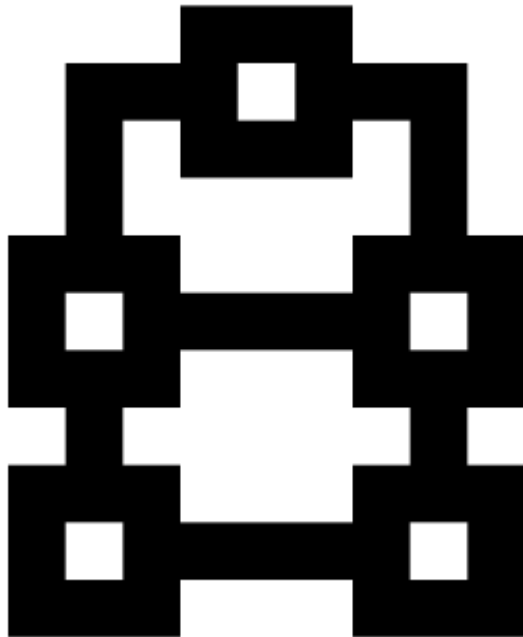
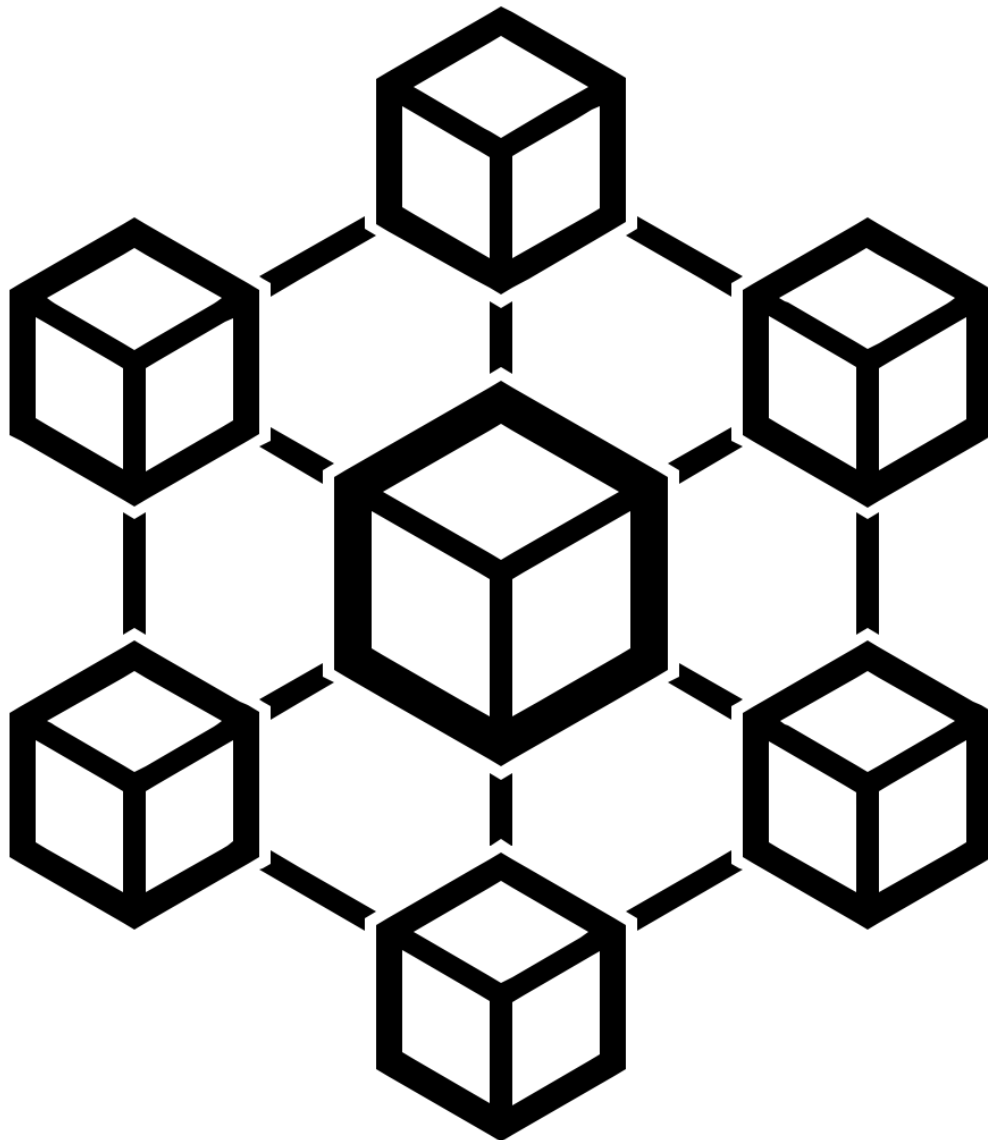


Architecture Microservices



Les microservices



Pourquoi ne pas encore plus découper les services?

