

redis

*RE*mote *DI*ctionary *S*erver

Présentations



whoami : **Chahr-Eddine Touil**

À PROPOS DE VOUS

- Expériences avec les bases de données
- Ce que vous attendez de ce cours



Introduction

NoSQL (Not Only SQL)

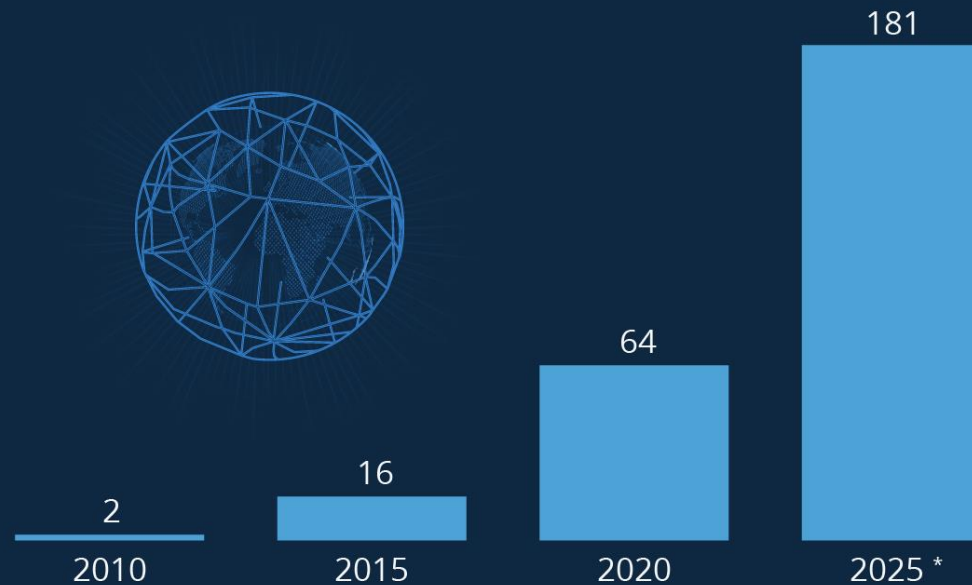
- ❖ Bases de données relationnelles (1970)
- ❖ Explosion des données grâce au web (2000)
 - Volume
 - Vitesse
 - Variété
- ❖ Caractéristiques du NoSQL :
 - privilégie souvent la disponibilité et le partitionnement à la cohérence (théorème de CAP)
 - schémas de stockage sont dynamiques
 - scalabilité verticale ET horizontale
 - Les données sont distribuées

+

•

Le Big Bang du Big Data

Estimation du volume de données numériques créées ou répliquées par an dans le monde, en zettaoctets



Un zettaoctet équivaut à mille milliards de gigaoctets.

* Prévision en date de mars 2021.

Sources : IDC, Seagate, Statista



statista 



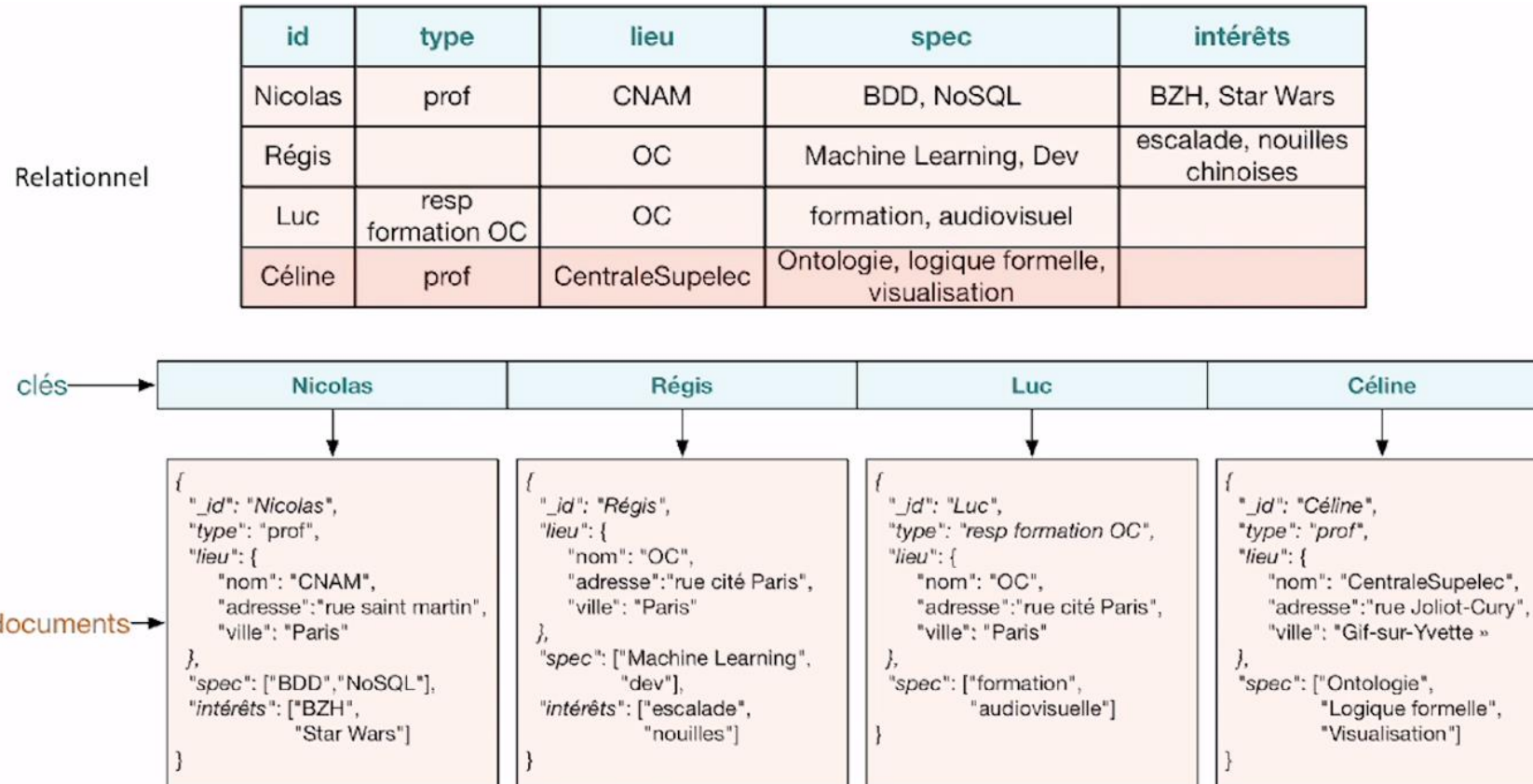
Types de bases de données NoSQL

+

•

BD orientées documents

Couchebase, DynamoDB, cassandra, MongoDB



BD orientées colonnes

Hbase, Spark SQL, elasticsearch

Relationnel

id	type	lieu	spec	intérêts
Nicolas	prof	CNAM	BDD, NoSQL	BZH, Star Wars
Régis		OC	Machine Learning, Dev	escalade, nouilles chinoises
Luc	resp formation OC	OC	formation, audiovisuel	
Céline	prof	CentraleSupelec	Ontologie, logique formelle, visualisation	

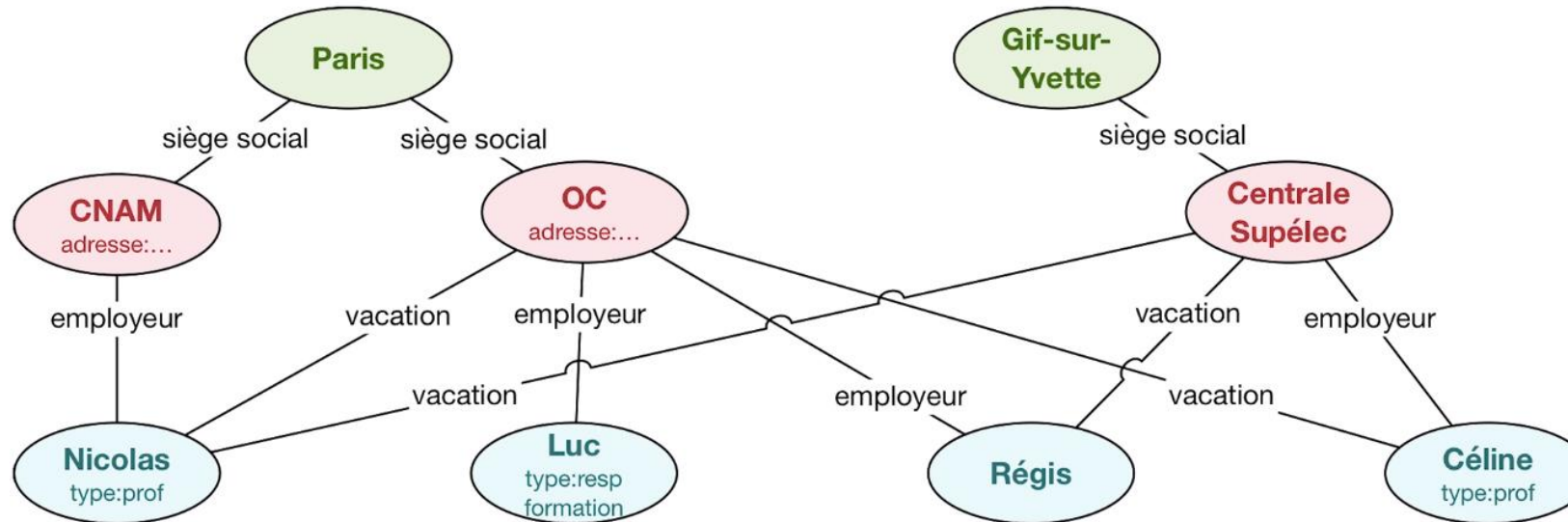
id	type	id	lieu	id	spec	id	intérêts
Nicolas	prof	Céline	Centrale Supelec	Nicolas	BDD	Nicolas	BZH
Céline	prof	Nicolas	CNAM	Nicolas	NoSQL	Nicolas	Star Wars
Luc	resp formation OC	Régis	OC	Régis	Machine Learning	Régis	escalade
		Luc	OC	Régis	Dev	Régis	nouilles chinoises
				Luc	formation		
				Luc	audiovisuel		
				Céline	Ontologie		
				Céline	logique formelle		
				Céline	visualisation		

BD orientées graphes

Neo4J, OrientDB, FlockDB

Relationnel

id	type	lieu	spec	intérêts
Nicolas	prof	CNAM	BDD, NoSQL	BZH, Star Wars
Régis		OC	Machine Learning, Dev	escalade, nouilles chinoises
Luc	resp formation OC	OC	formation, audiovisuel	
Céline	prof	CentraleSupélec	Ontologie, logique formelle, visualisation	

