Synthèse générale (Avril 2025)

En avril 2025, Google a inauguré **Agent-2-Agent (A2A)**, un protocole ouvert destiné à faire communiquer des agents d'IA hétérogènes entre eux ([

```
    Announcing the Agent2Agent Protocol (A2A)

    - Google Developers Blog
](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=To
    Announcing the Agent2Agent Protocol (A2A)

    - Google Developers Blog
](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=To
    Announcing the Agent2Agent Protocol (A2A)

    - Google Developers Blog
](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=A2
```

Durant avril, on observe plusieurs faits marquants : **publication de la spécification A2A en open-source**, avec un dépôt GitHub dédié, et premières itérations; **adoption initiale** par divers frameworks et plateformes (Google Vertex AI, projets open-source tiers) facilitant la mise en œuvre d'agents compatibles; retours d'expérience concrets via des démonstrations et articles techniques; et **focalisation sur la sécurité et la gouvernance** (authentification unifiée, délégation d'autorisations, etc.) pour encadrer ces nouveaux échanges inter-agents. L'ensemble de ces développements suggère qu'en l'espace d'un mois, A2A est passé du stade d'annonce à celui de véritable chantier collaboratif, mobilisant un écosystème large vers une **standardisation des agents** un peu analogue à ce que REST et HTTP furent pour les APIs web, avec à la clé des gains potentiels de productivité et de flexibilité importants pour les développeurs. Les sections suivantes détaillent ces tendances par thématiques.

Spécification & versions du protocole A2A

La **spécification initiale d'Agent2Agent** a été publiée en version *draft* ouverte (v0.1.0) début avril 2025 (What is The Agent2Agent Protocol (A2A) and Why You Must Learn It ...). Hébergée sur le dépôt GitHub `google/A2A`, elle définit un schéma JSON complet du protocole (Agent Card, tâches, messages, etc.) sous licence Apache 2.0. Le cœur de la spec repose sur des standards web éprouvés –

échanges HTTP(S) avec appels au format JSON-RPC 2.0 et flux SSE pour le streaming (From Silos to Synergy: Agent to Agent(A2A) Protocol in Action | by Renu Khandelwal | Apr, 2025 | Medium) ([

```
    Announcing the Agent2Agent Protocol (A2A)


    - Google Developers Blog

](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=,s

    Announcing the Agent2Agent Protocol (A2A)


    - Google Developers Blog

](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=,r
```

Évolutions prévues : étant donné le statut précoce d'A2A en avril, Google a annoncé travailler **ouvertement avec la communauté** sur de futures révisions (Google Open-Sources Agent2Agent Protocol for Agentic Collaboration - InfoQ). La feuille de route inclut l'enrichissement des *Agent Cards* (ex. inclusion native de schémas d'authentification OAuth/OIDC) (GitHub - google/A2A: An open protocol enabling communication and interoperability between opaque agentic applications.), l'ajout de mécanismes de **découverte d'agents** plus automatisés, une méthode de requête de capacités à la volée (`QuerySkill()` pour interroger un agent sur une compétence non déclarée) (GitHub - google/A2A: An open protocol enabling communication and interoperability between opaque agentic applications.), et le support de la négociation dynamique en cours de dialogue (par ex. passer en audio/vidéo en milieu de conversation) (GitHub - google/A2A: An open protocol enabling communication and interoperability between opaque agentic applications.). Des améliorations de fiabilité pour le streaming et les notifications push sont aussi planifiées (GitHub - google/A2A: An open protocol enabling communication and interoperability between opaque agentic applications.). Par ailleurs, la documentation générée à partir du schéma JSON sera étoffée (génération de docs HTML lisibles) et des exemples d'usage supplémentaires seront fournis, y compris avec divers frameworks d'agents (GitHub - google/A2A: An open protocol enabling communication and interoperability between opaque agentic applications.). Aucune version « stable » n'a encore été estampillée en avril, mais Google vise une **version production-ready d'ici fin 2025** en partenariat avec l'industrie ([

```
    Announcing the Agent2Agent Protocol (A2A)


    - Google Developers Blog

](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=ca

    Announcing the Agent2Agent Protocol (A2A)
```

```
    - Google Developers Blog
](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=To
    Announcing the Agent2Agent Protocol (A2A)

    - Google Developers Blog
](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=Bo
```

Implémentations (serveurs, clients & hôtes A2A)

Plusieurs implémentations de référence du protocole A2A ont émergé en avril, servant soit de **serveur A2A** (agent offrant un endpoint compatible) soit de **client A2A** (agent initiant des appels vers un autre agent) :

