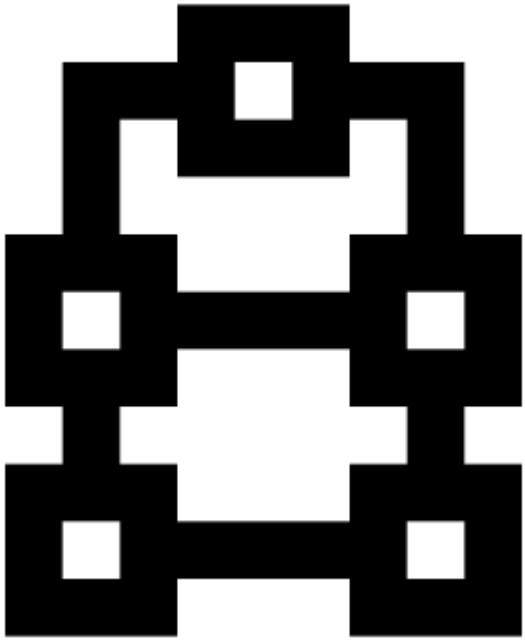
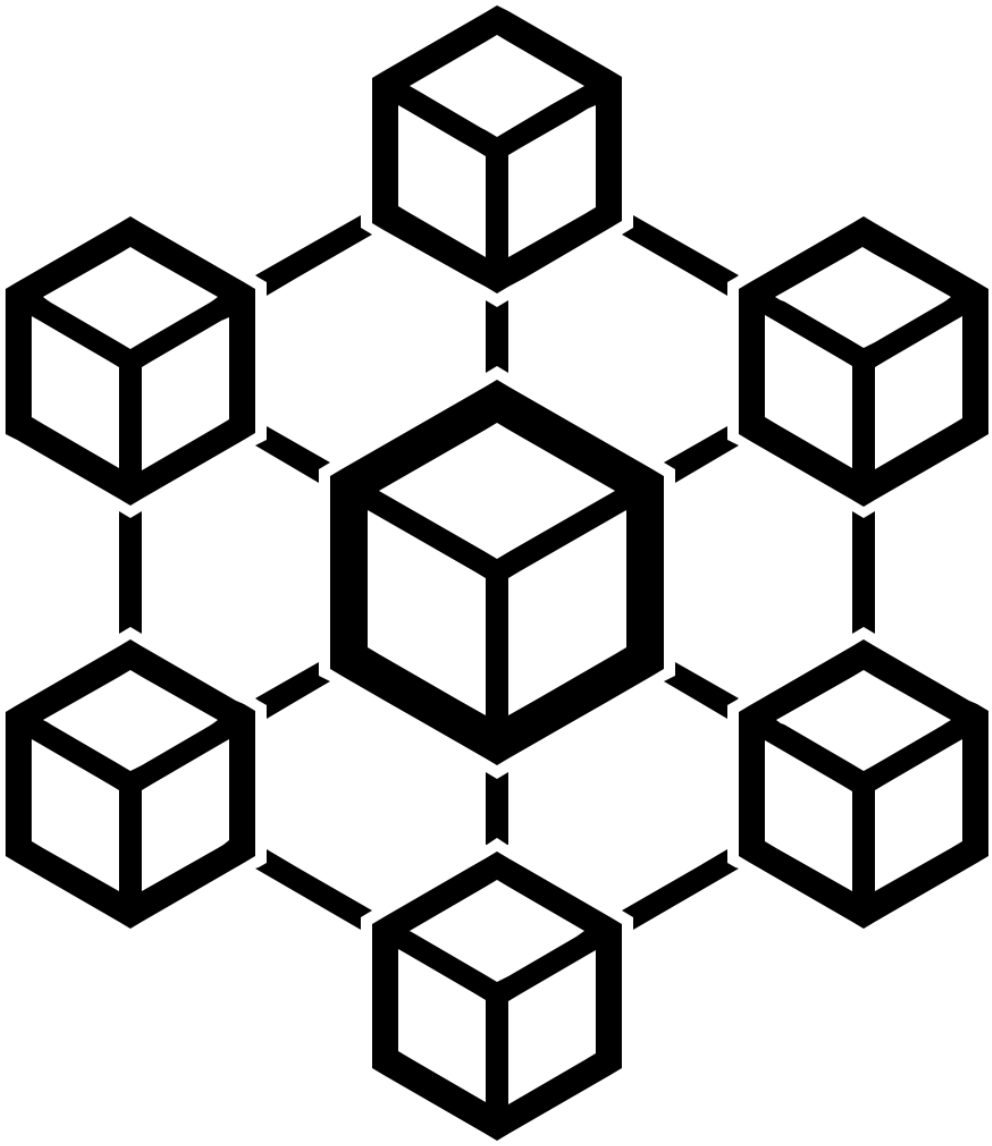
Architecture Microservices

