

**Consultation publique de l'ARCEP sur l'interopérabilité et la portabilité des services  
d'informatique en nuage (cloud) Juillet 2025 (« Consultation »)**

**Questions relatives à la proposition de favoriser la mise à disposition d'API stables et documentées**

**Question 3** : Le délai de préavis proposé vous semble-t-il approprié ? Dans le cas contraire, quel délai préconisez-vous ? Pourquoi ?

1. AWS reconnaît que la stabilité des API destinées aux clients est fondamentale pour une utilisation efficace des services cloud par les clients. Ces API doivent répondre à des exigences rigoureuses et aux attentes des clients en matière de résilience, de sécurité, de facilité d'utilisation et de stabilité, car elles sont intégrées dans leurs solutions. Elles doivent être sécurisées, intuitives, adaptées à un large éventail de cas d'utilisation et compatibles avec de multiples services, protocoles et langages de codage. Lorsque les clients intègrent ces API dans leurs applications et leurs systèmes, les changements peuvent impacter significativement leurs opérations. AWS partage donc les préoccupations précédemment exprimées à l'ARCEP selon lesquelles des changements soudains et fréquents des API destinées aux clients pourraient limiter la capacité de ces derniers à maintenir et à adapter leur architecture.
2. AWS fait déjà preuve d'un fort engagement en faveur de la stabilité des API et n'apporte pas de modifications substantielles non-rétrocompatibles à ses APIs destinées aux clients sans préavis [SDA]<sup>1</sup>. AWS fournit également des conseils aux clients sur la manière d'obtenir les mêmes résultats avec la version mise à jour d'une API donnée, lorsque les modifications affectent les fonctionnalités existantes. Cependant, bien qu'AWS prévoie déjà un préavis de [SDA], pour une mise en œuvre efficace des lignes directrices de l'ARCEP, il est essentiel de définir de manière appropriée ce qui constitue une « mise à jour majeure » nécessitant un préavis et de prévoir des exceptions appropriées lorsque cela est nécessaire, par exemple pour remédier à des failles de sécurité. Une interprétation trop large de cette notion pourrait entraver inutilement la capacité des fournisseurs à apporter des améliorations de routine, à mettre en œuvre des améliorations mineures ou à résoudre des problèmes techniques ou de sécurité en temps utile, ce qui finirait par nuire à la qualité du service et à l'innovation. AWS propose qu'une mise à jour majeure soit comprise comme une modification qui modifierait de manière significative le comportement des API destinées aux clients, de telle sorte que cela affecterait les charges de travail des clients. Plus précisément, cela devrait inclure les modifications qui : (1) après la modification, provoquent l'échec d'une requête/opération précédemment réussie, (2) obligent les clients à prendre des mesures et à repenser leur architecture pour éviter une interruption du service ou (3) suppriment une caractéristique ou une fonctionnalité du service utilisée par les clients.
3. Tout cadre réglementaire régissant les modifications apportées aux API destinées aux clients doit trouver un équilibre subtil entre la stabilité et d'autres considérations essentielles, notamment la sécurité et l'innovation. Les services peuvent être amenés à modifier leurs API pour diverses raisons, autres que des modifications de fonctionnalités, telles que l'application de correctifs, la correction de défauts, la résolution de problèmes de sécurité ou la mise à niveau pour suivre les dernières avancées en matière de dépendances en amont. Sur cette base, AWS recommande d'adopter un cadre plus souple qui prévoit des exceptions claires à toute obligation de notification lorsque la modification est nécessaire pour remédier à des failles de sécurité ou à des situations d'urgence, pour répondre à des exigences légales ou réglementaires, pour répondre à des revendications de tiers relatives à la propriété intellectuelle ou pour atténuer les risques pour l'intégrité du service. AWS souligne que les fournisseurs de services cloud cherchent naturellement à minimiser les modifications non rétro-compatibles, car elles créent des tensions avec les clients, augmentent la complexité opérationnelle, risquent d'entraîner la perte de clients et affectent la réputation et la confiance accordée au fournisseur. Par conséquent, les fournisseurs sont déjà

<sup>1</sup> Voir le Contrat Client AWS (<https://aws.amazon.com/agreement/>), Section 1.5

naturellement incités à minimiser ces modifications tout en maintenant la sécurité et l'innovation des services.