- **Référence Google (Python & TypeScript)** – Le dépôt officiel inclut des exemples complets de serveur et client A2A écrits en Python (FastAPI) et en TypeScript (GitHub - google/A2A: An open protocol enabling communication and interoperability between opaque agentic applications.). Ces exemples démontrent la mise en place d'un endpoint d'agent exposant son *Agent Card* et répondant aux requêtes de tâche. Par exemple, un *mini-serveur* Python peut publier un Agent Card sur `http://localhost:8080/.well-known/agent.json` et gérer des tâches via FastAPI en respectant le schéma A2A (Unlocking the Future of AI Collaboration with the Agent2Agent (A2A) Protocol - Abdul Aziz Ahwan) (Unlocking the Future of AI Collaboration with the Agent2Agent (A2A) Protocol - Abdul Aziz Ahwan). Côté client, un script peut interroger un agent distant en récupérant son Agent Card puis en envoyant des requêtes JSON-RPC sur l'URL fournie. Bien que ces implémentations de base soient fournies à titre de démonstration, elles ont servi de point de départ à de nombreux développeurs pour expérimenter A2A. Google a reconnu que **l'absence de SDK packagé au lancement** pouvait freiner l'adoption, et encourage la communauté à réutiliser ou adapter le code exemple en attendant des clients/serveurs officiels plus aboutis (Integrating Semantic Kernel Python with Google's A2A Protocol | Azure AI Foundry Blog).

- **Vertex AI Agent Builder (Agent Engine)** – La plateforme Google Cloud Vertex AI s'est rapidement intégrée à A2A. L'**Agent Engine** de Vertex (service d'orchestration d'agents managé) supporte nativement le protocole : il peut ainsi connecter des agents externes conformes A2A, quel que soit leur environnement d'origine (Vertex AI Agent Builder | Google Cloud). Concrètement, Vertex AI **peut héberger un agent orchestrateur** qui dialogue via HTTP avec d'autres agents tiers (par exemple un agent ServiceNow ou Salesforce compatible) sans nécessiter de connecteurs propriétaires. Google présente A2A comme un *"langage universel"* pour que son Agent Builder unifie des agents hétérogènes (ADK, LangGraph, Crew.ai ou autres) (Build and manage multi-system agents with Vertex AI | Google Cloud Blog). Des connecteurs spécifiques Vertex permettant de découvrir et déclarer des agents A2A ont été annoncés : par

exemple, un développeur peut enregistrer l'URL d'un agent externe dans l'interface Vertex, la plateforme allant lire l'Agent Card pour connaître ses capacités et établir la connexion. Cette intégration transparente s'inscrit dans la stratégie « **multi-system agents** » de Google Cloud visant à orchestrer des agents à travers toute l'entreprise sans couture (Vertex AI Agent Builder | Google Cloud).

- **Hôtes multi-agents & orchestrateurs** – Au-delà des simples clients, on a vu apparaître des **applications "hôte"** jouant le rôle de médiateur entre plusieurs agents A2A. Un exemple marquant est la **demo Web "Mesop"** publiée par Google : cette application front-end + backend FastAPI agit comme une interface unifiée où l'utilisateur peut converser avec un *pool* de 3 agents distincts (Getting Started with Google A2A: A Hands-on Tutorial for the Agent2Agent Protocol | by Heiko Hotz | Google Cloud - Community | Apr, 2025 | Medium). Concrètement, la démo lance trois serveurs d'agent (l'un basé sur LangGraph, un sur CrewAI, et un agent construit avec l'ADK Google) – chacun tournant sur un port différent avec son endpoint A2A propre. Un quatrième service fait office d'orchestrateur central : le backend FastAPI de l'UI web reçoit les messages de l'utilisateur, **découvre les capacités** de chaque agent en lisant leurs Agent Cards (via une étape d'enregistrement manuel dans la demo) (Getting Started with Google A2A: A Hands-on Tutorial for the Agent2Agent Protocol | by Heiko Hotz | Google Cloud - Community | Apr, 2025 | Medium) (Getting Started with Google A2A: A Hands-on Tutorial for the Agent2Agent Protocol | by Heiko Hotz | Google Cloud - Community | Apr, 2025 | Medium), puis route chaque requête utilisateur vers l'agent le plus approprié via A2A. Les réponses de ces agents distants sont ensuite agrégées et renvoyées côté front-end dans une vue conversation unique. Ce type d'« agent hôte » démontre comment bâtir un **hub conversationnel multi-agents** : l'UI présente une liste des tâches en cours, des events SSE reçus, etc., offrant une visibilité sur le travail collaboratif des agents. C'est une implémentation de référence pour orchestrer des *ensembles d'agents spécialisés*, modèle que Google nomme parfois *Agentspace*. Ce concept d'hôte A2A préfigure des **workflows complexes orchestrés** (ex. un agent pilote qui délègue une sous-tâche à un autre agent via A2A, collecte le résultat, puis le combine à d'autres sources).

