# GRAPHQL, UN NOUVEAU STANDARD POUR LES API

VEILLE TECHNOLOGIQUE

## 1 - GRAPHQL, HISTORIQUEMENT

GraphQL a été **développé en 2012** par Lee Byron, Dan Schafer et Nick Schrock pour le compte de Facebook. Son objectif était de **remplacer la pile de réseau de l'application mobile Facebook**. Les développeurs de Facebook ont créé GraphQL pour **résoudre les problèmes de surcharge de données et de latence** qui se posaient lors de l'utilisation de l'API de Facebook, une API REST traditionnelle, pour alimenter leur naissante application mobile.

En 2015, Facebook publie GraphQL en tant que projet **open source**, ce qui permet à d'autres développeurs de l'utiliser pour construire leurs propres API. Depuis lors, GraphQL est devenue une technologie populaire pour la construction d'API et est utilisée par de nombreuses entreprises et organisations. Entreprises parmi lesquels on peut compter GitHub, Pinterest et Airbnb.

En 2019, la Fondation Linux créait la **Linux Foundation GraphQL Foundation**, une organisation à but non lucratif qui a pour but de promouvoir l'adoption et l'amélioration de GraphQL en tant que standard ouvert pour les API. Aujourd'hui, GraphQL est une technologie largement utilisée et reconnue dans l'industrie de la technologie de l'information pour la **construction d'API flexibles et efficaces.**

## 2 - QU'EST-CE QUE GRAPHQL ?

GraphQL est un langage de requête et un système de runtime pour les API (Application Programming Interface, ou interface de programmation d'application en français) qui permet aux clients de décrire les données dont ils ont besoin ainsi que la façon dont elles doivent être structurées. GraphQL a été conçu pour offrir une flexibilité accrue et une meilleure expérience de développement aux clients (browser, mobile, desktop...) en leur permettant de demander exactement les données dont ils ont besoin, plutôt que de recevoir un ensemble de données prédéterminées n'étant pas forcément nécessaires.

GraphQL est souvent utilisé pour construire des API pour les applications Web et mobile, car il permet aux développeurs de créer des API plus robustes et flexibles qui répondent aux besoins

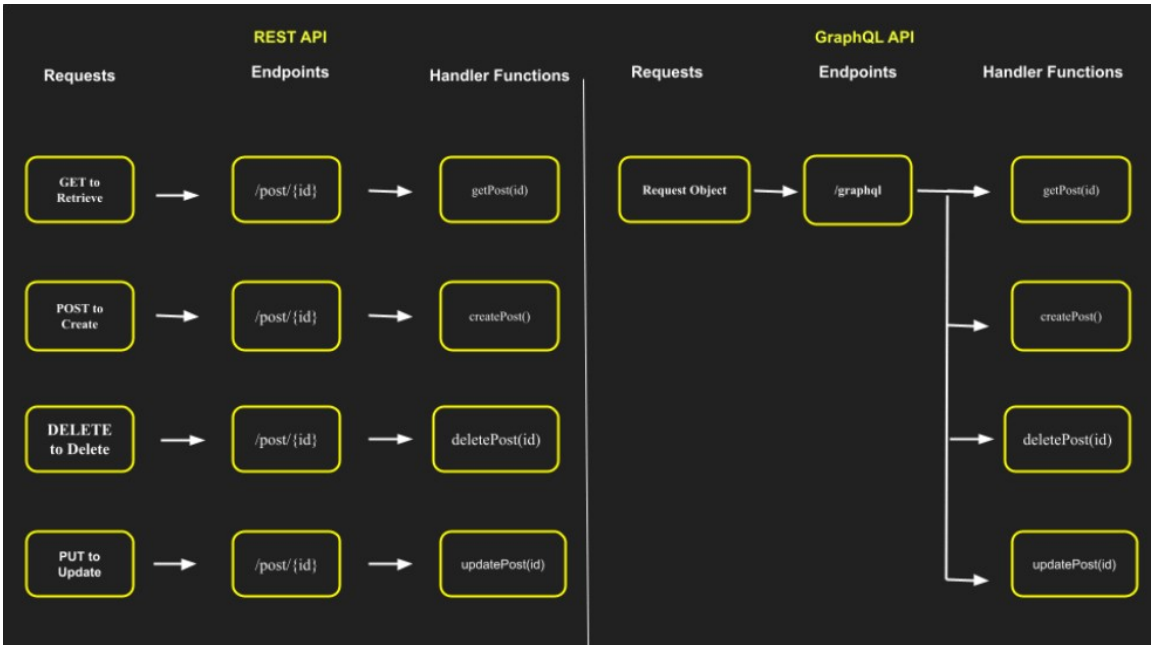
spécifiques de chaque client. GraphQL permet également d'éviter les problèmes de surcharge de données et de latence liés à l'envoi de données superflues sur le réseau.