**Question 4 :** L'adoption généralisée de la spécification OpenAPI vous semble-t-elle souhaitable, notamment afin de permettre une documentation des API harmonisée ?

**Question 5 :** Avez-vous d'autres commentaires sur cette recommandation ?

4. AWS soutient l'utilisation de standards ouvertes et indépendantes pour décrire les API et intègre les spécifications OpenAPI de différentes manières<sup>2</sup>. Par exemple, Amazon API Gateway permet aux clients d'importer et d'exporter des définitions d'API en utilisant OpenAPI 3.0. AWS fournit également des outils et des ressources pour aider les développeurs à travailler avec OpenAPI, notamment la possibilité de générer des spécifications OpenAPI à partir de ses modèles Smithy natifs, comme détaillé au paragraphe 12 ci-dessous. Toutefois, bien qu'AWS reconnaisse OpenAPI comme une norme de documentation précieuse, AWS n'est pas d'accord avec la proposition de l'ARCEP d'imposer l'adoption des définitions d'une spécification particulière dans le cadre de la mise en œuvre technique de toutes les API destinées aux clients, car cela créerait des limitations qui sont *in fine* nuisible à l'expérience client lors de l'utilisation des services cloud. A tout le moins, AWS invite vivement l'ARCEP à (i) reconnaître que certains services cloud ne peuvent pas du tout être représentés avec OpenAPI, (ii) reconnaître que différents services cloud sont susceptibles de nécessiter des approches variées en matière de spécification et de documentation des API afin d'exploiter pleinement leurs capacités, et que de nombreux clients auraient encore besoin de fonctionnalités allant au-delà de ce que OpenAPI peut décrire, et (iii) pour les services qui peuvent être représentés avec OpenAPI, accepter la disponibilité d'outils permettant la conversion vers une version largement adoptée du format OpenAPI soient considérés comme suffisants pour satisfaire l'exigence de documentation des API de l'ARCEP. Cette approche permettrait à l'ARCEP d'atteindre ses objectifs d'harmonisation tout en préservant la capacité des fournisseurs à utiliser les outils les mieux adaptés à leurs services, ce qui profiterait *in fine* aux clients grâce à une innovation et des performances accrues. Outre l'impact négatif sur les fournisseurs et les clients, AWS ne pense pas non plus que l'ARCEP ait besoin d'imposer une telle normalisation en vertu du Data Act et de la Loi SREN.

*Le Data Act et la Loi SREN n'imposent pas de normalisation pour les interfaces destinées aux clients.*

5. Tout d'abord, AWS note que le Data Act exige que les fournisseurs disposent « d'interfaces ouvertes » pour des services cloud qui ne sont pas considérés comme « d'éléments d'infrastructure ». « Interfaces ouvertes » est un terme plus large qui va au-delà des API. Si AWS met à disposition des API à destination des clients pour ses services, AWS prend plusieurs autres mesures pour faciliter l'interopérabilité et prendre en charge la communication entre systèmes et l'échange de données entre différents fournisseurs. Par exemple, AWS met à disposition un grand nombre de ses kits de développement logiciel (*software development kits* - « SDK ») sous licence open source et utilise des protocoles, des interfaces, des API et des formats de données ouverts pour l'ensemble de ses services. Les SDK sont des ensembles d'outils de développement spécifiques à un langage de programmation destinés aux développeurs d'applications et de logiciels. Ils fournissent aux développeurs des modules, des composants, des packages et des outils prêts à l'emploi pour créer, tester et déployer des applications de logiciels. Les SDK comprennent souvent des composants tels que des débogueurs, des compilateurs et des bibliothèques permettant de créer du code qui s'exécute sur un système d'exploitation ou un langage de programmation spécifique. Ces composants permettent aux développeurs de gagner un temps considérable qui était auparavant consacré au codage et au débogage à partir de zéro. Les SDK contiennent également des ressources

<sup>2</sup> Les clients AWS peuvent définir leurs API HTTP dans le service Amazon API Gateway à l'aide des fichiers de définition OpenAPI 3.0 (<https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-open-api.html>). Les outils AWS tels que SAM (Serverless Application Model) et CDK (Cloud Development Kit) permettent aux développeurs de déployer des API à l'aide des spécifications OpenAPI (<https://aws.amazon.com/blogs/opensource/create-restful-apis-on-aws-with-openapi-specification-with-no-coding/>).