- **Implémentations tierces** – La communauté open-source s'est emparée du protocole dès son annonce. On note par exemple une **bibliothèque Rust** non-officielle (`a2a-rs`) créée par un développeur indépendant pour implémenter A2A côté client et serveur en langage Rust (EmilLindfors/a2a-rs: Agent2Agent in Rust - GitHub). De même, des contributeurs ont expérimenté des *wrappers* en Go ou Node.js en s'appuyant sur la spécification JSON. Ces initiatives témoignent de l'intérêt pour intégrer A2A dans divers environnements technologiques dès à présent. Par ailleurs, certains projets d'agents autonomes existants ont annoncé travailler à la compatibilité A2A : par exemple, **LangChain** (framework populaire d'agents outillés) figure parmi les partenaires de lancement ([

```
Announcing the Agent2Agent Protocol (A2A)


- Google Developers Blog
```

```
](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=

([

    Announcing the Agent2Agent Protocol (A2A)

    - Google Developers Blog

](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/)) *Exemple

    Announcing the Agent2Agent Protocol (A2A)

    - Google Developers Blog

](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=,%

    Announcing the Agent2Agent Protocol (A2A)

    - Google Developers Blog

](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=,v
```

SDK & intégrations tierces (frameworks multi-agents)

L'annonce d'A2A s'est accompagnée d'une volonté d'en faire un **standard inter-framework**. En avril 2025, plusieurs SDK, bibliothèques ou frameworks d'agents ont annoncé leur prise en charge du protocole, permettant de connecter leurs agents via A2A sans effort supplémentaire. Voici les principales intégrations par thème :

- **Agent Development Kit (ADK)** – Google a lancé simultanément un kit de développement d'agent open-source baptisé ADK (Build and manage multi-system agents with Vertex AI | Google Cloud Blog). Disponible en Python (d'autres langages à venir), **ADK** fournit un cadre pour construire et orchestrer facilement des agents, avec moins de 100 lignes de code pour un agent de base (Build and manage multi-system agents with Vertex AI | Google Cloud Blog). Cet outil est optimisé pour Vertex AI et les modèles Google (Gemini, etc.), mais surtout, il a été conçu dès le départ pour être **interopérable**. ADK supporte nativement l'**Agent2Agent** : un agent créé avec l'ADK peut exposer un endpoint A2A ou appeler d'autres agents A2A sans complexité supplémentaire (Vertex AI Agent Builder | Google Cloud). Par exemple, un développeur ADK peut instancier un *A2A server* en quelques lignes, l'ADK se chargeant d'implémenter les méthodes du protocole. ADK facilite également l'intégration d'**outils via MCP** (il embarque *Model Context Protocol* pour brancher des datasources ou APIs tierces) (Build and manage multi-system agents

with Vertex AI | Google Cloud Blog) (Vertex AI Agent Builder | Google Cloud). En interne, ADK était utilisé chez Google avant l'open-source (notamment pour le projet *Agentspace*, un orchestrateur multi-agents mentionné pendant Next) (Google Releases Open-Source Agent Development Kit for Multi-Agent AI Applications - InfoQ) (Google Releases Open-Source Agent Development Kit for Multi-Agent AI Applications - InfoQ). Son ouverture permet à la communauté d'avoir un framework complet pour des systèmes multi-agents *interopérables* dès maintenant. Au-delà de la communication, ADK offre des fonctionnalités avancées utiles en contexte A2A : *guardrails* déterministes pour contrôler la logique des agents, support du **streaming bidirectionnel audio/vidéo** pour des interactions plus naturelles, outillage de test et de débogage (CLI interactive, interface web de visualisation des états) (Google Releases Open-Source Agent Development Kit for Multi-Agent AI Applications - InfoQ). Ces capacités multimodales se marient bien avec A2A (qui est *modality-agnostic*) ([

```
Announcing the Agent2Agent Protocol (A2A)


- Google Developers Blog

](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=
```

- **LangGraph** – Ce framework open-source émergent propose une approche *graph-based* pour orchestrer des agents (chaque agent étant un nœud dans un graphe de tâches). LangGraph met l'accent sur la coordination et la planification dans des workflows multi-agents. En avril, les auteurs de LangGraph ont démontré une intégration poussée avec A2A : un exemple publié montre un **agent de conversion de devises** construit avec LangGraph qui expose ses compétences via A2A (A2A/samples/python/agents/langgraph/README.md at main - GitHub). Concrètement, LangGraph peut servir à modéliser les interactions entre agents (qui appelle qui et dans quel ordre) tandis qu'A2A sert de couche transport pour les appels effectifs entre ces agents. Un article Medium compare ainsi LangGraph et A2A en indiquant que *« LangGraph aide à modéliser le quand et quoi de l'interaction agent, A2A en définit le comment »* (LangGraph & A2A: Simulating Multi-Agent AI Collaboration for ...). On a donc une **complémentarité** : LangGraph fournit un plan d'exécution multi-agent (par exemple d'abord un agent A, puis déléguer à B, etc.), et A2A exécute ce plan en assurant la communication inter-agent standardisée. L'équipe LangGraph a contribué des *samples* dans le dépôt A2A pour montrer cette synergie (les développeurs peuvent s'inspirer de ces exemples pour brancher leurs propres agents LangGraph en A2A).

