**Agent-to-Agent (A2A) Protocol — Technical Guide**

**1) What A2A is (and why it exists)**

**A2A** is an open, application-level protocol that lets **independent AI agents**—built by different vendors and frameworks—**discover each other, advertise capabilities, and exchange requests/responses** in a standard way. Google announced A2A in April 2025 and has been evolving it with tooling (ADK, Agent Engine) and an open implementation effort; the Linux Foundation is stewarding the project with growing industry support. ([developers.googleblog.com](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/?utm_source=chatgpt.com), [Google Cloud](https://cloud.google.com/blog/products/ai-machine-learning/agent2agent-protocol-is-getting-an-upgrade?utm_source=chatgpt.com), [linuxfoundation.org](https://www.linuxfoundation.org/press/linux-foundation-launches-the-agent2agent-protocol-project-to-enable-secure-intelligent-communication-between-ai-agents?utm_source=chatgpt.com))

**Why it matters to us**

* We’re composing a **multi-agent GraphRAG** system (Local/Global/DRIFT + Analysis). A2A provides a **vendor-neutral contract** so our orchestrator can call internal agents (our own) **and** external partner agents (translation, OCR, sanctions, etc.) without custom glue per vendor. ([google.github.io](https://google.github.io/adk-docs/?utm_source=chatgpt.com))

**2) Core concepts (terms you’ll see in the spec & docs)**

* **Provider Agent**: exposes a **capabilities catalog** and an **invoke** interface to other agents.
* **Client Agent**: discovers capabilities and invokes them (synchronous or streaming).
* **Capability**: a typed operation (name, description, **input/output JSON Schemas**, error model, policy).
* **Session/Conversation**: optional, long-lived interaction state.
* **Policy/Quota/Auth**: scopes, rate limits, tenant isolation, bearer/OIDC.

These primitives are the backbone across the official A2A write-ups, ADK guides, and codelabs. ([GitHub](https://github.com/a2aproject/A2A?utm_source=chatgpt.com), [google.github.io](https://google.github.io/adk-docs/a2a/?utm_source=chatgpt.com), [Google Codelabs](https://codelabs.developers.google.com/intro-a2a-purchasing-concierge?utm_source=chatgpt.com))

**3) Message flow (the minimum you need to implement)**

**3.1 Discovery**

Client agent fetches a provider’s **capabilities** document (HTTP GET). It includes agent metadata, a list of capabilities, each with schemas and auth requirements. ([GitHub](https://github.com/a2aproject/A2A?utm_source=chatgpt.com))

**3.2 Invocation**

Client agent **POSTs** an invoke request specifying capability, request\_id, inputs, and optional context (locale, tenant, trace). The provider returns a structured success or error payload; long tasks may stream progress/events. ([GitHub](https://github.com/a2aproject/A2A?utm_source=chatgpt.com))

**3.3 Streaming & status**

For long-running tasks, providers can emit **progress states/events** (e.g., SSE/WebSocket) and a final status: success|error. Google’s A2A materials and ADK examples show both sync and streaming patterns. ([google.github.io](https://google.github.io/adk-docs/a2a/?utm_source=chatgpt.com), [developers.googleblog.com](https://developers.googleblog.com/en/agents-adk-agent-engine-a2a-enhancements-google-io/?utm_source=chatgpt.com))

The codelab (“Purchasing Concierge”) is the fastest way to see end-to-end discovery → invoke → response with reference payloads. (Google Codelabs)

**4) Security & governance (what to enforce)**

* **AuthN/Z**: Bearer or OIDC tokens with **scopes per capability** (e.g., rag.read, analysis.write), plus **tenant-level quota/rate limits**. ([GitHub](https://github.com/a2aproject/A2A?utm_source=chatgpt.com))
* **Schema validation**: strictly validate inputs/outputs against JSON Schema to block prompt-injection via payloads and ensure contract stability. ([GitHub](https://github.com/a2aproject/A2A?utm_source=chatgpt.com))
* **Isolation**: A2A encourages **exchange of results/artifacts**, not internal memory or tool access—reducing data exfiltration risk between organizations. ([A2A Protocol](https://a2aprotocol.ai/?utm_source=chatgpt.com))
* **Observability**: include request\_id and trace\_id; collect latencies, token counts, per-capability success/error rates. Google’s dev blogs emphasize production-readiness and evaluation tooling alongside A2A. ([developers.googleblog.com](https://developers.googleblog.com/en/agents-adk-agent-engine-a2a-enhancements-google-io/?utm_source=chatgpt.com))