telles que de la documentation, des tutoriels et des guides, ainsi que des API et des structures cadres pour accélérer le développement d'applications. En outre, AWS met publiquement à disposition les modèles API utilisés pour générer ces SDK, ainsi que des spécifications open source détaillant la manière dont les SDK sont créés à partir de ces modèles et la logique qui sous-tend leur conception. Cette approche offre non seulement des outils préconstruits, mais permet également aux développeurs de comprendre comment les services AWS sont structurés, ce qui favorise une culture open source dans laquelle les clients peuvent améliorer ou personnaliser les intégrations avec les services AWS.

6. En matière de cloud computing, les développeurs utilisent les SDK pour s'intégrer à un environnement cloud dans le langage de leur choix ou pour créer et déployer des applications cloud. Les SDK d'AWS permettent aux clients d'utiliser plus facilement les services AWS dans leurs applications grâce à une API adaptée au langage de programmation de leur choix. AWS fournit des SDK pour de technologies et langages de programmation très répandus, facilitant ainsi le recours aux services AWS par les clients depuis leurs applications dans cette langue ou cette technologie. Les SDK d'AWS simplifient également le codage en fournissant des API spécifiques à chaque langage pour les services AWS. Par exemple, les clients peuvent utiliser AWS SDK pour .NET, AWS SDK pour Python (Boto3) et AWS SDK pour Ruby lorsqu'ils créent des applications web sur AWS. Par conséquent, AWS recommande vivement que la position de l'ARCEP concernant la conformité aux exigences du Data Act en matière d'interfaces ouvertes ne se limite pas aux API destinées aux clients, mais tienne compte de la gamme plus large d'outils et de ressources fournis pour faciliter la communication.
7. En outre, le Data Act prévoit l'adoption de spécifications communes et de normes harmonisées par la Commission Européenne uniquement pour les services de traitement des données qui ne sont pas qualifiés d'« éléments d'infrastructure ». Le Règlement ne contient aucune disposition prévoyant l'adoption d'une norme ou d'une spécification obligatoire sur la manière dont les API destinées aux clients doivent être conçues ou documentées. Au contraire, le Règlement sur les données prévoit une certaine souplesse à cet égard en faisant référence aux « interfaces ouvertes », comme expliqué ci-dessus, où les fournisseurs sont tenus (i) de mettre les « interfaces ouvertes » à la disposition de tous leurs clients et des fournisseurs destinataires dans la même mesure et gratuitement afin de faciliter le processus de changement, et (ii) de veiller à ce que ces interfaces incluent des informations suffisantes sur le service concerné pour permettre le développement de logiciels capables de communiquer avec les services, aux fins de la portabilité et de l'interopérabilité des données. L'article 28.II.3 de la Loi SREN contient une disposition similaire reflétant la même exigence.
8. Il convient de relever que les API destinées aux clients d'AWS, ainsi que celles proposées par de nombreux autres fournisseurs, sont déjà largement utilisées par les clients et sont bien documentées sans que l'adoption d'OpenAPI soit une exigence normative. Il n'existe aucune preuve qu'il existe un obstacle concret et spécifique à l'interopérabilité qui pourrait être efficacement éliminé en rendant obligatoire l'utilisation de l'OpenAPI, qui, comme décrit ci-après, ne créerait pas d'autres conséquences imprévues. En pratique, la diversité actuelle des spécifications API n'a pas entravé l'interopérabilité entre différents fournisseurs de cloud.

*Imposer l'usage d'OpenAPI risque d'être contreproductif et de limiter la capacité des clients à utiliser toutes les fonctionnalités offertes par un certain service cloud.*

9. Tout d'abord, l'interopérabilité entre services cloud du même type, c'est-à-dire la capacité de services du même type à échanger et à utiliser des données entre eux, telle que définie à l'article 2(40) du Data Act, ne dépend pas des formats de documentation des API. Cela est clairement démontré dans les services de stockage de cloud. Par exemple, l'API d'Amazon S3, qui est antérieure à OpenAPI, a été largement adoptée pour le stockage d'objets, avec de nombreux fournisseurs proposant des interfaces compatibles avec S3, quelle que soit leur documentation API. À l'inverse, même lorsque les services utilisent la même norme de documentation API, ils peuvent rester incompatibles. Par exemple, de nombreux fournisseurs s'appuient sur des personnalisations OpenAPI qui ne sont pas normalisées et peuvent nécessiter des personnalisations de la part des clients pour y avoir recours, pour lesquelles l'harmonisation n'aiderait pas.