- **Crew.ai (CrewAI)** – CrewAI est une bibliothèque ouverte pour créer des *équipes d'agents* collaboratifs (appelés *crew*). Orientée application pratique, elle a été popularisée via des tutoriels (DeepLearning.AI, DataCamp, etc.) et vise à orchestrer des agents spécialisés dans des pipelines de tâche complexes. CrewAI figure parmi les frameworks cités par Google comme compatibles A2A (Build and manage multi-system agents with Vertex AI | Google Cloud Blog). En pratique, cela signifie qu'un agent développé avec CrewAI peut être exposé via un *agent host* A2A, ou

inversement intégrer des agents externes en tant que « coéquipiers ». Par exemple, une démonstration en avril montrait un *workflow RAG (Retrieval-Augmented Generation)* où CrewAI gérait l'appel séquentiel de plusieurs agents pour rechercher des infos puis rédiger un rapport, chaque interaction étant modulable via A2A (Building a Multi-Agent RAG Pipeline with Crew AI - Medium). CrewAI apporte une approche haut-niveau centrée sur la coordination métier (chaque agent a un rôle bien défini dans le workflow), et A2A lui donne l'infrastructure technique pour communiquer avec des agents hors du framework. L'adoption d'A2A par CrewAI en était encore à ses débuts en avril (quelques discussions sur leur forum l'évoquent), mais le simple fait qu'il soit listé comme exemple par Google Cloud montre que CrewAI s'inscrit dans l'écosystème naissant d'interopérabilité. On peut s'attendre à des connecteurs ou adaptateurs plus formalisés dans les mois suivants pour faciliter l'enregistrement d'un agent CrewAI comme serveur A2A.

- **Semantic Kernel (Microsoft)** – Un événement notable a été l'annonce, le 17 avril, que **Semantic Kernel (SK)** de Microsoft supporte désormais A2A (Integrating Semantic Kernel Python with Google's A2A Protocol | Azure AI Foundry Blog). Semantic Kernel est la librairie .NET/Python de Microsoft pour orchestrer des plugins et *skills* autour des LLM (initialement conçue pour le plugin OpenAI). L'équipe Azure AI Foundry a publié un billet de blog expliquant que *« Semantic Kernel parle maintenant A2A »*, permettant des collaborations d'agents *cross-cloud* (Azure ↔ Google) (Integrating Semantic Kernel Python with Google's A2A Protocol | Azure AI Foundry Blog). Microsoft a même contribué du code au dépôt A2A pour faciliter l'intégration (ils ont proposé une première version de package Python) (Integrating Semantic Kernel Python with Google's A2A Protocol | Azure AI Foundry Blog). Concrètement, un développeur utilisant SK peut dorénavant utiliser un **connecteur A2A** pour qu'un agent SK envoie des requêtes A2A ou se comporte en serveur. L'exemple fourni est un **agent de voyage** construit avec SK qui délègue les requêtes à deux sous-agents spécialisés (un *CurrencyExchangeAgent* pour les taux de change et un *ActivityPlannerAgent* pour les activités touristiques) via A2A (Integrating Semantic Kernel Python with Google's A2A Protocol | Azure AI Foundry Blog) (Integrating Semantic Kernel Python with Google's A2A Protocol | Azure AI Foundry Blog). Chaque sous-agent est implémenté en SK mais communique par HTTP JSON-RPC avec l'agent principal. Cette intégration par Microsoft est doublement importante : elle valide l'intérêt d'A2A au-delà de l'écosystème Google, et elle montre que d'autres clouds majeurs embrassent l'idée d'un standard commun pour éviter le verrouillage. En avril, SK proposait cette intégration comme *expérimentale* (reposant sur le code d'exemple A2A en l'absence de SDK officiel) (Integrating Semantic Kernel Python with Google's A2A Protocol | Azure AI Foundry Blog), mais l'initiative a été saluée comme un pas vers l'**interopérabilité multi-cloud** des agents.

- **Autres bibliothèques et frameworks** – D'autres acteurs du domaine agentique ont indiqué leur compatibilité ou intention de support : par exemple **LangChain** (framework très répandu pour agents outillés) a été un contributeur à la spec A2A ([

```
Announcing the Agent2Agent Protocol (A2A)


- Google Developers Blog
```

```
](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=
```

Retours d’expérience, outillage & bonnes pratiques

Dès le mois d’avril, de premiers retours d’expérience et contenus pédagogiques sur A2A ont fleuri, issus aussi bien de Google que de la communauté. Ils permettent de dégager à la fois les **atouts concrets** du protocole et les **défis pratiques** rencontrés lors de sa mise en œuvre initiale.