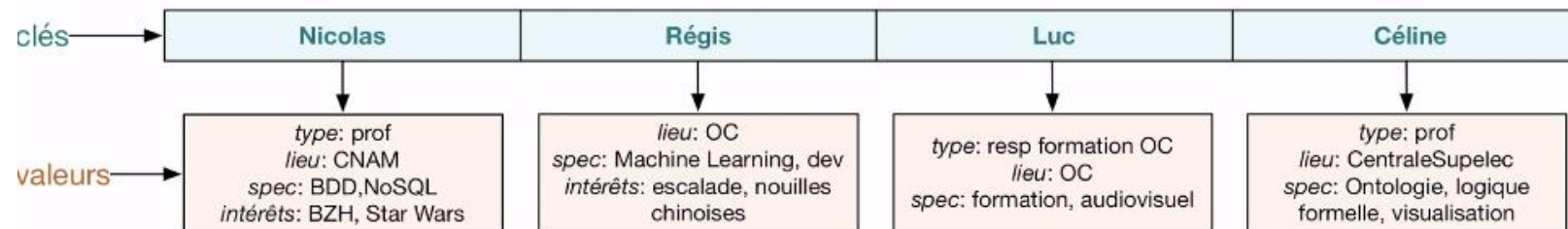


BD orientées clés-valeurs

Redis, Memcached, Amazon SimpleDB

Relationnel

id	type	lieu	spec	intérêts
Nicolas	prof	CNAM	BDD, NoSQL	BZH, Star Wars
Régis		OC	Machine Learning, Dev	escalade, nouilles chinoises
Luc	resp formation OC	OC	formation, audiovisuel	
Céline	prof	CentraleSupélec	Ontologie, logique formelle, visualisation	



Introduction à Redis



Redis :

- Est une solution open source de stockage de données
- Peut être utilisé comme :
 - ✓ Base de données
 - ✓ Système de cache
 - ✓ Broker de messages
- Maintient toutes les données en mémoire (RAM)
- Utilise le disque pour la persistance des données
- Est Basé sur le concept de stockage en « clé-valeur »
- Supporte de nombreuses structures de données
- Supporte un nombre important de langage de connexion (clients)
- Dispose de plusieurs modules pour étendre ses capacités

+

•

Redis :

In-Memory

Open Source

Fast

Key-Value
Data structure

+

•

Simple Access

Persistence

Data
Expiration

High
Availability

Distributed
Cluster

Clients

Modules

Scale out

Installation :

- **Packages :**
 - La plupart des distributions Linux majeures fournissent des paquets pour Redis
 - Il reste possible de partir des sources et compilation à des fins de customisation
- Voir TP 1



Gestion des données

Stockage des données :

- La structure des données est basée sur les clés : **Keys**
- Key = value
- Value peut être :
 - String
 - List
 - Hash
 - Sets
 - Et plus encore ...
- La donnée peut être retrouvée si on connaît exactement le nom de la « key » qui a servi à la stocker
- Toutes les données sont dans des « keys »
- Les données en sont accessibles que par les « keys »

+

•

Stockage des données :

- Ecriture : **SET** key value
- Lecture : **GET** key
- Suppression : **DEL** key
- Test d'existence : **EXISTS** key

```
127.0.0.1:6379>
127.0.0.1:6379> set var1 val1
OK
127.0.0.1:6379> set var2 val2
OK
127.0.0.1:6379> set var3 val3
OK
127.0.0.1:6379> get var1
"val1"
127.0.0.1:6379> KEYS var*
1) "var3"
2) "var2"
3) "var1"
127.0.0.1:6379> KEYS *
1) "var3"
2) "var2"
3) "var1"
127.0.0.1:6379> del var1
(integer) 1
127.0.0.1:6379> get var1
(nil)
127.0.0.1:6379>
[root@redis redis]# redis-cli KEYS "var*"
1) "var3"
2) "var2"
[root@redis redis]# redis-cli KEYS "var*" | xargs redis-cli del
(integer) 2
[root@redis redis]# redis-cli KEYS "var*"
(empty array)
[root@redis redis]#
[root@redis redis]#
```

+

•

Stockage des données : snapshots

```
##### SNAPSHOTTING #####  
  
# Save the DB to disk.  
#  
# save <seconds> <changes>  
#  
# Redis will save the DB if both the given number of seconds and the given  
# number of write operations against the DB occurred.  
#  
# Snapshotting can be completely disabled with a single empty string argument  
# as in following example:  
#  
# save ""  
#  
# Unless specified otherwise, by default Redis will save the DB:  
# * After 3600 seconds (an hour) if at least 1 key changed  
# * After 300 seconds (5 minutes) if at least 100 keys changed  
# * After 60 seconds if at least 10000 keys changed  
#  
# You can set these explicitly by uncommenting the three following lines.  
#  
# save 3600 1  
# save 300 100  
# save 60 10000
```

Stockage des données : répertoire des données

- Répertoire des données par défaut (customisable) :

```
# The Append Only file will also be created inside this directory.  
#  
# Note that you must specify a directory here, not a file name.  
dir /var/lib/redis  
  
##### REPLICATION #####
```

- Nom par défaut du fichier des données (customisable) :

```
# The filename where to dump the DB  
dbfilename dump.rdb  
  
# Remove RDB files used by replication in in
```

Données avec délai d'expiration :

- **SET** key value **ex time** (sec)
- **TTL** key → temps restant avant suppression
- **EXPIRE** key time (sec) → modification du délai d'expiration

NB : les délais peuvent être exprimés en ms avec les arguments :

px : set key value **px** 1200
pttl : **pttl** key
pexpire : **pexpire** key 2500

```
127.0.0.1:6379>
127.0.0.1:6379> KEYS *
(empty array)
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> set key1 value1 ex 120
OK
127.0.0.1:6379> ttl key1
(integer) 112
127.0.0.1:6379> ttl key1
(integer) 107
127.0.0.1:6379> expire key1 10
(integer) 1
127.0.0.1:6379> ttl key1
(integer) 7
127.0.0.1:6379> ttl key1
(integer) -2
127.0.0.1:6379> keys *
(empty array)
127.0.0.1:6379>
```

Données avec délai d'expiration :

Suppression du timeout :

PERSIST key

```
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> set key1 value1 ex 200
OK
127.0.0.1:6379> ttl key1
(integer) 195
127.0.0.1:6379> ttl key1
(integer) 194
127.0.0.1:6379> PERSIST key1
(integer) 1
127.0.0.1:6379> ttl key1
(integer) -1
127.0.0.1:6379>
```

Key spaces :

- « Key space » est équivalent à une base de données
- Par défaut, il y a 16 « key spaces » : 0, 1, 2,..., 15

- Pour passer d'une base à une autre on utilise la commande :

SELECT « space index »

- Des keys dans différentes bases peuvent porter les mêmes noms
- « 0 » est le key space par défaut
- **FLUSHDB** : tout supprimer dans un key space (**Attention !**)

```
[root@redis ~]#  
[root@redis ~]# grep -i ^databases /etc/redis/redis.conf  
databases 16  
[root@redis ~]#
```

```
127.0.0.1:6379>  
127.0.0.1:6379> set key1 base1  
OK  
127.0.0.1:6379> set key2 base1  
OK  
127.0.0.1:6379> keys *  
1) "key1"  
2) "key2"  
127.0.0.1:6379> select 1  
OK  
127.0.0.1:6379[1]> keys *  
(empty array)  
127.0.0.1:6379[1]> set key1 base2  
OK  
127.0.0.1:6379[1]> set key2 base2  
OK  
127.0.0.1:6379[1]> keys *  
1) "key1"  
2) "key2"  
127.0.0.1:6379[1]>
```

Conventions de nommage des keys :

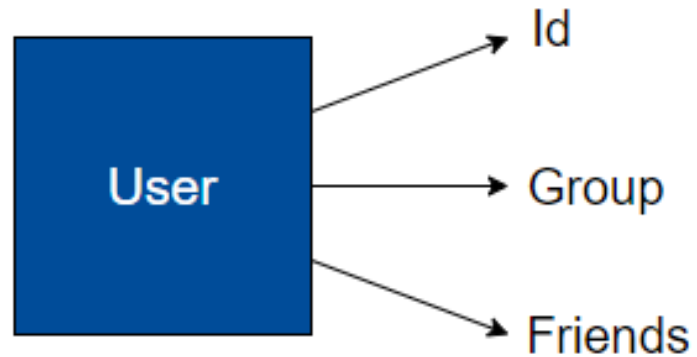
- Les noms des clés doivent être simples et intuitifs
- Éviter les noms courts
- Essayer de reproduire le schéma de la base dans les noms des clés

Exemple :

User :

- ID : *100*
- Nom : *Adam*
- Group : *Admin*
- Friends : *Scott*

Conventions de nommage des keys :



Noms des clés :

user:100
user:100:group
user:100:friends

user:101
user:101:group
user:101:friends

- Les données sont organisées
- Les recherches sont simplifiées

Pattern de recherche :

- H?**i**llo : un seul caractère quelconque après H
- H***i**llo : suite quelconque de caractères après H
- H[**ae**]llo : seulement « **a** » ou « **e** » après H
- H[^**e**]llo : tout caractère sauf « **e** » après H
- H[**a-g**]llo : tout caractère entre « **a** » et « **g** » après H

+

•

Renommage de clés :

- **RENAME** key newkey

```
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> set name1 "John"
OK
127.0.0.1:6379> set name2 "Doe"
OK
127.0.0.1:6379> get name1
"John"
127.0.0.1:6379> get name2
"Doe"
127.0.0.1:6379> rename name1 fname
OK
127.0.0.1:6379> get name1
(nil)
127.0.0.1:6379> get fname
"John"
127.0.0.1:6379> rename name2 lname
OK
127.0.0.1:6379> get name2
(nil)
127.0.0.1:6379> get lname
"Doe"
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
```

*NB : Si « newkey » existe déjà, elle est écrasée.
Dans ce cas, RENAME exécute une opération DEL implicite.*

Renommage de clés :

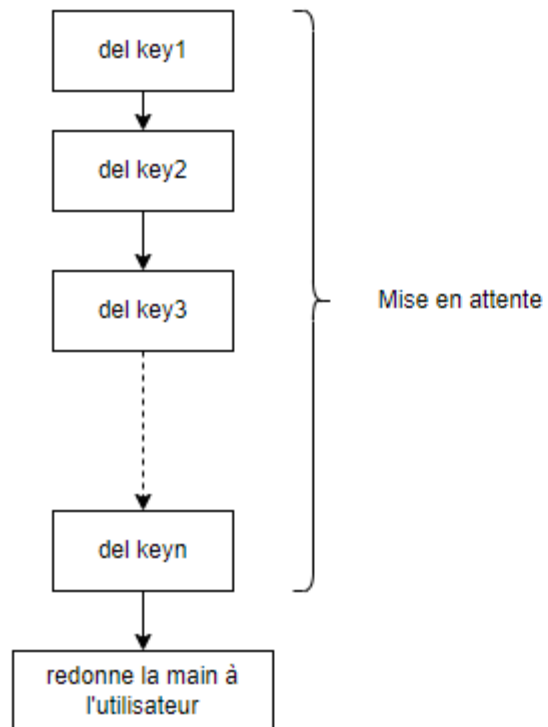
- **RENAMENX** key newkey

```
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> set k1 v1
OK
127.0.0.1:6379> set k2 v2
OK
127.0.0.1:6379> get k1
"v1"
127.0.0.1:6379> get k2
"v2"
127.0.0.1:6379> renamenx k1 k2
(integer) 0
127.0.0.1:6379> get k1
"v1"
127.0.0.1:6379> get k2
"v2"
127.0.0.1:6379> renamenx k1 k3
(integer) 1
127.0.0.1:6379> get k1
(nil)
127.0.0.1:6379> get k3
"v1"
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
```

