REDIS

Tondeur Hervé (2017)

Définition

Redis, est un moteur de base de données en mémoire.

Redis (REmote Dictionary Server) est un moteur de base de données en mémoire développé à partir de 2009 par un développeur italien nommé Salvatore Sanfilippo. Ce dernier ainsi qu'un contributeur important de Redis (Pieter Noordhuis) sont recrutés par VMWare en 2010 pour se consacrer à plein temps sur le développement de Redis tout en le laissant en licence libre BSD.
Redis est écrit avec le langage de programmation C ANSI. Redis fait partie des solutions NoSQL et il se distingue par sa rapidité, son efficacité et sa légèreté. Par exemple, en terme de rapidité il est très difficile à battre, tant en lecture qu'en écriture.
Il peut traiter plus de 100 000 opérations par seconde.

Présentation

Redis est un moteur non relationnel qui opère principalement en mémoire. Il manipule **des structures de données clé-valeur**.

Il est à la fois gestionnaire de ces types de données et un cache de données à la manière de memcached (un système d'usage général servant à gérer la mémoire cache distribuée.

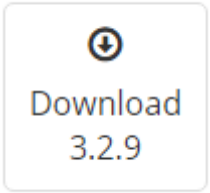
Il est souvent utilisé pour augmenter la vitesse de réponse des sites web (Facebook, Twitter, Wikipedia) créés à partir de bases de données.

Il gère les données et les objets en RAM de façon à réduire le nombre de fois qu'une même donnée stockée dans un périphérique externe est lue.

- Stockage de paires clé-valeur en mémoire (In-memory key-value store)
- Réplication Maitre/Esclave
- Persistance
- RDB (snapshots) à intervalles régulier des données.
- AOF (append-only log file) A chaque écriture dans le serveur (plus long)
- Supporte les langages
- Ruby, Python, Java, C, PHP, Javascript (Node.js)
- Mise à disposition de 16 bases de données au lancement du serveur.

Installation Linux

https://redis.io/download




```
$ wget http://download.redis.io/releases/redis-3.2.9.tar.gz
$ tar xzf redis-3.2.9.tar.gz
$ cd redis-3.2.9
$ make
```

Redis est disponible sous forme de paquet dans plusieurs distributions : Debian, Ubuntu, etc. Sous Ubuntu ou Debian, le paquet est nommé : redis-server. Pour l'installer, il faut tout simplement lancer la commande :

```
#$> sudo aptitude install redis-server
```

Installation Windows(packages non officiels)

https://github.com/MSOpenTech/redis/releases

https://www.nuget.org/packages/Redis-64/

C:>Install-Package Redis-64 -Version 3.0.503

https://chocolatey.org/packages/redis-64

C:>choco install redis-64

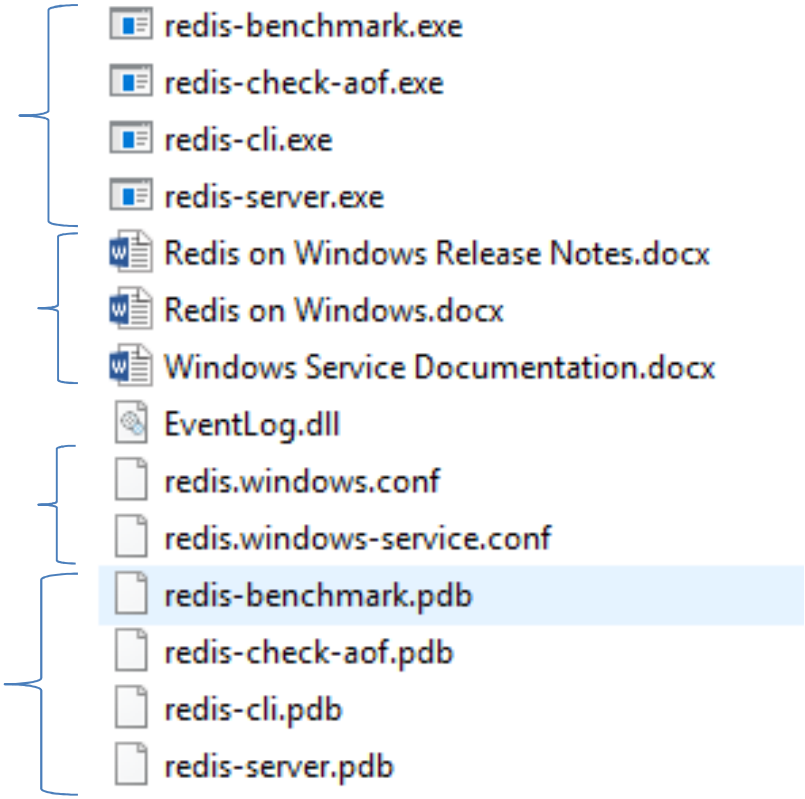
Installation Windows(packages non officiels)

Exécutables pour gérer REDIS

Un peu de DOC

Les fichiers de config pour REDIS

Les fichiers de Debug PDB C++



Lancer le serveur REDIS

Lancer simplement la commande redis-server.exe dans une console PowerShell sous Windows ou une console Linux sous Linux.
Redis travaille par défaut sur le port 6379

Lancer le serveur REDIS

Si nous souhaitons personnaliser les paramètres de démarrage, alors nous avons deux façons :

```
#$> nohup redis-server /etc/redis/redis.conf &
```

Ou

```
#$> nohup redis-server --port 5555 --slaveof 127.0.0.1 8888 &
```

Dans la première commande les valeurs des paramètres sont renseignées dans le fichier de configuration redis.conf.

Cependant dans la deuxième commande, les paramètres sont passés dans la ligne de commande.

Tester le serveur REDIS sur votre machine

REDIS est proposé avec l'application redis-benchmark, qui permet de tester votre installation et le serveur REDIS

```
PS C:\Users\maj\Downloads\Redis-x64-3.2.100> .\redis-benchmark.exe
====== PING_INLINE ======
  100000 requests completed in 0.73 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

99.93% <= 1 milliseconds
100.00% <= 2 milliseconds
100.00% <= 2 milliseconds
137362.64 requests per second

====== PING_BULK ======
  100000 requests completed in 0.62 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

99.95% <= 1 milliseconds
100.00% <= 1 milliseconds
160771.70 requests per second

====== SET ======
  100000 requests completed in 0.61 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

99.83% <= 1 milliseconds
99.90% <= 2 milliseconds
99.92% <= 3 milliseconds
99.95% <= 5 milliseconds
100.00% <= 5 milliseconds
163398.70 requests per second

====== GET ======
  100000 requests completed in 0.61 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

99.94% <= 1 milliseconds
100.00% <= 1 milliseconds
162866.44 requests per second
```

La commande « redis-cli.exe ping », doit répondre PONG si le serveur est UP

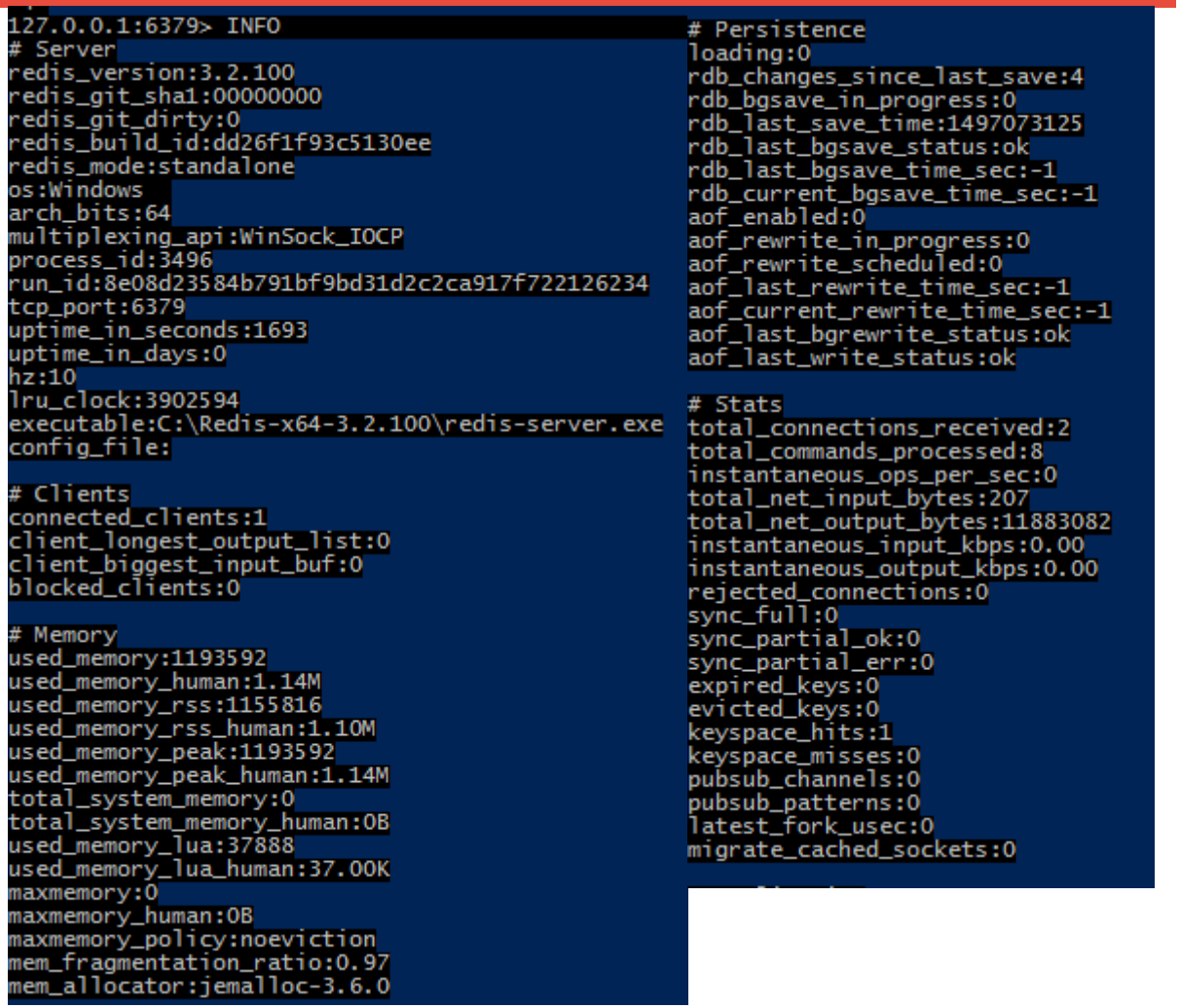
```
PS C:\Redis-x64-3.2.100> .\redis-cli.exe ping
PONG
PS C:\Redis-x64-3.2.100>
```