Dans une API Rest conventionnelle, pour accéder à des données imbriquées, cela nécessite plusieurs « allers-retours » (appels API et réponse. Et à chaque fois appel, la réponse renvoie toutes les données contenues dans l'objet). Cela crée des transfert de données inutiles et des appels inutiles, ce qui cause des lenteurs côté serveur car énormément de requêtes HTTP.

Graphql est donc un langage de requêtes pour les données des API (Graph provient de l'organisation des données, qui ressemblent à un arbre et QL provient de Query Langage) ainsi qu'un environnement d'exécution de ces requêtes.
Il s'agit d'une spécification pour implémenter une API (ce n'est ni un langage de programmation ni un Framework).

## 3 – GRAPHQL, COMMENT CELA FONCTIONNE ? QUELLE DIFFERENCE AVEC REST ?

Pour obtenir des données avec REST et tous ses endpoints il était nécessaire de faire plusieurs appels à la base de donnée (appels API) pour obtenir les données imbriquées.
Avec GraphQL, 1 seul appel suffit. Il ne possède qu'un seul endpoint.




Avec REST l'objet reçu d'un appel API est refaçonné côté back-end.
En effet l'objet complet est retourné a chaque appel avec impossibilité de définir les données spécifiques appelées ou non.

Cependant, avec GraphQL il est possible de définir directementet dynamiquement l'objet que l'ont reçoit côté client après un appel API (Voir différence dans les appels/réponse API, section GraphQL & API REST).
Dans une API REST comme dans une API GraphQL, l'architecture de l'API est la même.
Seule la couche de routage change passant de REST à GraphQL.
L'environnement d'exécution de GraphQl est une couche de routage de l'API, qui interprète des requêtes GraphQL, reçues via le payload d'un POST.
Dans une requête GraphQL l'ordre demandé des objets est respecté lors de la réponse de l'API.
GraphQL, en plus d'être un langage de requête, est également un environnement d'exécution qui interprète et structure ces requêtes à partir d'un schéma.
Dans le schéma GraphQL on définit tous les objets GraphQL, leurs types ainsi que toutes les requêtes possibles dans un objet global **Query**, qui est **endpoint unique** pour les appels API dans une API GraphQL, ainsi que les actions disponibles dans l'objet global **Mutation** (dont les entrées font office de crud et où on y définit les fonctions telles que createObject, delete, etc...)
Lorsqu'une requête est envoyée, **le serveur GraphQL vérifie que la requête est bien disponible** dans l'objet global Query  et que les champs demandés correspondent bien au **retour de la requête en question.**
Une fois cela accompli, le serveur GraphQL fait appel à ce que l'ont appel des **Resolvers**.
Un resolver associe une fonction de l'API (récupération de données de la base de données, appels à une API tierce, etc…) à une requête ou une action définie dans les objets globaux du schéma (Query et Mutation). Le resolver récupère les données et les réorganise (les champs demandés, dans l'ordre demandé) pour que la réponse corresponde à la requête.

## 4 - MISE EN PLACE

Il existe différentes façons de développer une API avec GraphQL et toutes ces façons d'implémentations sont listées dans la documentation officielle de GraphQL (https://graphql.org/code/). GraphQL est compatible avec une majorité de langages car il s'agit d'une spécification.

Il est possible d'implémenter une API GraphQL en utilisant un framework de développement de serveur GraphQL, comme Express-GraphQL ou Apollo Server. Ces frameworks offrent une interface simple pour définir les types de données ainsi que les resolvers de GraphQL, et pour configurer un serveur GraphQL qui peut être intégré à un serveur web existant.
Ou en utilisant une bibliothèque de client GraphQL, comme Apollo Client ou Relay, pour écrire une application web qui utilise GraphQL pour interroger un serveur distant.