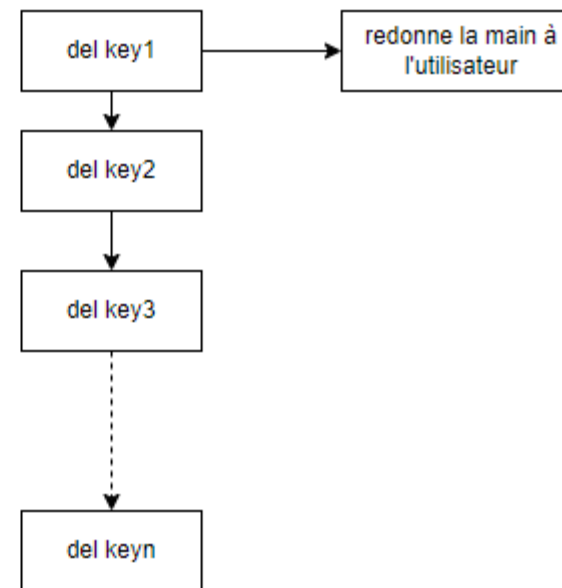
Renomme key en newkey *si celle-ci (newkey) n'existe pas encore.*

Suppression de clés (synchrone et asynchrone)

del key1 key2 key3 ... keyn



unlink key1 key2 key3 ... keyn



+

•

Structures de données : String

- String est la structure de donnée de base dans Redis
- On peut y mettre tous types de données
- String peut être utilisé comme un stockage générique de données
- Tous types de données peuvent être sérialisés et mis dans une clé string :
 - *Exemple : Image → Stream → byte array → string*
- Donnée volumineuse peut être encodée et mise dans une clé string
- Taille maximum d'une variable string est 512 Mo

+

•

Structures de données : String

Cas d'utilisation

Variables simples

- Pour contenir du contenu statique : pages web de sites
- Redis.io utilise redis pour contenir toutes les pages statiques

Cache

- L'utilisation la plus fréquente

Configurations

- Pour contenir des données de configuration :

app:config:hostname
app:config:username
app:config:password

Structures de données : String

Integer

INCR key \longrightarrow key = key + 1

DECR key: \longrightarrow key = key - 1

INCRBY key valeur : \longrightarrow key = key + valeur

DECRBY key valeur : \longrightarrow key = key - valeur

```
127.0.0.1:6379>
127.0.0.1:6379> set student:100:name "Dupont"
OK
127.0.0.1:6379> get student:100:name
"Dupont"
127.0.0.1:6379>
127.0.0.1:6379> INCR student:100:name
(error) ERR value is not an integer or out of range
127.0.0.1:6379>
```

```
127.0.0.1:6379>
127.0.0.1:6379> set student:score:math 15
OK
127.0.0.1:6379> get student:score:math
"15"
127.0.0.1:6379> INCR student:score:math
(integer) 16
127.0.0.1:6379> get student:score:math
"16"
127.0.0.1:6379> DECR student:score:math
(integer) 15
127.0.0.1:6379> get student:score:math
"15"
127.0.0.1:6379> INCRBY student:score:math 3
(integer) 18
127.0.0.1:6379> get student:score:math
"18"
127.0.0.1:6379> DECRBY student:score:math 5
(integer) 13
127.0.0.1:6379> get student:score:math
"13"
127.0.0.1:6379>
```

+

•

Structures de données : String

Float

INCRBYFLOAT key valeur \longrightarrow key = key + valeur

DECRBYFLOAT n'existe pas

```
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> set pi 3.14
OK
127.0.0.1:6379> get pi
"3.14"
127.0.0.1:6379> INCR pi
(error) ERR value is not an integer or out of range
127.0.0.1:6379>
127.0.0.1:6379> INCRBYFLOAT pi 1
"4.14"
127.0.0.1:6379> INCRBYFLOAT pi 1.8
"5.94"
127.0.0.1:6379>
127.0.0.1:6379> INCRBYFLOAT pi -2.8
"3.14"
127.0.0.1:6379> get pi
"3.14"
127.0.0.1:6379> █
```


Structures de données : String

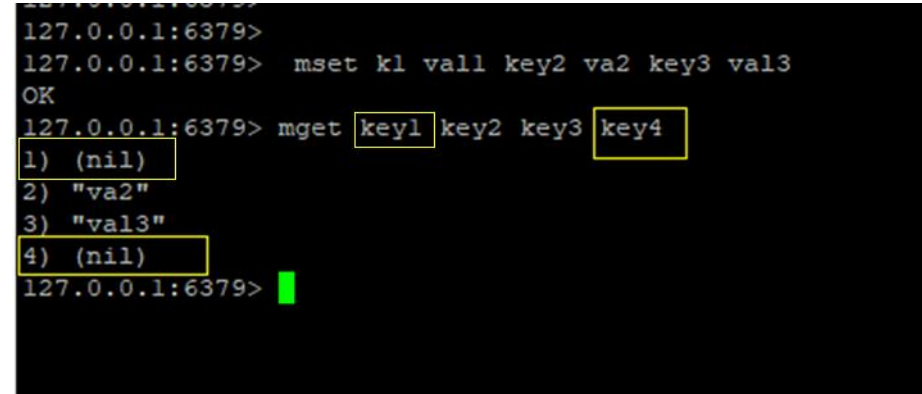
- **MSET** : Définit les keys à leurs valeurs respectives.

MSET remplace les valeurs existantes par de nouvelles valeurs,
tout comme SET

MSET Key1 Val1 key2 Val2 ...

- **MGET** : Renvoie les valeurs de toutes les keys spécifiées.

Pour chaque key qui ne contient pas de valeur de chaîne
ou qui n'existe pas, la valeur spéciale « **nil** » est renvoyée.



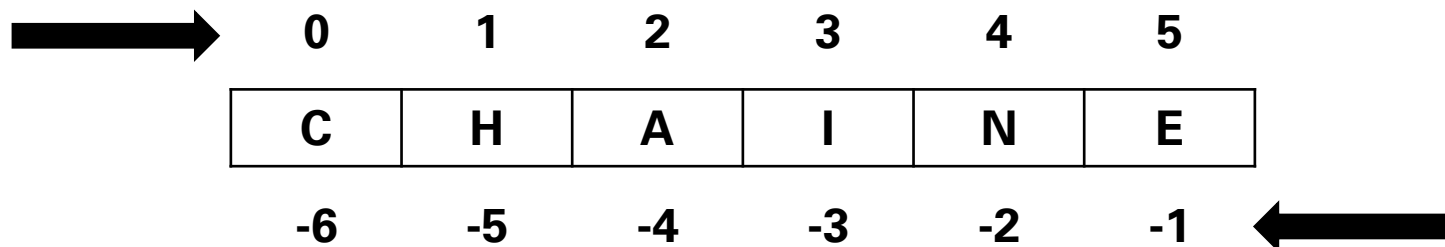
```
127.0.0.1:6379>
127.0.0.1:6379> mset k1 val1 key2 va2 key3 val3
OK
127.0.0.1:6379> mget key1 key2 key3 key4
1) (nil)
2) "va2"
3) "val3"
4) (nil)
127.0.0.1:6379>
```

Structures de données : String

- **GETRANGE** key start end : Renvoie la sous-chaîne de la valeur de la chaîne stockée dans key, déterminée par les décalages start et end.

Des décalages négatifs peuvent être utilisés pour fournir un décalage à partir de la fin de la chaîne.

```
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> set kl "Ceci est un message"
OK
127.0.0.1:6379> getrange kl 0 3
"Ceci"
127.0.0.1:6379> getrange kl -7 -1
"message"
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> getrange kl 5 200
"est un message"
127.0.0.1:6379>
127.0.0.1:6379>
```



Structures de données : String

- **SETRANGE** key offset value : Ecrase une partie de la chaîne stockée dans key, à partir du décalage spécifié, pour toute la longueur de value.

```
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> set message "Hello world"
OK
127.0.0.1:6379> get message
"Hello world"
127.0.0.1:6379>
127.0.0.1:6379> setrange message 6 "redis"
(integer) 11
127.0.0.1:6379>
127.0.0.1:6379> get message
"Hello redis"
127.0.0.1:6379>
127.0.0.1:6379> █
```

Structures de données : List

- List est une structure de données très flexibles
- Elle se présente comme une simple collection d'éléments :
 - 1,2,3,4, ...
- C'est une séquence d'objets ordonnés
- L'ordre des éléments dépend des séquences d'insertion
- Les éléments de List sont des strings
- List peut contenir > 4 milliards d'éléments

+

•

Structures de données : List

Use cases :

- **Event queue** : pour des outils traitant des données séquencées (comme logstash)
- Avoir les **données** les **plus récentes** : LIFO

Structures de données : List

- **LPUSH** key element1 element2 element3 ... :
ajoute les éléments à la tête de la liste
- **RPUSh** key element1 element2 element3 ... :
ajoute les éléments à la fin de la liste
- **LRANGE** key start stop :
affiche les éléments de la key entre les indices start et stop

```
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> LPUSH nbre 1 2 3 4 5
(integer) 5
127.0.0.1:6379> LRANGE nbre 0 -1
1) "5"
2) "4"
3) "3"
4) "2"
5) "1"
127.0.0.1:6379>
127.0.0.1:6379> RPUSH lettres A B C D E
(integer) 5
127.0.0.1:6379> LRANGE lettres 0 -1
1) "A"
2) "B"
3) "C"
4) "D"
5) "E"
127.0.0.1:6379>
```

Structures de données : List

- **LINDEX** key index : renvoie l'élément de la liste à l'indice « index »

Les indices commencent à 0 :

- 0 → premier élément
- 1 → deuxième élément
- -1 → dernier élément
- -2 → avant dernier élément

```
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> LPUSH lettres A B C D E F
(integer) 6
127.0.0.1:6379>
127.0.0.1:6379> LRange lettres 0 -1
1) "F"
2) "E"
3) "D"
4) "C"
5) "B"
6) "A"
127.0.0.1:6379> LINDEX lettres 2
"D"
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> LINDEX lettres -1
"A"
127.0.0.1:6379> LINDEX lettres -2
"B"
127.0.0.1:6379> █
```

Structures de données : List

- **LINSERT** key before | after pivot element :

Insert « element » avant ou après l'élément « pivot »

```
127.0.0.1:6379>
127.0.0.1:6379> LPUSH lettres A B C D E F
(integer) 6
127.0.0.1:6379>
127.0.0.1:6379> LRANGE lettres 0 -1
1) "F"
2) "E"
3) "D"
4) "C"
5) "B"
6) "A"
127.0.0.1:6379> LINSERT lettres before D Y
(integer) 7
127.0.0.1:6379> LRANGE lettres 0 -1
1) "F"
2) "E"
3) "Y"
4) "D"
5) "C"
6) "B"
7) "A"
127.0.0.1:6379> LINSERT lettres after C Z
(integer) 8
127.0.0.1:6379> LRANGE lettres 0 -1
1) "F"
2) "E"
3) "Y"
4) "D"
5) "C"
6) "Z"
7) "B"
8) "A"
127.0.0.1:6379>
```


Structures de données : List

- **LPOP** key count :

Supprime de « key » « count » éléments depuis la haut de list

NB : si count est omis, le premier élément est supprimé

- **RPOP** key count :