Paramètre du serveur REDIS

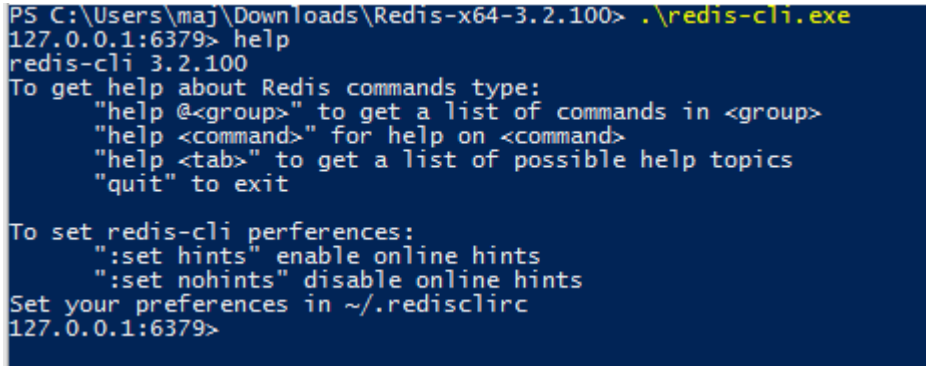
Avant de personnaliser les paramètres de serveur Redis, il est utile de jeter un coup d'oeil sur la commande : INFO Pour lancer cette commande, il faut passer par redis-cli.

Ce dernier est livré avec Redis-server.

Redis-cli est une invite interactive qui permet d'envoyer des commandes au serveur.



Accéder à la console REDIS



Il est possible de travailler avec REDIS en mode console
Il faut lancer cette console via la commande « redis-cli.exe »

La première commande à découvrir est la commande « HELP »

Help + espace + tab n fois pour obtenir la liste des commandes possibles…

Help @group (cluster, connection, geo, hashes, hyperloglog, keys, lists, pub/sub, Scripting, server, sets, sorted sets, strings, transactions)

Commandes REDIS

- AUTH password Authentication vers le serveur (Si active)
- ECHO message Echo le message données si le serveur est UP
- PING [message] Ping le serveur
- QUIT Fermer la connexion
- SELECT index Modifier la base de données courante (1 à 16)
- SWAPDB index index Echanger deux base de données Redis

Les types de données REDIS

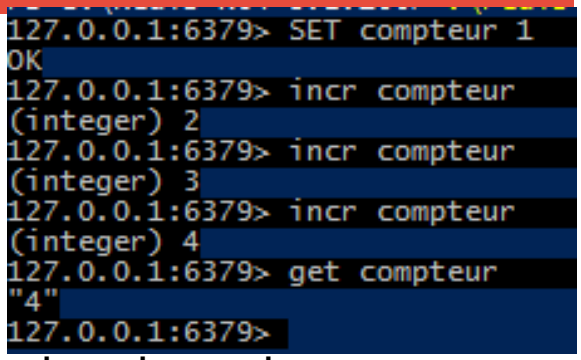
Contrairement aux autres solutions NoSQL qui opèrent sur des documents (json, xml, etc),

Redis manipule des paires clé-valeur, les valeurs peuvent être de cinq types de données : chaines, listes, ensembles, hachages et ensembles triés.

Ci-dessous, nous détaillons ces cinq types de données.

Strings Redis

Le terme "String" porte à confusion dans la mesure où dans Redis, un objet de type String désigne aussi bien une chaîne de caractères, un entier ou encore une image jpeg



Chaînes

Une chaîne en Redis est plus qu'une chaîne standard.

Elle permet de stocker une chaine de caractère, mais aussi de stocker des valeurs numériques et des valeurs binaires. Ces dernières permettent par exemple de stocker une image en mémoire.

Ceci afin d'éviter le codage et décodage de format image.

La taille maximale est de 512 Mo.

Les commandes SET et GET permettent de manipuler les Strings.

D'autres commandes sont intégrées spécialement pour modifier des chaînes de type numérique INCR, INCRBY, DECR, DECRBY.

Nous verrons toutes ces commandes dans les sections de travaux pratiques.

Les commandes de gestion des Strings

- APPEND key value *Append a value to a key*
- BITCOUNT key [start end] *Count set bits in a string*
- BITFIELD key [GET type offset] [SET type offset value] [INCRBY type offset increment] [OVERFLOW WRAP|SAT|FAIL] *Perform arbitrary bitfield integer operations on strings*
- BITOP operation destkey key [key ...] Perform bitwise operations between strings
- BITPOS key bit [start] [end] Find first bit set or clear in a string
- DECR key Decrement the integer value of a key by one
- DECRBY key decrement Decrement the integer value of a key by the given number
- GET key Get the value of a key
- GETBIT key offset Returns the bit value at offset in the string value stored at key
- GETRANGE key start end Get a substring of the string stored at a key
- GETSET key value Set the string value of a key and return its old value
- INCR key Increment the integer value of a key by one
- INCRBY key increment Increment the integer value of a key by the given amount
- INCRBYFLOAT key increment Increment the float value of a key by the given amount
- MGET key [key ...] Get the values of all the given keys
- MSET key value [key value ...] Set multiple keys to multiple values
- MSETNX key value [key value ...] Set multiple keys to multiple values, only if none of the keys exist
- PSETEX key milliseconds value Set the value and expiration in milliseconds of a key
- SET key value [EX seconds] [PX milliseconds] [NX|XX] Set the string value of a key
- SETBIT key offset value Sets or clears the bit at offset in the string value stored at key
- SETEX key seconds value Set the value and expiration of a key
- SETNX key value Set the value of a key, only if the key does not exist
- SETRANGE key offset value Overwrite part of a string at key starting at the specified offset
- STRLEN key Get the length of the value stored in a key

Les “Lists” REDIS

Listes

Les listes de Redis sont des listes liées avec pour conséquence que même si un grand nombre d’enregistrements apparaissent dans une liste, le coût d’insertion d’un élément en tête ou en queue de liste reste très faible. La contrepartie de cette vitesse est que l’accès à un élément dans la liste doit se faire par son index. Les utilisateurs qui ont l'habitude avec **ArrayList** ou **LinkedList** en Java connaissent déjà cette différence.

Les principales commandes (dont les noms commencent en général par **L** comme List) sont :

- **RPUSH** et **LPUSH** qui permettent respectivement d’ajouter un élément en fin ou en début de liste (et leurs fonctions inverses de type **xPOP**) ;
- **LRANGE** pour obtenir une partie des éléments de la liste ;
- **LINDEX** pour obtenir un seul élément de la liste ;
- **LLEN** pour obtenir la taille de la liste.

La commande **KEYS *** permet d’obtenir l'énumération de toutes les listes disponibles dans le système Redis.
La commande **DEL <nom de la liste>** permet de supprimer complètement une liste avec l’ensemble de son contenu.