Il est également possible d'utiliser une bibliothèque de serveur GraphQL telle que GraphQL.js pour écrire un serveur GraphQL from scratch en utilisant Node.js.
Cette approche nécessite une connaissance approfondie de GraphQL et de l'implémentation de serveur GraphQL, mais elle peut être utile dans des cas où l'ont souhaiterait avoir un contrôle total sur la façon dont le serveur GraphQL est mis en place et exécuté.
Il existe également des services en ligne qui permettent de développer une API GraphQL sans avoir à gérer l'infrastructure de serveur.
Par exemple, AWS AppSync et Graphcool, qui sont des services de back-end basés sur GraphQL qui permettent de développer une API GraphQL sans avoir à écrire de code de serveur.

## 5 - QUELS AVANTAGES ?

GraphQL présente de nombreux avantages par rapport aux API REST traditionnelles.
Tout d'abord sa **flexibilité est accrue**. En effet, avec GraphQL, le client peut spécifier exactement les données dont ils a besoin ainsi que la façon dont elles doivent être structurées.
Cela signifie que les clients peuvent obtenir exactement les données dont ils ont besoin, sans données superflues.  Cela permet une optimisation des données réseau grâce à un seul appel API ainsi qu'une optimisation des requêtes pour palier à l'over-fetching et à l'under-fetching.
GraphQL permet donc aux clients de demander exactement les données dont ils ont besoin, ce qui peut réduire la quantité de données envoyées sur le réseau et améliorer les temps de chargement des pages.
Ensuite, graphQL c'est une très **bonne expérience de développement** : GraphQL offre une **syntaxe simple et intuitive** qui facilite la création de requêtes et la gestion des réponses des serveurs. Cela rend le développement d'applications plus rapide et plus simple.
De même, GraphQL est **conçu pour être évolutif** et facilite l'ajout de nouvelles fonctionnalités et de nouvelles sources de données **sans avoir à modifier les interfaces existantes.**
Les standard de GraphQL sont clairs et facile à suivre et à appliquer. Le schéma, qui est une nouvelle pratique de gestion des données, offre une possibilité d'introspection vis à vis de l'API (demande du schéma à l'API, retour des erreurs en direct, voir le playground).
Malgré cela, GraphQL n'est pas forcément la meilleure solution pour tous les cas d'utilisation et qu'il peut y avoir des cas où une API REST semble plus appropriée.
Cependant, pour de nombreux projets de développement d'API, GraphQL offre une **meilleure flexibilité et une meilleure expérience de développement que les API REST traditionnelles.**

## 6 - LES INCONVENIENTS ?

Malgré tous ses avantages GraphQL comprend tout de même quelques inconvénients.

En effet, GraphQL est difficile de mise en place . Sa mise en place nécessite un serveur dédié et une implémentation côté serveur pour fonctionner, ce est plus complexe à mettre en place que certaines autres technologies d'API.

D'un point de vue de l'apprentissage, GraphQL est un nouveau langage de requête et peut nécessiter un temps d'apprentissage pour bien comprendre et cerner son fonctionnement ainsi que la meilleure façon de l'appréhender.

GraphQL présente un problème de **limitation de la pile de données**. En effet, GraphQL **ne peut pas être utilisé pour travailler avec des données non structurées**, comme les données binaires. De même le téléversement de fichiers multimédias est compliqué à implémenter. Il existe cependant des techniques pour contourner ces limites (notamment la conversion des fichiers en base 64...)

La **mise en cache serveur est difficilement envisageable** avec graphQL également. En effet les requêtes sont pour la plupart personnalisées et les réponses très souvent différentes, contrairement aux requêtes d'une API REST qui, elles, retournent souvent les même résultats.

Pour palier à cela, on utilise une mise en cache côté applicatif.

Meta fournit également un outil de batching et de caching (Dataloader, un agrégateur de requêtes récurrentes) pour aider à la mise en cache.

Ainsi, graphQL présente tout de même quelques désavantages. Même si de nombreuses autres technologies d'API présentent également certaines limitations et complexités.

Cependant, il semble important de comprendre en considération ces limitations lors de la prise de décision entre le développement d'une API GraphQL ou d'une API REST.