semifir



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

 microservices

---

Du monolithique aux microservices

`Monolithique` = Un seul batiment 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 `micro`services 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 débugger.
- 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

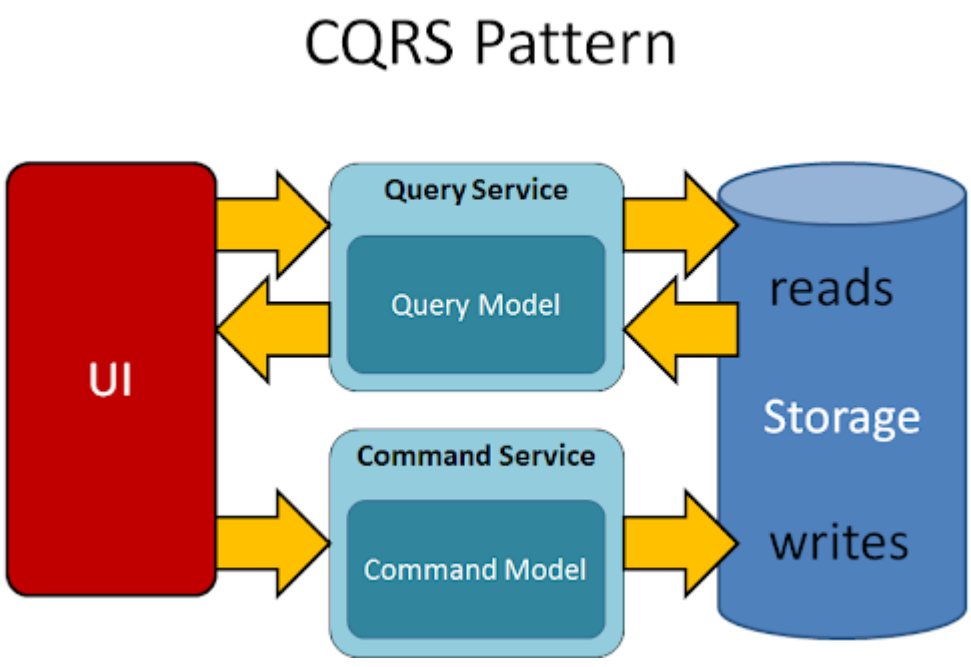
= Plus de temps de développement à long terme