Les “Lists” REDIS

```
127.0.0.1:6379> RPUSH msg "BONJOUR"
(integer) 1
127.0.0.1:6379> RPUSH msg "Les licences ISTV de l'UVHC"
(integer) 2
127.0.0.1:6379> RPUSH msg "BYE!"
(integer) 3
127.0.0.1:6379> LRANGE msg 0 2
1) "BONJOUR"
2) "Les licences ISTV de l'UVHC"
3) "BYE!"
127.0.0.1:6379> KEYS *
1) "compteur"
2) "counter:__rand_int__"
3) "key:__rand_int__"
4) "mylist"
5) "msg"
127.0.0.1:6379> LLEN msg
(integer) 3
127.0.0.1:6379>
```

Les “Hashes” REDIS

Hash

Un **hash** est une structure de données permettant de stocker dans un même enregistrement plusieurs couples clé/valeur.
Les commandes de hash commencent généralement par un **H** (comme Hash).
On trouve les commandes **HSET**, **HGET**, **HLEN**, mais aussi **HGETALL** pour obtenir tous les couples clé/valeur, **HINCRBY** pour incrémenter un compteur dans le hash, **HKEYS** et **HVALS** pour obtenir toutes les clés ou valeurs et **HDEL** "pour faire le ménage".

Les “Sets” REDIS

Set

Les **Sets** sont des collections d’objets non ordonnés. Les commandes qui commencent toutes avec un **S** (comme Set) sont les suivantes :

- **SADD** pour ajouter une valeur à un set ;
- **SCARD** pour obtenir la taille (cardinalité) d’un set ;
- **SINTER**, **SUNION** et **SDIFF** qui permettent respectivement d’obtenir l’intersection, l’union et la différence entre deux sets.

Ces commandes existent en version "STORE". Par exemple, **INTERSTORE** permet de stocker dans un nouveau set l’intersection de deux autres.

Les “Sorted Sets” REDIS

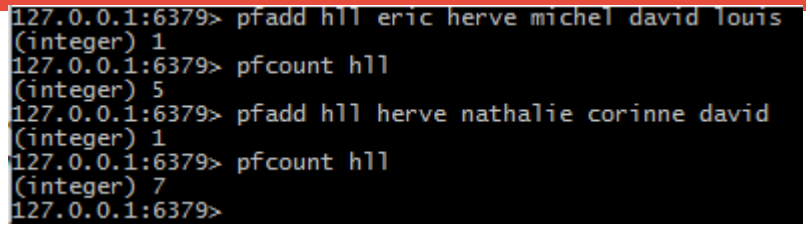
Sorted Set

"Sorted Set" pour définir une sorte de **Set** avec une notion de "score" associé aux valeurs ajoutées, ce qui permet de faire des tris. Les commandes commencent toutes par Z comme "Zorted" Set.

On trouve donc les équivalents des commandes précédentes, à savoir **ZADD**, **ZCARD**, **ZINTER**, **ZUNION**, **ZDIFF** mais aussi **ZRANGE**, **ZRANGEBYSCORE** et **ZRANK**.

Les “HyperLogLogs” REDIS

Un HyperLogLog est une structure de données probabilistes utilisée pour compter des choses uniques.



En comptant généralement des éléments uniques, cela nécessitent l'utilisation d'une quantité de mémoire proportionnelle au nombre d'éléments que vous souhaitez compter.

Dans le cas de REDIS un algorithme est proposé, vous finissez par une mesure estimée avec une erreur standard qui est inférieure à 1%.

La magie de cet algorithme est que vous n'avez plus besoin d'utiliser une quantité de mémoire proportionnelle au nombre d'éléments comptés et que vous pouvez utiliser une quantité de mémoire constante! 12k octets dans le pire des cas, ou beaucoup moins si votre HyperLogLog (HLL) est très peu d'éléments.

Les “Geos” REDIS

La structure Geo permet de stocker et manipuler des coordonnées spatiale avec une technique appelée Geohash.

Les bits Latitude et Longitude sont entrelacés pour former un nombre entier de 52 bits unique.
Ce format permet une interrogation d’un rayon en vérifiant les zones nécessaires pour couvrir tout le rayon et éliminer les éléments en dehors du rayon.
Les zones sont vérifiées en calculant la portée de la boîte couverte, en supprimant suffisamment de morceaux de la partie moins significative du score défini et en calculant la plage de pointage pour interroger dans l'ensemble trié pour chaque zone.
Quel modèle utilise-t-il?
Il suppose simplement que la Terre est une sphère, puisque la formule à distance utilisée est la formule Haversine.
Cette formule n'est qu'une approximation lorsqu'elle est appliquée à la Terre, ce qui n'est pas une sphère parfaite.
Les erreurs introduites ne sont pas un problème lorsqu'il est utilisé dans le contexte de sites de réseaux sociaux qui doivent interroger par approximation et la plupart des autres applications.
Cependant, dans le pire des cas, l'erreur peut atteindre 0,5%, vous devrez envisager d'autres systèmes pour les applications critiques.

Les “Geos” REDIS

Les commandes prennent des arguments au format standard x, la longitude doit être spécifiée avant la latitude.

Il existe des limites aux coordonnées qui peuvent être indexées:
- les zones très proches des pôles ne sont pas indexables.
- Les limites exactes, telles que spécifiées par EPSG: 900913 / EPSG: 3785 / OSGEO: 41001 sont les suivantes:

- Les longitudes valides sont de -180 à 180 degrés.
- Les latitudes valides sont de -85.05112878 à 85.05112878 degrés.
- Les commandes signalerons des erreurs lorsque l'utilisateur tente d'indexer des coordonnées en dehors des plages spécifiées.

Ce qui n’est pas du DATA dans REDIS

Les sections précédentes ont montré comment faire des manipulations de base à partir d’un environnement de type ligne de commande. La richesse de Redis consiste à proposer en plus une API Pub/Sub qui permet de poster et de recevoir des messages sur des "channels" et d'autre part à offrir les possibilités suivantes :

- donner une durée de vie à une clé (avec les commandes **EXPIRE clé secondes** ou **EXPIREAT clé timestamp**), ce qui permet de laisser redis purger ses données sans avoir à écrire des scripts de purge ou de l’utiliser comme cache distribué ;
- passer une série de commandes dans une transaction (**MULTI** et **EXEC**) avec la possibilité de l’exécuter seulement si la clé change (avec **WATCH** à la place de **EXEC**)
- utiliser plusieurs bases de données (16 disponibles) avec **SELECT n** ;
- brancher un ou plusieurs "slaves" (tous sur le master ou les uns à la suite des autres, en cascade) : la réplication est extrêmement rapide.
- Scripter des commandes directement dans le serveur avec le langage Lua !

Les transactions dans REDIS

MULTI, **EXEC**, **DISCARD** et **WATCH** sont à la base des transactions dans Redis.

Ils permettent l'exécution d'un groupe de commandes en une seule étape, avec deux garanties importantes:

- Toutes les commandes d'une transaction sont sérialisées et exécutées séquentiellement. Il ne peut jamais arriver qu'une demande émise par un autre client soit diffusée au milieu de l'exécution d'une transaction Redis. Cela garantit que les commandes sont exécutées en une seule opération isolée.

- Soit toutes les commandes, soit aucune sont traitées, de sorte qu'une transaction Redis est également atomique.

La commande **EXEC** déclenche l'exécution de toutes les commandes de la transaction.

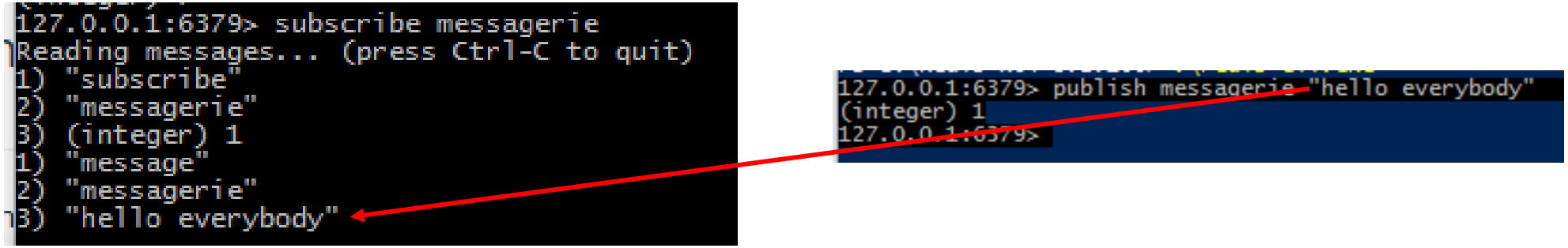
Pub/Sub sous REDIS

SUBSCRIBE, UNSUBSCRIBE et PUBLISH permet la mise en œuvre du paradigme de messagerie par publication/inscription où les expéditeurs (éditeurs) ne sont pas programmés pour envoyer leurs messages à des récepteurs spécifiques (abonnés).

Au lieu de cela, les messages publiés se caractérisent par des canaux, sans savoir quels sont (s'il en existe) les abonnés.

Les abonnés manifestent un intérêt pour un ou plusieurs canaux et ne reçoivent que des messages intéressants, sans connaître les éditeurs (le cas échéant).

Ce découplage des éditeurs et des abonnés peut permettre une plus grande évolutivité et une topologie de réseau plus dynamique (voir également les bus MOM AMQP ou JMS).



Pub/Sub sous REDIS

Database & Scoping

Pub / Sub n'a aucune relation avec l'espace clé. Il a été fait pour ne pas interférer quelque soit le niveau, y compris les nombres de base de données.