Les microservices

Les microservices sont une évolution de l'architecture SOA.

Ils sont:

- plus petits.
- plus indépendants.
- plus spécialisés.
- plus facile à déployer.
- plus facile à maintenir.

Les microservices



Du monolithique aux microservices

Monolithique = Un seul bâtiment avec tout dedans.

3 tiers = Un supermarcher, Une usine et un entrepot.

SOA = Un supermarcher, des usines et des entrepots.

Microservices = Des marché, de petit producteurs spécialisés et des entrepots spécialisés.

Plus de services!

Le but des **microservices** est de découper les services en plus petits services.

- **données utilisateur**: ne gère que les données utilisateur.
 - **authentification**: ne gère que l'authentification.
 - **gestion des produits**: ne gère que la gestion des produits.
 - **gestion des messages**: ne gère que la gestion des messages.
 - ...
-

Des services plus petits

Les microservices sont plus petits et plus spécialisés:

- plus facile à développer.
 - plus facile à débbugger.
 - moins de risques de dettes techniques.
 - plus facile à maintenir.
 - plus facile à déployer.
 - plus facile à tester.
 - plus facile à mettre à l'échelle.
-

La meilleure technologie pour chaque service

Chaque service peut utiliser la meilleure technologie pour répondre à ses besoins.

- Traitement des données et calculs d'indicateurs: **spark**.
- Gestion de l'authentification: **keycloak**.
- Gestion des messages: **firebase**.
- Gestion des produits: **Spring Boot**.

La meilleure technologie pour chaque base de données

Chaque base de données peut utiliser la meilleure technologie pour répondre à ses besoins.

- Données plates: **PostgreSQL**.
 - Données complexe: **MongoDB**.
 - Données temporelles: **InfluxDB**.
 - Données sous forme de graphes: **Neo4j**.
 - Données temporaires: **Redis**.
-

Les mises à jours

Elles sont plus facile à faire.

Il faut juste mettre à jour le service et le redéployer.

Les nouvelles fonctionnalités

Elles sont plus facile à faire.

Il faut juste développer le service et le déployer.

Scalabilité

Les microservices sont plus facile à mettre à l'échelle.

Il suffit de mettre à l'échelle le service qui a besoin d'être mis à l'échelle.

Granularité

Le but des **microservices** est de **découper** les **responsabilités** et de **simplifier** les **évolutions**.

Il faut donc plus ou moins **découper** les **microservices** en fonction de la **granularité** des **fonctionnalités**.

Plus de granularité

Plus de **microservices**

= Plus de **code** et de **conception**

= Plus de temps de **développement**

= Plus de **configurations**

Plus de granularité

Plus petit **microservices**

= Plus **indépendant**

= Plus facile à **développer**

= Plus facile à **maintenir**

Moins de granularité

Moins de **microservices**

= Moins de **code** et de **conception**

= Moins de temps de **développement**

= Moins cher à court terme

Moins de granularité

Moins de **microservices**

= Plus de **dépendances**

= Plus de **complexité**

= Plus de temps de **développement** à long terme

= Plus cher à long terme

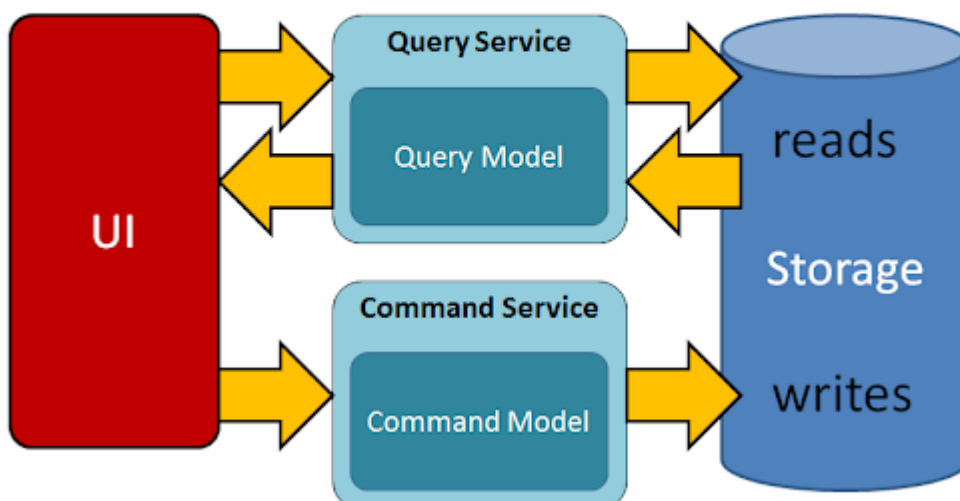
Une histoire d'équilibre!



CQRS et Event Sourcing

Les microservices permettent de mettre en place des architectures **CQRS** et **Event Sourcing**.