**5) How A2A relates to MCP (Model Context Protocol)**

Think **MCP** for *agent ↔ tool/data* (inside one agent’s runtime) and **A2A** for *agent ↔ agent* (across runtimes/organizations). They’re complementary; ADK docs and codelabs show agents using MCP tools internally while speaking A2A externally. ([Google Codelabs](https://codelabs.developers.google.com/intro-a2a-purchasing-concierge?utm_source=chatgpt.com), [google.github.io](https://google.github.io/adk-docs/?utm_source=chatgpt.com))

**6) Implementation plan for our project (Next.js + GraphRAG)**

**6.1 Expose our orchestrator as an A2A Provider**

Create two endpoints in our Node/Next.js API:

1. GET /a2a/capabilities

Returns agent metadata and a list of capabilities—at minimum:

* local\_graph\_rag (1–2 hop neighborhood with citations)
* global\_graph\_rag (multi-hop, cross-code synthesis)
* drift\_graph\_rag (recency/change-aware)
* contradiction\_miner (AI Analysis pipeline)

Each capability includes **input\_schema** and **output\_schema** in JSON Schema (versioned), plus **auth.scopes**. This matches A2A’s discovery pattern and what ADK’s A2A integration expects. ([google.github.io](https://google.github.io/adk-docs/a2a/?utm_source=chatgpt.com))

1. POST /a2a/invoke

Accepts { capability, request\_id, inputs, context }.

* Validate against the declared schemas.
* Execute the corresponding agent or sub-workflow.
* Respond { request\_id, status, outputs, meta }, and optionally stream progress for long jobs. Pattern mirrors the A2A invoke guidance. ([GitHub](https://github.com/a2aproject/A2A?utm_source=chatgpt.com))

If we adopt Google’s ADK, there’s a first-class A2A section that shows how to adapt an existing agent to speak A2A, and how to convert ADK agents into A2A-ready components. ([google.github.io](http://google.github.io), Google Cloud)

**6.2 Consume external A2A agents**

Add a simple “A2A client” utility:

* Fetch /a2a/capabilities from the remote agent.
* Check auth.scopes and negotiate limits.
* Build an invoke() helper that sends typed requests and handles retries/streaming.

Use this to integrate third-party **translation/OCR** agents or enterprise services that already advertise A2A. Box’s public write-up is a good reference of a partner integrating with A2A. ([Google Cloud](https://cloud.google.com/blog/topics/customers/box-ai-agents-with-googles-agent-2-agent-protocol?utm_source=chatgpt.com))

**6.3 Versioning & compatibility**

Include a2a\_version and a per-capability version. Use additive changes where possible; gate breaking changes with a new version slice in the capabilities doc. This is consistent with the standardization guidance. ([GitHub](https://github.com/a2aproject/A2A?utm_source=chatgpt.com))

**7) Minimal payload examples (provider-side)**

**Discovery (response)**

{

"a2a\_version": "1.0",

"agent": { "name": "uae-legal-graph", "version": "1.2.0" },

"capabilities": [

{

"name": "local\_graph\_rag",

"version": "1.0",

"description": "Neighborhood (1–2 hops) with citations",

"input\_schema": { "$schema":"<https://json-schema.org/draft/2020-12/schema>",

"type":"object","properties":{"query":{"type":"string","minLength":3}},

"required":["query"] },

"output\_schema": { "type":"object",

"properties":{"answer":{"type":"string"},

"citations":{"type":"array","items":{"type":"object"}}},

"required":["answer"] },

"auth": { "scopes": ["rag.read"] }

}

]

}

**Invoke (request & response)**

// POST /a2a/invoke (request)