= Plus cher à long terme

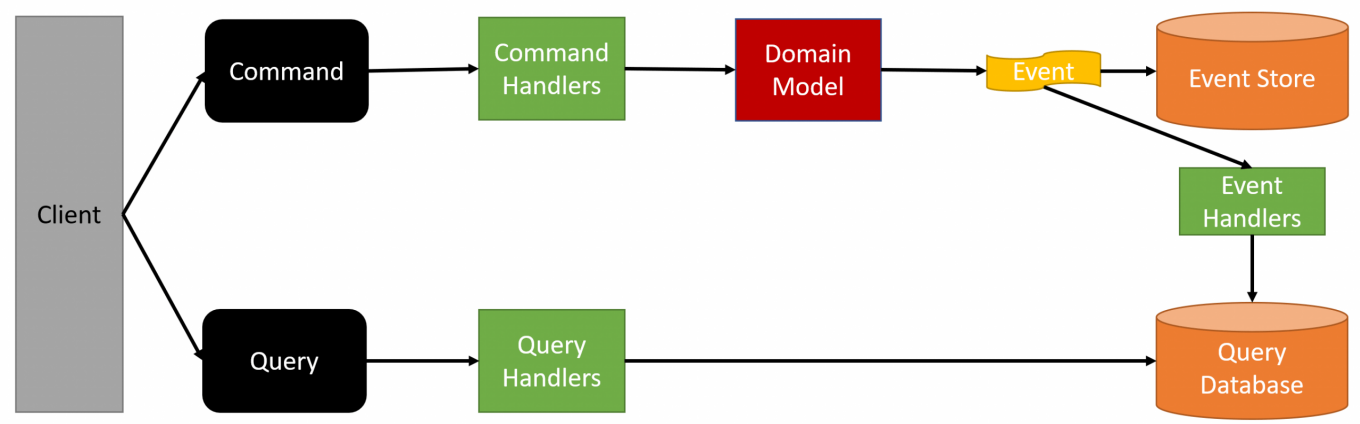
Une histoire d'équilibre!