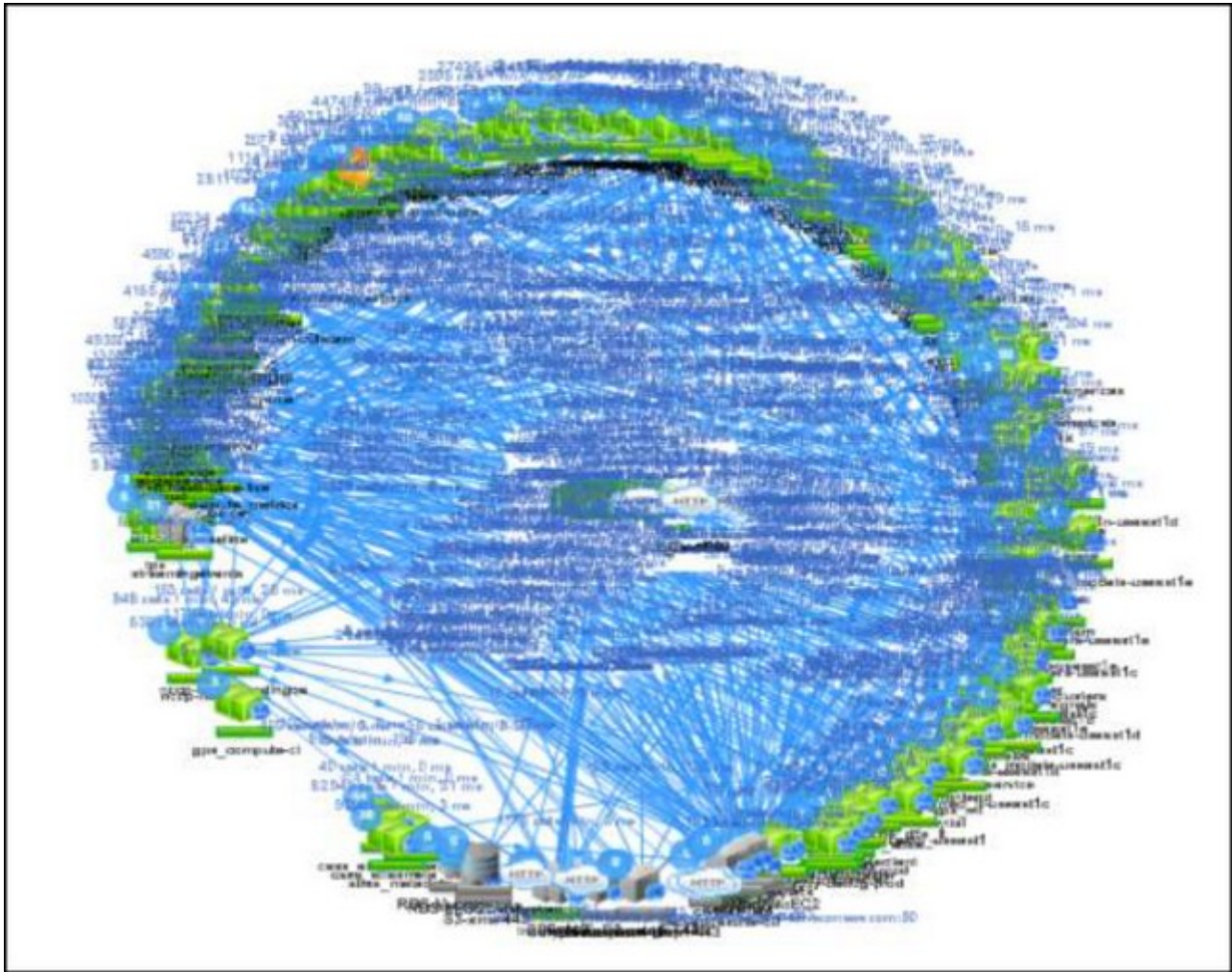
CQRS Pattern



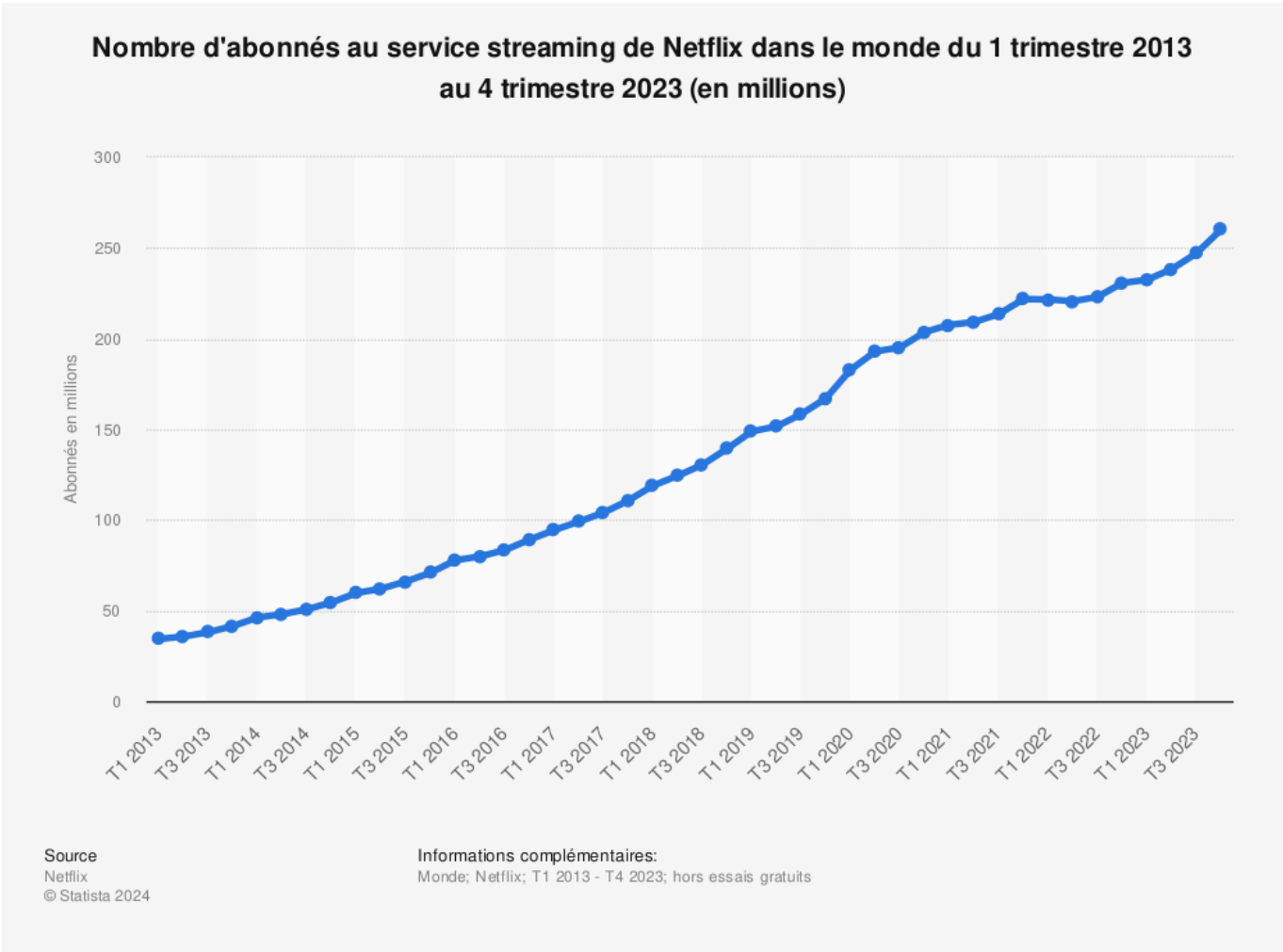
```
graph LR; Client[Client] --> Command[Command]; Client --> Query[Query]; Command --> CH[Command Handlers]; CH --> DM[Domain Model]; DM --> Event[Event]; Event --> ES[(Event Store)]; Event --> EH[Event Handlers]; EH --> QD[(Query Database)]; Query --> QH[Query Handlers]; QH --> QD;
```

The diagram illustrates the Event Sourcing architecture. It starts with a **Client** (grey rectangle) on the left. The Client branches into two paths: **Command** (black rounded rectangle) and **Query** (black rounded rectangle). The **Command** path flows through **Command Handlers** (green rectangle) to the **Domain Model** (red rectangle), which then generates an **Event** (yellow cylinder). The **Event** is stored in the **Event Store** (orange cylinder) and also triggers **Event Handlers** (green rectangle), which update the **Query Database** (orange cylinder). The **Query** path flows through **Query Handlers** (green rectangle) directly to the **Query Database**.

