**CALL OF DUTY**

**STACK TECHNIQUE**

* Langage de programmation : C++

**ARCHITECTURE**

**MICRO SERVICES**

Nous avons opté pour l’architecture en micro services pour les raisons suivantes :

* Résilience / disponibilité : Afin de découpler les différents modules et ainsi éviter une panne généralisée lorsqu’un seul module en erreur. Seul le service en erreur sera ainsi indisponible laissant le reste de l’application en bon fonctionnement.  
  Par exemple, il serait problématique que les parties en ligne cessent de fonctionner à cause d’une exception non gérée sur le service de la boutique.
* Organisation / Maintenabilité : Avoir un micro-service par grande fonctionnalité permet de bien différencier et séparer les responsabilités de l’application, il est donc plus simple pour le développeur de s’occuper d’une partie précise de l’application car le périmètre est déjà bien défini.
* Scalabilité : Si un micro-service en particulier est surchargé par les requêtes clients, comme le service de paiement, il sera plus efficace de simplement dupliquer le micro service de paiement sur d’autres serveur et que sur une architecture monolithique.
* Rapidité de déploiement : Lors d’une opération de maintenance ou du développement d’une fonctionnalité, la tâche ne sera à effectuer que sur un module, il sera donc seulement nécessaire de déployer le micro-service en question, les autres n’ayant pas à changer d’état.  
  On ne va pas non plus exécuter tous les tests de l’application mais juste ceux du service en question.

**LOAD BALANCER**

Nous allons mettre en place un load balancer afin de rediriger chaque requête client vers le serveur le plus apte à la réceptionner afin d’éviter une surcharge serveur.

**MESSAGE BROKER**

Pour toutes actions ne nécessitant pas de communication en temps réel, un courtier de message sera utilisé pour éviter la surcharge des serveurs. Par exemple, lors d’un crash jeu, l’utilisateur a la possibilité d’envoyer un rapport d’incident à l’équipe de développement, si de nombreux crash surviennent en même temps, les rapports seront consommés par les serveurs au rythme qui leur convient.

Périmètre : actions ne nécessitant pas de communication en temps réel

Exemples : Rapport de crash,

Pourquoi : éviter les surcharges de serveur lors d’un envoi simultané d’un grand nombre de requête

**CLUSTER DATABASE**

**FRISE CHRONOLOGIQUE DES ACTIONS DE LA CONCEPTION**