



Étude de cas Distribution des données avec Elasticsearch

Loïc Herman et Jonas Troeltsch

10 juin 2024

HEIG-VD — MAC

2024-06-10

Étude de cas Distribution des données avec Elasticsearch



Étude de cas
Distribution des données avec Elasticsearch

Loïc Herman et Jonas Troeltsch
10 juin 2024
HEIG-VD — MAC

- 1 Un système distribué par défaut
 - 1.1 Architecture
 - 1.2 Mécanismes : disaster recovery
 - 1.3 Mécanismes : gestion des nœuds et équilibrage
 - 1.4 Mécanismes : gestion des répliques
 - 1.5 Tolérance au partitionnement
 - 1.6 Service discovery

- 2 Une refonte récente en prévention des dirty reads
 - 2.1 Un besoin de restructuration
 - 2.2 Ingestion des documents
 - 2.3 Système de prévention
 - 2.4 Problèmes évités

- 3 Bibliographie

Plan

- 1 Un système distribué par défaut
 - 1.1 Architecture
 - 1.2 Mécanismes : disaster recovery
 - 1.3 Mécanismes : gestion des nœuds et équilibrage
 - 1.4 Mécanismes : gestion des répliques
 - 1.5 Tolérance au partitionnement
 - 1.6 Service discovery
- 2 Une refonte récente en prévention des dirty reads
 - 2.1 Un besoin de restructuration
 - 2.2 Ingestion des documents
 - 2.3 Système de prévention
 - 2.4 Problèmes évités
- 3 Bibliographie

2024-06-10

- Étude de cas Distribution des données avec Elasticsearch
 - └ Un système distribué par défaut
 - └ Architecture

Un système distribué par défaut

Architecture

Un système distribué par défaut

Architecture

Elasticsearch utilise une architecture sans partage, un cluster contiendra un nœud master et un système de coordination entre les nœuds pour coordonner les opérations.

- Chaque nœud gère ses processeurs, RAM et disques
- Un nœud contient des shards et fait automatiquement partie d'un cluster
- Chaque shard est une instance d'Apache Lucene

- └─ Un système distribué par défaut
 - └─ Architecture
 - └─ Architecture sans partage

Une shard est une instance de Apache Lucene, un noeud une instance d'elasticsearch, plusieurs noeuds communiquent ensemble via le système de cluster coordination

Elasticsearch utilise une architecture sans partage, un cluster contiendra un nœud master et un système de coordination entre les nœuds pour coordonner les opérations.

- Chaque nœud gère ses processeurs, RAM et disques
- Un nœud contient des shards et fait automatiquement partie d'un cluster
- Chaque shard est une instance d'Apache Lucene

- **Nœud master** : création/suppression d'indexes, management du cluster, connaît l'emplacement des documents
- **Nœud de données** : indexation, recherche, suppression et autres opérations liée aux documents
- **Nœud de coordination** : agit en tant que gestionnaire de travail, distribuant les requêtes entrantes aux nœuds appropriés et répondant au client

- └ Un système distribué par défaut
 - └ Architecture
 - └ Les nœuds et leurs responsabilités

Noeud maître : Gère les opérations de haut niveau comme la création/suppression d'index et les tâches administratives. Un seul noeud maître suffit pour le cluster. En cas de panne, un autre noeud est élu maître. Il ne gère pas les opérations CRUD des documents mais connaît leur emplacement.

Noeud de données : Gère les opérations liées aux documents comme l'indexation, la recherche et la suppression. Ces noeuds stockent les documents indexés et effectuent des opérations intensives en I/O disque et en mémoire.

Noeud de coordination : Chaque noeud peut gérer les demandes des clients, les distribuer aux noeuds appropriés et renvoyer les résultats au client.

Les nœuds et leurs responsabilités	
•	Nœud master : création/suppression d'indexes, management du cluster, connaît l'emplacement des documents
•	Nœud de données : indexation, recherche, suppression et autres opérations liée aux documents
•	Nœud de coordination : agit en tant que gestionnaire de travail, distribuant les requêtes entrantes aux nœuds appropriés et répondant au client

- Chaque index est séparé équitablement en shards sur les nœuds
- Une shard primaire peut avoir des répliques qui ne font que des lectures

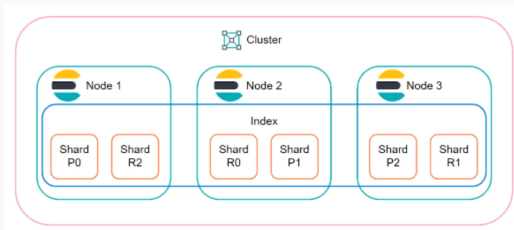


Figure 1 : Shards et répliques dans un cluster

2024-06-10

Étude de cas Distribution des données avec Elasticsearch

- └ Un système distribué par défaut
 - └ Architecture
 - └ Partitionnement et réplication

- Chaque index est séparé équitablement en shards sur les nœuds
- Une shard primaire peut avoir des répliques qui ne font que des lectures



Figure 1 : Shards et répliques dans un cluster

la shard primaire s'occupe de recevoir toutes les demandes de create update et delete
les shards contiennent toutes les données, donc c'est important d'avoir une taille appropriée

Elasticsearch définit trois agrégations pour représenter l'état du système :

- 1. **RED** : Certains nœuds possèdent des shards qui ne sont pas assignés,
- 2. **YELLOW** : Les shards sont toutes assignés et prêts, mais pas les replicas,
- 3. **GREEN** : Toutes les shards et les replicas sont assignés et prêts à l'emploi.

Elasticsearch définit trois agrégations pour représenter l'état du système :

- 1. **RED** : Certains nœuds possèdent des shards qui ne sont pas assignés,
- 2. **YELLOW** : Les shards sont toutes assignés et prêts, mais pas les replicas,
- 3. **GREEN** : Toutes les shards et les replicas sont assignés et prêts à l'emploi.