10. Les fournisseurs de services cloud développent leurs spécifications API et leurs approches en matière de documentation en fonction de l'architecture de leurs services, des besoins de leurs clients et de leurs capacités technologiques. Le paysage des services cloud est diversifié et évolue rapidement, avec les fournisseurs qui innovent en permanence pour répondre aux besoins complexes et variés de leurs clients. Dans ce contexte, si OpenAPI est l'un des standards industriels existants pouvant être utilisé pour la documentation des API, OpenAPI présente toutefois des limitations techniques intrinsèques qui la rendent inadaptée à la description de nombreux services cloud complexes. Pour cette même raison, OpenAPI n'est pas adopté par l'ensemble des fournisseurs pour la totalité des services qu'ils proposent. Ces limitations font que l'obligation d'utiliser exclusivement les définitions OpenAPI restreindrait aussi la capacité des clients à utiliser pleinement les fonctionnalités proposées par un service cloud.
11. AWS, par exemple, a développé Smithy<sup>3</sup>, qui offre des avantages significatifs par rapport à OpenAPI en termes de flexibilité, d'expressivité et de capacité à prendre en charge des interactions de services complexes. Smithy s'appuie sur l'expertise acquise par AWS dans le développement d'outils clients depuis près de vingt ans d'activité dans le cloud. Smithy ajoute des fonctionnalités telles que les métadonnées de ressources, un comportement client plus riche, des caractéristiques auto-descriptives et un système de validation personnalisable. Cela permet à AWS de décrire ses services de manière plus précise et plus complète, tout en conservant la rétrocompatibilité et en faisant évoluer les API destinées aux clients afin de répondre à leurs besoins changeants. De la même manière, Microsoft utilise TypeSpec et Google s'appuie sur Protocol Buffers. Chacun de ces choix reflète l'architecture sous-jacente, les besoins des clients qui utilisent ces API et les exigences opérationnelles de leurs environnements cloud respectifs. Ces approches spécifiques à chaque fournisseur ne sont pas des choix arbitraires, faits parce que « plus simple » pour le fournisseur mais plutôt des composants essentiels de la fourniture de services cloud. Ils permettent aux fournisseurs d'optimiser les performances et l'évolutivité de leurs services, tout en prenant en charge des opérations complexes telles que le streaming bidirectionnel. Ces spécifications sont conçues pour gérer efficacement différents protocoles et formats de données, permettant ainsi aux services d'évoluer tout en conservant leur rétrocompatibilité. De plus, elles sont soigneusement intégrées aux mécanismes de sécurité et d'authentification propres à chaque fournisseur, garantissant ainsi des services robustes et sécurisés.
12. Afin d'illustrer les raisons pour lesquelles un fournisseur de cloud peut choisir de développer son propre langage de définition d'API ou d'opter pour une spécification différente d'OpenAPI, voici les principales caractéristiques de Smithy et leur pertinence pour les utilisateurs des services AWS :
  - 12.1. *Smithy fonctionne de manière neutre, quel que soit le protocole utilisé* : Smithy a été développé avec pour objectif d'être agnostique des protocoles. Un langage de définition d'interface agnostique du protocole (*interface definition language* - « IDL ») est défini à un niveau d'abstraction supérieur à celui des données transmises sur le réseau. Cela permet à un IDL de se concentrer sur les fonctionnalités qu'il prend en charge et sur la manière dont elles sont exposées aux clients plutôt que sur les octets transmis sur le réseau. Grâce à cette fonctionnalité, Smithy décrit un plus large éventail de services, de métadonnées et de capacités. Par exemple, Smithy peut définir des services MQTT tels que l'API AWS IoT Jobs<sup>4</sup> et constitue la base des versions mises à jour des SDK AWS IoT. Ces IDL agnostiques permettent aux services de passer plus facilement à d'autres formats de sérialisation et protocoles de couche application. OpenAPI ne pourrait pas devenir agnostique de tout protocole, comme l'ont déjà indiqué ses mainteneurs<sup>5</sup>, car OpenAPI a été fondamentalement construite autour du protocole HTTP et conçue pour échanger des données dans des formats tels que JSON. La

