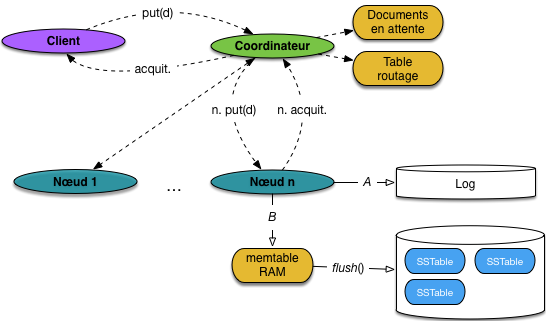
**Cassandra, une base relationnelle étendue**

Cassandra est un système de gestion de données à grande échelle. C’est un système NoSQL très utilisé, et sans doute un bon point de départ pour passer du relationnel à un système distribué.



Un cluster Cassandra fonctionne en mode multi-nœuds. La notion de nœud maître et nœud esclave n’existe donc pas. Chaque nœud du cluster a le même rôle et la même importance, et jouit donc de la capacité de lecture et d’écriture dans le cluster. Un nœud ne sera donc jamais préféré à un autre pour être interrogé par le client.

Pour que ce système fonctionne, chaque nœud du cluster a la connaissance de la topologie de l’anneau. Chaque nœud sait donc où sont les autres nœuds, quels sont leurs identifiants, quels nœuds sont disponibles et lesquels ne le sont pas.

Un client qui interroge Cassandra contacte un nœud au hasard parmi tous les nœuds du cluster. Le partitionnement implique que tous les nœuds ne possèdent pas localement l’information recherchée. Cependant, tous les nœuds sont capables de dire quel est le nœud du cluster qui possède la ressource recherchée. Dans ce cas donc, ce nœud que l’on appellera le coordinateur va rediriger la requête au bon nœud, c’est-à-dire celui qui est capable de la traiter.

Installation

Avec Docker, il vous sera possible d’utiliser Cassandra dans un environnement distribué. C’est de loin le mode d’installation le plus simple, il est rapide et ne pollue pas la machine avec des services qui tournent en tâche de fond et dont on ne se sert pas.

Créons maintenant un cluster Cassandra, avec 4 nœuds. Pour cela, nous créons un premier nœud qui nous servira de point d’accès (seed dans la terminologie Cassandra) pour en ajouter d’autres.

Sur fiteccabd1-ls-4

docker run --name cass1 --net=host -d -e CASSANDRA\_BROADCAST\_ADDRESS=10.0.2.31 cassandra:latest

Sur fiteccabd1-ls-1

docker run --name cass2 --net=host -d -e CASSANDRA\_BROADCAST\_ADDRESS=10.0.2.25 -e CASSANDRA\_SEEDS=10.0.2.31 cassandra:latest

Sur fiteccabd1-ls-3

docker run --name cass3 --net=host -d -e CASSANDRA\_BROADCAST\_ADDRESS=10.0.2.39 -e CASSANDRA\_SEEDS=10.0.2.31 cassandra:latest

Sur fiteccabd1-ls-19

docker run --name cass4 --net=host -d -e CASSANDRA\_BROADCAST\_ADDRESS=10.0.2.38 -e CASSANDRA\_SEEDS=10.0.2.31 cassandra:latest

Nous venons de créer un cluster de 4 nœuds Cassandra, qui tournent tous en tâche de fond grâce à Docker. Vous pouvez tester la bonne configuration du cluster en tapant la commande suivante :

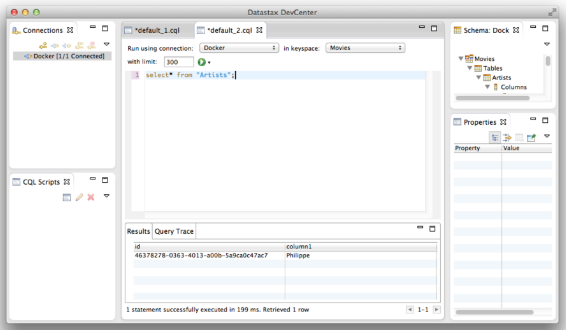
docker exec -it cass1 sh -c 'nodetool status'

Exécuter vos commandes CQL avec la commande suivante :

docker exec -ti cass1 cqlsh localhost

Le client

Des clients graphiques existent. Le plus complet semble le Datastax DevCenter. Vous pouvez le télécharger à l’adresse suivante : <https://downloads.datastax.com/devcenter/DevCenter-1.6.0-win-x86_64.zip>



N.B : Connectez-vous sur le port **9042**.