Elasticsearch utilise un modèle de partitionnement par hachage de l'ID pour déterminer sur quelle shard un document doit aller.

$$S_n = \text{hash}(\text{doc}_{\text{id}}) \bmod N_{\text{shards}}$$

- Introduite en 2022
- Séparation du stockage des données de l'indexation et de la recherche.
- 2 types de nœuds
 - nœud d'index
 - nœud de recherche
- Permet l'utilisation d'un service de stockage externe (S3, GCS, Azure Blob Storage)

2024-06-10

Étude de cas Distribution des données avec Elasticsearch

- └ Un système distribué par défaut
 - └ Architecture
 - └ Alternative : architecture stateless

- Introduite en 2022
- Séparation du stockage des données de l'indexation et de la recherche.
- 2 types de nœuds
 - nœud d'index
 - nœud de recherche
- Permet l'utilisation d'un service de stockage externe (S3, GCS, Azure Blob Storage)

Avantages de l'architecture stateless :

- Évolutivité : Les nœuds Elasticsearch peuvent être ajoutés ou supprimés sans affecter les données stockées dans le service de stockage externe.
- Élasticité : L'architecture stateless permet de s'adapter dynamiquement à la charge de travail en ajoutant ou supprimant des nœuds Elasticsearch au besoin.
- Simplicité : La séparation du stockage des données de l'indexation et de la recherche simplifie la gestion et l'administration du cluster Elasticsearch.

Alternative : architecture stateless

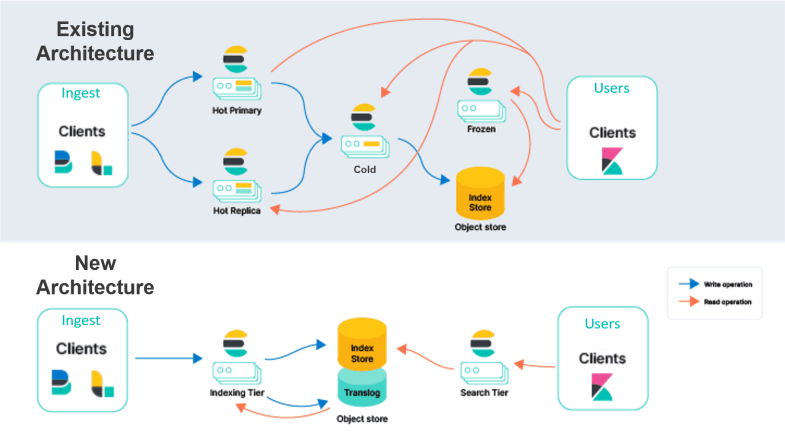
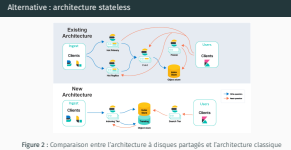


Figure 2 : Comparaison entre l'architecture à disques partagés et l'architecture classique

Un système distribué par défaut

2024-06-10

Étude de cas Distribution des données avec Elasticsearch
└ Un système distribué par défaut
└ Architecture
└ Alternative : architecture stateless



- Avantages de l'architecture stateless :
- Évolutivité : Les noeuds Elasticsearch peuvent être ajoutés ou supprimés sans affecter les données stockées dans le service de stockage externe.
 - Élasticité : L'architecture stateless permet de s'adapter dynamiquement à la charge de travail en ajoutant ou supprimant des noeuds Elasticsearch au besoin.
 - Simplicité : La séparation du stockage des données de l'indexation et de la recherche simplifie la gestion et l'administration du cluster Elasticsearch.

2024-06-10

- Étude de cas Distribution des données avec Elasticsearch
 - └ Un système distribué par défaut
 - └ Mécanismes : disaster recovery

Un système distribué par défaut
Mécanismes : disaster recovery

Un système distribué par défaut

Mécanismes : disaster recovery

2024-06-10

Étude de cas Distribution des données avec Elasticsearch

- └ Un système distribué par défaut
 - └ Mécanismes : disaster recovery
 - └ En cas de panne

Elasticsearch implémente un modèle simple de disaster recovery

- **Panne d'un nœud data** : une des répliques de chaque shards primaires sera élue primaire
- Panne d'un nœud master : élection d'un nouveau nœud master par les nœuds éligibles
- Panne d'un cluster entier : possibilité de failover sur un cluster de copie qui reprend la charge

Elasticsearch implémente un modèle simple de disaster recovery

- **Panne d'un nœud data** : une des répliques de chaque shards primaires sera élue primaire
- Panne d'un nœud master : élection d'un nouveau nœud master par les nœuds éligibles
- Panne d'un cluster entier : possibilité de failover sur un cluster de copie qui reprend la charge

évtl. maintient un transaction log

- └ Un système distribué par défaut
 - └ Mécanismes : disaster recovery
 - └ En cas de panne

Elasticsearch implémente un modèle simple de disaster recovery

- **Panne d'un nœud data** : une des répliques de chaque shards primaires sera élue primaire
- **Panne d'un nœud master** : élection d'un nouveau nœud master par les nœuds éligibles
- **Panne d'un cluster entier** : possibilité de failover sur un cluster de copie qui reprend la charge

Elasticsearch implémente un modèle simple de disaster recovery

- **Panne d'un nœud data** : une des répliques de chaque shards primaires sera élue primaire
- **Panne d'un nœud master** : élection d'un nouveau nœud master par les nœuds éligibles
- **Panne d'un cluster entier** : possibilité de failover sur un cluster de copie qui reprend la charge

évtl. maintient un transaction log

2024-06-10

- Étude de cas Distribution des données avec Elasticsearch
 - └ Un système distribué par défaut
 - └ Mécanismes : disaster recovery
 - └ En cas de panne

Elasticsearch implémente un modèle simple de disaster recovery

- Panne d'un nœud data : une des répliques de chaque shards primaires sera élue primaire
- Panne d'un nœud master : élection d'un nouveau nœud master par les nœuds éligibles
- Panne d'un cluster entier : possibilité de failover sur un cluster de copie qui reprend la charge