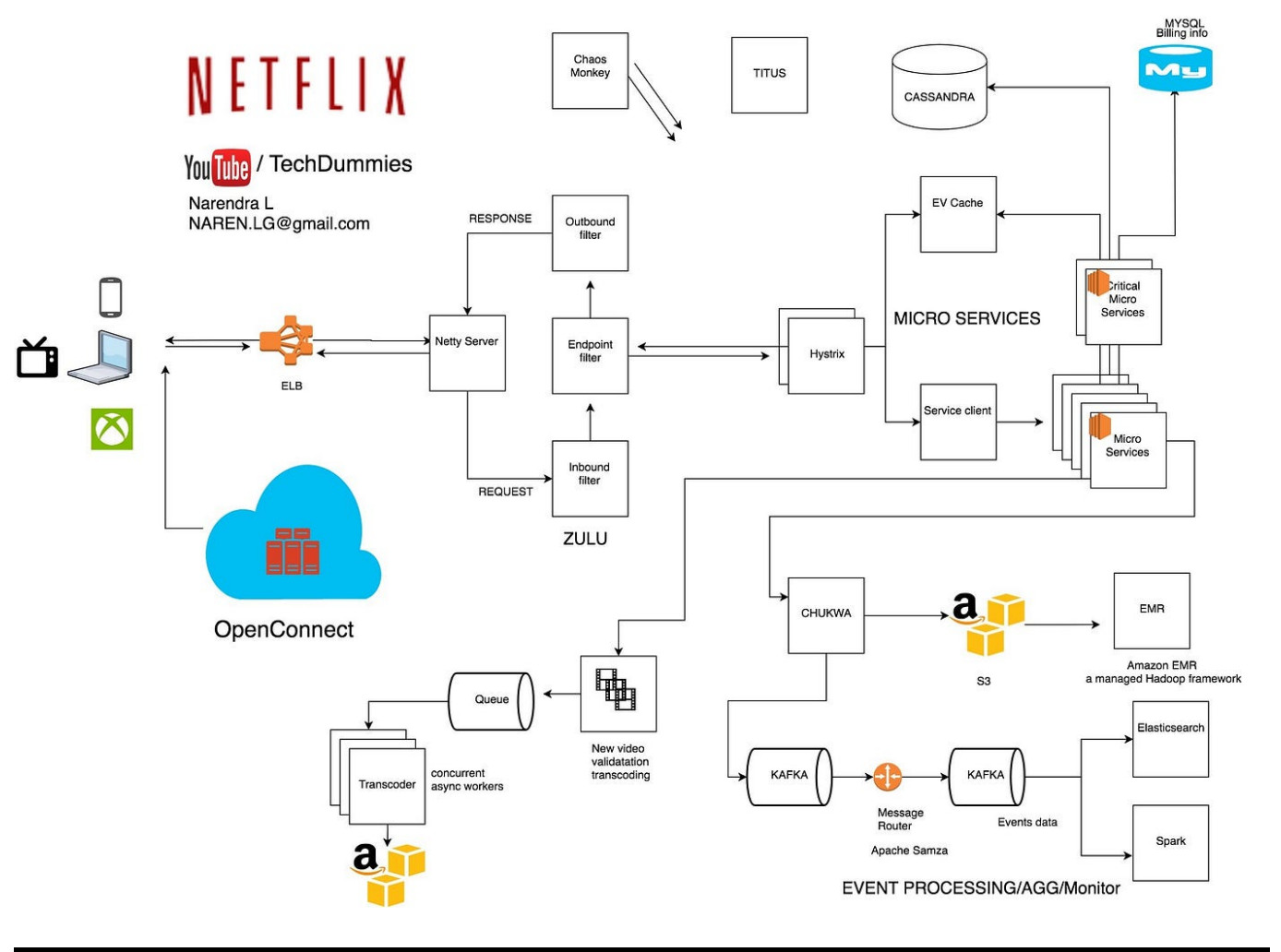
The image features the Netflix logo, consisting of a large red 'N' and the word 'NETFLIX' in white capital letters, centered over a dark background. The background is a collage of various Asian movie and TV show posters, including titles like '水行俠' (Aquaman), '格雷的五十道陰影' (Fifty Shades of Grey), '轉學來的女生' (The New Girl), '黑道律師文森佐' (Vincent & the Doctor), '惡怕怕' (The Worst of Us), '黑道主夫' (The Worst of Us), 'AV 狂熱' (AV Fever), '我在你心底的名字' (I Am Not a Gay), '我親愛的小潔癖' (My Obsessive Compulsive Disorder), '臥底追緝' (The Hidden Man), '臥底追緝 2' (The Hidden Man 2), '臥底追緝 3' (The Hidden Man 3), '臥底追緝 4' (The Hidden Man 4), '臥底追緝 5' (The Hidden Man 5), '臥底追緝 6' (The Hidden Man 6), '臥底追緝 7' (The Hidden Man 7), '臥底追緝 8' (The Hidden Man 8), '臥底追緝 9' (The Hidden Man 9), '臥底追緝 10' (The Hidden Man 10), '臥底追緝 11' (The Hidden Man 11), '臥底追緝 12' (The Hidden Man 12), '臥底追緝 13' (The Hidden Man 13), '臥底追緝 14' (The Hidden Man 14), '臥底追緝 15' (The Hidden Man 15), '臥底追緝 16' (The Hidden Man 16), '臥底追緝 17' (The Hidden Man 17), '臥底追緝 18' (The Hidden Man 18), '臥底追緝 19' (The Hidden Man 19), '臥底追緝 20' (The Hidden Man 20), '臥底追緝 21' (The Hidden Man 21), '臥底追緝 22' (The Hidden Man 22), '臥底追緝 23' (The Hidden Man 23), '臥底追緝 24' (The Hidden Man 24), '臥底追緝 25' (The Hidden Man 25), '臥底追緝 26' (The Hidden Man 26), '臥底追緝 27' (The Hidden Man 27), '臥底追緝 28' (The Hidden Man 28), '臥底追緝 29' (The Hidden Man 29), '臥底追緝 30' (The Hidden Man 30), '臥底追緝 31' (The Hidden Man 31), '臥底追緝 32' (The Hidden Man 32), '臥底追緝 33' (The Hidden Man 33), '臥底追緝 34' (The Hidden Man 34), '臥底追緝 35' (The Hidden Man 35), '臥底追緝 36' (The Hidden Man 36), '臥底追緝 37' (The Hidden Man 37), '臥底追緝 38' (The Hidden Man 38), '臥底追緝 39' (The Hidden Man 39), '臥底追緝 40' (The Hidden Man 40), '臥底追緝 41' (The Hidden Man 41), '臥底追緝 42' (The Hidden