Une publication sur db 10, sera entendu par un abonné sur db 1.

Pattern-matching subscriptions

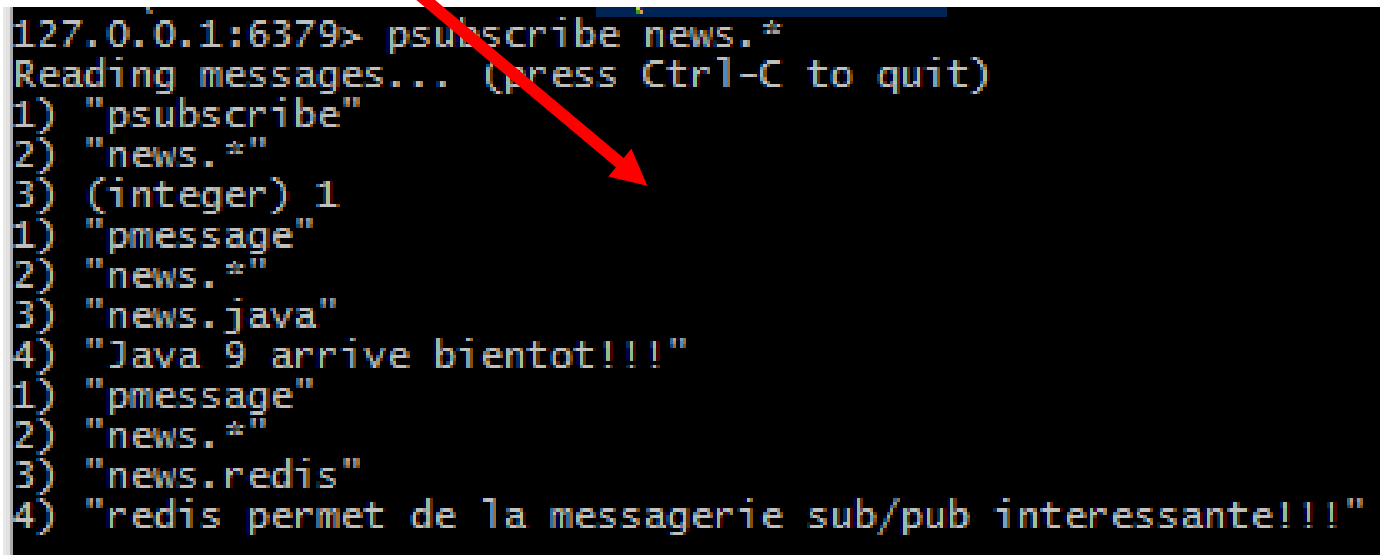
L'implémentation Redis Pub / Sub supporte la correspondance des modèles. Les clients peuvent s'abonner à des modèles de styles globaux afin de recevoir tous les messages envoyés aux noms de chaînes correspondant à un modèle donné.

Par exemple:
PSUBSCRIBE news. *

Recevra tous les messages envoyés à la chaîne news.java, news.lua, news.redis etc.
Tous les motifs «glob-style» sont valides, plusieurs caractères génériques sont pris en charge.

Pub/Sub sous REDIS

```
127.0.0.1:6379> publish news.java "Java 9 arrive bientot!!!"
(integer) 1
127.0.0.1:6379> publish news.redis "redis permet de la messagerie sub/pub interessante!!!"
(integer) 1
127.0.0.1:6379>
```



Pub/Sub sous REDIS

•PSUBSCRIBE pattern [pattern ...] Ecouter les messages publiés sur les « channels » qui répondent à certains Pattern.

•PUBSUB subcommand [argument [argument ...]] Inspecte l'état du sous système Pub/Sub.

•PUBLISH channel message Poster un message sur un Channel.

•PUNSUBSCRIBE [pattern [pattern ...]] Stopper l'écoute de messages postés sur les Channels qui match avec le pattern défini.

•SUBSCRIBE channel [channel ...] Ecouter les messages publiés sur certains Channels données.

•UNSUBSCRIBE [channel [channel ...]] Stopper l'écoute de messages postés sur certains Channels.

Développer en Java avec REDIS – Prérequis

Installer le jdk 8 dans sa dernière version

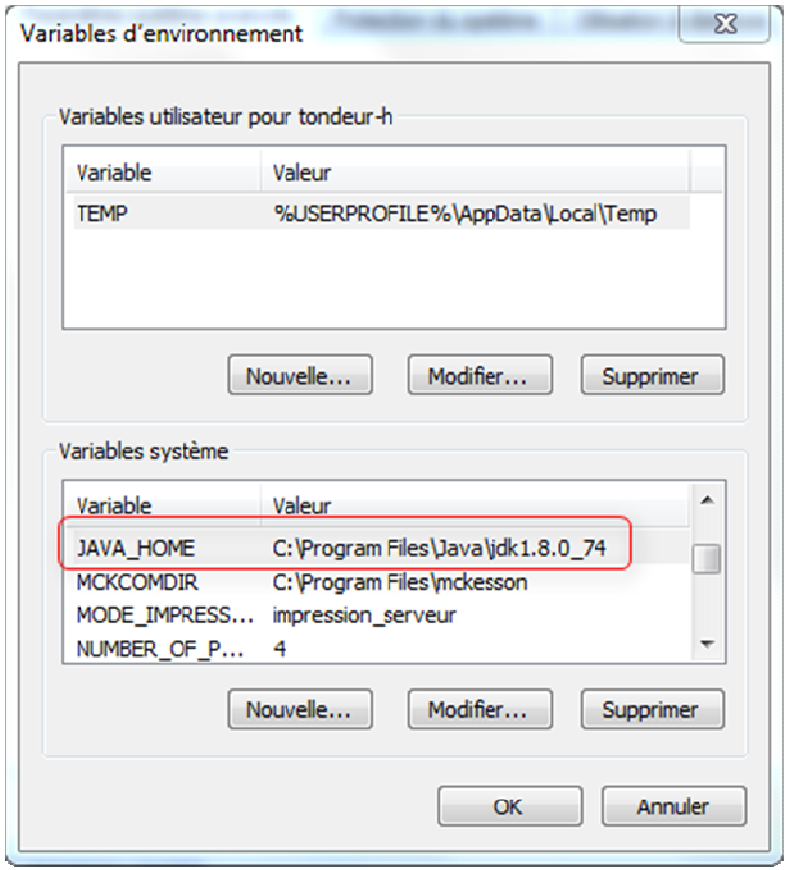
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Java SE Development Kit 8u131

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement Decline License Agreement

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.87 MB	jdk-8u131-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.81 MB	jdk-8u131-linux-arm64-vfp-hflt.tar.gz
Linux x86	164.66 MB	jdk-8u131-linux-i586.rpm
Linux x86	179.39 MB	jdk-8u131-linux-i586.tar.gz
Linux x64	162.11 MB	jdk-8u131-linux-x64.rpm
Linux x64	176.95 MB	jdk-8u131-linux-x64.tar.gz
Mac OS X	226.57 MB	jdk-8u131-macosx-x64.dmg
Solaris SPARC 64-bit	139.79 MB	jdk-8u131-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.13 MB	jdk-8u131-solaris-sparcv9.tar.gz
Solaris x64	140.51 MB	jdk-8u131-solaris-x64.tar.Z
Solaris x64	96.96 MB	jdk-8u131-solaris-x64.tar.gz
Windows x86	191.22 MB	jdk-8u131-windows-i586.exe
Windows x64	198.03 MB	jdk-8u131-windows-x64.exe



Penser à définir la variable **JAVA_HOME** correctement,
Ainsi que votre **PATH**