- **Tutoriels et démonstrations techniques** – Google a publié un guide pas-à-pas très détaillé pour *“Getting Started with A2A”*, illustrant la mise en place d’un mini-système multi-agent complet (Getting Started with Google A2A: A Hands-on Tutorial for the Agent2Agent Protocol | by Heiko Hotz | Google Cloud - Community | Apr, 2025 | Medium). Rédigé par un ingénieur Cloud, ce tutoriel explique comment lancer trois micro-agents (en utilisant LangGraph, CrewAI et ADK) et les faire collaborer via une interface unique grâce à A2A. L’article couvre la configuration des endpoints, la création des fichiers Agent Card, et la **connexion des agents dans un chat unifié** (enregistrement manuel des URLs puis envoi de prompts, etc.) (Getting Started with Google A2A: A Hands-on Tutorial for the Agent2Agent Protocol | by Heiko Hotz | Google Cloud - Community | Apr, 2025 | Medium) (Getting Started with Google A2A: A Hands-on Tutorial for the Agent2Agent Protocol | by Heiko Hotz | Google Cloud - Community | Apr, 2025 | Medium). Ce type de guide a été précieux pour la communauté, montrant de manière concrète le *flow* complet A2A (du *handshake* initial via l’Agent Card jusqu’à la complétion d’une tâche multi-agent). En complément, des blogueurs indépendants ont partagé leurs propres expériences : **Renu Khandelwal** par exemple a décrit comment implémenter un serveur A2A en Python Flask couplé à un agent LangGraph (From Silos to Synergy: Agent to Agent(A2A) Protocol in Action | by Renu Khandelwal | Apr, 2025 | Medium). Elle y décortique le format des messages échangés et insiste sur l’importance de ne pas exposer d’information sensible (chaque agent reste une “boîte noire” qui ne partage ni son *prompt* interne ni son état mental, seulement les infos explicites autorisées) (From Silos to Synergy: Agent to Agent(A2A) Protocol in Action | by Renu Khandelwal | Apr, 2025 | Medium). Ce point, souligné aussi par Google, clarifie que A2A permet une collaboration *contrôlée* (les agents s’échangent des tâches et des résultats, pas leurs pensées internes) – ce qui évite notamment les fuites de prompts ou le mélange de mémoires entre agents, problématiques en termes de confidentialité.

- **Bénéfices constatés** – Les premiers retours confirment les bénéfices attendus de l’approche standardisée. D’une part, les développeurs apprécient la **simplicité d’intégration** : en s’appuyant sur HTTP et JSON, A2A s’insère facilement dans des stacks web existantes, et nombre d’outils (proxy, client HTTP) peuvent être réutilisés ([

```
Announcing the Agent2Agent Protocol (A2A)


- Google Developers Blog
```

```
](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=
```

- **Limites et difficultés rencontrées** – Malgré l'enthousiasme, les retours d'avril soulignent aussi des **points d'attention** liés à la maturité encore limitée du protocole. D'abord, l'absence de **découverte automatique** est un frein pratique : dans les démos, il faut manuellement indiquer l'URL des agents à l'orchestrateur (via un fichier de config ou une saisie utilisateur) (Getting Started with Google A2A: A Hands-on Tutorial for the Agent2Agent Protocol | by Heiko Hotz | Google Cloud - Community | Apr, 2025 | Medium). En production, on aimerait un registre ou un mécanisme de découverte dynamique. L'équipe A2A en est consciente et mentionne ce besoin dans les évolutions à venir (GitHub - google/A2A: An open protocol enabling communication and interoperability between opaque agentic applications.). Ensuite, la **gestion des identités et autorisations** n'est pas totalement standardisée dans la v0.1. Par défaut, A2A s'aligne sur les schémas d'auth d'OpenAPI (API key, OAuth2…) ([

```
    Announcing the Agent2Agent Protocol (A2A)


    - Google Developers Blog

](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=
```

- **Outils d'accompagnement et bonnes pratiques** – Pour adresser ces défis, quelques outils commencent à apparaître. Google a intégré à l'ADK une **interface de monitoring** où l'on peut voir chaque étape de la décision d'un agent et les messages qu'il envoie/reçoit (Google Releases Open-Source Agent Development Kit for Multi-Agent AI Applications - InfoQ). Couplé à A2A, cela permet de déboguer visuellement un scénario multi-agent (en suivant le cheminement d'une tâche de l'agent A au B, etc.). Des sociétés d'observabilité partenaires, comme **Arize AI**, envisagent d'étendre leurs solutions à la **traçabilité inter-agents** : Arize, spécialiste du suivi de modèles en production, a salué A2A et pourrait offrir des moyens de suivre les métriques de performance de chaque agent collaboratif ([

```
    Announcing the Agent2Agent Protocol (A2A)


    - Google Developers Blog

](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=
```

En résumé, les retours d'avril 2025 sur A2A sont globalement positifs quant à sa **pertinence et son potentiel**, tout en soulignant que nous en sommes aux **prémices de sa courbe de maturité**. L'outillage va devoir s'enrichir (meilleure découverte, SDK officiels, outils de test) et des **patterns d'architecture** vont émerger avec l'expérience. Les pionniers recommandent d'adopter

progressivement A2A sur des cas concrets tout en respectant les bonnes pratiques de conception d'API et de sécurité, gages de succès pour ce nouveau paradigme.