CQRS et Event Sourcing

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



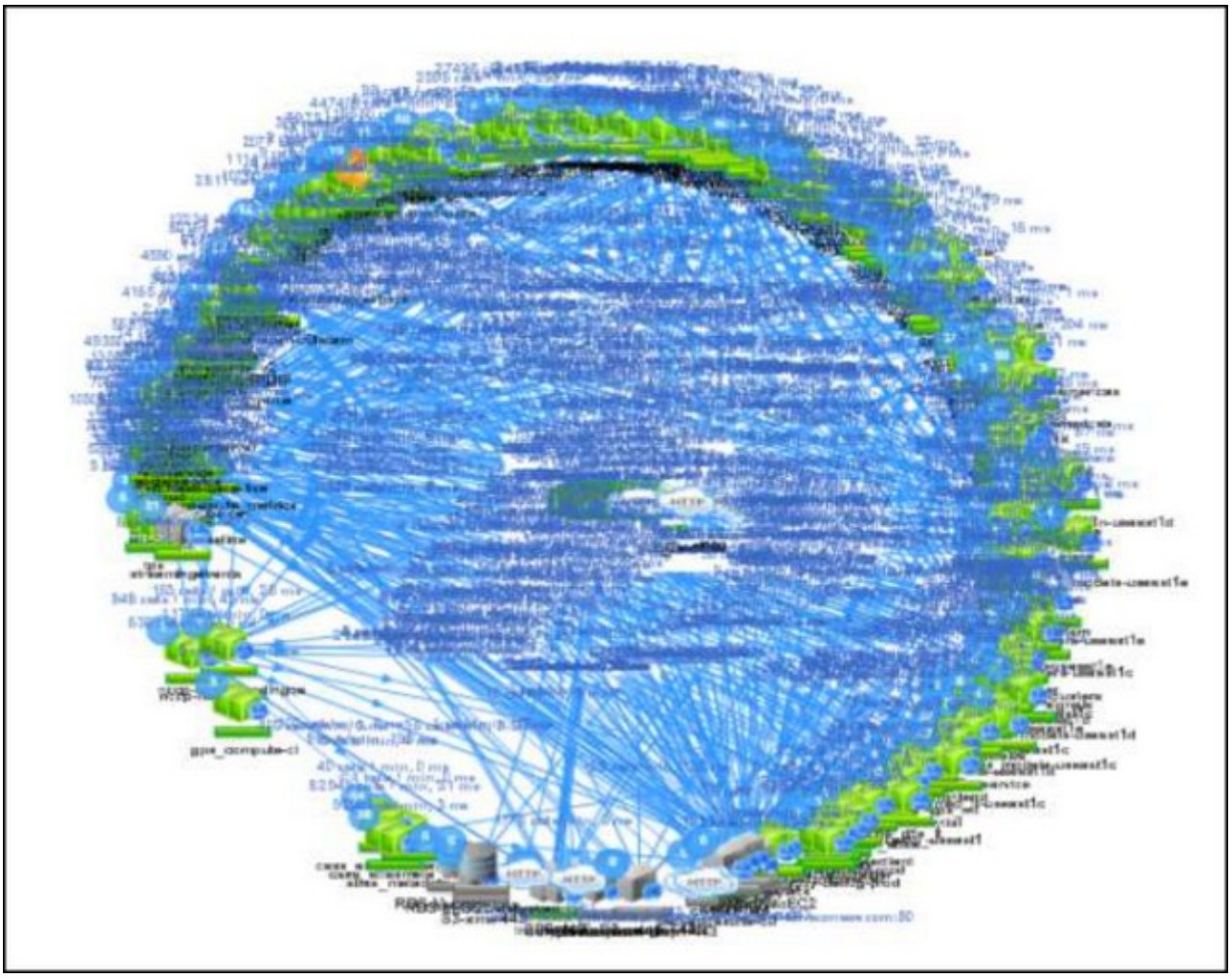
