aware policy.

WHAT'S MISSING

A policy decision point between agent intent and tool execution.

Tool-aware. Versioned. Auditable.

WHAT GOES WRONG

Three failure modes auth alone will not catch

01

Injection

Adversarial input convinces the agent to call a tool it should not.

OWASP LLM-01

02

Drift

Agent is on v1. Tool team quietly shipped v2 with a new field.

SILENT SHAPE MISMATCH

03

Scope creep

Tool used safely in one path is reused in a higher-trust path with no review.

NO SECOND LOOK

The gap is not more auth. It is tool-aware policy.

THE PATTERN

Quote a contract. Do not call a tool.

BEFORE

```
// agent SDK
tools.call(
  "prom_query",
  { query, range }
);
```

- No policy decision point
- No version pin
- No structured audit

AFTER

```
// agent SDK
gateway.invoke({
  contract: { id: "metrics",
              version: "v1" },
  tool: "prom_query",
  args: { query, range }
});
```

- id + version are the lock
- Gateway resolves what runs
- One audit row per call

THE DATA MODEL

Anatomy of a contract — eight fields, three buckets

01 · THE LOCK

`id`

`version`

Together they uniquely identify a contract. Different version = different contract.

02 · THE CALL

`tool`

`input_schema`

`limits`

What runs. What shape it must take. How much it can cost.

03 · ACCOUNTABILITY

`required_scopes`

`owner`

`audit_class`

Who can use it. Who owns it. How loud the audit gets.

id + version are the lock. Everything else is policy attached to that key.

IN THE WILD

Example — metrics.v1 contract

```yaml
id: metrics
version: v1
tool: prom_query
owner: observability-platform
audit_class: routine
required_scopes: [obs:read]
limits: { max_payload_bytes: 4096, rps: 50 }
input_schema:
  type: object
  required: [query, range]
  properties:
    query: { type: string }
    range:
      type: object
      required: [from, to]
```

Source of truth

YAML in git. Reviewed in PRs.

Hot reload

Gateway watches for changes.

Schema is policy

Not a doc. Validator runs it.

HOW IT RUNS

Architecture and the seven-step pipeline

Only the gateway is new code.

Tools and APIs stay as they are.

Validation pipeline · cheap → expensive

1	Token decode	401
2	Contract resolution	404 / 409
3	Tool match	403
4	Scope check	403
5	Schema validation	400
6	Limits + rate	429
7	Forward + audit	200

First failure wins. Codes are distinguishable.

DEMO 1 OF 4

Demo 1 — Happy path

HTTP 200

OK

WHAT HAPPENS

- Agent quotes metrics.v1 with valid args.
- All seven checks pass in order.
- Audit row written. Tool response returned.

GATEWAY LOG (kubectl logs -f)

```
[gateway] tok.ok        sub=agent-svc-1
scopes=[obs:read]
[gateway] ctr.resolved  metrics@v1  status=current
[gateway] tool.match    requested=prom_query
[gateway] scope.ok      required=[obs:read]
[gateway] schema.ok     pointer=/  errors=0
[gateway] limit.ok      bytes=312/4096
[gateway] forward.ok    target=prom-stub  status=200
[audit]   write class=routine  hash=sha256:9f...e2  ✓

→ HTTP 200  body={ result: [ ... ] }
```

DEMO 2 OF 4

Demo 2 — Contract mismatch

HTTP 403

`contract_tool_mismatch`

WHAT HAPPENS

Adversarial agent on metrics.v1 tries vault.read_secret.

Fails at step 3 — tool match — before scope check.

Defense in depth: contract id alone forbids the tool.

GATEWAY LOG (kubectl logs -f)

```
[gateway] tok.ok        sub=agent-svc-1
scopes=[obs:read]
[gateway] ctr.resolved  metrics@v1  status=current
[gateway] tool.MISMATCH requested=vault.read_secret
          contract.tool=prom_query
[audit]   write class=sensitive  decision=deny
          reason=tool_mismatch

→ HTTP 403
  { error:'contract_tool_mismatch',
    reason:'metrics@v1 binds tool=prom_query' }
```

DEMO 3 OF 4

Demo 3 — Version drift

HTTP 409

`contract_version_drift`

WHAT HAPPENS

Tool team shipped metrics.v2 with tenant_id.

Agent is still on v1 — registry has it deprecated.

Gateway returns 409 with current version + migration.

GATEWAY LOG (kubectl logs -f)

```
[gateway] tok.ok        sub=agent-svc-1
[gateway] ctr.resolved  metrics@v1  status=deprecated
[gateway] ctr.DRIFT     current=v2
          introduced=[tenant_id]
[audit]   write decision=deny  reason=version_drift

→ HTTP 409
  { error:'contract_version_drift',
    current_version:'v2',
    migration:'/docs/contracts/metrics#v1-to-v2' }
```

DEMO 4 OF 4

Demo 4 — Schema violation

HTTP 400

schema_violation

WHAT HAPPENS

Agent passes range as a string, not an object.

Validator returns JSON Pointer + expected/got + hint.

The error itself is documentation.

GATEWAY LOG (kubectl logs -f)

```
[gateway] tok.ok          sub=agent-svc-1
[gateway] ctr.resolved    metrics@v1   status=current
[gateway] tool.match      requested=prom_query
[gateway] scope.ok
[gateway] schema.FAIL     pointer=/args/range
          expected=object   got=string
[audit]   write decision=deny
reason=schema_violation

→ HTTP 400
  { errors:[{ pointer:'/args/range',
              expected:'object', got:'string' }] }
```

RECAP

How each demo fails — at a glance

Scenario	Token?	Contract?	Tool?	Scope?	Schema?	Stops at	HTTP
happy	✓	✓ current	✓	✓	✓	step 7 (forwards)	**200**
mismatch	✓	✓ current	**✗ wrong tool**	—	—	step 3	**403**
drift	✓	**✗ deprecated**	—	—	—	step 2b	**409**
schema	✓	✓ current	✓	✓	**✗ wrong shape**	step 5	**400**

OPERATE

Free out of the box, and how to roll it in

FREE OUT OF THE BOX

- ✔ **OTel spans**
  one per validation step
- ✔ **Audit log**
  payload hash, never raw body
- ✔ **Platform metrics**
  deny rate · drift · schema-fail top-N
- ✔ **Health & readiness**
  stale-but-readable on registry outage

ROLLOUT — FOUR PHASES

1. **Dry run / shadow**
   gateway sees traffic, denies nothing
2. **First contract live**
   one team, lowest-risk tool first
3. **Team-by-team opt-in**
   expand by audit class
4. **GitOps for promotion**
   v1 → v2 is a PR with deny_after

THREE THINGS TO REMEMBER

If nothing else, take these.

01

Make agents quote a contract.

The smallest change that buys a real policy decision point.

02

Versioning is non-negotiable.

v1 and v2 are different keys. Drift becomes solvable.

03

Audit gets easier when calls are typed.

One row per call. Decision and reason captured.