Elasticsearch implémente un modèle simple de disaster recovery

- Panne d'un nœud data : une des répliques de chaque shards primaires sera élue primaire
- Panne d'un nœud master : élection d'un nouveau nœud master par les nœuds éligibles
- Panne d'un cluster entier : possibilité de failover sur un cluster de copie qui reprend la charge

évtl. maintient un transaction log

2024-06-10

- Étude de cas Distribution des données avec Elasticsearch
 - └ Un système distribué par défaut
 - └ Mécanismes : gestion des nœuds et équilibrage

Un système distribué par défaut

Mécanismes : gestion des nœuds et équilibrage

Un système distribué par défaut

Mécanismes : gestion des nœuds et équilibrage

Ajout d'un nœud à un cluster

En cas d'ajout d'un nœud, Elasticsearch va automatiquement commencer à équilibrer la charge de données entre tous les nœuds présents dans le cluster.

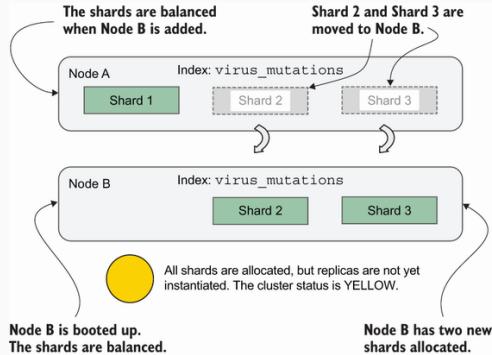
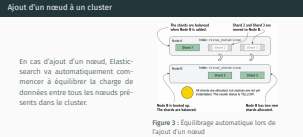


Figure 3 : Équilibrage automatique lors de l'ajout d'un nœud

- 2024-06-10
- Étude de cas Distribution des données avec Elasticsearch
 - Un système distribué par défaut
 - Mécanismes : gestion des nœuds et équilibrage
 - Ajout d'un nœud à un cluster

Commenter le diagramme et comment les shards primaires ont été déplacées.



Une fois les shards « primaires » initialisées, les répliques peuvent être ajoutées.

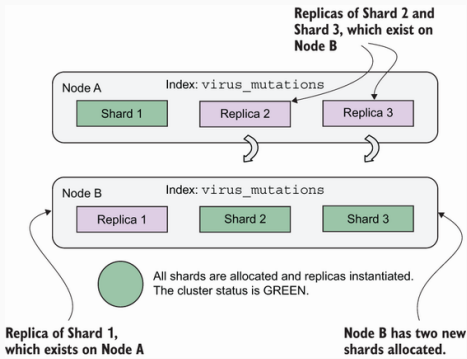
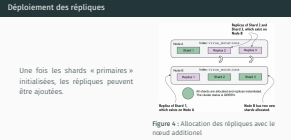


Figure 4 : Allocation des répliques avec le nœud additionel

2024-06-10

- Étude de cas Distribution des données avec Elasticsearch
- Un système distribué par défaut
 - Mécanismes : gestion des nœuds et équilibrage
 - Déploiement des répliques

Commenter le diagramme avec l'apparition des répliques, ne pas préciser comment la réplication est déroulée (présenté ensuite)



2024-06-10

Étude de cas Distribution des données avec Elasticsearch

└─ Un système distribué par défaut

└─ Mécanismes : gestion des répliques

Un système distribué par défaut
Mécanismes : gestion des répliques

Un système distribué par défaut

Mécanismes : gestion des répliques

Lors de la création d'une nouvelle réplique, un processus standard est utilisé :

1. Snapshot de la shard « primaire » avec le *sequence number* associé
2. Copie de la snapshot dans la réplique
3. Mise à jour de la copie selon la différence du *sequence number* actuel de la shard « primaire »

Lors de la création d'une nouvelle réplique, un processus standard est utilisé :

1. Snapshot de la shard « primaire » avec le *sequence number* associé
2. Copie de la snapshot dans la réplique
3. Mise à jour de la copie selon la différence du *sequence number* actuel de la shard « primaire »

Lors de la création d'une nouvelle réplique, un processus standard est utilisé :

1. Snapshot de la shard « primaire » avec le *sequence number* associé
2. Copie de la snapshot dans la réplique
3. Mise à jour de la copie selon la différence du *sequence number* actuel de la shard « primaire »

- Il n'est **pas possible** d'augmenter le nombre de shards d'un index **actif**.
- ⇒ Le nombre de shards doit donc être spécifié **à la création** d'un index (par défaut 1 shard et 1 réplique).
- Il est néanmoins possible de changer le nombre de répliques de chaque shards via la configuration de l'index.
- ⇒ Les nouvelles répliques seront automatiquement **dispersées** dans les nœuds du cluster de sorte à garantir une **tolérance aux pannes**.

- └ Un système distribué par défaut
 - └ Mécanismes : gestion des répliques
 - └ Ajout de shards et répliques

- Il n'est **pas possible** d'augmenter le nombre de shards d'un index **actif**.
- ⇒ Le nombre de shards doit donc être spécifié **à la création** d'un index (par défaut 1 shard et 1 réplique).
- Il est néanmoins possible de changer le nombre de répliques de chaque shards via la configuration de l'index.
- ⇒ Les nouvelles répliques seront automatiquement dispersées dans les nœuds du cluster de sorte à garantir une tolérance aux pannes.

- Il n'est **pas possible** d'augmenter le nombre de shards d'un index **actif**.
- ⇒ Le nombre de shards doit donc être spécifié **à la création** d'un index (par défaut 1 shard et 1 réplique).
- Il est néanmoins possible de changer le nombre de répliques de chaque shards via la configuration de l'index.
- ⇒ Les nouvelles répliques seront automatiquement **dispersées** dans les nœuds du cluster de sorte à garantir une **tolérance aux pannes**.