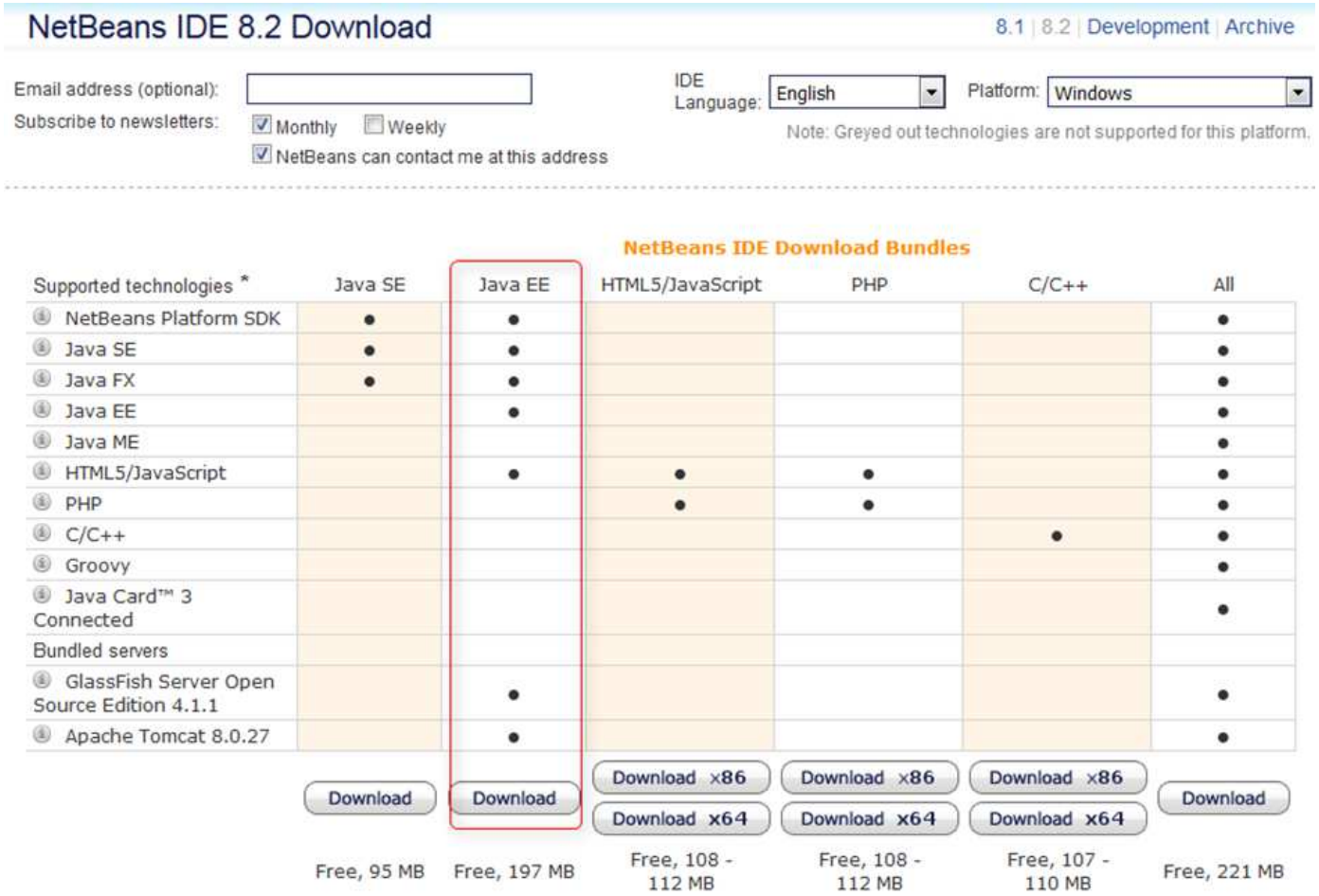
```
echo %JAVA_HOME%
echo %PATH%
```

Développer en Java avec REDIS – Prérequis

Installer NetBeans 8.2 pour JEE

https://netbeans.org/downloads/

Essentiellement pour la prise en charge du HTML et JavaScript



NetBeans IDE 8.2 Download

8.1 | 8.2 | Development | Archive

Email address (optional):

Subscribe to newsletters: ☑ Monthly ☐ Weekly

☑ NetBeans can contact me at this address

IDE Language: English

Platform: Windows

Note: Greyed out technologies are not supported for this platform.

NetBeans IDE Download Bundles

Supported technologies *	Java SE	Java EE	HTML5/JavaScript	PHP	C/C++	All
NetBeans Platform SDK	•	•				•
Java SE	•	•				•
Java FX	•	•				•
Java EE		•				•
Java ME						•
HTML5/JavaScript		•	•	•		•
PHP			•	•		•
C/C++					•	•
Groovy						•
Java Card™ 3 Connected						•
Bundled servers						
GlassFish Server Open Source Edition 4.1.1		•				•
Apache Tomcat 8.0.27		•				•
	Download	Download	Download x86 Download x64	Download x86 Download x64	Download x86 Download x64	Download
	Free, 95 MB	Free, 197 MB	Free, 108 - 112 MB	Free, 108 - 112 MB	Free, 107 - 110 MB	Free, 221 MB

Développer en Java avec REDIS – Prérequis

installer Maven version 3.5.0 https://maven.apache.org/download.cgi

	Link
Binary tar.gz archive	apache-maven-3.5.0-bin.tar.gz
Binary zip archive	apache-maven-3.5.0-bin.zip
Source tar.gz archive	apache-maven-3.5.0-src.tar.gz
Source zip archive	apache-maven-3.5.0-src.zip

Ajouter le chemin

C:\apache-maven-3.5.0-bin\apache-maven-3.5.0\bin Dans la variable PATH

Tester l'installation et la version

```
C:\Users\tondeur-h.CHU>mvn -v
Apache Maven 3.5.0 (ff8f5e7444045639af65f6095c62210b5713f426; 2017-04-03T21:39:06+02:00)
Maven home: C:\apache-maven-3.5.0-bin\apache-maven-3.5.0\bin\..
Java version: 1.8.0_74, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.8.0_74\jre
Default locale: fr_FR, platform encoding: Cp1252
OS name: "windows 7", version: "6.1", arch: "amd64", family: "windows"
```

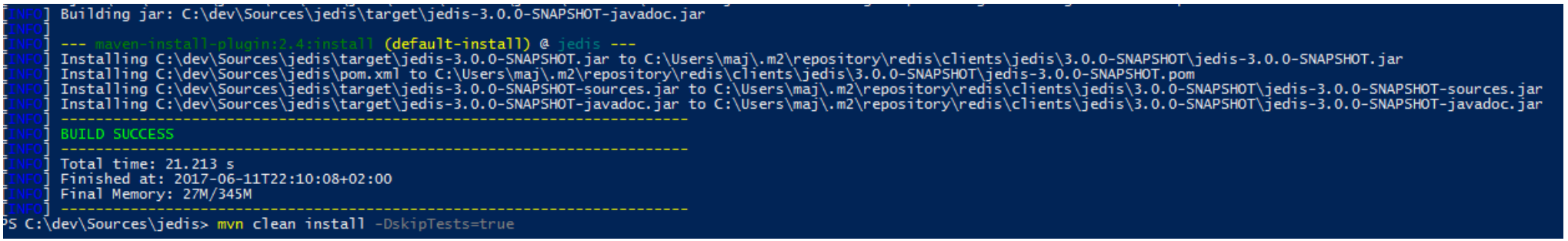
Développer en Java avec REDIS – client Jedis

Télécharger les sources du client Jedis à l'adresse GitHub ci-dessous :

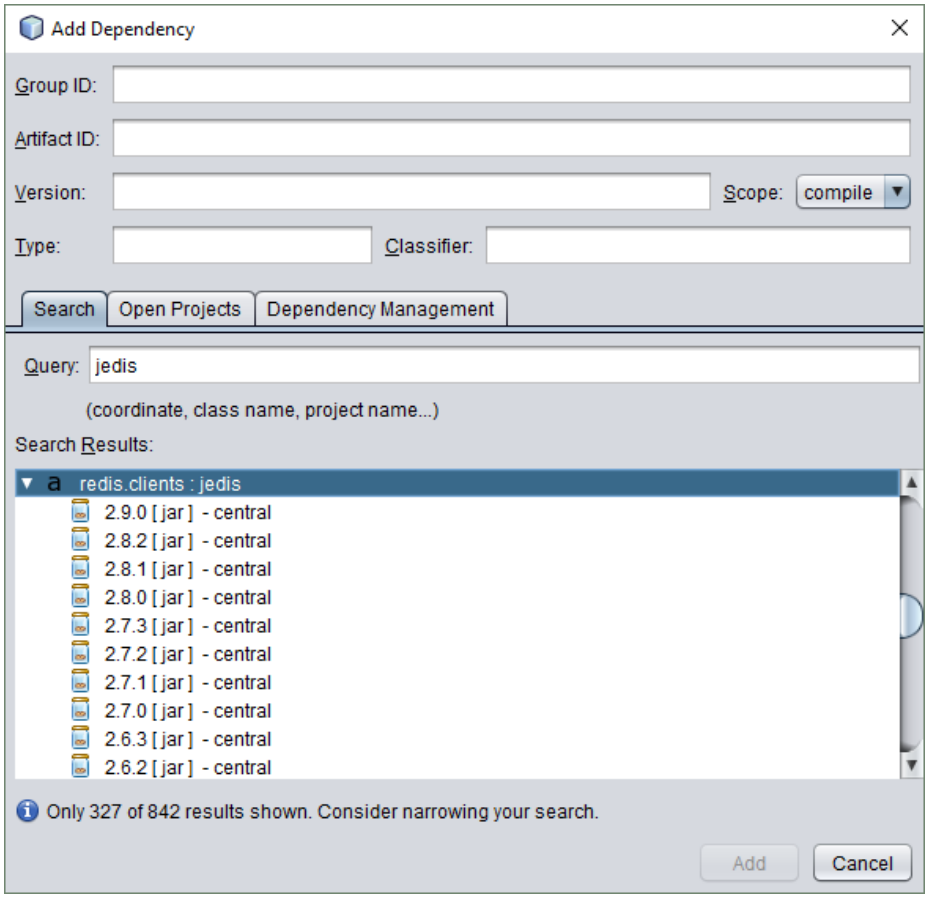
https://github.com/xetorthio/jedis.git

Compiler Jedis avec maven en utilisant la commande suivante :

