Table des matières

Documentation GameForge

1. Contexte du projet

Le jeu vidéo GameForge qui va être conçu a pour but de réunir des milliers de joueurs dans un environnement où chacun pourra interagir en temps réel. Ce contexte pose une problématique complexe qui est : comment concevoir une infrastructure de bases de données performante tout en étant capable de traiter des volumes élevés de données en temps réel ; mais aussi, comment gérer une potentielle croissance continue de notre jeu.

L'architecture a pour objectif de :

1. Gérer efficacement les profils mais aussi les inventaires des joueurs.

2. Avoir la possibilité d'assurer les interactions en temps réel entre les joueurs (les combats, les déplacements des différents joueurs…).

3. Calculer les classements ainsi que les statistiques des joueurs de façon dynamique afin de suivre les performances de tous les challenger de notre jeu.

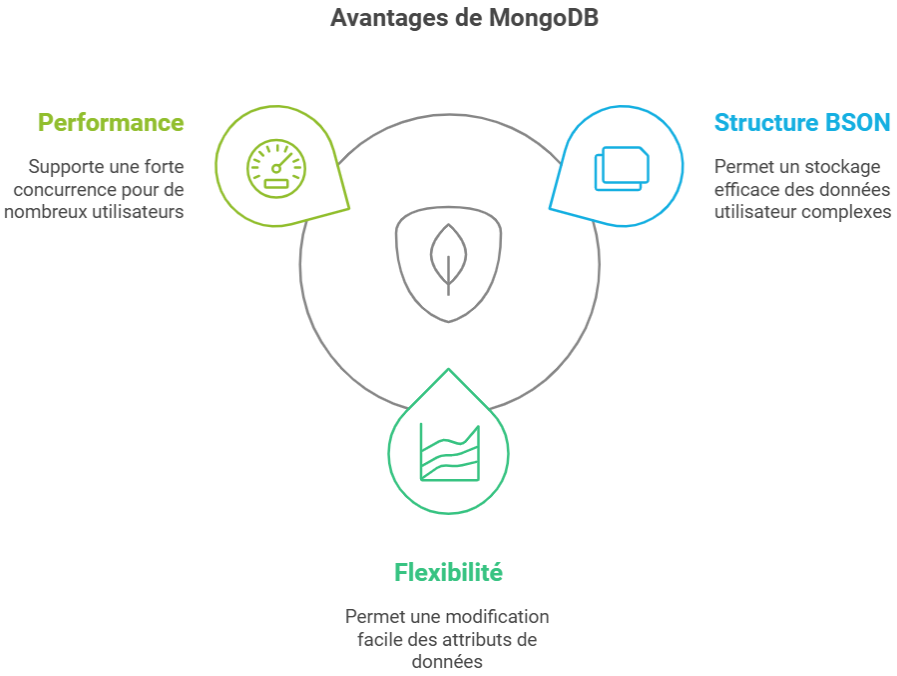
2. Choix des technologies

Les objectifs décrits précédemment font face à des problématiques technologiques fortes. Le but de cette partie sera de présenter différentes technologies et pourquoi nous avons choisi une solution plutôt qu'une autre pour chacun de nos objectifs pour notre jeu vidéo.