Le modèle de données

Cassandra est un système est un modèle relationnel étendu, avec typage fort et schéma contraint. Une colonne peut avoir différents types :

* des types atomiques, comme par exemple entier, texte, date;
* des types complexes (ensembles, listes, dictionnaires);
* des types construits et nommés.

Nous sommes effectivement proche d’un modèle de documents structurés de type JSON, avec imbrication de structures, mais avec un schéma qui assure le contrôle des données insérées. La gestion de la base est donc très contrainte et doit se faire en cohérence avec la structure de chaque table (son schéma). C’est une différence notable avec de nombreux systèmes NoSQL.

La conception d’un schéma Cassandra est nécessairement différente de celle d’un schéma relationnel à cause de l’absence du système de clé étrangère et de l’opération de jointure. Cassandra oblige à réfléchir en priorité à la façon dont le modèle de données va être utilisé. Quelles requêtes vont être exécutées ? Dans quel sens mes données seront-elles traitées ? C’est à partir de ces questions que pourra s’élaborer un modèle optimisé, dénormalisé et donc performant. L’inconvénient d’une démarche basée sur les besoins est que si ces derniers évoluent (ou si une application différente veut accéder à une base existante), l’organisation de la base devient inadaptée.

En résumé :

* Cassandra permet de stocker des tables dénormalisées dans lesquelles les valeurs ne sont pas nécessairement atomiques ; il s’appuie sur une plus grande diversité de types (pas uniquement des entiers et des chaînes de caractères, mais des types construits comme les listes ou les dictionnaires).
* La modélisation d’une architecture de données dans Cassandra est beaucoup plus ouverte qu’en relationnel ce qui rend notamment la modélisation plus difficile à évaluer, surtout à long terme.
* La dénormalisation (souvent considérée comme la bête noire à pourchasser dans un modèle relationnel) devient recommandée avec Cassandra, en restant conscient que ses inconvénients (notamment la duplication de l’information, et les incohérences possibles) doivent être envisagés sérieusement.
* En contrepartie des difficultés accrues de la modélisation, et surtout de l’impossibilité de garantir formellement la qualité d’un schéma grâce à des méthodes adaptées, Cassandra assure un passage à l’échelle par distribution basé sur des techniques de partitionnement et de réplication. C’est un système qui offre des performances jugées très satisfaisantes dans un environnement Big Data.

Rappelons que keyspace est le nom que Cassandra donne à une base de données. Cassandra est fait pour fonctionner dans un environnement distribué. Pour créer un keyspace, il faut donc préciser la stratégie de réplication à adopter. Voici la commande :

CREATE KEYSPACE IF NOT EXISTS velib\_db

WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication\_factor': 1};

Sous csqlsh, vous pouvez maintenant sélectionner la base de données pour vos prochaines requêtes et créer la structure de notre base de données :

USE velib\_db;

create table station (number int,

contract\_name text,

name text,

address text,

position map<text,float>,

banking boolean,

bonus boolean,

status text,

bike\_stands int,

available\_bike\_stands int,

available\_bikes int,

last\_update timestamp,

datetime timestamp,

primary key (number, last\_update) );

create table station\_static (number int,

contract\_name text,

name text,

address text,

position map<text,float>,

banking boolean,

bonus boolean,

last\_update timestamp,

primary key (number) );

create table station\_dynamic (number int,

status text,

bike\_stands int,

available\_bike\_stands int,

available\_bikes int,

last\_update timestamp,

datetime timestamp,

primary key (number, last\_update));

Pour créer notre base de données, vous pouvez plus simplement exécuter le fichier create\_db.cql directement dans un docker ici dans cass2 sur fiteccabd1-ls-1 :

docker cp create\_db.cql cass2:/tmp/

docker exec -it cass2 cqlsh -f /tmp/create\_bd.cql