mvn clean install -DskipTests=true



Dépendance Maven pour le client REDIS



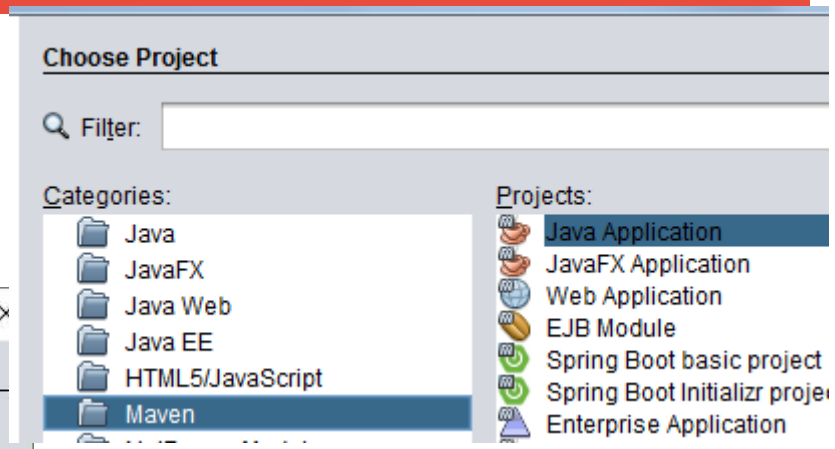
Ce client se trouve également dans le repository Maven en version 2.9 actuellement…

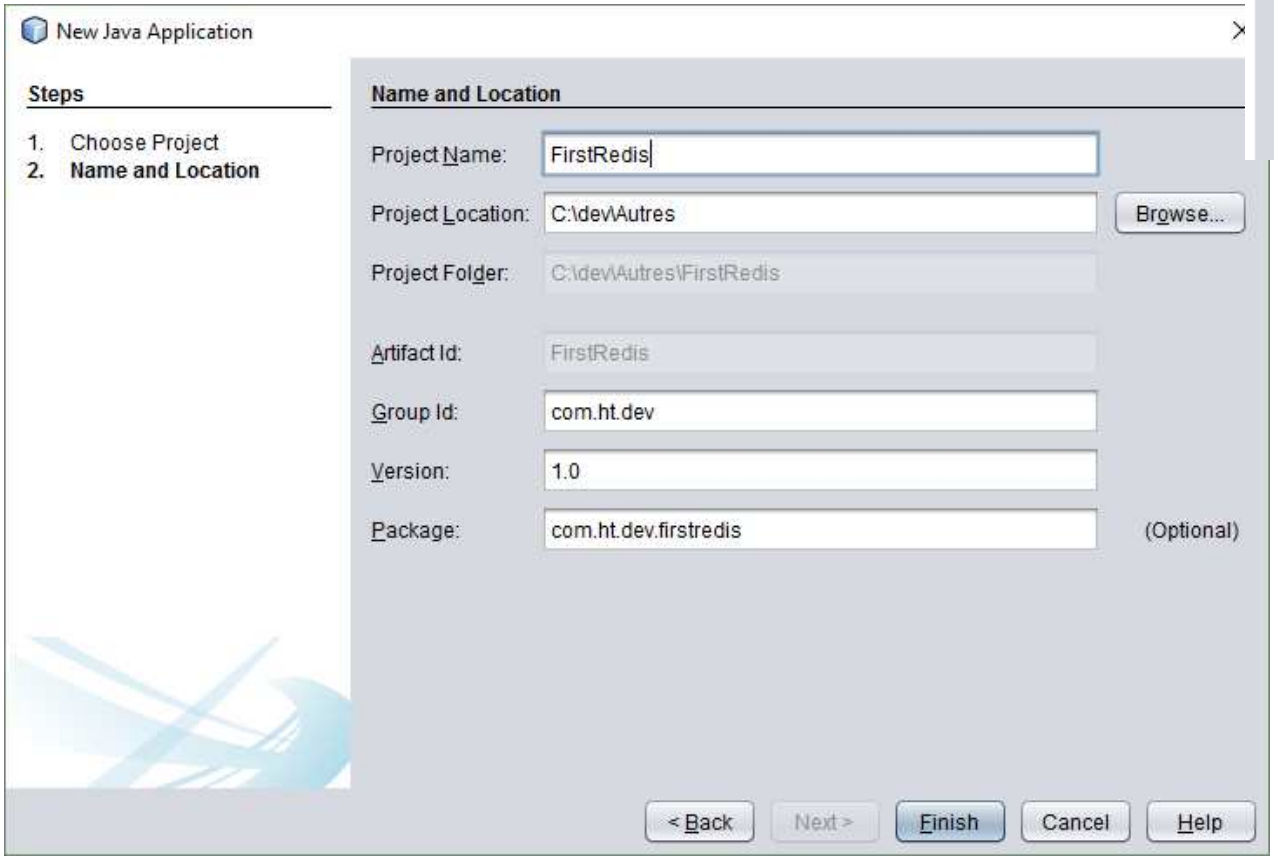
```
<dependency>
        <groupId>redis.clients</groupId>
        <artifactId>jedis</artifactId>
        <version>2.9.0</version>
        <type>jar</type>
        <scope>compile</scope>
</dependency>
```

Premier programme Java avec le client REDIS

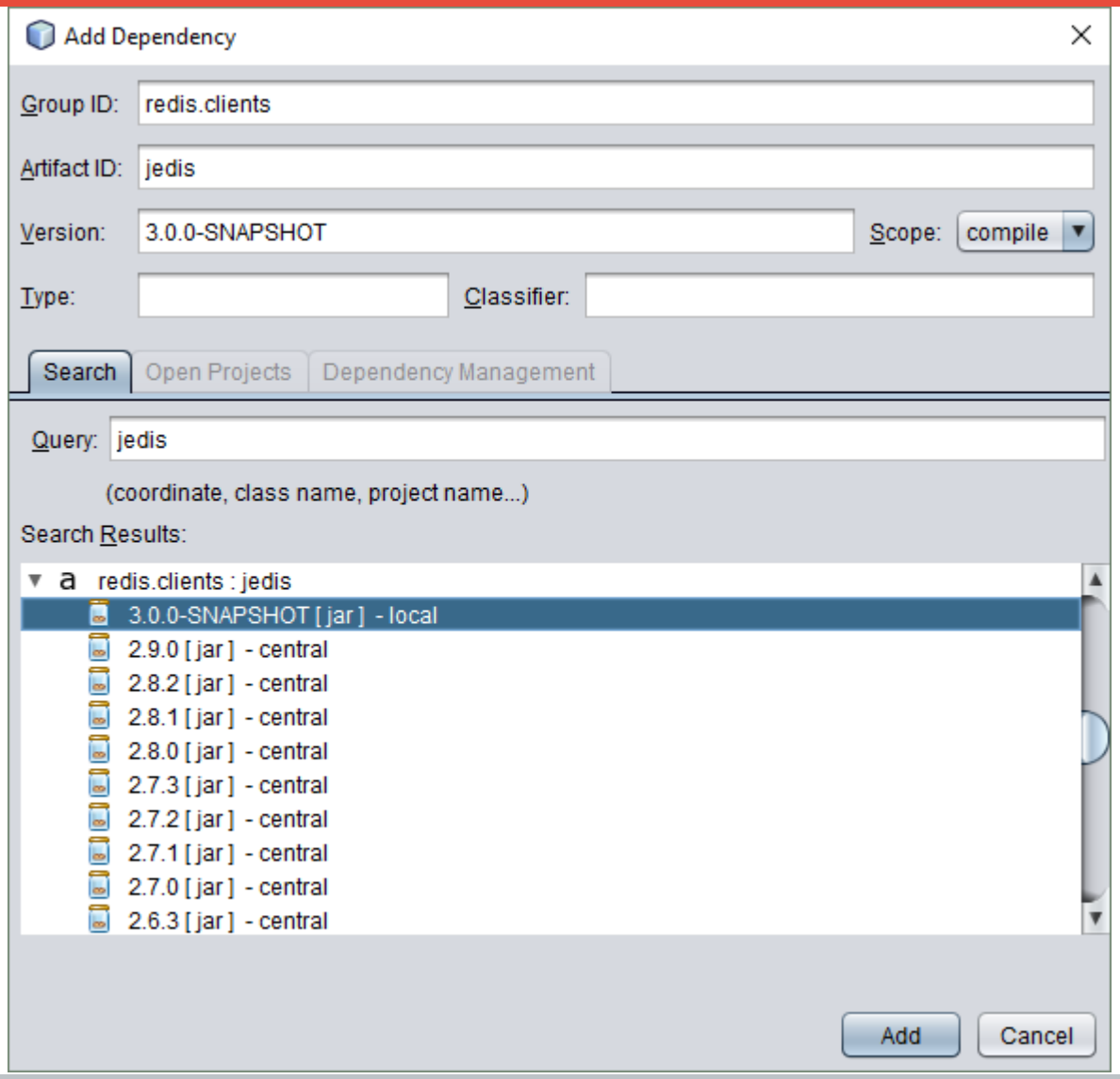
<u>Sous NetBeans</u> :