CQRS et Event Sourcing



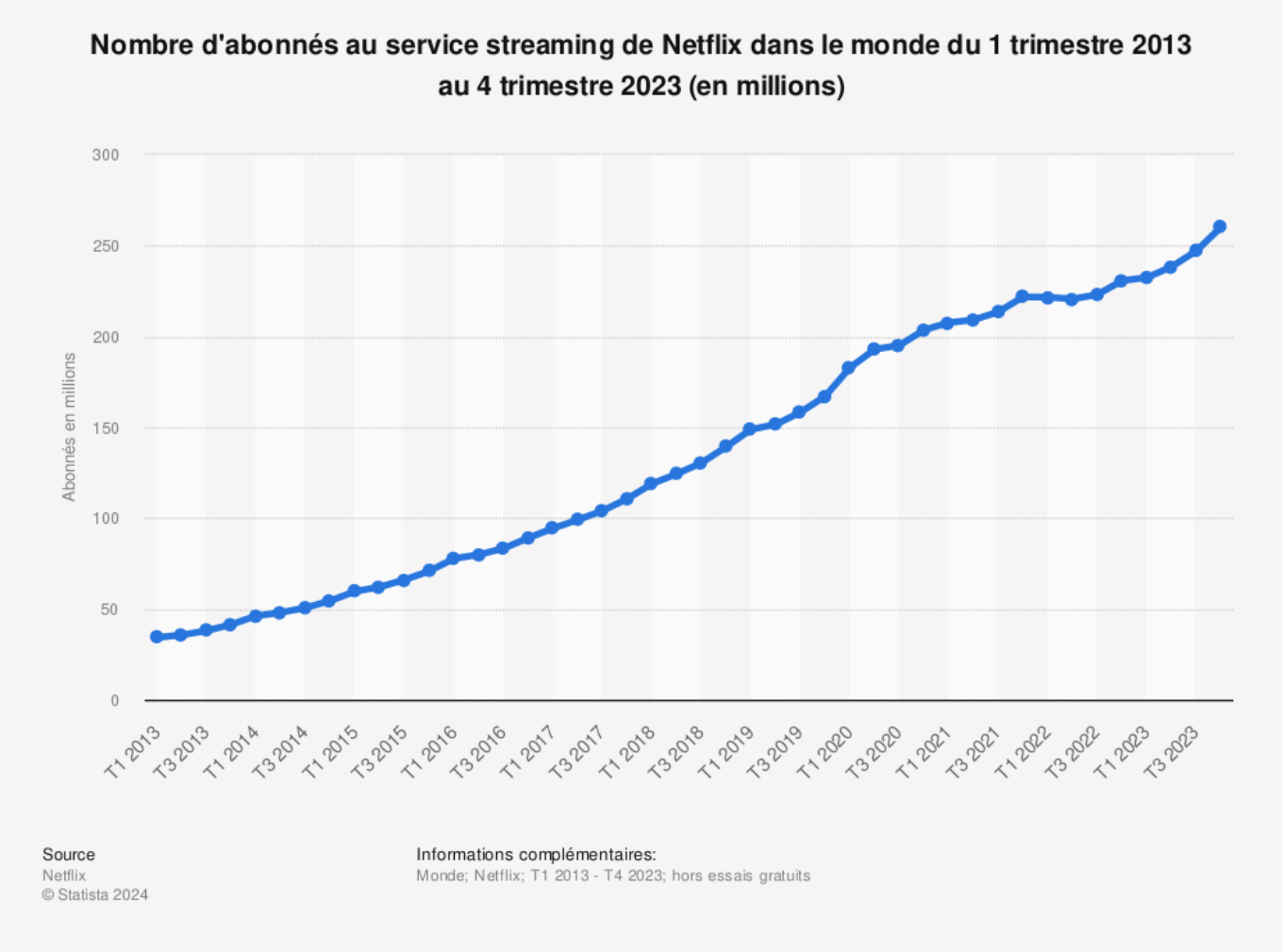
Exemples de systèmes utilisant des microservices