Sécurité & gouvernance

Sécurité par design – Le protocole A2A a été conçu en mettant l'accent sur la sécurité **dès sa conception**. L'un de ses principes directeurs est d'être *« secure by default »*, avec un support des schémas d'authentification et d'autorisation de grade entreprise équivalent à ceux d'OpenAPI dès la sortie ([

```
        Announcing the Agent2Agent Protocol (A2A)


        - Google Developers Blog

    ](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=in
```

Gouvernance des permissions et des actions – Au niveau *métier*, la gouvernance consiste à contrôler **qui peut faire quoi via A2A**. Dans un scénario multi-agent, cela signifie s'assurer qu'un agent client n'abuse pas des capacités d'un agent serveur au-delà de ce qui lui est permis. Par exemple, un agent "Support Client" ne devrait déclencher via A2A qu'un remboursement plafonné et approuvé, et non un virement libre. Actuellement, A2A délègue cette gouvernance aux implémenteurs : c'est au propriétaire de chaque agent de valider les requêtes reçues. Néanmoins, la communauté identifie le besoin de standardiser certaines pratiques. Une proposition déposée (issue #19) recommande un mécanisme de **délégation d'autorisation utilisateur** : l'idée est qu'un agent client passe un jeton représentant l'utilisateur final et les permissions accordées, que l'agent serveur peut vérifier avant exécution (Delegated User Authorization for Agent2Agent Servers · Issue #19 · google/A2A · GitHub) (Delegated User Authorization for Agent2Agent Servers · Issue #19 · google/A2A · GitHub). Cela ressemble au modèle OAuth où une appli obtient un *access token* pour agir pour le compte d'un user. Appliqué à A2A, cela permettrait par exemple qu'un agent "RH" délègue à un agent "Finance" une action de payer une facture, mais dans le cadre de l'identité d'un employé particulier (avec ses droits budgétaires). En avril, ce n'est qu'une proposition, mais elle reflète une exigence de gouvernance forte des entreprises : **garder le contrôle** sur les actions que des agents autonomes prennent entre eux, en particulier lorsque celles-ci engagent des données ou opérations sensibles. D'un point de vue gouvernance plus large, Google a également rallié des partenaires conseil (BCG, Deloitte, Wipro…) pour définir des *guidelines* de déploiement responsable (Building the industry's best agentic AI ecosystem with partners | Google Cloud Blog). Les discussions incluent l'établissement de **quotas** ou limites d'usage : par exemple, limiter le nombre de tâches qu'un agent peut déléguer par minute, pour éviter des effets de surcharge ou des coûts imprévus (un agent qui appellerait 1000 fois un agent facturé à l'usage, par inadvertance ou malveillance). Aucune fonctionnalité native de quota n'est présente dans A2A, mais on peut s'appuyer sur des API Gateway en amont pour cela. De même, en cas de **vulnérabilité (CVE)** découverte dans la façon dont un agent traite les messages A2A (ex: injection dans un champ JSON non filtré, ou attaque par fichiers malveillants via `FilePart`), la

correction dépendra de chaque implémentation. D'où l'importance de suivre les bonnes pratiques de sécurité API (validation de schéma JSON côté serveur, restrictions sur les tailles de fichiers acceptés, etc.).

Au 30 avril 2025, **aucune faille de sécurité publique** n'avait été remontée sur A2A (ce qui n'est pas surprenant étant donné sa jeunesse). Toutefois, un œil attentif est porté sur les risques de *prompt injection* et de *jailbreak indirect* : si un agent A envoie un texte à B, pourrait-il contenir un prompt malveillant amenant B à se comporter de façon non désirée ? Ce genre de problématique est surtout du ressort de la logique des agents (les LLM sous-jacents) plutôt que du protocole lui-même. Néanmoins, la gouvernance pourrait inclure des politiques de filtrage de contenu inter-agent (ex: un agent de niveau élevé pourrait "sanitiser" les instructions avant de les transmettre à un sous-agent). Enfin, sur le plan de la **standardisation externe**, Google a indiqué travailler *"en partenariat ouvert"* avec l'industrie sur ces enjeux de sécurité ([

```
        Announcing the Agent2Agent Protocol (A2A)


        - Google Developers Blog

    ](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=To
```

Conseils et enseignements (adoption, sécurité, mise à l'échelle)

Après ce premier mois riche en retours, voici quelques recommandations concrètes dégagées pour les équipes souhaitant adopter A2A :

- **Commencer petit et concret** : Il est conseillé de débuter par un cas d'usage ciblé impliquant 2 ou 3 agents maximum, afin de se familiariser avec le protocole. Par exemple, mettre en place un agent « question-answer » qui fait appel via A2A à un agent « base de connaissances » séparé. Ce type de *pilot* permet de comprendre le cycle *découverte → requête → réponse* et de valider l'infrastructure (networking, auth) en conditions réelles, sans complexité excessive. Google souligne qu'il est déjà possible de **déployer A2A en prod sur Vertex AI** (le runtime gère la scalabilité), donc on peut envisager un petit déploiement isolé dans un environnement cloud pour test (Build and manage multi-system agents with Vertex AI | Google Cloud Blog). L'expérience acquise servira de modèle pour étendre à davantage d'agents.