File | New Project | Maven | Java Application

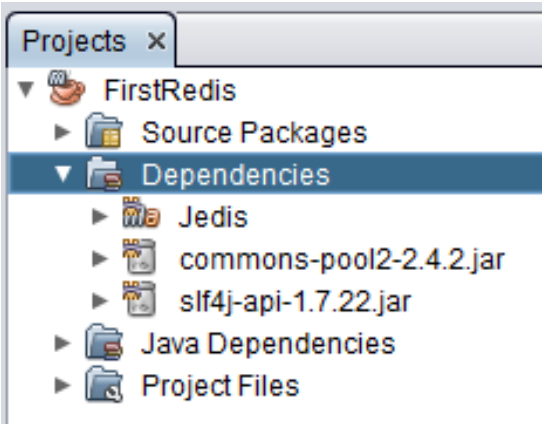




Premier programme Java avec le client REDIS



Ici on utilise notre fichier Snapshot que l'on vient de compiler et qui se trouve dans notre repository Maven local…



Premier programme Java avec le client REDIS

```
package com.ht.dev.firstredis;

import redis.clients.jedis.Jedis;

public class FirstRedis {

    Jedis jedis;

    public FirstRedis() {
        jedis=new Jedis("localhost");
        jedis.set("msg1", "bonjour");
        jedis.set("msg2", "le monde!");

        System.out.println(jedis.get("msg1")+" "+jedis.get("msg2"));
        jedis.close();
    }


    public static void main(String[] args) {
        FirstRedis firstRedis = new FirstRedis();
    }

}
```

Premier programme Java avec le client REDIS

Comme vous avez pu le constater sur ce simple exemple, développer en Java pour REDIS n'est pas très compliqué.
La classe Jedis met à disposition toutes les méthodes qui portent en général le même nom que les commandes REDIS que l'on utilisera.

Le client propose également une gestion des clusters REDIS au travers de la classe JedisCluster.

Il est possible de consulter la documentation des API qui est construite en même temps que le package Jar

Mise en Cluster de REDIS

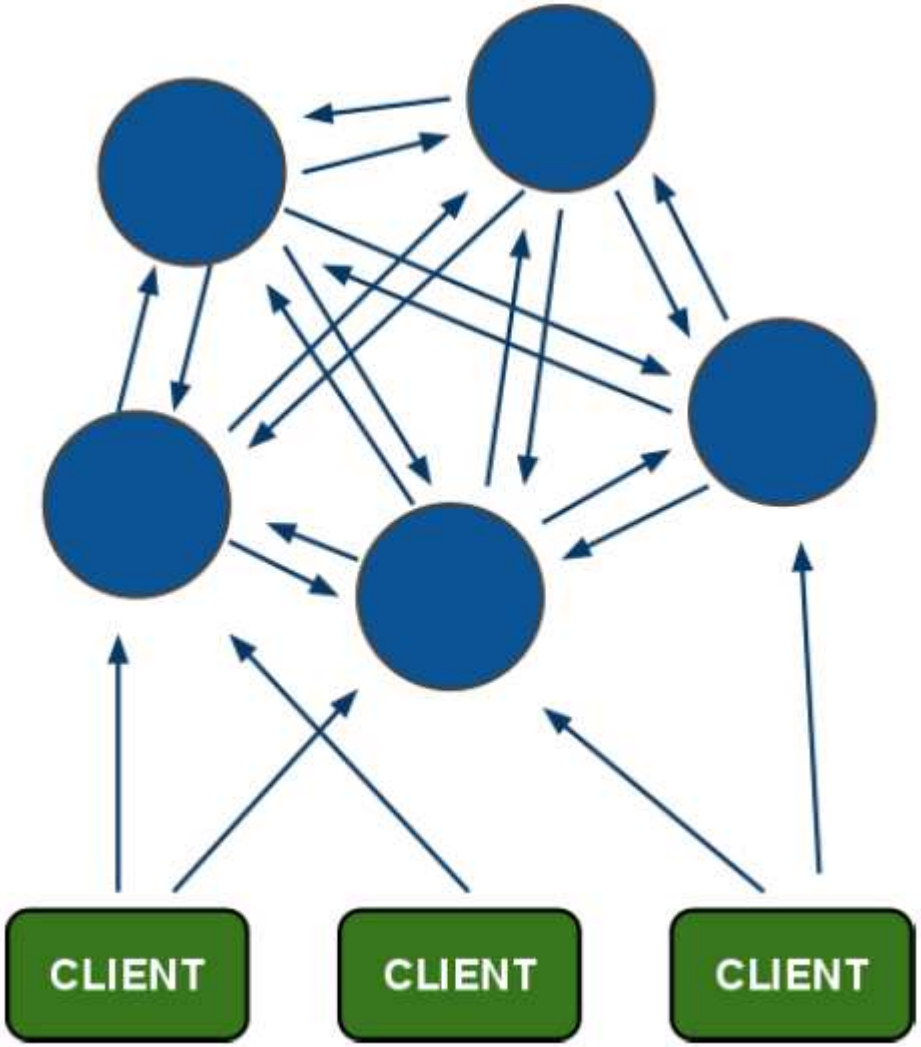
Un Cluster Redis permet d'exécuter une instance Redis ou les données sont éclatées sur plusieurs nœuds.

Redis permet d'un certains niveau de disponibilité, ce qui permet de continuer les opérations malgré l'arrêt de certains nœuds. Le Cluster n'est pas complétement infaillible et peut s'arrêter si un grand nombre de nœuds se sont arrêtés.

En pratique, que peut on espéré d'un Cluster Redis?

- La possibilité de dispatcher les données sur de multiples nœuds.
- La possibilité de continuer les opérations quand un certains nombres de nœuds sont arrêtés.

Mise en Cluster de REDIS



Tous les nœuds sont connecté directement via un Channel de service sur le port TCP de base + 10000,
exemple 6379 ->16379.

Le protocole nœud à nœud est de type binaire, optimisé pour la bande passante et la vitesse

Mise en Cluster de REDIS

Ports TCP d’un Cluster Redis

Chaque nœuds d’un cluster Redis nécessite deux ports pour se communiquer.
Le port standard du serveur Redis par exemple 6379 et un port complémentaire que l’on obtient en additionnant 10000 au port standard soit dans notre exemple 16379.

Ce second port permet au bus du cluster de communiquer de nœud à nœud en utilisant un protocole binaire.
Ce bus de cluster est utilisé par les nœuds pour détecter les « faillures », les mise à jour de configurations, etc…
Ce bus de communication est réservé au communications inter nœuds et vous n’avez jamais besoin de communiquer sur ces ports.