Netflix

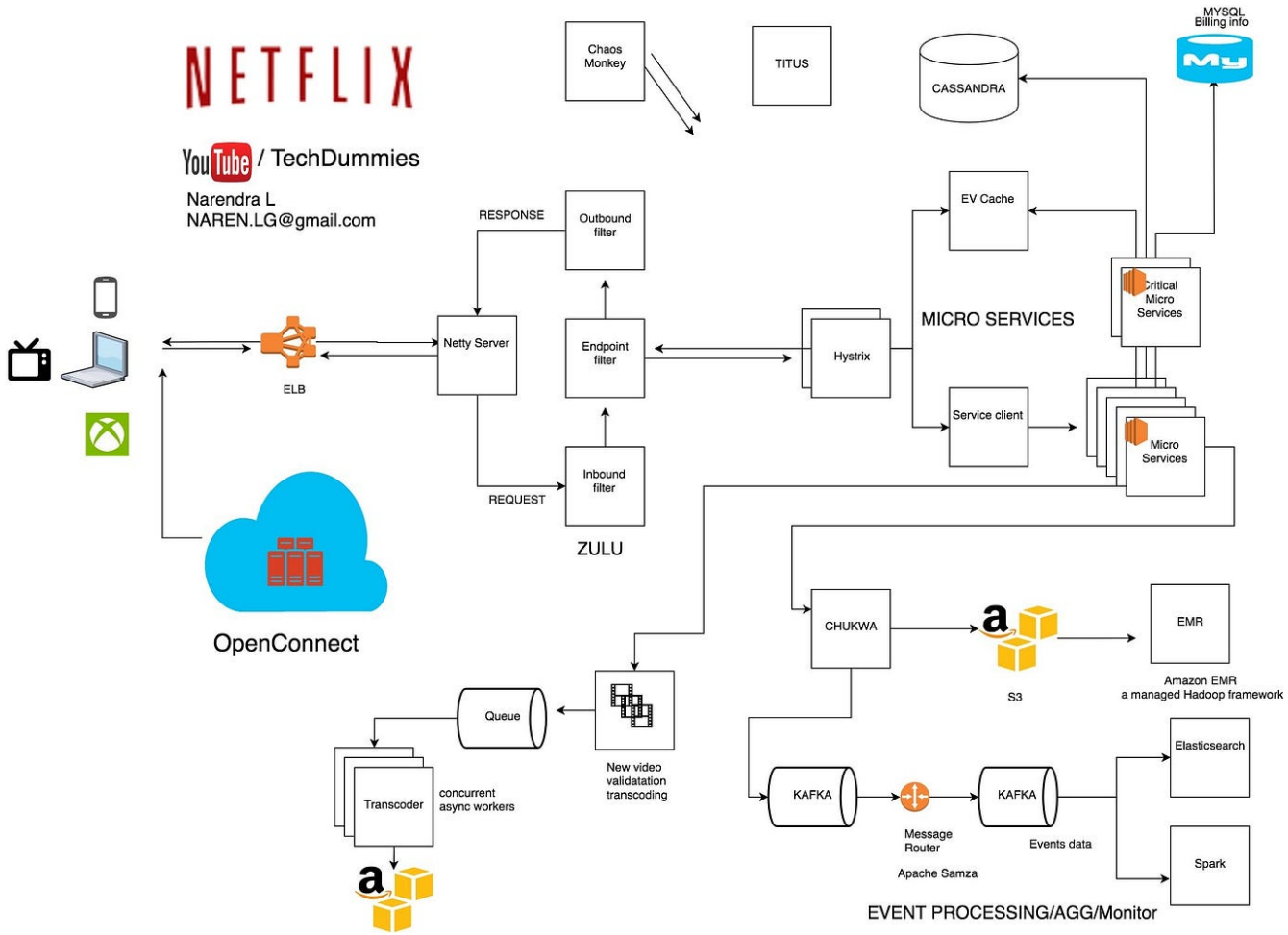
Plus de 1000 microservices.