Supprime de « key » « count » éléments depuis le bas de list

NB : si count est omis, le dernier élément est supprimé

```
127.0.0.1:6379>
127.0.0.1:6379> LPUSH lettres A B C D E F
(integer) 6
127.0.0.1:6379>
127.0.0.1:6379> LRANGE lettres 0 -1
1) "F"
2) "E"
3) "D"
4) "C"
5) "B"
6) "A"
127.0.0.1:6379> LPOP lettres
"F"
127.0.0.1:6379> LRANGE lettres 0 -1
1) "E"
2) "D"
3) "C"
4) "B"
5) "A"
127.0.0.1:6379> LPOP lettres 2
1) "E"
2) "D"
127.0.0.1:6379> LRANGE lettres 0 -1
1) "C"
2) "B"
3) "A"
127.0.0.1:6379> RPOP lettres 2
1) "A"
2) "B"
127.0.0.1:6379> LRANGE lettres 0 -1
1) "C"
127.0.0.1:6379>
```

Structures de données : List

- **LTRIM** key start stop :

Découpe une liste existante de manière à ce qu'elle ne contienne que la plage d'éléments spécifiée.

```
127.0.0.1:6379>
127.0.0.1:6379> LPUSH lettres A B C D E F
(integer) 6
127.0.0.1:6379>
127.0.0.1:6379> LRANGE lettres 0 -1
1) "F"
2) "E"
3) "D"
4) "C"
5) "B"
6) "A"
127.0.0.1:6379> LTRIM lettres 2 4
OK
127.0.0.1:6379> LRANGE lettres 0 -1
1) "D"
2) "C"
3) "B"
127.0.0.1:6379>
127.0.0.1:6379>
```

Structures de données : List

- **LSET** key index element :

modifie la valeur d'une key à un index donné

Une erreur est renvoyée pour les index hors plage de la key

```
127.0.0.1:6379> LRANGE nbre 0 -1
1) "1"
2) "2"
3) "3"
4) "500"
5) "5"
6) "6"
7) "7"
127.0.0.1:6379> LSET nbre 111 55
(error) ERR index out of range
127.0.0.1:6379> █
```

```
127.0.0.1:6379>
127.0.0.1:6379> RPUSH nbre 1 2 3 4 5 6 7
(integer) 7
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> keys *
1) "nbre"
127.0.0.1:6379>
127.0.0.1:6379> LRANGE nbre 0 -1
1) "1"
2) "2"
3) "3"
4) "4"
5) "5"
6) "6"
7) "7"
127.0.0.1:6379> LSET nbre 3 500
OK
127.0.0.1:6379> LRANGE nbre 0 -1
1) "1"
2) "2"
3) "3"
4) "500"
5) "5"
6) "6"
7) "7"
127.0.0.1:6379> █
```

Structures de données : List

- **LLEN** key :

Renvoie la longueur de la liste stockée en clé.

Si key n'existe pas, elle est interprétée comme une liste vide et 0 est renvoyé.

Une erreur est renvoyée si la valeur stockée dans key n'est pas une liste.



```
127.0.0.1:6379>
127.0.0.1:6379> LRANGE nbre 0 -1
1) "1"
2) "2"
3) "3"
4) "500"
5) "5"
6) "6"
7) "7"
127.0.0.1:6379> LLEN nbre
(integer) 7
127.0.0.1:6379>
127.0.0.1:6379> LLEN toto
(integer) 0
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> keys *
1) "nbre"
127.0.0.1:6379>
```

Structures de données : List

- **LPOS** key element RANK rank :

renvoie l'index des éléments correspondants dans une liste

Rank est le rang du premier element à renvoyer :

- rank = 1 signifie que le premier élément doit être retourné,
- rank = 2 signifie que le deuxième élément doit être retourné, et ainsi de suite.

Par défaut, si aucune option n'est donnée, elle parcourt la liste du début à la fin, à la recherche du premier élément correspondant à "element"

```
127.0.0.1:6379>
127.0.0.1:6379> RPUSH mylist a b c 1 2 3 c c
(integer) 8
127.0.0.1:6379>
127.0.0.1:6379> LRANGE mylist 0 -1
1) "a"
2) "b"
3) "c"
4) "1"
5) "2"
6) "3"
7) "c"
8) "c"
127.0.0.1:6379>
127.0.0.1:6379> LPOS mylist c
(integer) 2
127.0.0.1:6379> LPOS mylist c RANK 2
(integer) 6
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
```

Structures de données : List

- **LRM** key count element :

Supprime les premières occurrences d'éléments égaux à l'élément de la liste stockée dans key.

L'argument count influence l'opération de la manière suivante :

- count > 0 : Supprime les éléments égaux à l'élément se déplaçant du début vers la fin.
- count < 0 : Supprime les éléments égaux à l'élément se déplaçant de la fin vers le début.
- count = 0 : Supprime tous les éléments égaux à l'élément.

```
127.0.0.1:6379>
127.0.0.1:6379> RPUSH mylist a b a a b c c a
(integer) 8
127.0.0.1:6379> LRANGE mylist 0 -1
1) "a"
2) "b"
3) "a"
4) "a"
5) "b"
6) "c"
7) "c"
8) "a"
127.0.0.1:6379> LREM mylist 3 a
(integer) 3
127.0.0.1:6379> LRANGE mylist 0 -1
1) "b"
2) "b"
3) "c"
4) "c"
5) "a"
127.0.0.1:6379> LREM mylist 1 c
(integer) 1
127.0.0.1:6379> LRANGE mylist 0 -1
1) "b"
2) "b"
3) "c"
4) "a"
127.0.0.1:6379>
```

Structures de données : List

- **LMOVE** source destination <LEFT | RIGHT> <LEFT | RIGHT>

Renvoie et supprime atomiquement le premier ou le dernier élément de la liste source donnée, et place l'élément au premier ou au dernier élément de la liste de destination spécifiée.

```
127.0.0.1:6379>
127.0.0.1:6379> rpush orders:pending 1 2 3 4 5
(integer) 5
127.0.0.1:6379> LRANGE orders:pending 0 -1
1) "1"
2) "2"
3) "3"
4) "4"
5) "5"
127.0.0.1:6379> lmove orders:pending orders:completed right left
"5"
127.0.0.1:6379> LRANGE orders:pending 0 -1
1) "1"
2) "2"
3) "3"
4) "4"
127.0.0.1:6379> LRANGE orders:completed 0 -1
1) "5"
127.0.0.1:6379> lmove orders:pending orders:completed right left
"4"
127.0.0.1:6379> LRANGE orders:completed 0 -1
1) "4"
2) "5"
127.0.0.1:6379>
127.0.0.1:6379>
```

Structures de données : Hashes

- Sont des listes de paires champ : valeur
- Champ1:valeur1, champ2:valeur2, champ3:valeur3, ...
- Similaires aux objets json
- Représentent un mapping entre les champs et les valeurs
- Ils ne possèdent pas de schéma (Schemaless)
- Un hash peut contenir > 4 Milliards d'éléments
- Utilisés généralement pour des objets à plusieurs propriétés :
 - User : nom, prénom, âge, tél, adresse, ...

Structures de données : Hashes

- **HSET** key field value [field value ...]
- **HGET** key field
- **HGETALL** key
 - Retourne tous les champs et toutes les valeurs

```
127.0.0.1:6379> HSET user fname "John" lname "Doe" age "30"
(integer) 3
127.0.0.1:6379> HGET user lname
"Doe"
127.0.0.1:6379> HGETALL user
1) "fname"
2) "John"
3) "lname"
4) "Doe"
5) "age"
6) "30"
127.0.0.1:6379> HSET user status 1
(integer) 1
127.0.0.1:6379> HGETALL user
1) "fname"
2) "John"
3) "lname"
4) "Doe"
5) "age"
6) "30"
7) "status"
8) "1"
127.0.0.1:6379> HSET user age 25
(integer) 0
127.0.0.1:6379> HGETALL user
1) "fname"
2) "John"
3) "lname"
4) "Doe"
5) "age"
6) "25"
7) "status"
8) "1"
127.0.0.1:6379>
127.0.0.1:6379>
```

Ajout d'un champ

modification d'un champ

Structures de données : Hashes

- **HMGET** key field [field ...]

Renvoie les valeurs associées aux champs spécifiés

- **HLEN** key

Renvoie nombre de champs

```
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> HLEN user
(integer) 4
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
```

```
127.0.0.1:6379>
127.0.0.1:6379> HGETALL user
1) "fname"
2) "John"
3) "lname"
4) "Doe"
5) "age"
6) "25"
7) "status"
8) "1"
127.0.0.1:6379>
127.0.0.1:6379> HMGET user fname lname
1) "John"
2) "Doe"
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> HMGET user fname lname login
1) "John"
2) "Doe"
3) (nil)
```

Structures de données : Hashes

- **HDEL** key field [field ...]

Supprime les champs spécifiés de key

```
127.0.0.1:6379>
127.0.0.1:6379> HGETALL user
1) "fname"
2) "John"
3) "lname"
4) "Doe"
5) "age"
6) "25"
7) "status"
8) "1"
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> HDEL user age status
(integer) 2
127.0.0.1:6379>
127.0.0.1:6379> HGETALL user
1) "fname"
2) "John"
3) "lname"
4) "Doe"
127.0.0.1:6379>
127.0.0.1:6379>
```

Structures de données : Hashes

- **HEXISTS** key field

Teste l'existence d'un champ dans key

Code retour :

- **1** si key contient le champ « field »
- **0** si key ne contient pas « field », ou si key n'existe pas

```
127.0.0.1:6379>
127.0.0.1:6379> HGETALL user
1) "fname"
2) "John"
3) "lname"
4) "Doe"
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> HEXISTS user lname
(integer) 1
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> HEXISTS key age
(integer) 0
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> HEXISTS toto lname
(integer) 0
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
```

Structures de données : Hashes

- **HKEYS** key

Renvoie tous les champs de key

Uniquement les champs sans les valeurs

```
127.0.0.1:6379>
127.0.0.1:6379> HSET user fname "John" lname "Doe" age 35 status 1
(integer) 4
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> HGETALL user
1) "fname"
2) "John"
3) "lname"
4) "Doe"
5) "age"
6) "35"
7) "status"
8) "1"
127.0.0.1:6379> HKEYS user
1) "fname"
2) "lname"
3) "age"
4) "status"
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
```

Structures de données : Hashes

- **HVALS** key

Renvoie toutes les valeurs des champs de key

Uniquement les valeurs sans les champs

```
127.0.0.1:6379>
127.0.0.1:6379> HGETALL user
1) "fname"
2) "John"
3) "lname"
4) "Doe"
5) "age"
6) "35"
7) "status"
8) "1"
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> HVALS user
1) "John"
2) "Doe"
3) "35"
4) "1"
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
```

Structures de données : Hashes

- **HSETNX** key field value

Définit « field » avec sa valeur uniquement si « field » n'existe pas encore. Si « key » n'existe pas, une nouvelle clé contenant le hash est créée.

Si « field » existe déjà, cette opération n'a aucun effet.

```
127.0.0.1:6379> HGETALL user
1) "fname"
2) "John"
3) "lname"
4) "Doe"
5) "age"
6) "35"
7) "status"
8) "1"
127.0.0.1:6379>
127.0.0.1:6379> HSETNX user age 45
(integer) 0
127.0.0.1:6379> HGETALL user
1) "fname"
2) "John"
3) "lname"
4) "Doe"
5) "age"
6) "35"
7) "status"
8) "1"
127.0.0.1:6379> HSETNX user login "jd"
(integer) 1
127.0.0.1:6379> HGETALL user
1) "fname"
2) "John"
3) "lname"
4) "Doe"
5) "age"
6) "35"
7) "status"
8) "1"
9) "login"
10) "jd"
```

Structures de données : Sets

- Listes de strings :
 - Uniques : → éviter d'avoir des doublons
 - Non ordonnées
- Utilisent des opérations mathématiques (les ensembles) :
 - Intersection
 - Différence
 - Union
- Max nombre d'éléments ~ 4 milliards

+

•

Structures de données : Sets

- **SADD** key membre [membre ...]