{

"capability": "local\_graph\_rag",

"request\_id": "req-7b2a",

"inputs": { "query": "What changed for UBO obligations in 2024?" },

"context": { "tenantId": "demo", "locale": "en-US", "trace\_id": "t-123" }

}

// 200 (response)

{

"request\_id": "req-7b2a",

"status": "success",

"outputs": {

"answer": "Summary …",

"citations": [{ "nodeId":"Article-123", "snippet":"…" }]

},

"meta": { "latency\_ms": 842, "model": "gpt-4o" }

}

*Note:* Exact field names may differ by A2A version/SDK; align with the spec/ADK generator you adopt. ([GitHub](https://github.com/a2aproject/A2A?utm_source=chatgpt.com), [google.github.io](https://google.github.io/adk-docs/a2a/?utm_source=chatgpt.com))

**8) Testing, observability, and safety checklist**

* **Contract tests**: generate clients from JSON Schemas; validate requests in CI (Pact or similar).
* **Canaries**: run a tiny A2A probe that exercises capabilities + invoke and asserts response shape/latency.
* **Metrics**: per-capability SLI/SLO (availability, P95 latency, error budget), usage by tenant/scope.
* **Threat model**: authenticate every call; enforce schema limits; sanitize text fields (prompt-injection payloads); rate-limit and deduplicate request\_id for idempotency. These are called out across Google’s A2A materials and general security guidance. ([developers.googleblog.com](https://developers.googleblog.com/en/agents-adk-agent-engine-a2a-enhancements-google-io/?utm_source=chatgpt.com))

**9) Where to start (recommended path this sprint)**

1. **Scaffold** /a2a/capabilities and /a2a/invoke in our Next.js API (Node runtime).
2. Register four capabilities: local\_graph\_rag, global\_graph\_rag, drift\_graph\_rag, contradiction\_miner.
3. Add **Bearer auth + scopes** and per-tenant quotas.
4. Build a tiny **A2A client** to call a partner agent (e.g., translation) using their capabilities doc.
5. Run the **“Purchasing Concierge”** codelab to compare payloads and streaming behavior with ours. ([Google Codelabs](https://codelabs.developers.google.com/intro-a2a-purchasing-concierge?utm_source=chatgpt.com))

**10) Authoritative references**

* **Official announcement & updates** (Google Developers Blog / Cloud Blog). ([developers.googleblog.com](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/?utm_source=chatgpt.com), [Google Cloud](https://cloud.google.com/blog/products/ai-machine-learning/agent2agent-protocol-is-getting-an-upgrade?utm_source=chatgpt.com))
* **Agent Development Kit (ADK) & A2A docs** (how to build/convert agents). ([google.github.io](https://google.github.io/adk-docs/?utm_source=chatgpt.com))
* **Getting Started codelab** (hands-on, end-to-end). ([Google Codelabs](https://codelabs.developers.google.com/intro-a2a-purchasing-concierge?utm_source=chatgpt.com))
* **Linux Foundation press** (governance & ecosystem). ([linuxfoundation.org](https://www.linuxfoundation.org/press/linux-foundation-launches-the-agent2agent-protocol-project-to-enable-secure-intelligent-communication-between-ai-agents?utm_source=chatgpt.com))
* **Background explainers** (IBM/Descope) and community posts (use with caution for non-normative guidance). ([IBM](https://www.ibm.com/think/topics/agent2agent-protocol?utm_source=chatgpt.com), [Descope](https://www.descope.com/learn/post/a2a?utm_source=chatgpt.com))

**TL;DR for the team**

* **A2A = standard “language” for agents to call each other** across orgs/stacks.
* We’ll **expose** our GraphRAG capabilities over A2A and **consume** external agents the same way.
* Follow the **ADK A2A guide** + **codelab** to bootstrap in days; enforce **scopes, quotas, and strict schemas** from day one. ([google.github.io](https://google.github.io/adk-docs/a2a/?utm_source=chatgpt.com), [Google Codelabs](https://codelabs.developers.google.com/intro-a2a-purchasing-concierge?utm_source=chatgpt.com))