- La gestion des profils et l'inventaire des joueurs

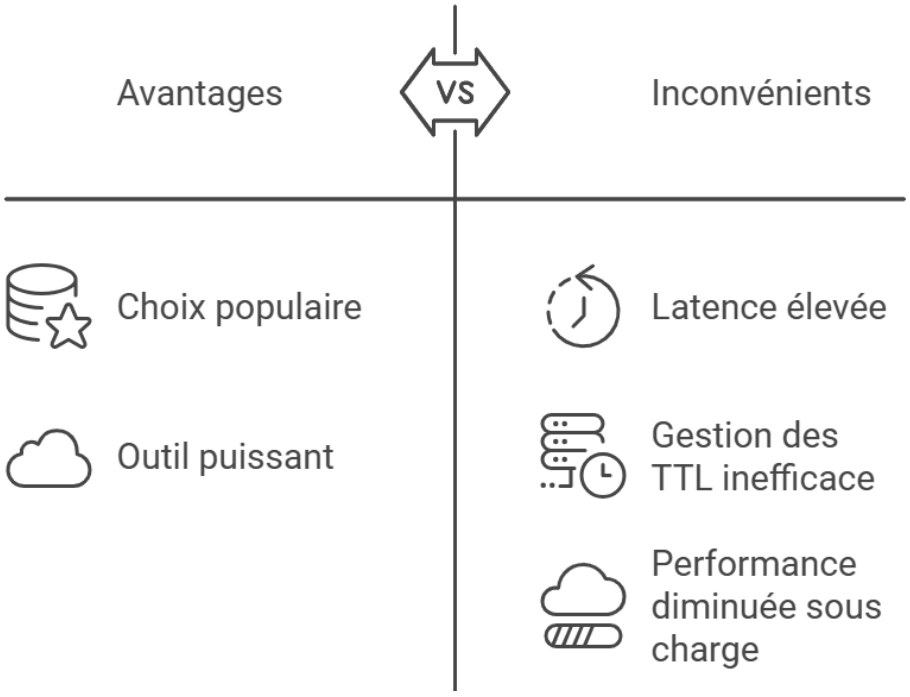
Nous avons donc choisi d'utiliser la solution technique MongoDB afin de gérer les inventaires des joueurs puisqu'il est important d'avoir une structure en document BSON puisqu'il faut stocker plusieurs données pour un seul utilisateur. Le système MongoDB permet de gérer cette architecture.
De plus, le stockage en document est flexible et permet d'ajouter ou de supprimer facilement du contenu dans les attributs compétences et inventaire. En revanche, pour que les performances soient suffisantes pour un grand nombre de joueurs connectés en simultané

(ici jusqu'à 40000), il faut disposer d'une architecture plus complexe, grâce à des clusters plus importants et une répartition des charges.
Le choix n'était pas difficile puisque MongoDB est le seul à proposer l'architecture nécessaire à la gestion des inventaires.



- **<u>Les interactions en temps réel pour les joueurs</u>**

Dans GameForge, nos joueurs pourront interagir en temps réel. De nombreuses fonctionnalités sont disponibles, notamment : les différentes actions dans les combats, les échanges d'objets entre les utilisateurs et les déplacements de ces derniers. Afin de pouvoir gérer ces interactions de manière fluide et rapide. Le choix ne se porte pas de nouveau sur la technologie MongoDB. Pourquoi cela ?



- **Latence élevée**

MongoDB stocke directement ses données sur disque, cela entraîne une latence plus élevée. Sachant que les actions de combats et déplacements génèrent un nombre important d'action, cela pourrait provoquer des soucis au niveau de la latence pour les joueurs.

- **Gestion des TTL inefficace**

Oui, MongoDB propose une expiration des données. Cependant, cette fonctionnalité n'est pas optimisée pour des suppressions rapides et fréquentes. Ce qui pourrait rapidement encombrer notre base de données si il y a un enregistrement à chaque changement de position d'un joueur.

- **Performance diminuée sous charge**

Si il y a un ou des pics de données massives, MongoDB peut ralentir et par conséquent, les données de traitement des interactions des joueurs pourraient provoquer des lags ou des réactions tardives. Par exemple, un joueur pourrait être contraint d'effectuer une action (ex : attaque spéciale) avec un retard, ce qui pourrait être frustrant pour un joueur.

Par conséquent, il a été décidé de partir sur la solution de **Redis** pour la gestion des interactions en temps réel en raison de ses caractéristiques qui répondent aux exigences mises en place. Redis est conçu pour la haute performance ainsi qu'une faible latence. Cela est crucial pour les fonctionnalités mises en place comme les combats en temps réel, les échanges d'objets et les déplacements des joueurs.

Pourquoi REDIS ?

Les points vus précédemment vont être remis sur la table afin de voir où REDIS va pouvoir nous aider et proposer une solution plus optimales pour la problématique sur les données en temps réel.