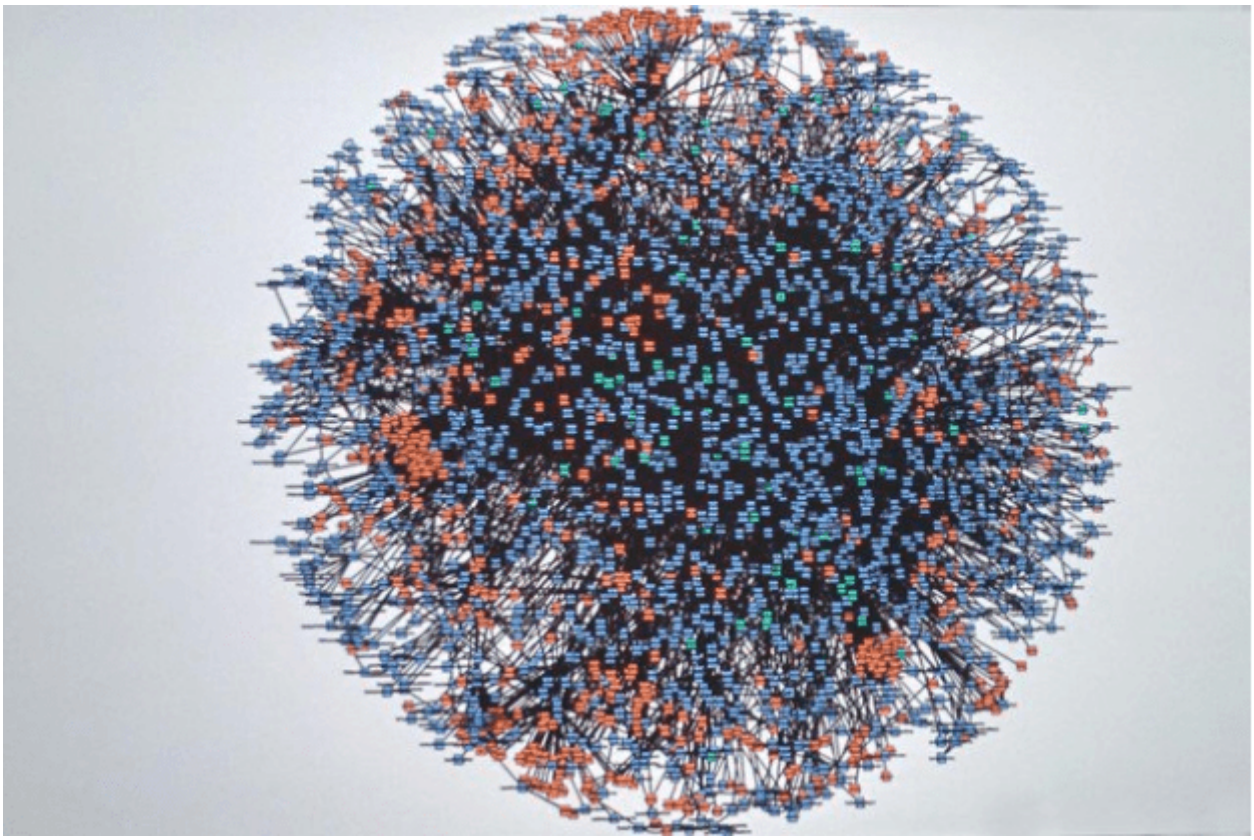
Netflix


Nombre d'abonnés au service streaming de Netflix dans le monde du 1 trimestre 2013 au 4 trimestre 2023 (en millions)
Abonnés en millions
300
250
200
150
100
50
0
T1 2013
T3 2013
T1 2014
T3 2014
T1 2015
T3 2015
T1 2016
T3 2016
T1 2017
T3 2017
T1 2018
T3 2018
T1 2019
T3 2019
T1 2020
T3 2020
T1 2021
T3 2021
T1 2022
T3 2022
T1 2023
T3 2023
Source
Netflix
© Statista 2024
Informations complémentaires:
Monde; Netflix; T1 2013 - T4 2023; hors essais gratuits

Netflix

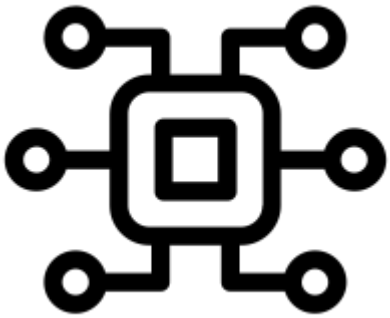

NETFLIX
YouTube / TechDummies
Narendra L
NAREN.LG@gmail.com
Chaos Monkey
TITUS
CASSANDRA
MYSQL Billing info
EV Cache
RESPONSE
Outbound filter
Critical Micro Services
MICRO SERVICES
Netty Server
Endpoint filter
Hystrix
ELB
Service client
Micro Services
Inbound filter
REQUEST
ZULU
OpenConnect
CHUKWA
S3
EMR
Amazon EMR
a managed Hadoop framework
Queue
New video validatation transcoding
Transcoder
concurrent async workers
KAFKA
Message Router
KAFKA
Events data
Elasticsearch
Spark
Apache Samza
EVENT PROCESSING/AGG/Monitor

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

Consul

APACHE 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 réplicas.

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 perlet 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 demndes 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 faie 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.

> `Disponilité` et `Scalabilité` sont des propriétés de Kubernetes.

Le cloud et les microservices

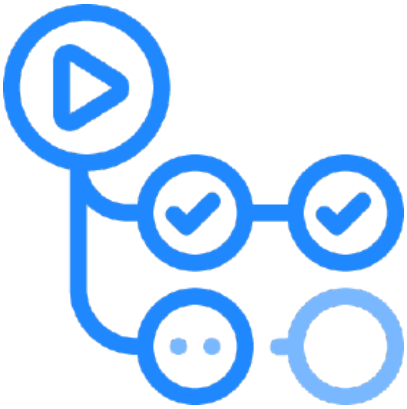
Le cloud permet de déployer des architectures microservices.

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 seul 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.

Pou cela il faut décomposer le projet en services.

Conclusion

Conclusions