Man 42), '臥底追緝 43' (The Hidden Man 43), '臥底追緝 44' (The Hidden Man 44), '臥底追緝 45' (The Hidden Man 45), '臥底追緝 46' (The Hidden Man 46), '臥底追緝 47' (The Hidden Man 47), '臥底追緝 48' (The Hidden Man 48), '臥底追緝 49' (The Hidden Man 49), '臥底追緝 50' (The Hidden Man 50), '臥底追緝 51' (The Hidden Man 51), '臥底追緝 52' (The Hidden Man 52), '臥底追緝 53' (The Hidden Man 53), '臥底追緝 54' (The Hidden Man 54), '臥底追緝 55' (The Hidden Man 55), '臥底追緝 56' (The Hidden Man 56), '臥底追緝 57' (The Hidden Man 57), '臥底追緝 58' (The Hidden Man 58), '臥底追緝 59' (The Hidden Man 59), '臥底追緝 60' (The Hidden Man 60), '臥底追緝 61' (The Hidden Man 61), '臥底追緝 62' (The Hidden Man 62), '臥底追緝 63' (The Hidden Man 63), '臥底追緝 64' (The Hidden Man 64), '臥底追緝 65' (The Hidden Man 65), '臥底追緝 66' (The Hidden Man 66), '臥底追緝 67' (The Hidden Man 67), '臥底追緝 68' (The Hidden Man 68), '臥底追緝 69' (The Hidden Man 69), '臥底追緝 70' (The Hidden Man 70), '臥底追緝 71' (The Hidden Man 71), '臥底追緝 72' (The Hidden Man 72), '臥底追緝 73' (The Hidden Man 73), '臥底追緝 74' (The Hidden Man 74), '臥底追緝 75' (The Hidden Man 75), '臥底追緝 76' (The Hidden Man 76), '臥底追緝 77' (The Hidden Man 77), '臥底追緝 78' (The Hidden Man 78), '臥底追緝 79' (The Hidden Man 79), '臥底追緝 80' (The Hidden Man 80), '臥底追緝 81' (The Hidden Man 81), '臥底追緝 82' (The Hidden Man 82), '臥底追緝 83' (The Hidden Man 83), '臥底追緝 84' (The Hidden Man 84), '臥底追緝 85' (The Hidden Man 85), '臥底追緝 86' (The Hidden Man 86), '臥底追緝 87' (The Hidden Man 87), '臥底追緝 88' (The Hidden Man 88), '臥底追緝 89' (The Hidden Man 89), '臥底追緝 90' (The Hidden Man 90), '臥底追緝 91' (The Hidden Man 91), '臥底追緝 92' (The Hidden Man 92), '臥底追緝 93' (The Hidden Man 93), '臥底追緝 94' (The Hidden Man 94), '臥底追緝 95' (The Hidden Man 95), '臥底追緝 96' (The Hidden Man 96), '臥底追緝 97' (The Hidden Man 97), '臥底追緝 98' (The Hidden Man 98), '臥底追緝 99' (The Hidden Man 99), '臥底追緝 100' (The Hidden Man 100).