---

<sup>3</sup> Voir : <https://smithy.io>

<sup>4</sup> Voir : <https://docs.aws.amazon.com/iot/latest/developerguide/jobs-api.html>

<sup>5</sup> Voir <https://github.com/OAI/OpenAPI-Specification/issues/777> pour une demande visant à rendre OpenAPI indépendant du protocole, où il est indiqué que « le style architectural REST est indépendant du protocole, contrairement à OpenAPI » et également <https://github.com/OAI/OpenAPI-Specification/issues/523>.

spécification API actuelle d'AWS permet d'optimiser la manière dont les requêtes sont envoyées entre les services et les clients, ce qui permet d'utiliser des formats de sérialisation plus efficaces que JSON lorsque cela est approprié. Le couplage d'OpenAPI à des protocoles spécifiques éliminerait cette flexibilité et pourrait limiter la capacité des fournisseurs à offrir des services innovants et performants à leurs clients.

- 12.2. *Smithy prend en charge le streaming bidirectionnel* : contrairement à OpenAPI, Smithy prend en charge la diffusion bidirectionnelle, une fonctionnalité essentielle pour décrire les services utilisant des flux de données dans les deux sens ou des formats d'API complexes (XML et chaînes de requêtes). Cette capacité est particulièrement importante pour des services AWS comme Amazon Bedrock, Lex ou Polly qui s'appuient sur le streaming bidirectionnel via HTTP/2, un cas d'usage qu'OpenAPI ne peut pas décrire de manière adéquate.
- 12.3. *Smithy définit des métadonnées d'entrée et d'erreur plus riches, permettant aux SDK de mieux gérer les erreurs et les problèmes de disponibilité* : Smithy sépare les caractéristiques et les attentes d'une erreur de la représentation HTTP d'une erreur. Smithy définit divers traits qui améliorent la capacité d'un client à réessayer intelligemment les requêtes ayant échoué. Les opérations de cycle de vie définies sur les ressources Smithy garantissent l'application correcte de ces traits. Cette exécution permet de s'assurer que les concepteurs d'API fournissent la meilleure interface possible pour leur service. Alors qu'OpenAPI définit les API HTTP et que le protocole HTTP définit implicitement des caractéristiques similaires, les auteurs d'API ne comprennent souvent pas les subtilités du protocole HTTP et ceux qui les comprennent ont souvent des opinions divergentes.
- 12.4. *Smithy génère un code sans ambiguïté* : le code généré par Smithy est plus prévisible que celui généré par OpenAPI. Cela s'explique par le fait que Smithy est un modèle hautement standardisé dans lequel chaque forme a un nom explicite. C'est également cette propriété qui confère à Smithy son puissant système de traits. Les générateurs de code OpenAPI doivent déduire les noms en fonction du contexte, ce qui peut donner lieu à des noms peu parlants tels que « `__listOfGroupCertificateAuthorityProperties` ». De plus, la prise en charge de l'héritage dans la génération de code par OpenAPI est principalement un comportement indéfini avec des résultats imprévisibles, car des fonctionnalités telles que `oneOf`, `allOf` et `discriminator` sont combinées. En outre, une prise en charge imprudente de l'héritage dans un système distribué entraînera des problèmes de rétrocompatibilité pour les clients<sup>6</sup>.
13. Les limitations d'OpenAPI évoquées précédemment expliquent pourquoi un fournisseur pourrait développer sa propre spécification ou préférer une alternative. À cet égard, une liste détaillée des fonctionnalités offertes par Smithy, mais non prises en charge par OpenAPI, est consultable à l'adresse <https://smithy.io/2.0/guides/model-translations/convertng-to-openapi.html#unsupported-features>. Du point de vue de l'utilisateur, l'essentiel est d'avoir accès à des services cloud fiables, performants et sécurisés, accompagnés d'une documentation exhaustive. L'utilisation d'un code client généré uniquement à partir d'une représentation OpenAPI de l'API peut présenter lacunes importantes, cruciales pour la fiabilité des opérations cloud. Cela inclut notamment des fonctionnalités de résilience comme les mécanismes de réessaie, ou des éléments de sécurité tels que la gestion des identifiants. En se limitant à l'utilisation de code client généré à partir des spécifications OpenAPI, les utilisateurs se priveraient de ces fonctionnalités essentielles, compromettant ainsi la robustesse et la sécurité de leurs interactions avec les services cloud