Ajoute les membres spécifiés à l'ensemble stocké

dans key.

Les membres spécifiés qui font déjà partie de cet ensemble sont ignorés.

```
127.0.0.1:6379> SADD ip 1.1.1.1
(integer) 1
127.0.0.1:6379> SADD ip 1.1.1.2
(integer) 1
127.0.0.1:6379> SADD ip 1.1.1.3 1.1.1.4 1.1.1.5
(integer) 3
127.0.0.1:6379>
127.0.0.1:6379> SMEMBERS ip
1) "1.1.1.4"
2) "1.1.1.3"
3) "1.1.1.2"
4) "1.1.1.1"
5) "1.1.1.5"
127.0.0.1:6379> SADD ip 1.1.1.3
(integer) 0
127.0.0.1:6379> SMEMBERS ip
1) "1.1.1.3"
2) "1.1.1.2"
3) "1.1.1.1"
4) "1.1.1.4"
5) "1.1.1.5"
127.0.0.1:6379>
```

Structures de données : Sets

- **SREM** key membre [membre ...]

Supprime les membres spécifiés de l'ensemble stocké dans key.

Les membres spécifiés qui ne font pas partie de cet ensemble sont ignorés

```
127.0.0.1:6379> SMEMBERS ip
1) "1.1.1.3"
2) "1.1.1.2"
3) "1.1.1.1"
4) "1.1.1.4"
5) "1.1.1.5"
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> SREM ip 1.1.1.4
(integer) 1
127.0.0.1:6379> SMEMBERS ip
1) "1.1.1.2"
2) "1.1.1.1"
3) "1.1.1.5"
4) "1.1.1.3"
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
```

Structures de données : Sets

- **SPOP** key [count]

Retire et renvoie un ou plusieurs membres

aléatoires du set.

```
127.0.0.1:6379> SADD alpha a b c d e f g h
(integer) 8
127.0.0.1:6379>
127.0.0.1:6379> SMEMBERS alpha
1) "e"
2) "a"
3) "d"
4) "g"
5) "c"
6) "b"
7) "f"
8) "h"
127.0.0.1:6379> SPOP alpha 3
1) "d"
2) "a"
3) "h"
127.0.0.1:6379> SMEMBERS alpha
1) "e"
2) "g"
3) "c"
4) "b"
5) "f"
127.0.0.1:6379> SPOP alpha
"f"
127.0.0.1:6379> SMEMBERS alpha
1) "e"
2) "g"
3) "c"
4) "b"
127.0.0.1:6379>
```

Structures de données : Sets

- **SISMEMBER** key membre

Renvoie si le « membre » est un membre du set stocké dans key.

- Ne pas confondre avec SMEMBERS
- La commande est sensible à la casse.

- **SMISMEMBER** key membre [membre]

Même fonctionnement pour des vérifications multiples

```
127.0.0.1:6379> SMEMBERS alpha
1) "e"
2) "g"
3) "c"
4) "b"
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> SISMEMBER alpha c
(integer) 1
127.0.0.1:6379>
127.0.0.1:6379> SISMEMBER alpha h
(integer) 0
127.0.0.1:6379>
```

```
127.0.0.1:6379> SMEMBERS alpha
1) "e"
2) "g"
3) "c"
4) "b"
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> SMISMEMBER alpha g b h q
1) (integer) 1
2) (integer) 1
3) (integer) 0
4) (integer) 0
127.0.0.1:6379>
```

Structures de données : Sets

- **SRANDMEMBER** key [count]

Renvoie un nombre aléatoire de membres du set

Si count est omis alors count=1

```
127.0.0.1:6379> SMEMBERS num
1) "1"
2) "2"
3) "3"
4) "4"
5) "5"
6) "6"
7) "7"
8) "8"
9) "9"
127.0.0.1:6379> SRANDMEMBER num
"5"
127.0.0.1:6379> SRANDMEMBER num
"2"
127.0.0.1:6379> SRANDMEMBER num 3
1) "7"
2) "8"
3) "9"
127.0.0.1:6379> SRANDMEMBER num 3
1) "2"
2) "5"
3) "8"
127.0.0.1:6379>
```



Structures de données : Sets

- **SMOVE** source destination membre

Déplace un membre du set « source » vers set

« destination ».

Cette opération est atomique : À chaque instant,

l'élément apparaîtra comme un membre de la source

ou de la destination pour tous les clients.

```
127.0.0.1:6379> KEYS *
1) "num"
127.0.0.1:6379> SMEMBERS num
1) "1"
2) "2"
3) "3"
4) "4"
5) "5"
6) "6"
127.0.0.1:6379> SMOVE num pairs 2
(integer) 1
127.0.0.1:6379> SMOVE num pairs 4
(integer) 1
127.0.0.1:6379> SMOVE num pairs 6
(integer) 1
127.0.0.1:6379> KEYS *
1) "pairs"
2) "num"
127.0.0.1:6379> SMEMBERS num
1) "1"
2) "3"
3) "5"
127.0.0.1:6379> SMEMBERS pairs
1) "2"
2) "4"
3) "6"
127.0.0.1:6379>
```

Structures de données : Sets

- **SUNION** key [key ...]

Renvoie les membres de l'ensemble résultant de l'union de tous les sets donnés.

- **SUNIONSTORE** destination key [key ...]

Equivalut à **SUNION**, mais au lieu de renvoyer le résultat, il est stocké dans « destination »

NB : si « destination » existe, elle est écrasée.

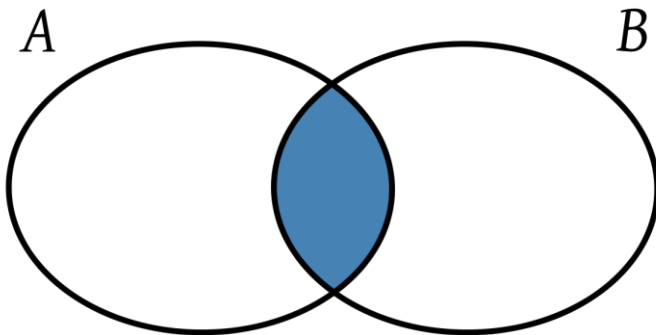
```
127.0.0.1:6379> SADD un 1 2 3 4 5
(integer) 5
127.0.0.1:6379> SADD deux 4 5 6 7 8 9
(integer) 6
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> SUNION un deux
1) "1"
2) "2"
3) "3"
4) "4"
5) "5"
6) "6"
7) "7"
8) "8"
9) "9"
127.0.0.1:6379>
```

```
127.0.0.1:6379> SUNIONSTORE trois un deux
(integer) 9
127.0.0.1:6379> SMEMBERS trois
1) "1"
2) "2"
3) "3"
4) "4"
5) "5"
6) "6"
7) "7"
8) "8"
9) "9"
127.0.0.1:6379> KEYS *
1) "trois"
2) "deux"
3) "un"
127.0.0.1:6379>
```

Structures de données : Sets

- **SINTER** key [key ...]

Renvoie les membres de l'ensemble résultant de l'intersection de tous les sets donnés.

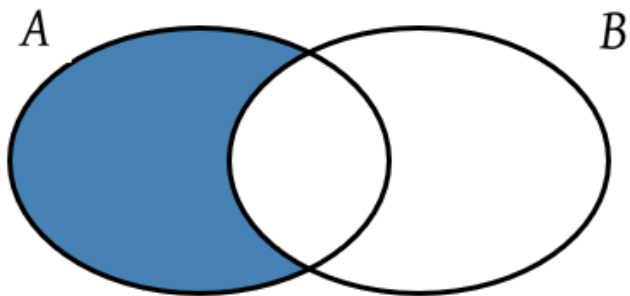


```
127.0.0.1:6379> SMEMBERS un
1) "1"
2) "2"
3) "3"
4) "4"
5) "5"
127.0.0.1:6379> SMEMBERS deux
1) "4"
2) "5"
3) "6"
4) "7"
5) "8"
6) "9"
127.0.0.1:6379> SINTER un deux
1) "4"
2) "5"
```


Structures de données : Sets

- **SDIFF** key [key ...]

Renvoie les membres de l'ensemble résultant de la **différence** entre le **premier** set et tous les sets successifs. Autrement dit : les membres appartenant au premier set et n'appartenant pas aux suivants.



```
127.0.0.1:6379> SMEMBERS k1
1) "d"
2) "c"
3) "b"
4) "a"
127.0.0.1:6379> SMEMBERS k2
1) "c"
127.0.0.1:6379> SMEMBERS k3
1) "e"
2) "c"
3) "a"
127.0.0.1:6379>
127.0.0.1:6379> SDIFF k1 k2 k3
1) "d"
2) "b"
127.0.0.1:6379>
```

Structures de données : Sorted Sets

- Mix entre les structures de données **Set** et **Hash**
- Comme les Hash, les Sorted Sets stockent :
 - Multitude de membres et
 - Des valeurs numériques appelées « scores »
- Tous les membres sont uniques
- Ils sont ordonnés selon les valeurs du « score »
- Les « Sorted Sets » sont utilisés pour des données nécessitant un classement
- Aucun tri supplémentaire n'est nécessaire pour le client qui récupère les données.

+

•

Structures de données : Sorted Sets

- **ZADD** key [NX | XX] [GT | LT] [CH] [INCR] score member [score member ...]

Ajoute tous les membres spécifiés avec les scores correspondants à « key »

NX : MAJ des éléments existants. Pas de nouveaux éléments

XX : Ajout de nouveaux éléments. Pas de MAJ d'éléments existant

GT : MAJ des éléments existants si le nouveau score est inférieur au score actuel.

LT : MAJ des éléments existants si le nouveau score est supérieur au score actuel.

CH : Renvoie le nombre d'éléments modifiés. Cela inclut les membres nouvellement ajoutés et les membres dont les scores ont été modifiés.

INCR : Incrémente la valeur du score du membre

Structures de données : Sorted Sets

- **ZADD** key score member [score member ...]
- **ZRANGE** key start stop [WITHSCORES]
Tri ascendant
- **ZREVRANGE** key start stop [WITHSCORES]
Tri descendant

NB : stop = -1 → dernier élément

```
127.0.0.1:6379> ZADD jeu:resultats 10 j1 34 j2 5 j3 3 j4
(integer) 4
127.0.0.1:6379>
127.0.0.1:6379> ZRANGE jeu:resultats 0 -1 withscores
1) "j4"
2) "3"
3) "j3"
4) "5"
5) "j1"
6) "10"
7) "j2"
8) "34"
```

```
127.0.0.1:6379> ZREVRANGE jeu:resultats 0 -1 withscores
1) "j2"
2) "34"
3) "j1"
4) "10"
5) "j3"
6) "5"
7) "j4"
8) "3"
```

Structures de données : Sorted Sets

- **ZINCRBY** key incrément membre

Augmente le score d'un membre de de la valeur « incrément ».