- └ Un système distribué par défaut
 - └ Mécanismes : gestion des répliques
 - └ Ajout de shards et répliques

- Il n'est **pas possible** d'augmenter le nombre de shards d'un index **actif**.
- ⇒ Le nombre de shards doit donc être spécifié **à la création** d'un index (par défaut 1 shard et 1 réplique).
- Il est néanmoins possible de changer le nombre de répliques de chaque shards via la configuration de l'index.
- ⇒ Les nouvelles répliques seront automatiquement dispersées dans les nœuds du cluster de sorte à garantir une tolérance aux pannes.

- Il n'est **pas possible** d'augmenter le nombre de shards d'un index **actif**.
- ⇒ Le nombre de shards doit donc être spécifié **à la création** d'un index (par défaut 1 shard et 1 réplique).
- Il est néanmoins possible de changer le nombre de répliques de chaque shards via la configuration de l'index.
- ⇒ Les nouvelles répliques seront automatiquement **dispersées** dans les nœuds du cluster de sorte à garantir une **tolérance aux pannes**.

- └ Un système distribué par défaut
 - └ Mécanismes : gestion des répliques
 - └ Ajout de shards et répliques

Ajout de shards et répliques

- Il n'est pas possible d'augmenter le nombre de shards d'un index actif.
- ⇒ Le nombre de shards doit donc être spécifié à la création d'un index (par défaut 1 shard et 1 réplique).
- Il est néanmoins possible de changer le nombre de répliques de chaque shards via la configuration de l'index.
- ⇒ Les nouvelles répliques seront automatiquement **dispersées** dans les nœuds du cluster de sorte à garantir une **tolérance aux pannes**.

- Il n'est **pas possible** d'augmenter le nombre de shards d'un index **actif**.
- ⇒ Le nombre de shards doit donc être spécifié **à la création** d'un index (par défaut 1 shard et 1 réplique).
- Il est néanmoins possible de changer le nombre de répliques de chaque shards via la configuration de l'index.
- ⇒ Les nouvelles répliques seront automatiquement **dispersées** dans les nœuds du cluster de sorte à garantir une **tolérance aux pannes**.

- └ Un système distribué par défaut
 - └ Mécanismes : gestion des répliques
 - └ Ajout de shards et répliques

Ajout de shards et répliques

- Il n'est pas possible d'augmenter le nombre de shards d'un index actif.
- ⇒ Le nombre de shards doit donc être spécifié à la création d'un index (par défaut 1 shard et 1 réplique).
- Il est néanmoins possible de changer le nombre de répliques de chaque shards via la configuration de l'index.
- ⇒ Les nouvelles répliques seront automatiquement **dispersées** dans les nœuds du cluster de sorte à garantir une **tolérance aux pannes**.

Elasticsearch utilise plusieurs facteurs décisifs pour déterminer qu'un équilibrage des shards est nécessaire et, si oui, comment y procéder :

- **ClusterRebalance** : autorise un redémarrage selon le statut du cluster,
- **DiskThreshold** : permet de décider si un nœud est capable d'accueillir une shard,
- et bien d'autres facteurs permettant d'accomplir la décision.

En utilisant la combinaison des facteurs configurés, les shards seront déplacées sur les nœuds correspondants, ce processus se déroule **automatiquement**.

- └ Un système distribué par défaut
 - └ Mécanismes : gestion des répliques
 - └ Équilibre des répliques par nœud

2024-06-10

- Étude de cas Distribution des données avec Elasticsearch
 - └ Un système distribué par défaut
 - └ Tolérance au partitionnement

Un système distribué par défaut
Tolérance au partitionnement

Un système distribué par défaut

Tolérance au partitionnement

- Elasticsearch se caractérise par la réplication et la coordination entre les nœuds d'un cluster.
 - Les données sont toujours répliquées sur plusieurs shards, idéalement selon la configuration sur plusieurs nœuds.
- ⇒ La tolérance au partitionnement de l'infrastructure est toujours garantie.

2024-06-10

Étude de cas Distribution des données avec Elasticsearch

- └ Un système distribué par défaut
 - └ Tolérance au partitionnement
 - └ Garanties au sens du théorème de CAP

- Elasticsearch se caractérise par la réplication et la coordination entre les nœuds d'un cluster.
 - Les données sont toujours répliquées sur plusieurs shards, idéalement selon la configuration sur plusieurs nœuds.
- ⇒ La tolérance au partitionnement de l'infrastructure est toujours garantie.

- Elasticsearch se caractérise par la réplication et la coordination entre les nœuds d'un cluster.
 - Les données sont toujours répliquées sur plusieurs shards, idéalement selon la configuration sur plusieurs nœuds.
- ⇒ La tolérance au partitionnement de l'infrastructure est toujours garantie.

- └ Un système distribué par défaut
 - └ Tolérance au partitionnement
 - └ Garanties au sens du théorème de CAP

- ⇒ La tolérance au partitionnement de l'infrastructure est toujours garantie.
- Toutefois, Elasticsearch traite de deux manières différentes les accès :
 1. **En cas d'écriture** : la cohérence est privilégiée à la disponibilité.
 - Une écriture acquittée ne devrait jamais être perdue
 - Mais, un nœud rompu du système aura toujours les écritures bloquées
 - ⇒ La disponibilité en écriture n'est pas garantie en cas de partition du système
 2. **En cas de lecture** : la disponibilité est privilégiée à la cohérence
 - ⇒ Il y a une plus forte garantie en disponibilité à la lecture, en échange d'une pénalité de cohérence

2024-06-10

- ## Étude de cas Distribution des données avec Elasticsearch
- └ Un système distribué par défaut
 - └ Tolérance au partitionnement
 - └ Garanties au sens du théorème de CAP

Garanties au sens du théorème de CAP

⇒ La tolérance au partitionnement de l'infrastructure est toujours garantie.