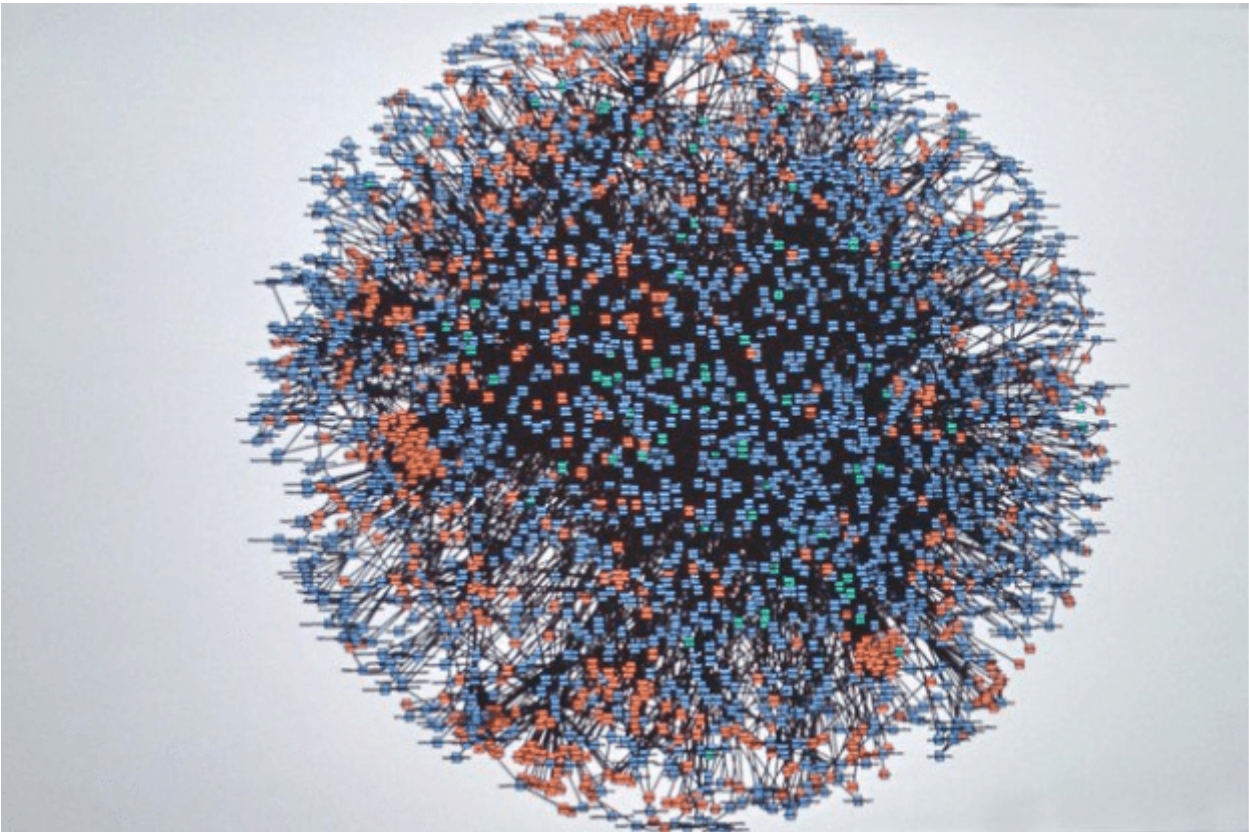
Netflix



Netflix



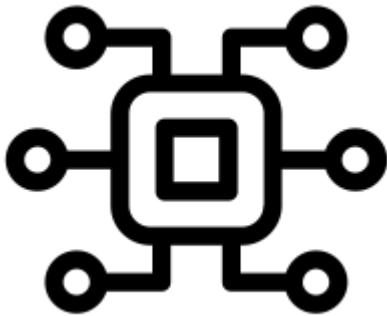
Amazon



Uber



Les composants d'une architecture microservices



Les architectures microservices ont des composants spécifiques

- API Gateway
- Service Registry ou Service Discovery
- Load Balancer

API Gateway

Un **API Gateway** est un point d'entrée unique pour les clients.

Il permet de n'avoir qu'une seule adresse IP pour accéder à l'ensemble des services.

Il vérifie les autorisations des clients.

Technologie API Gateway

- nginx
- Krakend
- Traefik



Service Registry ou Service Discovery

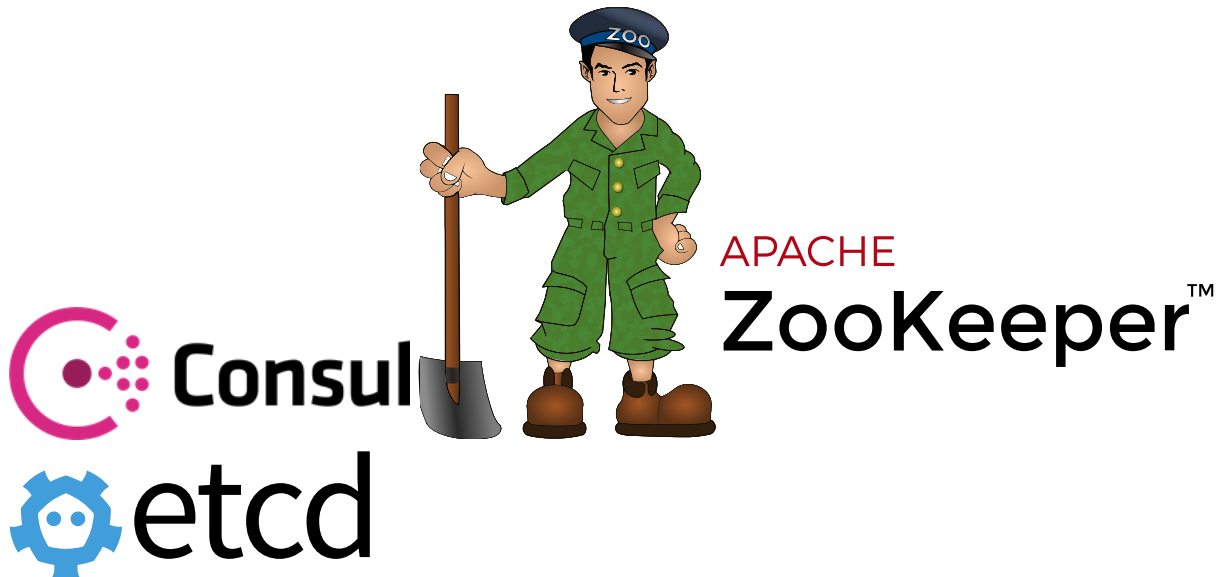
Le **Service Registry** permet de connaître l'ensemble des services disponibles.