« incrément » peut être > 0 ou < 0

- Si « membre » n'existe pas, il est ajouté avec le score « incrément »

```
127.0.0.1:6379> ZRANGE jeu:resultats 0 -1 withscores
1) "j4"
2) "3"
3) "j3"
4) "5"
5) "j1"
6) "10"
7) "j2"
8) "34"
127.0.0.1:6379> ZINCRBY jeu:resultats 15 j4
"18"
127.0.0.1:6379> ZRANGE jeu:resultats 0 -1 withscores
1) "j3"
2) "5"
3) "j1"
4) "10"
5) "j4"
6) "18"
7) "j2"
8) "34"
127.0.0.1:6379>
```

Structures de données : Sorted Sets

- **ZRANK** key membre

Renvoie le rang du membre du set stocké dans « key »,
les scores étant classées de la plus faible valeur à la plus
élevée.

NB : Le rang (ou l'indice) est basé sur 0, c'est à dire, **le
membre ayant le score le plus bas a le rang 0.**

```
127.0.0.1:6379> ZRANGE articles 0 -1 withscores
1) "art6" 0
2) "2" 1
3) "art3" 2
4) "5" 3
5) "art1" 4
6) "10" 5
7) "art7" 6
8) "13" 7
9) "art4" 8
10) "22" 9
11) "art2" 10
12) "35" 11
13) "art5" 12
14) "150" 13

127.0.0.1:6379> ZRANK articles art2
(integer) 10
127.0.0.1:6379> ZRANK articles art6
(integer) 0
127.0.0.1:6379> ZRANK articles art5
(integer) 12
127.0.0.1:6379>
```



Structures de données : HyperLogLog

- HyperLogLog est une structure de données probabiliste qui estime la cardinalité d'un ensemble.
- Utilisé généralement pour estimer le nombre d'éléments uniques dans un set de très grande cardinalité.
- Permet d'estimer la cardinalité d'un set ayant jusqu'à 2^{64} membres.
- **HyperLogLog ne stocke pas les données** mais la **cardinalité** de la « key »

+

•

Structures de données : HyperLogLog

- **PFADD** key element [element ...]

Ajoute les éléments donnés en arguments à la « key »

- **PFCOUNT** key

Renvoie la cardinalité de key

- **PFMERGE** key [key ...]

Fusionner plusieurs HyperLogLog en une seule dont la cardinalité se rapproche de la cardinalité de l'union des HyperLogLog sources.

```
127.0.0.1:6379>
127.0.0.1:6379> PFADD nbre1 1 2 3 4 5 6 7 8 9 10
(integer) 1
127.0.0.1:6379>
127.0.0.1:6379> PFADD nbre2 1 2 3 4 5 6 11 12 13 14
(integer) 1
127.0.0.1:6379> PFCOUNT nbre1
(integer) 10
127.0.0.1:6379> PFCOUNT nbre2
(integer) 10
127.0.0.1:6379> PMERGE nbre nbre1 nbre2
OK
127.0.0.1:6379> PFCOUNT nbre
(integer) 14
127.0.0.1:6379>
```


Modèle Publication / Abonnement



Pub / Sub

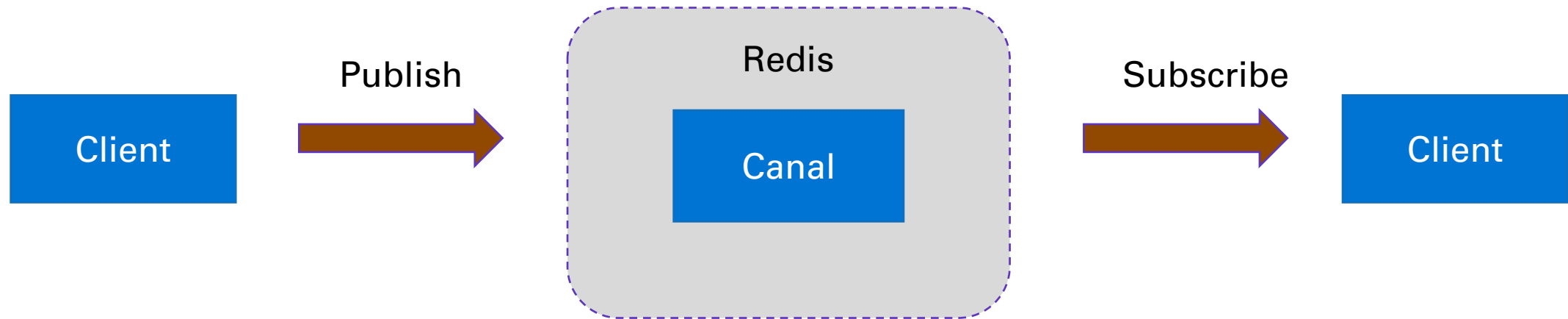
- Permet de créer des bus simples de messages
- Ceci se réalise par :
 - Publication de messages sur un canal
 - Abonnement aux messages d'un canal
- Il est basé sur « fire and forget messages »
- Redis va se positionner comme broker (courtier) central entre plusieurs « clients » pour :
 - Poster simplement des messages
 - Consommer ces messages

+

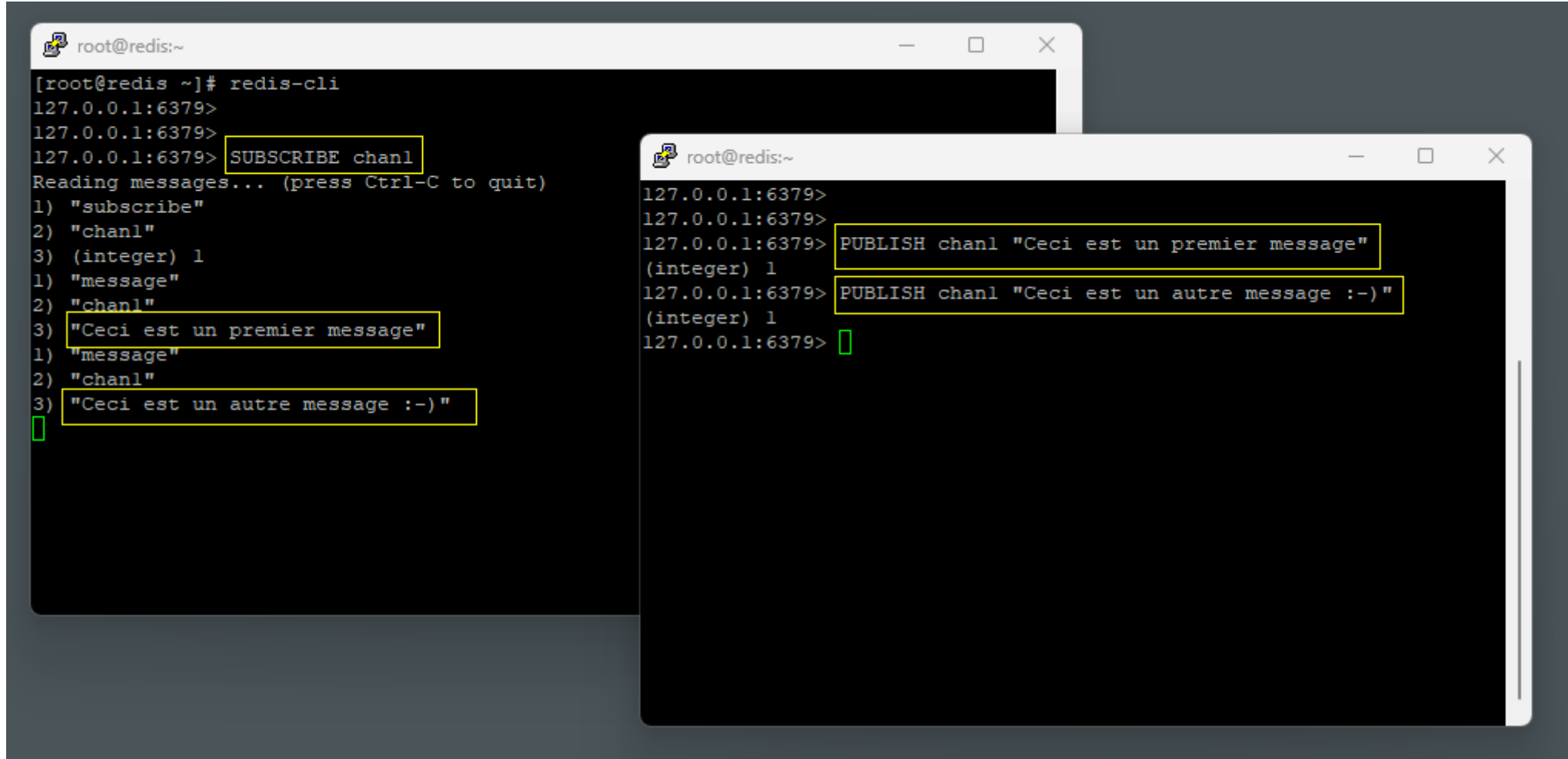
•

Pub / Sub

REDIS

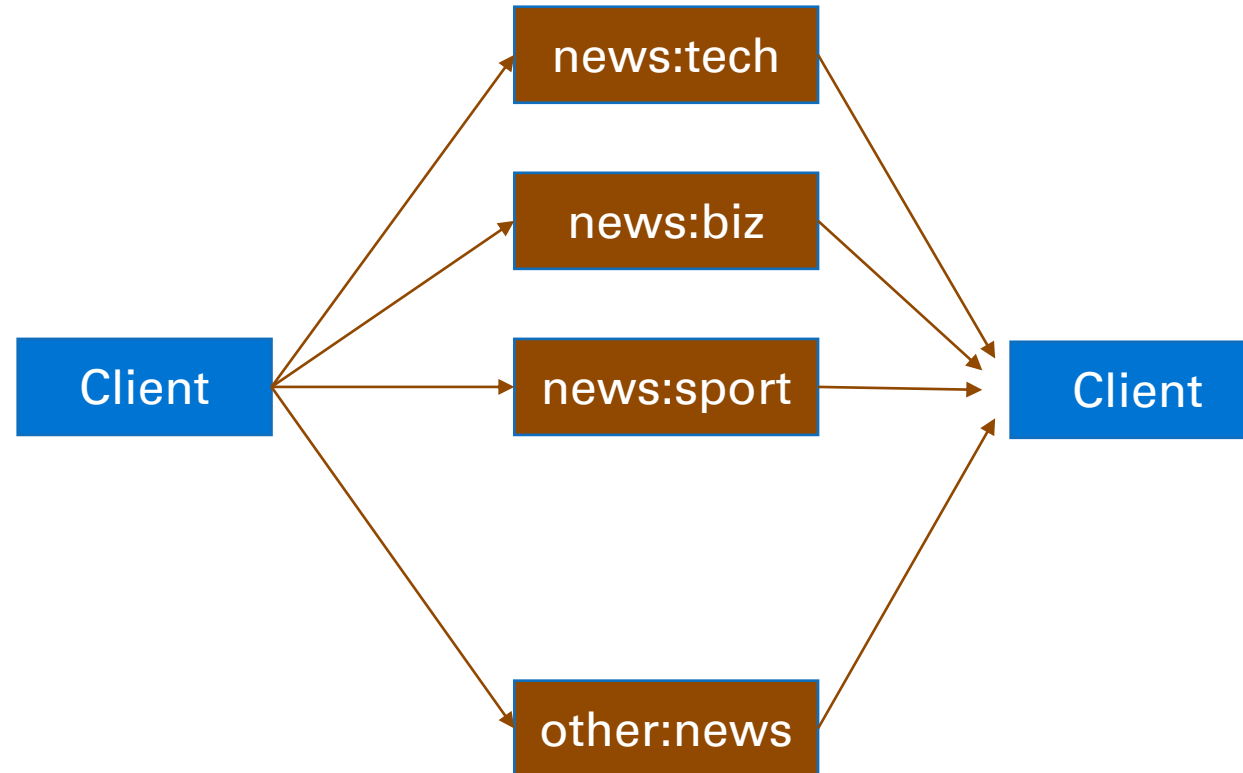


Pub / Sub



```
root@redis:~  
[root@redis ~]# redis-cli  
127.0.0.1:6379>  
127.0.0.1:6379>  
127.0.0.1:6379> SUBSCRIBE chan1  
Reading messages... (press Ctrl-C to quit)  
1) "subscribe"  
2) "chan1"  
3) (integer) 1  
1) "message"  
2) "chan1"  
3) "Ceci est un premier message"  
1) "message"  
2) "chan1"  
3) "Ceci est un autre message :-)"  
[br/>  
root@redis:~  
127.0.0.1:6379>  
127.0.0.1:6379>  
127.0.0.1:6379> PUBLISH chan1 "Ceci est un premier message"  
(integer) 1  
127.0.0.1:6379> PUBLISH chan1 "Ceci est un autre message :-)"  
(integer) 1  
127.0.0.1:6379> [
```