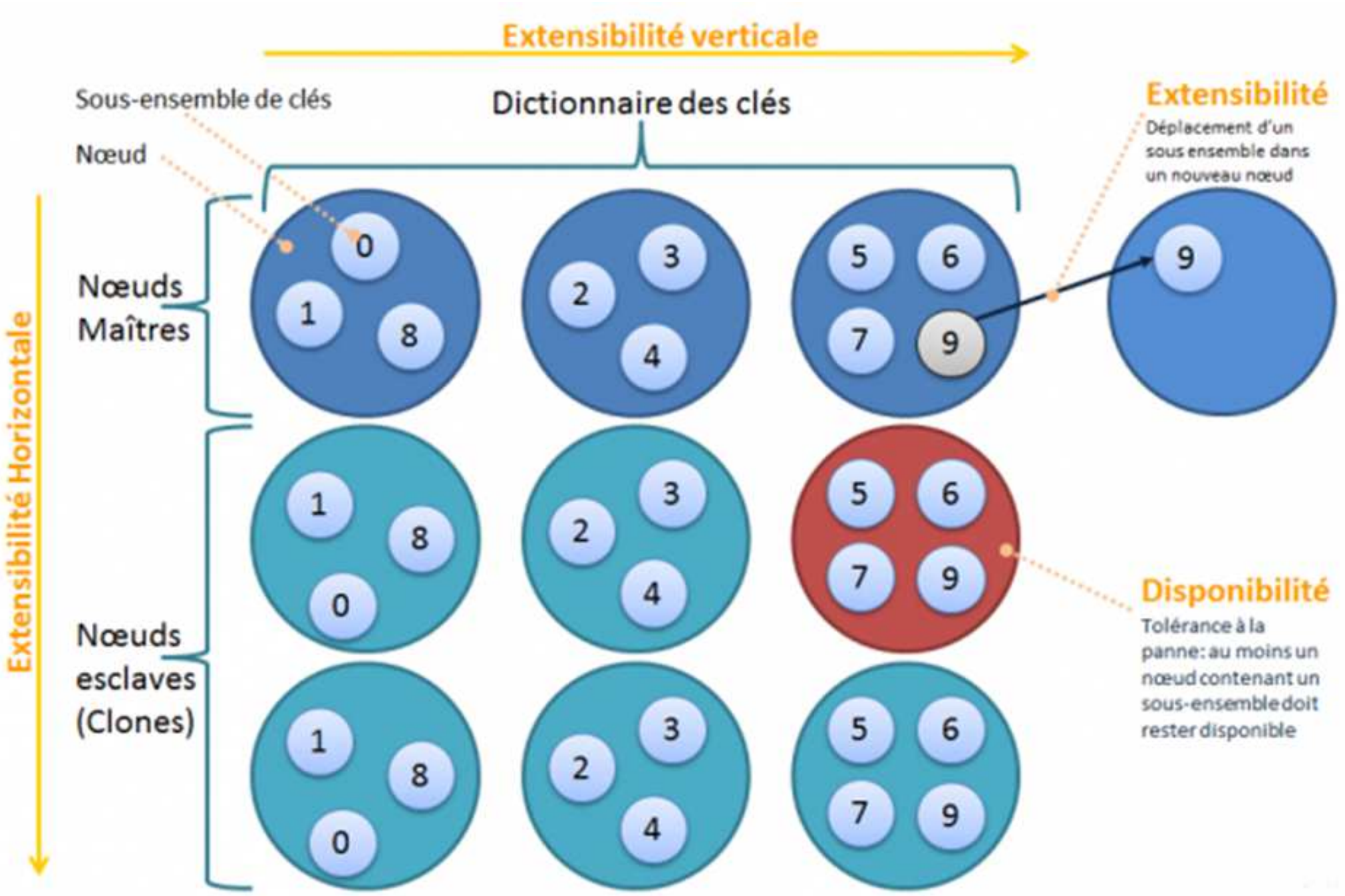
Mise en Cluster de REDIS

En mode *cluster*, un *node* se voit attribuer un ***hash slot*** (une part des données) du *cluster*. **Un *cluster* contient 16384 *hashs* repartis sur les différents *nodes* utilisés**.

Le but est de pouvoir découpler la quantité des données stockées pour le répartir sur plusieurs machines.

Ce partage est effectué à partir d'un calcul utilisant la clé de stockage. Redis est une technologie performante, **mais qui nécessite rapidement beaucoup de mémoire**.

Ce *sharing* (partage) des données va permettre **de partager la charge et la consommation sur plusieurs serveurs**.


Extensibilité verticale
Sous-ensemble de clés
Nœud
Dictionnaire des clés
Extensibilité
Déplacement d'un sous ensemble dans un nouveau nœud
Extensibilité Horizontale
Nœuds Maîtres
Nœuds esclaves (Clones)
0 1 8
2 3 4
5 6 7 9
9
1 8 0
2 3 4
5 6 7 9
1 8 0
2 3 4
5 6 7 9
Disponibilité
Tolérance à la panne: au moins un nœud contenant un sous-ensemble doit rester disponible

Mise en Cluster de REDIS

Pour commencer un *cluster*, Redis préconise **l'utilisation de 3 *nodes*** (A, B et C).

Dans notre cas notre *cluster* sera déployé sur 3 serveurs. Par défaut, **chaque node est démarré en master**.

En cas de défaillance d'un des deux serveurs, nous perdons les nodes qui étaient hébergés et donc une partie des données.

Mise en Cluster de REDIS

Pire, dans ce cas-là, le cluster perd son intégrité **et ne peut plus fonctionner**.

Il est donc nécessaire **d'utiliser l'option Replica pour permettre d'allouer un slave** (A1, B1, C1) pour **chaque** master créé.

Le principe du *slave* est **de répliquer les données du master**.

Le cluster Redis va ensuite g**érer automatiquement les défaillances** d'un *node master* en promulguant le *slave* associé.

Dans ce cas-là, **aucune donnée n'est perdue et le service reste disponible!**

Pré-requis pour la mise en Cluster de REDIS

En prérequis, vous devez avoir installé sur vos trois machines :

1. *build-essential,*
2. *ruby,*
3. *ruby-dev,*
4. *rubygems*.

En pratique : Cluster REDIS

Le fichier de configuration minimum d'un nœud Redis cluster est le suivant

Redis7000.conf

```
port 7000
cluster-enabled yes
cluster-config-file nodes-7000.conf
cluster-node-timeout 5000
cluster-require-full-coverage yes
cluster-migration-barrier 1
cluster-slave-validity-factor 10
appendonly yes
```

Pour démarrer il suffit de lancer : redis-server redis7000.conf

En pratique : Cluster REDIS

Redis est donc en écoute sur le port TCP/7000 pour Redis lui-même, et TCP/17000 pour le bus cluster. Par défaut, Redis n'alloue pas les slots de manière automatique. Il faut donc le faire soi-même :

```
redis-cli -p 7000 CLUSTER ADDSLOTS 0
redis-cli -p 7000 CLUSTER ADDSLOTS 1
redis-cli -p 7000 CLUSTER ADDSLOTS 2
redis-cli -p 7000 CLUSTER ADDSLOTS 3
[...]
redis-cli -p 7000 CLUSTER ADDSLOTS 16383
```

Redis fourni, avec les sources, un script Ruby qui permet de réaliser cela pour efficacement redis-trib.rb

Cet outil va inspecter l'état de l'allocation des slots et vous proposer de corriger ce qui doit l'être.

En pratique : Cluster REDIS

Naturellement, un cluster a un nœud n'est pas d'une utilité extraordinaire. Créons donc une autre instance Redis

```
port 7002
cluster-enabled yes
cluster-config-file nodes7002.conf
cluster-node-timeout 5000
cluster-require-full-coverage yes
cluster-migration-barrier 1
cluster-slave-validity-factor 10
appendonly yes
```

Démarrons cette seconde instance et ajoutons-là au cluster

```
redis-server redis7002.conf redis-cli -p 7002 CLUSTER MEET 127.0.0.1 7000
```

Etat du Cluster de REDIS

```
redis-trib check 127.0.0.1:7000
Connecting to node 127.0.0.1:7000: OK
Connecting to node 127.0.0.1:7002: OK
>>> Performing Cluster Check (using node 127.0.0.1:7000)
M: 1327103b8e5b4921bf47941943c4b279a50d8ed5 127.0.0.1:7000
   slots:0-16383 (16384 slots) master
   0 additional replica(s)
M: 6a5a36527d49b1c21275feffa0f4c3b6b62b49c5 127.0.0.1:7002
   slots: (0 slots) master
   0 additional replica(s)
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
```

Le nouveau nœud a été ajouté, mais il n'est responsable d'aucun slot. Il est nécessaire de les répartir manuellement.

ReSharding du Cluster de REDIS

Dans ce cas précis, nous allons déplacer la moitié les slots

```
redis-trib reshard 127.0.0.1:7000
Connecting to node 127.0.0.1:7000: OK
Connecting to node 127.0.0.1:7002: OK
>>> Performing Cluster Check (using node 127.0.0.1:7000)
M: 1327103b8e5b4921bf47941943c4b279a50d8ed5 127.0.0.1:7000
   slots:0-16383 (16384 slots) master
   0 additional replica(s)
M: 6a5a36527d49b1c21275feffa0f4c3b6b62b49c5 127.0.0.1:7002
   slots: (0 slots) master
   0 additional replica(s)
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
How many slots do you want to move (from 1 to 16384)? 8192
What is the receiving node ID? 6a5a36527d49b1c21275feffa0f4c3b6b62b49c5
Please enter all the source node IDs.
  Type 'all' to use all the nodes as source nodes for the hash slots.
  Type 'done' once you entered all the source nodes IDs.
Source node #1:1327103b8e5b4921bf47941943c4b279a50d8ed5
Source node #2:done
[...]
```

ReSharding du Cluster de REDIS

Ajout d'un troisième nœud sur le port 7004 et resharding des slots

```
redis-trib.rb check 127.0.0.1:7000
Connecting to node 127.0.0.1:7000: OK
Connecting to node 127.0.0.1:7004: OK
Connecting to node 127.0.0.1:7002: OK
>>> Performing Cluster Check (using node 127.0.0.1:7000)
M: 9b6e118f13081f6c8ded4d2419cdb21a0ad1bbc7 127.0.0.1:7000
   slots:8192-16383 (8192 slots) master
   0 additional replica(s)
M: 978ec5bc35cf57332d0c0672a7ff3f2fe3a43666 127.0.0.1:7004
   slots: (0 slots) master
   0 additional replica(s)
M: 1dfc1b008c56c8b9c415d930b7e53f4677f8b1bd 127.0.0.1:7002
   slots:0-8191 (8192 slots) master
   0 additional replica(s)
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
```

Ajout des esclaves du Cluster REDIS

Une fois la répartition effectuée, on peut ajouter des esclaves à chacun des nœuds maîtres. Créez les nœuds Redis sur les ports 7001, 7003 et 7005 et démarrez-les. Ensuite, on peut les intégrer au cluster en tant que réplica d'un maître :

```
redis-trib add-node --slave 127.0.0.1:7001 127.0.0.1:7000
redis-trib add-node --slave 127.0.0.1:7003 127.0.0.1:7002
redis-trib add-node --slave 127.0.0.1:7005 127.0.0.1:7004
```

Test du FailOver du Cluster REDIS

Pour tester le failover ?

Il suffit pour cela de tuer un des process master 7000, ou 7002 ou 7004