- **Latence minimale**

MongoDB stocke les données sur disque, ce qui peut agir négativement sur la latence, là où, Redis fonctionne principalement en mémoire, ce qui permet une latence plus faible. Cette méthode est nécessaire pour les actions instantanées qu'un joueur peut effectuer en jeu. Au vu du nombre d'actions et d'événements qui peuvent se produire en jeu, cela est primordial.

- **Gestion efficace des Time-To-Live**

Redis gère nativement les TTL (Time-To-Live) pour les données éphémères. Pour notre jeu, les actions pourront être supprimées automatiquement avec une latence moins élevée. Redis gère les suppressions en mémoire sans pour autant encombrer la base de données, contrairement à MongoDB.

- **Capacité à gérer des charges élevées**

Redis est plus optimisé pour supporter des volumes de requêtes massifs plus importants. Sachant que les pics de charge de notre jeu peuvent atteindre 175 000 interactions par secondes, Redis va permettre d'avoir une réactivité nécessaire sous lourdes charges. Cela va permettre d'éviter les ralentissements ou les crashs potentiels.

Statistiques et classements des joueurs

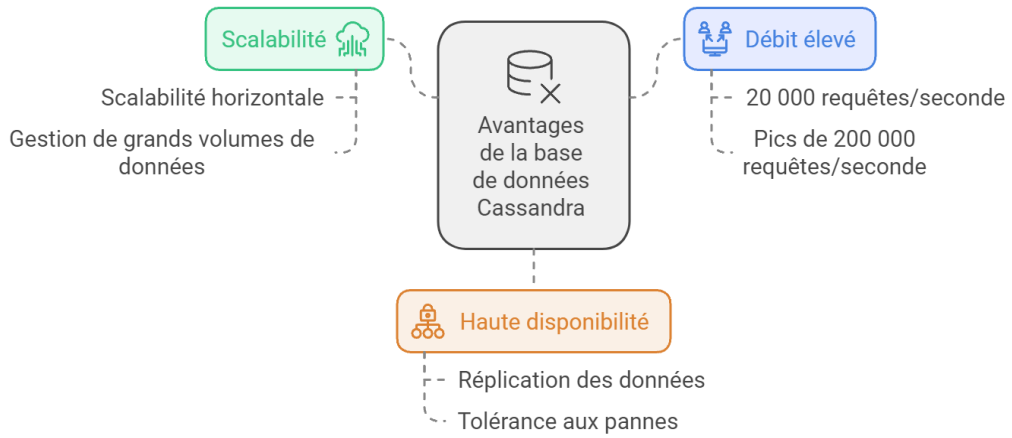
Pour une gestion efficace des statistiques et du classement des joueurs, nous avons choisi d'implémenter une base de données NOSQL orientée colonne: **Cassandra**.

Cassandra propose un certain nombre d'avantages:

- **Scalabilité horizontale** :
  Cassandra est conçu pour se répartir sur de nombreux serveurs, c'est-à-dire que l'on peut répartir les différentes colonnes sur différents serveurs. Ce qui le rend idéal pour gérer les très grandes quantités d'enregistrements de statistiques prévus et supporter les pics de charge prévisionnel.

- **Modèle adapté aux requêtes à fort débit** :
  L'application doit traiter environ 20 000 requêtes par seconde en moyenne, avec des pointes allant jusqu'à 200 000 requêtes par seconde. Cassandra est optimisé pour des opérations d'écriture et de lecture rapides, ce qui le rend adapté pour le stockage et la récupération des statistiques de joueur en temps réel. Son modèle de données en colonnes permet de structurer les informations des joueurs afin de faciliter des requêtes rapides sur des périodes de temps pour obtenir un classement.

- **Haute disponibilité** :
  Il offre une haute tolérance aux pannes grâce à la réplication de données. C'est une composante essentielle pour garantir que le système soit disponible tout le temps. En particulier, lors des événements de mise à jour de classements.



Nous avons aussi envisagé REDIS pour établir un classement global car celui-ci peut gérer un très grand volume de données par seconde ce qui permet d'avoir un classement

constant en temps réel malgré tout celui-ci est peu pratique pour des requêtes complexes sur de l'agrégation de données. Par exemple: un classement de nombre d’exp sur une période spécifique ou un classement du nombre d'attaques.

Schéma