Il permet de connaître l'adresse IP et le port d'un service.

Les autres services peuvent rechercher un service dans le **Service Registry**.

Technologie Service Registry

- Consul
- Zookeeper
- ETCD



Load Balancer

Le **Load Balancer** permet de répartir la charge entre les services.

Dans le cas où un service il est répliqué, le **Load Balancer** permet de répartir la charge entre les répliques.

Si un service est indisponible, le **Load Balancer** peut rediriger les requêtes vers un autre service.

Kubernetes

Kubernetes est un ensemble de composants qui permet de gérer des conteneurs.

Il contient un **Service Registry** (ETCD) et un **Load Balancer**.

Il permet de configurer un **API Gateway** avec **Ingress**.



kubernetes

Kubernetes et les microservices

Kubernetes est souvent utilisé pour gérer des architectures microservices.

Il permet de simplifier le déploiement, la mise à l'échelle et la maintenance des services.

Kubernetes et automatisation

Kubernetes permet de déployer des services de manière automatisée.

- Lors de l'ajout d'un nouveau service, Kubernetes va automatiquement le déployer et le rendre disponible pour les autres services.
 - Lors de la mise à l'échelle d'un service, Kubernetes va automatiquement répartir la charge entre les réplicas.
 - Lors de la mise à jour d'un service, Kubernetes va automatiquement mettre à jour les réplicas en assurant la disponibilité du service.
-

Kubernetes & Docker

Kubernetes est souvent utilisé avec Docker.

Docker permet de créer des images de conteneurs.

Ils peuvent ensuite être déployés sur Kubernetes.

Kubernetes peut automatiquement mettre à jour les conteneurs.

Kubernetes

Le fonctionnement de Kubernetes est basé sur des demandes d'état.

L'état désiré est défini dans un fichier YAML.

Je veux 3 réplicas du service `utilisateurs` avec une image `utilisateurs:1.0.0`.

Kubernetes

Pour faire une modification, il suffit de modifier le fichier YAML.

Je veux 5 réplicas du service `utilisateurs` avec une image `utilisateurs:1.0.1`.

Kubernetes va alors lancer la nouvelle version du service. Puis supprimer les anciennes réplicas.

Kubernetes

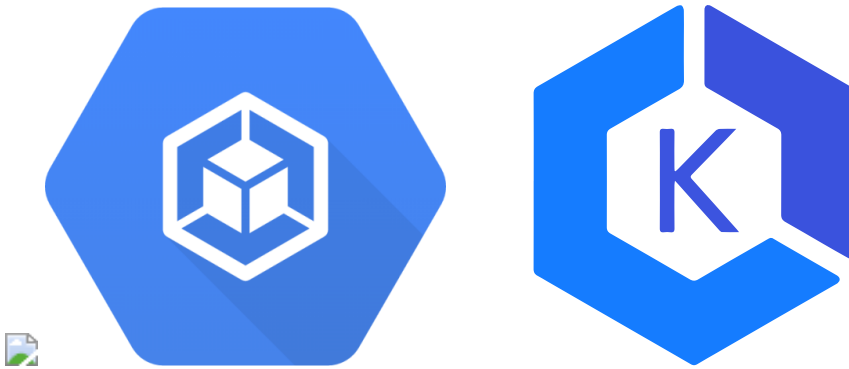
Il se charge de répartir les services sur les différentes machines (nodes).

Si une machine tombe en panne, Kubernetes va automatiquement déplacer les services sur une autre machine.

`Disponibilité` et `Scalabilité` sont des propriétés de Kubernetes.

Le cloud et les microservices

Le cloud permet de déployer des architectures microservices.



Le déploiement continu

Dans une architecture microservices, le déploiement continu est une pratique courante et nécessaire.

Il peut y avoir plusieurs centaines de services qui doivent être déployés.

Le déploiement continu permet de déployer les services de manière automatisée.



Une seule architecture par solution?

Non! Il est possible d'utiliser plusieurs architectures dans une même solution.

Par exemple, une architecture avec un gros monolithe et des microservices pour les fonctionnalités critiques.

Microservices et projets legacy

Il est possible de migrer un projet legacy vers une architecture microservices.

Pour cela il faut décomposer le projet en services.

Conclusion

[Conclusions](#)