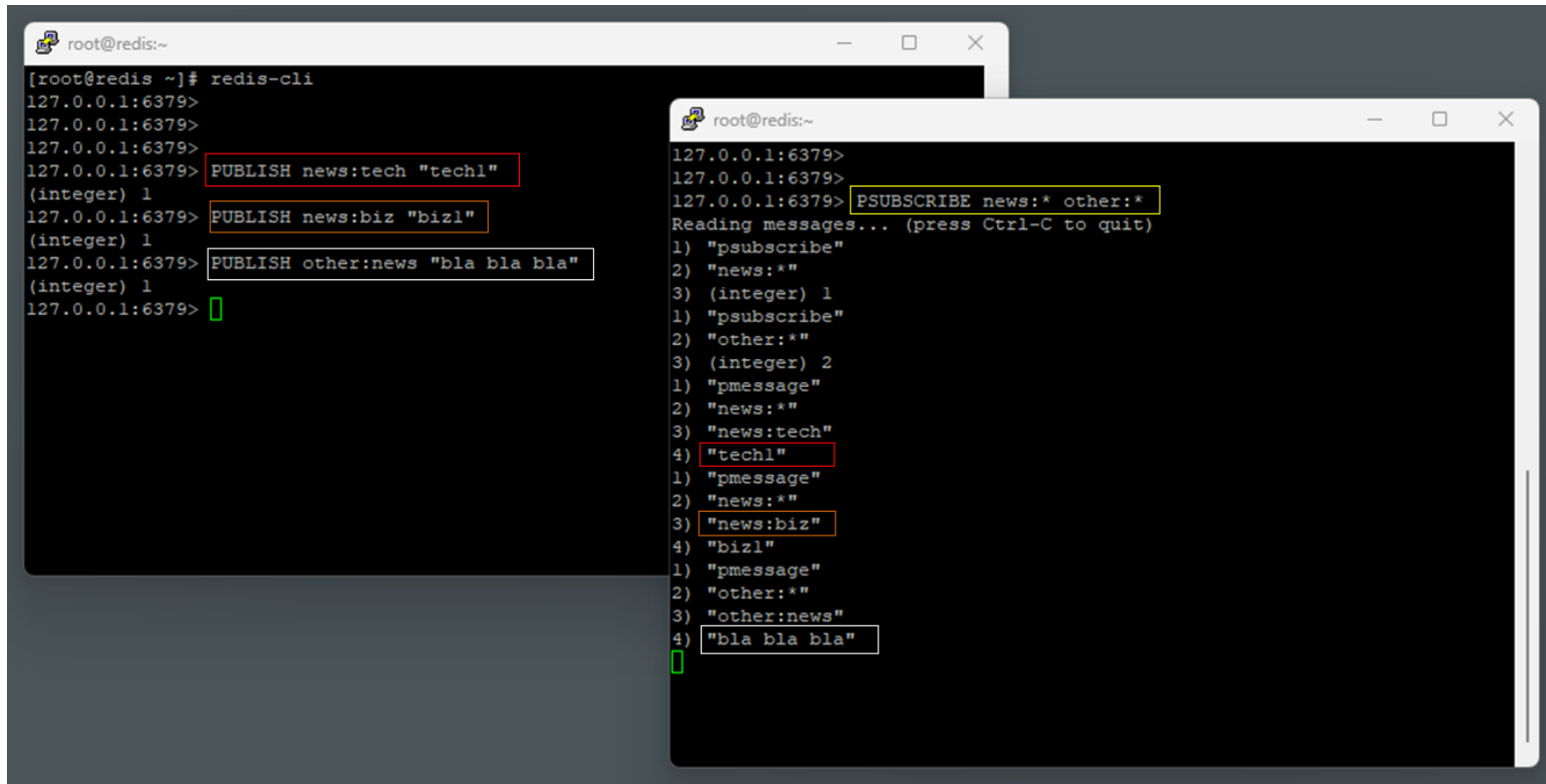
Pub / Sub : Pattern



Pub / Sub : Pattern

- **PSUBSCRIBE** pattern1 pattern2...

Abonne aux canaux contenant les patterns donnés



```
root@redis:~  
[root@redis ~]# redis-cli  
127.0.0.1:6379>  
127.0.0.1:6379>  
127.0.0.1:6379>  
127.0.0.1:6379> PUBLISH news:tech "tech1"  
(integer) 1  
127.0.0.1:6379> PUBLISH news:biz "biz1"  
(integer) 1  
127.0.0.1:6379> PUBLISH other:news "bla bla bla"  
(integer) 1  
127.0.0.1:6379>   
  
root@redis:~  
127.0.0.1:6379>  
127.0.0.1:6379>  
127.0.0.1:6379> PSUBSCRIBE news:* other:*  
Reading messages... (press Ctrl-C to quit)  
1) "psubscribe"  
2) "news:*"   
3) (integer) 1  
1) "psubscribe"  
2) "other:*"   
3) (integer) 2  
1) "pmessage"  
2) "news:*"   
3) "news:tech"  
4) "tech1"  
1) "pmessage"  
2) "news:*"   
3) "news:biz"  
4) "biz1"  
1) "pmessage"  
2) "other:*"   
3) "other:news"  
4) "bla bla bla"
```

Pub / Sub : Pattern

- `PSUBSCRIBE pattern1 pattern2 ...`

Supported glob-style patterns:

- `h?llo` subscribes to `hello`, `hallo` and `hxlllo`
- `h*llo` subscribes to `hllo` and `heeeello`
- `h[ae]llo` subscribes to `hello` and `hallo`, but not `hillo`

Pub / Sub : Pattern

- **PUBSUB** channels [pattern1]

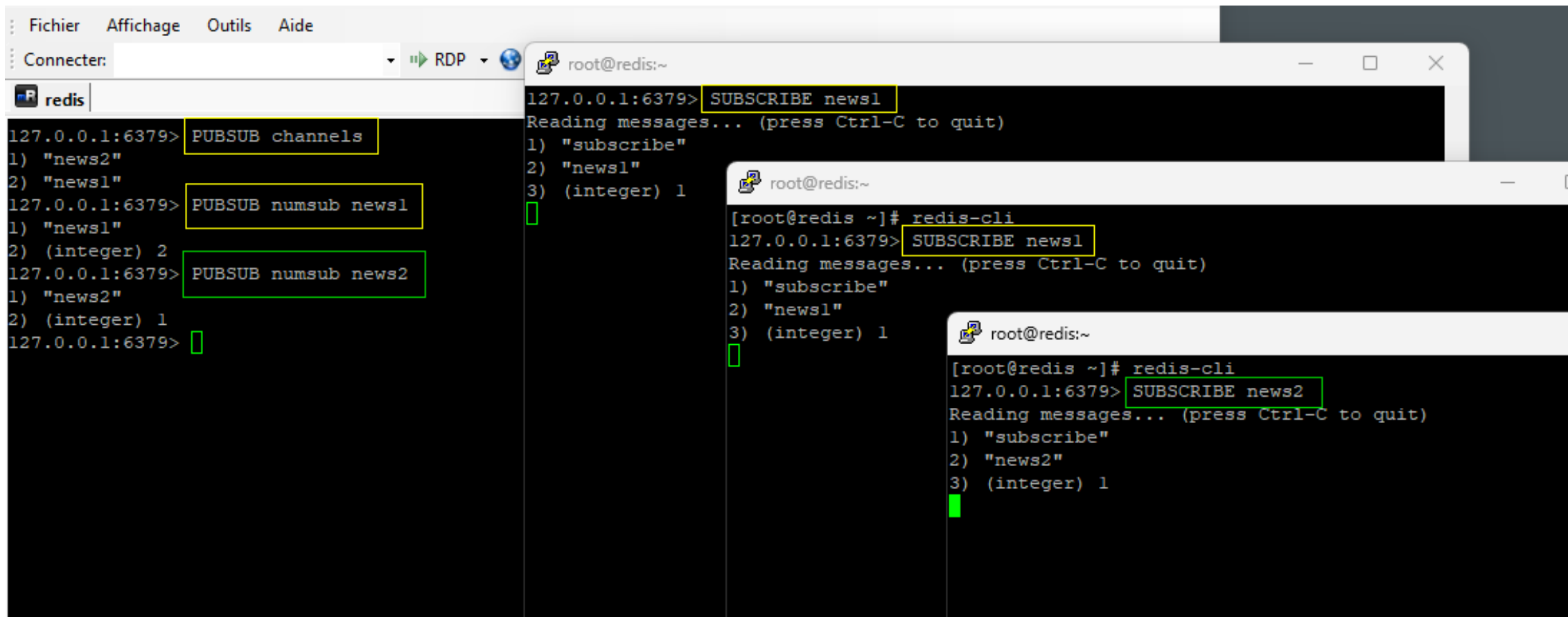
Liste les canaux actuellement actifs.

Un canal actif = avec un ou plusieurs abonnés.

- **PUBSUB** numsub [channel [channel ...]]

Renvoie le nombre d'abonnés pour les canaux spécifiés.