- Toutefois, Elasticsearch traite de deux manières différentes les accès :
 1. En cas d'écriture : la cohérence est privilégiée à la disponibilité.
 - Une écriture acquittée ne devrait jamais être perdue
 - Mais, un nœud rompu du système aura toujours les écritures bloquées
 - ⇒ La disponibilité en écriture n'est pas garantie en cas de partition du système
 2. En cas de lecture : la disponibilité est privilégiée à la cohérence
 - ⇒ Il y a une plus forte garantie en disponibilité à la lecture, en échange d'une pénalité de cohérence

- ⇒ La tolérance au partitionnement de l'infrastructure est toujours garantie.
- Toutefois, Elasticsearch traite de deux manières différentes les accès :
 1. **En cas d'écriture** : la cohérence est privilégiée à la disponibilité.
 - Une écriture acquittée ne devrait jamais être perdue
 - Mais, un nœud rompu du système aura toujours les écritures bloquées
- ⇒ La disponibilité en écriture n'est pas garantie en cas de partition du système
2. **En cas de lecture** : la disponibilité est privilégiée à la cohérence
- ⇒ Il y a une plus forte garantie en disponibilité à la lecture, en échange d'une pénalité de cohérence

2024-06-10

- ## Étude de cas Distribution des données avec Elasticsearch
- └ Un système distribué par défaut
 - └ Tolérance au partitionnement
 - └ Garanties au sens du théorème de CAP

Garanties au sens du théorème de CAP

⇒ La tolérance au partitionnement de l'infrastructure est toujours garantie.

- Toutefois, Elasticsearch traite de deux manières différentes les accès :
 1. En cas d'écriture : la cohérence est privilégiée à la disponibilité.
 - Une écriture acquittée ne devrait jamais être perdue
 - Mais, un nœud rompu du système aura toujours les écritures bloquées
 - ⇒ La disponibilité en écriture n'est pas garantie en cas de partition du système
 2. En cas de lecture : la disponibilité est privilégiée à la cohérence

⇒ Il y a une plus forte garantie en disponibilité à la lecture, en échange d'une pénalité de cohérence

- ⇒ La tolérance au partitionnement de l'infrastructure est toujours garantie.
- Toutefois, Elasticsearch traite de deux manières différentes les accès :
 1. **En cas d'écriture** : la cohérence est privilégiée à la disponibilité.
⇒ La disponibilité en écriture n'est pas garantie en cas de partition du système
 2. **En cas de lecture** : la disponibilité est privilégiée à la cohérence
 - Selon la configuration, une recherche peut privilégier de retourner un vieux résultat plutôt que de retourner une erreur
- ⇒ Il y a une plus forte garantie en disponibilité à la lecture, en échange d'une pénalité de cohérence

2024-06-10

- ## Étude de cas Distribution des données avec Elasticsearch
- └ Un système distribué par défaut
 - └ Tolérance au partitionnement
 - └ Garanties au sens du théorème de CAP

Garanties au sens du théorème de CAP

⇒ La tolérance au partitionnement de l'infrastructure est toujours garantie.

- Toutefois, Elasticsearch traite de deux manières différentes les accès :
 1. En cas d'écriture : la cohérence est privilégiée à la disponibilité.
⇒ La disponibilité en écriture n'est pas garantie en cas de partition du système
 2. En cas de lecture : la disponibilité est privilégiée à la cohérence
 - Selon la configuration, une recherche peut privilégier de retourner un vieux résultat plutôt que de retourner une erreur

⇒ Il y a une plus forte garantie en disponibilité à la lecture, en échange d'une pénalité de cohérence

- ⇒ La tolérance au partitionnement de l'infrastructure est toujours garantie.
- Toutefois, Elasticsearch traite de deux manières différentes les accès :
 1. **En cas d'écriture** : la cohérence est privilégiée à la disponibilité.
⇒ La disponibilité en écriture n'est pas garantie en cas de partition du système
 2. **En cas de lecture** : la disponibilité est privilégiée à la cohérence
 - Selon la configuration, une recherche peut privilégier de retourner un vieux résultat plutôt que de retourner une erreur
- ⇒ Il y a une plus forte garantie en disponibilité à la lecture, en échange d'une pénalité de cohérence

2024-06-10

- ## Étude de cas Distribution des données avec Elasticsearch
- └ Un système distribué par défaut
 - └ Tolérance au partitionnement
 - └ Garanties au sens du théorème de CAP

Garanties au sens du théorème de CAP

⇒ La tolérance au partitionnement de l'infrastructure est toujours garantie.

- Toutefois, Elasticsearch traite de deux manières différentes les accès :
 1. En cas d'écriture : la cohérence est privilégiée à la disponibilité.
⇒ La disponibilité en écriture n'est pas garantie en cas de partition du système
 2. En cas de lecture : la disponibilité est privilégiée à la cohérence
 - Selon la configuration, une recherche peut privilégier de retourner un vieux résultat plutôt que de retourner une erreur

⇒ Il y a une plus forte garantie en disponibilité à la lecture, en échange d'une pénalité de cohérence

- ⇒ La tolérance au partitionnement de l'infrastructure est toujours garantie.
- Toutefois, Elasticsearch traite de deux manières différentes les accès :
 1. **En cas d'écriture** : la cohérence est privilégiée à la disponibilité.
 - ⇒ La disponibilité en écriture n'est pas garantie en cas de partition du système
 2. **En cas de lecture** : la disponibilité est privilégiée à la cohérence
 - ⇒ Il y a une plus forte garantie en disponibilité à la lecture, en échange d'une pénalité de cohérence
- On peut malgré tout définir Elasticsearch comme étant un système principalement Consistent-Partition (CP) au sens de la conjecture de Brewer, car le fonctionnement principal tourne autour du stockage de documents répliqués et tolérants aux pannes.

2024-06-10

- Étude de cas Distribution des données avec Elasticsearch
- └ Un système distribué par défaut
 - └ Tolérance au partitionnement
 - └ Garanties au sens du théorème de CAP

Garanties au sens du théorème de CAP

⇒ La tolérance au partitionnement de l'infrastructure est toujours garantie.