#### 14. [SDA]

15. En conclusion, OpenAPI ne serait pas adaptée à tous les services proposés par les fournisseurs de services cloud tels qu'AWS, qui stimulent l'innovation dans le secteur grâce à des centaines de services sophistiqués. Ces services nécessitent souvent un comportement personnalisé de la part des clients afin d'optimiser leurs capacités et de faire progresser l'innovation dans le secteur, ce qui

---

<sup>6</sup> Voir par exemple : <https://github.com/swagger-api/swagger-codegen/pull/5597>.



rend OpenAPI inadéquat si elle était imposée dans le cadre de la mise en œuvre technique de services aussi complexes. L'avis d'AWS sur la portée des exigences du Data Act étant réservé (voir paragraphes 5-8 ci-dessus), AWS propose une approche plus flexible, telle que décrite ci-dessous, si l'ARCEP adopte des exigences basées sur OpenAPI dans ses lignes directrices.

*Les IDLs personnalisés peuvent encore prendre en charge la conversion vers les spécifications OpenAPI pour les représentations basées sur HTTP du modèle, ce qui semble atteindre les objectifs d'harmonisation de l'ARCEP.*

16. Comme indiqué ci-dessus, AWS reconnaît la valeur d'OpenAPI en tant que norme de documentation et l'utilise déjà dans plusieurs services. Cependant, AWS estime que toute directive réglementaire devrait viser principalement à garantir que les fournisseurs de services cloud fournissent des informations de référence techniques suffisantes, notamment une documentation API complète dans un format lisible par machine, ce qui peut être réalisé de différentes manières.
17. Le processus de construction, de maintenance et d'évolution des API dans des environnements cloud complexes nécessite des techniques spécialisées adaptées à l'architecture et aux exigences opérationnelles propres à chaque fournisseur. Cela est démontré par le fait que les principaux fournisseurs de cloud, tels qu'Azure et AWS, ont développé de manière indépendante leurs propres langages de définition d'interface, non par choix, mais par nécessité. Ces structures cadres personnalisées ont été développées pour répondre aux exigences techniques spécifiques de leurs environnements cloud respectifs. Cependant, ces spécifications personnalisées ne doivent pas être considérées comme des systèmes fermés, car elles peuvent toujours être traduites en standards industriels ouverts si nécessaire. Par exemple, Smithy et TypeSpec peuvent tous deux être traduits en formats OpenAPI. Dans le cas de Smithy, les utilisateurs peuvent, s'ils le souhaitent, utiliser un plugin pour convertir les modèles Smithy en spécifications OpenAPI (versions 3.0.2 et 3.1.0), assurant la compatibilité avec des outils tels que SwaggerUI, Postman ou Amazon API Gateway<sup>7</sup>. Cependant, en raison des limitations d'OpenAPI mentionnées ci-dessus, cette conversion entraîne nécessairement une perte de fonctionnalités, notamment en ce qui concerne le traitement des métadonnées et les comportements avancés. Cela démontre une fois de plus la nécessité de permettre aux fournisseurs d'utiliser les outils qu'ils souhaitent pour leurs spécifications API.
18. Par conséquent, bien qu'AWS maintienne ses préoccupations fondamentales concernant l'imposition d'une norme de documentation spécifique pour des interfaces à destination des clients, si l'ARCEP devait poursuivre dans cette voie, AWS recommande vivement que la possibilité de traduire des structures cadres d'API personnalisées en formats OpenAPI dans le but de fournir des informations de référence techniques suffisantes dans un format standardisé lisible par une machine soit considérée comme suffisante pour répondre à cette exigence. Cette approche souple permet aux fournisseurs de maintenir une prestation de services optimale tout en garantissant que les clients ont accès à une documentation standardisée, lorsqu'ils en ont besoin.