- **Utiliser les frameworks adaptés** : Plutôt que de coder tout from-scratch, il est souvent plus efficace de s'appuyer sur les SDK/frameworks disponibles. Si votre équipe est orientée Google Cloud, l'**ADK** est un choix naturel pour créer des agents compatibles rapidement, avec en prime les outils de test et déploiement intégrés (Build and manage multi-system agents with Vertex AI | Google Cloud Blog) (Build and manage multi-system agents with Vertex AI | Google Cloud Blog). Si vous avez déjà des agents développés avec LangChain, CrewAI ou autre, guettez les mises à jour : des modules d'intégration A2A pourraient être publiés rapidement (plusieurs sont en beta).

L'intégration de **Semantic Kernel** par Microsoft suggère que même hors écosystème Google, des librairies faciliteront la connexion d'agents existants à A2A (Integrating Semantic Kernel Python with Google's A2A Protocol | Azure AI Foundry Blog). L'objectif est de **minimiser le code "colle"**, pour se concentrer sur la logique métier de vos agents.

- **Sécuriser dès le départ** : Appliquez d'emblée une politique de sécurité stricte sur vos endpoints A2A. En pratique, cela signifie : exiger une authentification (par token, certificat client…) sur l'URL de votre agent, et vérifier ces identités côté serveur. Évitez de déployer un agent A2A sur internet sans protection, tout comme vous ne le feriez pas pour une API REST publique. Si possible, limitez dans un premier temps les communications A2A à un **réseau interne** ou un VPN corporatif : ainsi seuls vos agents "maison" peuvent communiquer, ce qui réduit la surface d'attaque le temps de gagner en maturité. Mettez en place du **logging** sur les requêtes A2A (qui a appelé, quelle tâche a été demandée) pour pouvoir auditer les interactions. En cas de comportement anormal d'un agent, ces logs seront précieux pour retracer l'incident. Enfin, tenez-vous informés des mises à jour du protocole sur la partie sécurité : par exemple, si un champ pour annoncer la prise en charge d'OIDC apparaît dans la spec, envisagez de l'utiliser et de vous brancher sur votre fournisseur d'identité d'entreprise – cela alignera la sécurité d'A2A avec vos pratiques existantes.

- **Cloisonner et limiter les permissions** : Du point de vue gouvernance, définissez clairement le **périmètre d'action de chaque agent** et implémentez des garde-fous. Par exemple, si un agent peut potentiellement effectuer des actions critiques (suppression de données, transactions financières), ne l'exposez pas tel quel via A2A sans contrôles. Vous pouvez insérer une couche de validation métier : l'agent reçoit la requête A2A, mais avant d'exécuter, il appelle un service d'autorisation interne ou vérifie un jeton incluant des claims de permission. Cette approche "zéro confiance" interne évitera les mauvaises surprises si un jour un agent client tente une action inattendue (que ce soit un bug ou une compromission). De plus, profitez des capacités du protocole pour demander confirmation si nécessaire : A2A permet à un agent de répondre qu'une *input additionnelle* est requise (GitHub - google/A2A: An open protocol enabling communication and interoperability between opaque agentic applications.). Un pattern de sécurisation peut être qu'un agent ne réalise une action sensible qu'après une confirmation explicite reçue (sous forme de message de l'utilisateur humain ou d'un autre agent superviseur). En d'autres termes, utilisez la flexibilité du *dialogue agent-agent* pour instaurer un **contrôle en plusieurs étapes** quand c'est pertinent.

- **Superviser et mesurer** : Comme pour toute application distribuée, la mise à l'échelle d'un système A2A nécessite de la supervision. Surveillez les latences des appels A2A, les taux de succès/erreur des tâches, et la charge sur chaque agent. Des solutions APM existantes peuvent être configurées pour tracer les requêtes HTTP entre vos agents. Il est aussi recommandé de définir des **SLAs internes** : par ex, un agent de recherche doit répondre en <## Mise en perspective : impact sur l'architecture multi-agents et l'écosystème
  En filigrane, l'émergence d'A2A traduit un changement de paradigme dans la conception des systèmes d'IA. Jusqu'ici, les agents opéraient souvent en silos, chaque plateforme proposant *son* agent autonome difficilement connectable aux autres – une situation décrite comme des *"îles*

d'intelligence" (The AI Agent Breakthrough: Why Google's A2A + Anthropic's MCP Are the Power Couple of 2025 | by Hossen | Apr, 2025 | Level Up Coding). Avec A2A, ces îles peuvent enfin être reliées par un **langage commun**. Couplé au protocole **MCP d'Anthropic** (qui adresse l'accès aux outils et données externes des LLMs), A2A complète le puzzle en résolvant le volet communication inter-agents ([

```
    Announcing the Agent2Agent Protocol (A2A)


    - Google Developers Blog
](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=
    Announcing the Agent2Agent Protocol (A2A)


    - Google Developers Blog
](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=
    Announcing the Agent2Agent Protocol (A2A)


    - Google Developers Blog
](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=
```

