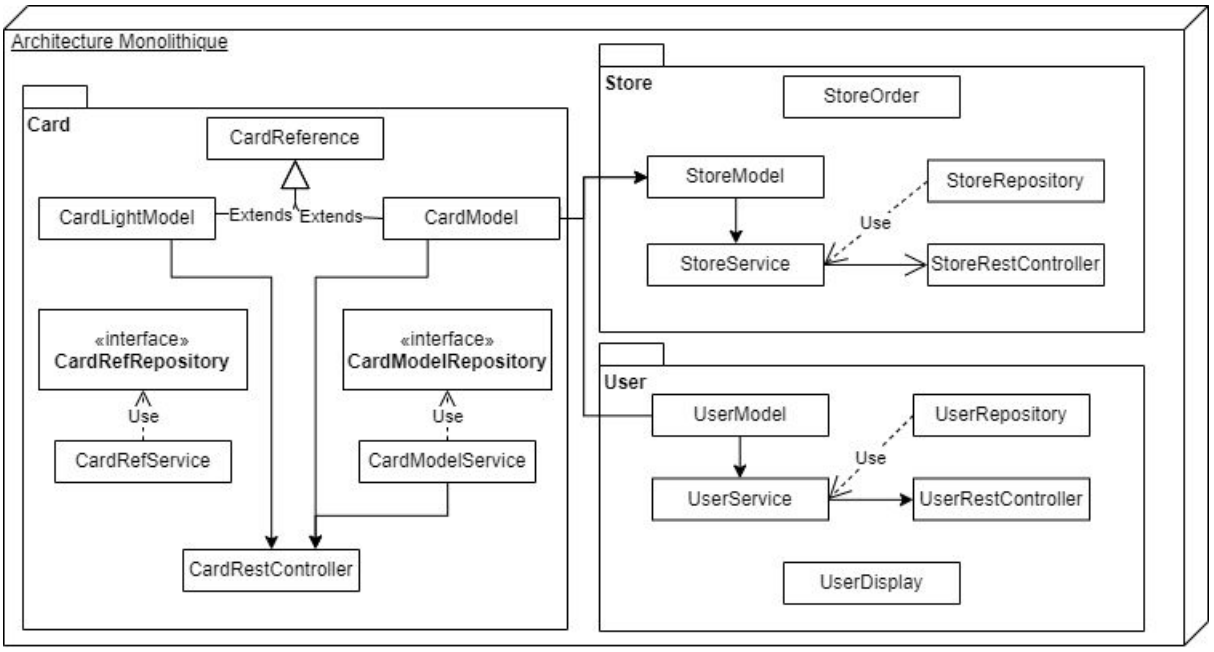

Comparatif des bus de communication

Introduction

Un **message broker** sur le principe permet de gérer des files d'attente de messages. Il est possible ainsi de publier un message qui sera donc en attente d'être consommé. Pour pouvoir gérer des files d'attente, il faut un **serveur de message**. Un message broker est donc une solution qui s'accompagne de concepts qui en font un projet assez conséquent à concevoir rigoureusement. Mais celle-ci reste très intéressante pour améliorer les performances et l'évolution à grande échelle de vos applications ou de votre SI.

Critères

Le premier critère étudié est la **documentation** pour comparer le niveau d'informations disponibles sur chaque bus de communication. Le symbole **+** signifie que la documentation est riche et le symbole **–** signifie l'inverse.

Le second critère est le **langage** de développement de ces solutions.

Le troisième critère est les **protocoles** implémentés par chacun des éléments étudiés.

Le quatrième critère indique le lieu de **stockage** des données

Le cinquième critère est la **transaction de messages** sécurisée et fiable.

Le sixième critère est **l'équilibrage des charges** qui rend l'utilisation et le déploiement simple.

Le septième critère est la possibilité d'avoir un **cluster** pour augmenter la performance des solutions.

Le huitième critère est la qualité de **l'interface de gestion**.

Le neuvième critère est la **haute disponibilité** qui est également un critère de performance.

Le dixième critère est le type de **distribution** des messages.

Tableau comparatif

	RabbiMQ	ZeroMQ	ActiveMQ
Documentation	+	-	+
Langage	Erlang	C++	Java
Protocoles	AMQP, STOMP	TCP	AMQP AUTO MQTT OpenWire REST RSS and Atom Stomp ...
Stockage	Mémoire, disque	Mémoire	Mémoire, disque, BD
Transaction de message	Prise en charge	Non prise en charge	Prise en charge
Équilibrage des charges	Prise en charge	Non prise en charge	Prise en charge
Cluster	Clustering simple	Non prise en charge	Prend en charge les modèles de cluster simple

Interface gestion	Bon	Aucun	Général
Disponibilités	Elevé	Elevé	Elevé
Distribution	Direct, thématique, en-têtes	Peer to peer	Peer to peer

Résumé

RabbitMQ est l'une des principales implémentations du protocole AMQP (avec Apache Qpid). Par conséquent, il implémente une architecture de courtier, ce qui signifie que les messages sont mis en file d'attente sur un nœud central avant d'être envoyés aux clients. Cette approche rend RabbitMQ très facile à utiliser et à déployer, car des scénarios avancés tels que le routage, l'équilibrage de charge ou la file d'attente de messages persistante sont pris en charge en quelques lignes de code. Cependant, cela le rend également moins évolutif et «plus lent» car le nœud central ajoute de la latence et les enveloppes de messages sont assez grandes.

ZeroMQ est un système de messagerie très léger spécialement conçu pour les scénarios à haut débit / faible latence comme celui que vous pouvez trouver dans le monde financier. Zmq prend en charge de nombreux scénarios de messagerie avancés, mais contrairement à RabbitMQ, vous devrez implémenter la plupart d'entre eux vous-même en combinant divers éléments du cadre (par exemple: les sockets et les périphériques). Zmq est très flexible mais vous devrez étudier les 80 pages environ du guide (que je recommande de lire pour quiconque écrit un système distribué, même si vous n'utilisez pas Zmq) avant de pouvoir faire quelque chose de plus compliqué que d'envoyer des messages entre 2 pairs.

ActiveMQ est au milieu. Il peut être déployé avec les topologies de courtier et P2P. Comme RabbitMQ, il est plus facile d'implémenter des scénarios avancés, mais généralement au détriment des performances brutes. C'est le couteau suisse de la messagerie

Comparatifs des frameworks Front End JavaScript

	React	**Angular**	**Vue**
Type	Librairie	Framework	Librairie
Popularité	+++	++	+
Difficulté d'apprentissage	+++	+++	+

Avantages	• JSX • DOM virtuel • Testabilité • Server-Side Rendering (SSR) • Data binding uni-direction nel	• Data binding bi-directionn el • Manipulation de DOM	• Taille des fichiers très petite • Implémentati on facile à mettre en place • Intégration simple • Documentati on complète
Inconvénients	• Faible documentati on • Apprentissag e nécessaire du JSX • Utilisation de librairie type Redux pour du MVC	• Communaut é divisée • Mises à jours fréquentes • Temps de chargement initial parfois long	• Petite communauté • Trop de flexibilité