- Toutefois, Elasticsearch traite de deux manières différentes les accès :
 1. **En cas d'écriture** : la cohérence est privilégiée à la disponibilité.
 - ⇒ La disponibilité en écriture n'est pas garantie en cas de partition du système
 2. **En cas de lecture** : la disponibilité est privilégiée à la cohérence
 - ⇒ Il y a une plus forte garantie en disponibilité à la lecture, en échange d'une pénalité de cohérence

■ On peut malgré tout définir Elasticsearch comme étant un système principalement Consistent-Partition (CP) au sens de la conjecture de Brewer, car le fonctionnement principal tourne autour du stockage de documents répliqués et tolérants aux pannes.

2024-06-10

Étude de cas Distribution des données avec Elasticsearch

└─ Un système distribué par défaut

└─ Service discovery

Un système distribué par défaut
Service discovery

Un système distribué par défaut

Service discovery

Elasticsearch prend en charge plusieurs types de *service discovery* :

- **Seed-based discovery** : une liste d'adresses IP et/ou une liste de noms de domaines dont les adresses liées seront résolues unes-à-unes
- **Cloud-based discovery** : des plugins sont disponibles pour connecter le service discovery avec les services de cloud providers (AWS EC2, Azure Classic, Google Compute Engine)

Ces listes d'adresses seront ensuite évaluées et si un nœud elastic est présent alors il sera rajouté dans le cluster avec la gestion d'état partagée.

- └ Un système distribué par défaut
 - └ Service discovery
 - └ Résolution par adresse ou cloud providers

Elasticsearch prend en charge plusieurs types de *service discovery* :

- **Seed-based discovery** : une liste d'adresses IP et/ou une liste de noms de domaines dont les adresses liées seront résolues unes-à-unes
- **Cloud-based discovery** : des plugins sont disponibles pour connecter le service discovery avec les services de cloud providers (AWS EC2, Azure Classic, Google Compute Engine)

Ces listes d'adresses seront ensuite évaluées et si un nœud elastic est présent alors il sera rajouté dans le cluster avec la gestion d'état partagée.

Comme indiqué dans l'architecture, chaque nœud d'un cluster Elasticsearch possède des rôles.

Il y a un rôle que tous les nœuds reçoivent automatiquement : **Coordinateur**.

Ce rôle demande donc de répondre aux requêtes des clients, en transmettant la requête à tous les nœuds connus du système, agréant les résultats obtenus.

2024-06-10

- Étude de cas Distribution des données avec Elasticsearch
 - └ Un système distribué par défaut
 - └ Service discovery
 - └ Tous les nœuds peuvent répondre

Comme indiqué dans l'architecture, chaque nœud d'un cluster Elasticsearch possède des rôles.
Il y a un rôle que tous les nœuds reçoivent automatiquement : **Coordinateur**.
Ce rôle demande donc de répondre aux requêtes des clients, en transmettant la requête à tous les nœuds connus du système, agréant les résultats obtenus.

Une refonte récente en prévention des dirty reads

Un besoin de restructuration

2024-06-10

- Étude de cas Distribution des données avec Elasticsearch
 - └ Une refonte récente en prévention des dirty reads
 - └ Un besoin de restructuration

Une refonte récente en prévention
des dirty reads

Un besoin de restructuration

La version 7.0 d'Elasticsearch a été une version plus que majeure, comprenant une série d'améliorations conséquentes quant au système de **coordination des clusters** et l'ajout de *log sequence numbers* sur toutes les opérations d'écriture.

Ces changements ont par la suite permis d'offrir le système de réindexation permettant de cloner un index vers un autre en appliquant éventuellement une mutation.

Dans les composantes internes, cela a pu permettre d'optimiser le temps de récupération des shards désynchronisées en permettant d'éviter de devoir sourcer les fichiers sous-jacents.

2024-06-10

- Étude de cas Distribution des données avec Elasticsearch
 - └ Une refonte récente en prévention des dirty reads
 - └ Un besoin de restructuration
 - └ Un petit peu d'histoire

La version 7.0 d'Elasticsearch a été une version plus que majeure, comprenant une série d'améliorations conséquentes quant au système de **coordination des clusters** et l'ajout de *log sequence numbers* sur toutes les opérations d'écriture. Ces changements ont par la suite permis d'offrir le système de réindexation permettant de cloner un index vers un autre en appliquant éventuellement une mutation. Dans les composantes internes, cela a pu permettre d'optimiser le temps de récupération des shards désynchronisées en permettant d'éviter de devoir sourcer les fichiers sous-jacents.

Ces changements ont aussi pu permettre de limiter l’impact qu’avaient anciennement les partitionnements réseau, qui pouvaient causer des **divergences, pertes d’écriture** et des **dirty reads**.

Nous allons donc nous concentrer plus en détail sur la façon dont ces changements ont pu permettre d’améliorer la gestion de ces cas d’erreurs, principalement dans le cadre des lectures.

- 2024-06-10
- Étude de cas Distribution des données avec Elasticsearch
 - └ Une refonte récente en prévention des dirty reads
 - └ Un besoin de restructuration
 - └ Dirty reads

Dirty reads

Ces changements ont aussi pu permettre de limiter l’impact qu’avaient anciennement les partitionnements réseau, qui pouvaient causer des **divergences, pertes d’écriture** et des **dirty reads**.

Nous allons donc nous concentrer plus en détail sur la façon dont ces changements ont pu permettre d’améliorer la gestion de ces cas d’erreurs, principalement dans le cadre des lectures.