*La proposition de l'ARCEP visant à rendre obligatoire la prise en charge de la dernière version d'OpenAPI soulève de sérieuses préoccupations.*

19. Exiger l'adoption de la dernière version d'OpenAPI obligerait les fournisseurs à mettre fréquemment à jour leur documentation et leurs outils API, ce qui pourrait potentiellement entraîner des problèmes d'instabilité et de compatibilité. Par exemple, la transition d'OpenAPI 3.0 à 3.1 a introduit des changements majeurs qui ont affecté de nombreux outils et implémentations existants. Le paysage général des outils API est souvent en retard par rapport à la dernière spécification OpenAPI, et ce pour des raisons valables. De nombreux outils prennent encore principalement en charge OpenAPI 3.0.x, alors que la version 3.1.0 a été publiée en 2021. Imposer la dernière version pourrait contraindre les fournisseurs à prendre en charge des versions qui ne bénéficient pas d'outils suffisamment aboutis. L'obligation de prendre en charge la dernière version d'OpenAPI serait en contradiction directe avec l'objectif de l'ARCEP de minimiser les changements API non rétro-compatibles.

---

<sup>7</sup> Voir : <https://smithy.io/2.0/guides/model-translations/convert-to-openapi.html>

20. En outre, plusieurs fournisseurs de services cloud tels qu'AWS ont établi des accords de niveau de service (*service level agreements* - SLA) dans lesquels ils s'engagent à maintenir les SDK pendant des périodes déterminées. Une obligation de mise à niveau imposée par un organisme externe, tel qu'OpenAPI, entrerait en conflit avec ces engagements existants et pourrait potentiellement perturber les opérations des clients. Les clients se soucient avant tout de la fonctionnalité et de la compatibilité du service, et non de la version de la spécification API sous-jacente. Plus encore, les clients qui ont développé leurs propres outils et processus autour de versions spécifiques d'OpenAPI seraient confrontés à une contrainte supplémentaire pour suivre le rythme des mises à jour obligatoires. Cela pourrait particulièrement affecter les clients ayant des intégrations complexes ou des outils personnalisés.
21. Enfin, il est important de relever que la dernière version d'une spécification n'est pas nécessairement la version la plus sûre ou la plus stable. Dans le domaine du développement de logiciel, notamment les spécifications d'API, les nouvelles versions peuvent introduire des failles de sécurité ou des problèmes de stabilité qui ne sont pas nécessairement visibles dès leur publication. Ceci est particulièrement important dans le domaine du cloud computing, où les services doivent respecter des normes élevées en matière de sécurité et de fiabilité. Les fournisseurs de services cloud ont besoin de temps pour évaluer de manière approfondie les implications des nouvelles versions en matière de sécurité avant leur mise en œuvre, compte tenu notamment de la nature sensible de la documentation relative aux API et des risques de sécurité potentiels liés à l'analyse et au traitement des nouveaux formats de spécifications. Imposer l'adoption rapide de la dernière version pourrait exposer les fournisseurs et leurs clients à des risques de sécurité non nécessaires avant que les acteurs du secteur aient eu le temps d'évaluer et de corriger de manière adéquate les failles potentielles.
22. En conclusion, AWS recommande de :
- Éviter d'imposer une norme spécifique pour les API destinées aux clients, car cela excèderait les exigences du Data Act et pourrait limiter les capacités des fournisseurs à répondre au mieux aux besoins de leurs clients en raison des limites inhérentes à toute spécification unique.
  - Si l'ARCEP envisage néanmoins d'adopter des normes pour la conception et la documentation d'API destinées aux clients basée sur OpenAPI, AWS recommande vivement que ces lignes directrices (i) reconnaissent que certains services cloud ne peuvent pas être représentés du tout par OpenAPI, (ii) prévoient que différents services peuvent nécessiter des approches variées en matière de spécification et de documentation des API pour exploiter pleinement leurs capacités, et que de nombreux clients auraient encore besoin de fonctionnalités qui dépassent ce qu'OpenAPI peut décrire et (iii) pour les services pouvant être représentés avec OpenAPI (malgré la perte potentielle de certaines fonctionnalités), considère la disponibilité de capacités de conversion permettant de traduire des structures cadres d'API personnalisées vers n'importe quelle version d'OpenAPI largement adoptée comme suffisante pour répondre aux exigences de l'ARCEP en matière de documentation des API.