Pour les développeurs et architectes, cela se traduit par une **montée en productivité et en flexibilité** à moyen terme. D'une part, A2A réduit le *vendor lock-in* : un agent construit sur une plateforme X pourra interagir avec un agent Y, évitant d'enfermer l'utilisateur dans un seul écosystème (What is The Agent2Agent Protocol (A2A) and Why You Must Learn It Now). Cela encourage l'évaluation *objective* de chaque agent/LLM pour une tâche donnée (on pourra combiner un agent conversationnel d'un fournisseur avec un agent analyste d'un autre, si c'est optimal). D'autre part, la possibilité de **réutiliser des agents existants** (internes ou fournis par des tiers) au lieu de tout redévelopper permet de gagner du temps. Par exemple, une équipe pourra intégrer l'agent ServiceNow ITSM de son entreprise avec un agent d'assistance Google sans écrire de glue spécifique – chacun "parle A2A" et comprend les capacités de l'autre. On peut imaginer à terme un **marché d'agents A2A** interchangeables, comme il existe des marketplaces d'APIs. Dans cette optique, les compétences de demain pour les développeurs incluront l'**orchestration d'agents** autant que le développement d'un agent individuel. Il faudra savoir concevoir des *conversations inter-agents*, un peu à la manière de l'architecture de microservices asynchrones. Les retours d'avril montrent que cette orchestration est tout un art (choisir quel agent fait quoi, comment gérer les erreurs, etc.), mais les gains potentiels en valent la chandelle : plus besoin de "forcer" un seul agent à tout faire, on peut allouer la bonne sous-tâche au bon agent, ce qui améliore la qualité des résultats et la robustesse du système global.

Sur le plan de l'industrie technologique, le lancement d'A2A et l'enthousiasme qu'il suscite suggèrent une **convergence vers des normes ouvertes** pour l'IA agentique. Voir des concurrents comme Google et Microsoft contribuer à un même protocole est révélateur : il y a une prise de conscience que sans standards, le potentiel des agents restera bridé. L'initiative rappelle d'autres efforts passés de normalisation (on pense au passage de CORBA propriétaires vers SOAP/HTTP puis REST, etc.). Il est probable que si A2A continue sur sa lancée, des organismes de standardisation formels comme l'**OASIS** ou le **W3C** pourraient s'en saisir. A2A pourrait par exemple s'inscrire dans une démarche plus large d'**interopérabilité des agents sur le web**, où le fichier `/.well-known/agent.json` deviendrait un standard de facto (à l'image de `robots.txt` ou `openid-configuration` pour OIDC). En avril 2025, on n'en est pas encore là – A2A est en mode incubation open-source – mais l'existence d'un site communautaire (a2aprotocol.net) et de discussions publiques foisonnantes sur Slack, HackerNews, Reddit, etc., montre qu'une **communauté standardisante** est déjà à l'œuvre. Par ailleurs, on observe une effervescence autour des événements dédiés : au-delà de Next '25, qui a consacré plusieurs sessions aux agents (avec l'annonce d'ADK, de connecteurs Vertex et la présence de nombreux partenaires sur scène), des **webinaires et meetups** se sont tenus tout au long du mois d'avril. Par exemple, un webinaire intitulé *"The Future of AI Collaboration – Deep Dive into A2A & MCP"* mi-avril a réuni plus de 500 participants (architectes Cloud, data scientists) pour échanger sur les cas d'usage concrets et les bonnes pratiques d'implémentation d'A2A en entreprise. De même, des conférences en ligne chez des partenaires (InfoQ Live, podcasts techniques) ont mis en lumière A2A, signe que le sujet dépasse le cadre de Google pour devenir une **préoccupation de tout l'écosystème IA**.

En conclusion, l'Agent-2-Agent protocole et son écosystème naissant semblent poser les fondations d'une **nouvelle ère pour les systèmes multi-agents**. Si les promesses se concrétisent, on pourrait assister dans les prochains trimestres à l'émergence d'"architectures agentiques" aussi transformantes que le fut en son temps l'architecture orientée services. Les développeurs pourront piocher des agents spécialisés comme on utilise aujourd'hui des services cloud, en les orchestrant via A2A pour construire des solutions complexes, adaptatives et interopérables par défaut. Bien sûr, des défis subsistent – maturité technique, sécurité avancée, gouvernance fine – mais l'engagement conjoint de dizaines d'acteurs majeurs et de la communauté open-source est de bon augure pour les relever. Avril 2025 aura été le témoin du **ralliement de l'industrie** autour d'une vision commune : celle d'AI agents hétérogènes *collaborant sans frontières*, au bénéfice de la productivité et de l'innovation. Les mois suivants confirmeront si A2A s'impose comme le standard incontournable de cette vision – mais d'ores et déjà, son avènement catalyse un formidable élan vers des **agents intelligents interconnectés**, ouvrant la voie à des applications et interactions jusqu'alors inenvisageables. L'histoire de l'IA retiendra peut-être ce printemps 2025 comme le moment où nos agents ont appris à parler un même langage – et ce faisant, ont décuplé leur pouvoir de transformation.

Sources : Google Developers Blog (9 avr. 2025) ([

```
Announcing the Agent2Agent Protocol (A2A)

- Google Developers Blog
```

](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=To

Announcing the Agent2Agent Protocol (A2A)

- Google Developers Blog

](https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/#:~:text=,%