Une refonte récente en prévention des dirty reads

Ingestion des documents

2024-06-10

- Étude de cas Distribution des données avec Elasticsearch
 - └ Une refonte récente en prévention des dirty reads
 - └ Ingestion des documents

Une refonte récente en prévention
des dirty reads

Ingestion des documents

Elasticsearch possède deux pipelines d'accès aux données :

- 1. **Les écritures** passent via chaîne de nœuds par un processus synchrone avant d'être traité indépendamment dans la shard primaire et ensuite, parallèlement, ses répliques,
- 2. **Les lectures** se font en parallèle sur chaque shard qui contient les données pertinentes, chaque nœud du système peut recevoir une demande et sera responsable d'agréger les résultats retournés par toutes les shards pertinentes

2024-06-10

- Étude de cas Distribution des données avec Elasticsearch
 - └ Une refonte récente en prévention des dirty reads
 - └ Ingestion des documents
 - └ Un traitement en deux parties

Un traitement en deux parties

Elasticsearch possède deux pipelines d'accès aux données :

1. **Les écritures** passent via chaîne de nœuds par un processus synchrone avant d'être traité indépendamment dans la shard primaire et ensuite, parallèlement, ses répliques,

2. **Les lectures** se font en parallèle sur chaque shard qui contient les données pertinentes, chaque nœud du système peut recevoir une demande et sera responsable d'agréger les résultats retournés par toutes les shards pertinentes

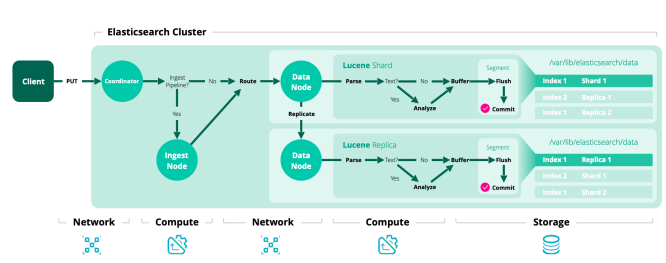


Figure 5 : Étapes lors de la réception des documents

Les opérations sont d'abord validées par la shard « primaire » routée pour le document, ensuite l'opération est effectuée par cette dernière. Les répliques vont ensuite effectuer la même opération en parallèle pour conserver les copies.

2024-06-10

- Étude de cas Distribution des données avec Elasticsearch
 - Une refonte récente en prévention des dirty reads
 - Ingestion des documents
 - Pipeline d'ingestion

Pipeline d'ingestion

Figure 5 : Étapes lors de la réception des documents

Les opérations sont d'abord validées par la shard « primaire » routée pour le document, ensuite l'opération est effectuée par cette dernière. Les répliques vont ensuite effectuer la même opération en parallèle pour conserver les copies.

Une refonte récente en prévention des dirty reads

Système de prévention

2024-06-10

- Étude de cas Distribution des données avec Elasticsearch
 - └ Une refonte récente en prévention des dirty reads
 - └ Système de prévention

Il existe donc trois opérations « modifcatrices » sur un index :

- 1. **Indexation** : ajouts de nouveaux documents
- 2. **Updation** : mise à jour des documents
- 3. **Deletion** : suppression de documents

Jusqu’à la version 7.0, Elasticsearch ne possédait pas de système pour tracer l’ordre d’exécution de ces trois types d’opérations lors de la copie dans les répliques.

Il a donc fallu rajouter un système de *log sequence numbers* à ces opérations d’écriture pour pouvoir détecter les différences de traitement en cas de partitionnement.

2024-06-10

- Étude de cas Distribution des données avec Elasticsearch
 - └ Une refonte récente en prévention des dirty reads
 - └ Système de prévention
 - └ Suivi des modifications

Mentionner rapport à la pipeline d’ingestion

Suivi des modifications

Il existe donc trois opérations « modifcatrices » sur un index :

- 1. **Indexation** : ajouts de nouveaux documents
- 2. **Updation** : mise à jour des documents
- 3. **Deletion** : suppression de documents

Jusqu’à la version 7.0, Elasticsearch ne possédait pas de système pour tracer l’ordre d’exécution de ces trois types d’opérations lors de la copie dans les répliques.
Il a donc fallu rajouter un système de *log sequence numbers* à ces opérations d’écriture pour pouvoir détecter les différences de traitement en cas de partitionnement.

Il existe donc trois opérations « modifcatrices » sur un index :

- 1. **Indexation** : ajouts de nouveaux documents
 - 2. **Updation** : mise à jour des documents
 - 3. **Deletion** : suppression de documents
- Jusqu’à la version 7.0, Elasticsearch ne possédait pas de système pour tracer l’ordre d’exécution de ces trois types d’opérations lors de la copie dans les répliques.
- Il a donc fallu rajouter un système de *log sequence numbers* à ces opérations d’écriture pour pouvoir détecter les différences de traitement en cas de partitionnement.

2024-06-10

- Étude de cas Distribution des données avec Elasticsearch
- └ Une refonte récente en prévention des dirty reads
 - └ Système de prévention
 - └ Suivi des modifications

Mentionner rapport à la pipeline d’ingestion

Suivi des modifications

Il existe donc trois opérations « modifcatrices » sur un index :

- 1. **Indexation** : ajouts de nouveaux documents
- 2. **Updation** : mise à jour des documents
- 3. **Deletion** : suppression de documents

Jusqu’à la version 7.0, Elasticsearch ne possédait pas de système pour tracer l’ordre d’exécution de ces trois types d’opérations lors de la copie dans les répliques.

Il a donc fallu rajouter un système de *log sequence numbers* à ces opérations d’écriture pour pouvoir détecter les différences de traitement en cas de partitionnement.

- └ Une refonte récente en prévention des dirty reads
 - └ Système de prévention
 - └ Définition d'une séquence

Deux nombres sont associés à chaque opération :

- **Terme primaire** : Incrémenté à chaque attribution à une shard primaire, déterminé par le nœud master.
- **Numéro de séquence (seq#)** : Incrémenté par la shard primaire pour chaque opération.

Protocole d'ordonnancement de deux opérations :

- $o1 < o2$ si $s1.seq\# < s2.seq\#$ ou $(s1.seq\# == s2.seq\# \text{ et } s1.term < s2.term)$.

Deux nombres sont associés à chaque opération :

- **Terme primaire** : Incrémenté à chaque attribution à une shard primaire, déterminé par le nœud master.
- **Numéro de séquence (seq#)** : Incrémenté par la shard primaire pour chaque opération.

Protocole d'ordonnancement de deux opérations :

- $o1 < o2$ si $s1.seq\# < s2.seq\#$ ou $(s1.seq\# == s2.seq\# \text{ et } s1.term < s2.term)$.

- └ Une refonte récente en prévention des dirty reads
 - └ Système de prévention
 - └ Définition d'une séquence

Variables utilisées pour déterminer si une shard est correctement synchronisée :

- **Local checkpoint#** : Plus haut seq# pour lequel tous les seq# inférieurs ont localement été traités.
- **Global checkpoint#** : Plus haut seq# traité sur toutes les répliques actives.

Chaque shard maintient en mémoire et dans les métadonnées de chaque commit de Lucene ces deux numéros, permettant de gérer la coordination lors de la réplication des données.

Variables utilisées pour déterminer si une shard est correctement synchronisée :

- **Local checkpoint#** : Plus haut seq# pour lequel tous les seq# inférieurs ont localement été traités.
- **Global checkpoint#** : Plus haut seq# traité sur toutes les répliques actives.

Chaque shard maintient en mémoire et dans les métadonnées de chaque commit de Lucene ces deux numéros, permettant de gérer la coordination lors de la réplication des données.

Une refonte récente en prévention des dirty reads

Problèmes évités

2024-06-10

- Étude de cas Distribution des données avec Elasticsearch
 - └ Une refonte récente en prévention des dirty reads
 - └ Problèmes évités

Une refonte récente en prévention
des dirty reads

Problèmes évités

Lorsqu’une shard « primaire » est isolée du cluster, elle continue à indexer localement.

L’isolement est découvert uniquement lors de la tentative de réplication et la shard ne va donc jamais acquitter l’écriture du document, en attendant une résolution en amont. Celle-ci peut engendrer des problèmes d’incohérences si la résolution ne peut plus déterminer l’ordre d’exécution à utiliser.

Solution : L’infrastructure mise en place permet une identification unique des documents en utilisant les champs de terme primaire et de numéro de séquence, permettant de mieux résoudre un conflit lié à un partitionnement réseau.

2024-06-10

- Étude de cas Distribution des données avec Elasticsearch
 - └ Une refonte récente en prévention des dirty reads
 - └ Problèmes évités
 - └ Indexation pendant un partitionnement

Indexation pendant un partitionnement

Lorsqu’une shard « primaire » est isolée du cluster, elle continue à indexer localement.

L’isolement est découvert uniquement lors de la tentative de réplication et la shard ne va donc jamais acquitter l’écriture du document, en attendant une résolution en amont. Celle-ci peut engendrer des problèmes d’incohérences si la résolution ne peut plus déterminer l’ordre d’exécution à utiliser.

Solution : L’infrastructure mise en place permet une identification unique des documents en utilisant les champs de terme primaire et de numéro de séquence, permettant de mieux résoudre un conflit lié à un partitionnement réseau.

Lorsqu’une shard primaire échoue, une réplique sera promue en primaire. Lors de la promotion, les autres répliques présentes pourraient devenir désynchronisées si des opérations en cours d’envoi ont été perdues lors de la partition.

Problème : Ces désynchronisations ne sont pas corrigées lors de la promotion de la shard primaire, mais sont différées jusqu’à la relocalisation des répliques (nécessitant une relecture des fichiers sous-jacents), rendant la période de désynchronisation indéfinie.

Solution : Les numéros de séquence permettent d’identifier les divergences entre répliques au niveau du document, facilitant la synchronisation efficace des répliques restantes avec le nouveau primaire.

2024-06-10

- Étude de cas Distribution des données avec Elasticsearch
 - └ Une refonte récente en prévention des dirty reads
 - └ Problèmes évités
 - └ Désynchronisation des répliques

Désynchronisation des répliques

Lorsqu’une shard primaire échoue, une réplique sera promue en primaire. Lors de la promotion, les autres répliques présentes pourraient devenir désynchronisées si des opérations en cours d’envoi ont été perdues lors de la partition.

Problème : Ces désynchronisations ne sont pas corrigées lors de la promotion de la shard primaire, mais sont différées jusqu’à la relocalisation des répliques (nécessitant une relecture des fichiers sous-jacents), rendant la période de désynchronisation indéfinie.

Solution : Les numéros de séquence permettent d’identifier les divergences entre répliques au niveau du document, facilitant la synchronisation efficace des répliques restantes avec le nouveau primaire.



Discovery and cluster formation.

<https://l8n.ch/q4wP0>.

[Dernière consultation le 09.06.2024].



Reading and writing documents.

<https://l8n.ch/LH3G4>.

[Dernière consultation le 09.06.2024].



Scalability and resilience : clusters, nodes, and shards.

<https://l8n.ch/JA9Tc>.

[Dernière consultation le 09.06.2024].

2024-06-10



Set up a cluster for high availability.

<https://l8n.ch/RyX7W>.

[Dernière consultation le 09.06.2024].



Size your shards.

<https://l8n.ch/cxRJm>.


[Dernière consultation le 09.06.2024].




C. Gormley and Z. Tong.

Elasticsearch : The Definitive Guide.

O'Reilly Media, Inc., 1st edition, 2015.

 M. Kleppmann.
Designing Data-Intensive Applications : The Big Ideas Behind Reliable, Scalable, and Maintainable Systems.
O'Reilly Media, Inc., 1st edition, 2017.

 M. Konda.
Elasticsearch in Action, Second Edition.
In Action. Manning, 2nd edition, 2023.

2024-06-10

“You allow things to be inconsistent and then you find ways
to compensate for mistakes, versus trying to prevent
mistakes altogether.”
— Eric Brewer

2024-06-10

Étude de cas Distribution des données avec Elasticsearch

└ Bibliographie

“You allow things to be inconsistent and then you find ways
to compensate for mistakes, versus trying to prevent
mistakes altogether.”
— Eric Brewer