Pub / Sub : Pattern



The screenshot shows three overlapping terminal windows demonstrating Redis Pub/Sub patterns. The leftmost window shows the 'PUBSUB channels' command listing 'news1' and 'news2', and 'PUBSUB numsub news1' showing 2 subscribers. The middle window shows 'SUBSCRIBE news1' receiving 'subscribe', 'news1', and a count of 1. The rightmost window shows 'SUBSCRIBE news2' receiving 'subscribe', 'news2', and a count of 1.

```
Fichier Affichage Outils Aide
Connecter: RDP
redis
127.0.0.1:6379> PUBSUB channels
1) "news2"
2) "news1"
127.0.0.1:6379> PUBSUB numsub news1
1) "news1"
2) (integer) 2
127.0.0.1:6379> PUBSUB numsub news2
1) "news2"
2) (integer) 1
127.0.0.1:6379>

root@redis:~
127.0.0.1:6379> SUBSCRIBE news1
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "news1"
3) (integer) 1

[root@redis ~]# redis-cli
127.0.0.1:6379> SUBSCRIBE news1
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "news1"
3) (integer) 1

[root@redis ~]# redis-cli
127.0.0.1:6379> SUBSCRIBE news2
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "news2"
3) (integer) 1
```



Piping

Insertion de données en masse

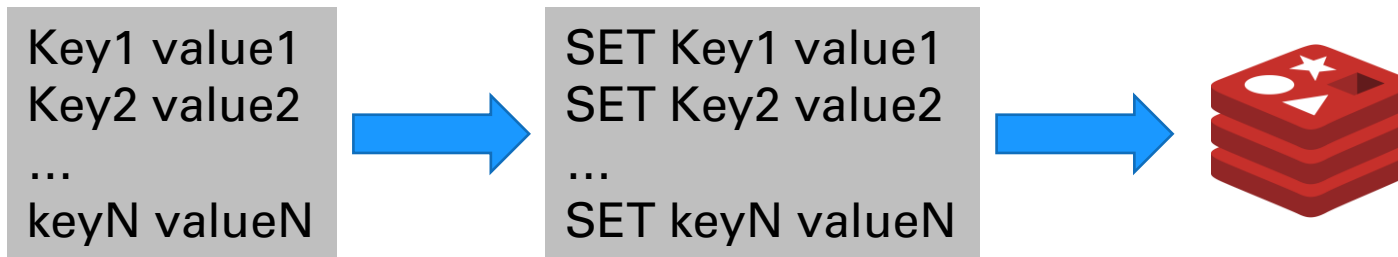
Insertion de données en masse

Pour insérer des données, on utilise la commande **SET KEY VALUE** :

```
SET key1 value1  
SET key2 value 2  
...  
SET keyN valueN
```

Comment faire avec des milliers (millions, ...) de données ?

Insertion de données en masse



Insertion de données en masse

```
[root@redis ~]#  
[root@redis ~]# cat codes | wc -l  
165  
[root@redis ~]# tail -7 codes  
UA,Ukraine  
UY,Uruguay  
UZ,Uzbekistan  
VU,Vanuatu  
YE,Yemen  
ZM,Zambia  
ZW,Zimbabwe  
[root@redis ~]# awk -F ',' '{print "SET data:"$1 " " $2}' codes | tail -7  
SET data:UA Ukraine  
SET data:UY Uruguay  
SET data:UZ Uzbekistan  
SET data:VU Vanuatu  
SET data:YE Yemen  
SET data:ZM Zambia  
SET data:ZW Zimbabwe  
[root@redis ~]# awk -F ',' '{print "SET data:"$1 " " $2}' codes | redis-cli --pipe  
All data transferred. Waiting for the last reply...  
Last reply received from server.  
errors: 0, replies: 165  
[root@redis ~]#  
[root@redis ~]#
```

```
[root@redis ~]# redis-cli  
127.0.0.1:6379>  
127.0.0.1:6379> KEYS *  
1) "data:SD"  
2) "data:GE"  
3) "data:SZ"  
4) "data:DZ"  
5) "data:LV"  
6) "data:MR"  
7) "data:TH"  
8) "data:NI"  
9) "data:BD"  
10) "data:BJ"  
11) "data:JO"
```

```
127.0.0.1:6379>  
127.0.0.1:6379> GET data:SD  
"Sudan"  
127.0.0.1:6379> GET data:IT  
"Italy"  
127.0.0.1:6379> GET data:PA  
"Panama"  
127.0.0.1:6379>  
127.0.0.1:6379>
```

+

•



Design

des bases de données

Design : Migration RDBMS → Redis



+

•

- Ne pas hésiter à générer un très grand nombre de keys (limite $\sim 2^{32}$)
- Chaque ligne de la table peut être stockée dans une key
- Utiliser les structures de données **Hashes**
- Les **noms des keys** peuvent se baser sur les **clés primaires** avec « : » en séparateur
- Le reste des champs forme les éléments des Hashes
- Les requêtes de recherches utilisent les commandes des Hashes : HSET, HGET, ...

Design : Migration RDBMS → Redis



<u>Salariées</u>
salarié_id nom prenom age date_embauche departement salaire

salarié_id	nom	prenom	age	date_embauche	departement	salaire
1	nom1	prenom1	age1	date_embauche1	departement1	salaire1
2	nom2	prenom2	age2	date_embauche2	departement2	salaire2
3	nom3	prenom3	age3	date_embauche3	departement3	salaire3

Design : Migration RDBMS → Redis

```
127.0.0.1:6379>
127.0.0.1:6379> HMSET salarie_1 nom nom1 prenom prenom1 age age1 date_embauche date_embauche1 departement departement1 salaire salaire1
OK
127.0.0.1:6379> HMSET salarie_2 nom nom2 prenom prenom2 age age2 date_embauche date_embauche2 departement departement2 salaire salaire2
OK
127.0.0.1:6379> HMSET salarie_3 nom nom3 prenom prenom3 age age3 date_embauche date_embauche3 departement departement3 salaire salaire3
OK
127.0.0.1:6379> HGETALL salarie_2
1) "nom"
2) "nom2"
3) "prenom"
4) "prenom2"
5) "age"
6) "age2"
7) "date_embauche"
8) "date_embauche2"
9) "departement"
10) "departement2"
11) "salaire"
12) "salaire2"
127.0.0.1:6379> HMGET salarie_2 nom prenom age salaire
1) "nom2"
2) "prenom2"
3) "age2"
4) "salaire2"
127.0.0.1:6379> █
```

Design : Migration RDBMS → Redis

Cas des : clés primaires composées

```
Table product {  
  order_id integer [primary key]  
  product_id integer [primary key]  
  unit_price real  
  quantity smallint  
  discount real  
}
```

product	
order_id	integer
product_id	integer
unit_price	real
quantity	smallint
discount	real

Table: product

INDEXES:
(order_id, product_id)

order_id	product_id	unit_price	quantity	discount
1200	351	200,00	20	0,00
1200	759	1245,50	12	0,00

Design : Migration RDBMS → Redis

Cas des : clés primaires composées

+

•

```
127.0.0.1:6379>
127.0.0.1:6379> HMSET order:product:1200:351 unit_price 200 quantity 20 discount 0
OK
127.0.0.1:6379> HMSET order:product:1200:759 unit_price 1245,50 quantity 12 discount 0
OK
127.0.0.1:6379>
127.0.0.1:6379> HGETALL order:product:1200:351
1) "unit_price"
2) "200"
3) "quantity"
4) "20"
5) "discount"
6) "0"
127.0.0.1:6379> keys order:product:1200:*
1) "order:product:1200:351"
2) "order:product:1200:759"
127.0.0.1:6379> keys order:product:*:351
1) "order:product:1200:351"
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> keys order:product:*:351
1) "order:product:1200:351"
2) "order:product:1200:759"
127.0.0.1:6379> █
```



Administration

Administration : Dump & Restore des keys

- **PDUMP** key :

Sérialise la valeur stockée dans key dans un format spécifique à Redis et la renvoie à l'utilisateur.

La valeur renvoyée peut être restaurée à nouveau dans une clé Redis à l'aide de la commande :

- **RESTORE** key ttl serialized-value

```
127.0.0.1:6379>
127.0.0.1:6379> set name Dupond
OK
127.0.0.1:6379>
127.0.0.1:6379> get name
"Dupond"
127.0.0.1:6379>
127.0.0.1:6379> DUMP name
"\x00\x06Dupond\t\x00\xfd\xfb\x8b%\x8d\x83\x82\xbe"
127.0.0.1:6379>
127.0.0.1:6379> DEL name
(integer) 1
127.0.0.1:6379>
127.0.0.1:6379> get name
(nil)
127.0.0.1:6379>
127.0.0.1:6379> RESTORE name 0 "\x00\x06Dupond\t\x00\xfd\xfb\x8b%\x8d\x83\x82\xbe"
OK
127.0.0.1:6379>
127.0.0.1:6379> get name
"Dupond"
127.0.0.1:6379>
127.0.0.1:6379>
```

Administration : Info

- **PINFO** section :
Renvoi des informations et des statistiques sur le serveur.

Sections :

```
# Server
# Clients
# Memory
# Persistence
# Stats
# Replication
# CPU
# Modules
# Errorstats
# Cluster
# Keyspace
```

```
127.0.0.1:6379>
127.0.0.1:6379> info server
# Server
redis_version:6.2.12
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:7db4903a77bc953b
redis_mode:standalone
os:Linux 3.10.0-1160.95.1.el7.x86_64 x86_64
arch_bits:64
monotonic_clock:POSIX clock_gettime
multiplexing_api:epoll
atomicvar_api:atomic-builtin
gcc_version:4.8.5
process_id:912
process_supervised:no
run_id:2225894dd06108ed2bfaff27a974e87d90e8c1fb
tcp_port:6379
server_time_usec:1697703289440036
uptime_in_seconds:2990
uptime_in_days:0
hz:10
configured_hz:10
lru_clock:3204473
executable:/usr/local/bin/redis-server
config_file:/etc/redis/redis.conf
io_threads_active:0
127.0.0.1:6379>
```

Administration : Historiques des commandes

- Le fichier de l'historique est sous la « home directory » de l'utilisateur

```
[root@redis ~]# ll -altr
total 44
-rw-r--r--. 1 root root 129 Dec 29 2013 .tcshrc
-rw-r--r--. 1 root root 100 Dec 29 2013 .cshrc
-rw-r--r--. 1 root root 176 Dec 29 2013 .bashrc
-rw-r--r--. 1 root root 176 Dec 29 2013 .bash_profile
-rw-r--r--. 1 root root 18 Dec 29 2013 .bash_logout
-rw-----. 1 root root 1259 Sep 8 09:49 anaconda-ks.cfg
drwxr-xr-x. 3 root root 18 Sep 8 09:53 .cache
drwxr-xr-x. 3 root root 18 Sep 8 09:53 .config
dr-xr-xr-x. 17 root root 244 Sep 8 10:01 ..
-rw-----. 1 root root 7109 Oct 15 00:33 .bash_history
-rw-----. 1 root root 4915 Oct 19 10:07 .viminfo
dr-xr-x---. 4 root root 205 Oct 19 10:07 .
-rw-----. 1 root root 2250 Oct 19 10:14 .rediscli_history
[root@redis ~]#
[root@redis ~]#
[root@redis ~]# head -10 .rediscli_history
PUBSUB channels
PUBSUB channels *
PUBSUB numsub *
PUBSUB numsub news*
PUBSUB numsub news1
PUBSUB channels
PUBSUB channels top*
PUBSUB channels top
PUBSUB channels
PUBSUB numsub news1
[root@redis ~]#
[root@redis ~]#
```

Administration : Scan keys avec redis-cli depuis Shell

DEBUG POPULATE nombre :

Alimente rapidement une instance Redis avec des données de test fictives qui s'avèrent utiles pendant le développement.

```
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> DEBUG populate 100
OK
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
[root@redis ~]#
[root@redis ~]#
[root@redis ~]# redis-cli --scan --pattern 'key:3*'
"key:39"
"key:30"
"key:38"
"key:32"
"key:35"
"key:37"
"key:3"
"key:34"
"key:33"
"key:36"
"key:31"
[root@redis ~]#
[root@redis ~]#
```


Administration : keys et values depuis Bash

+

•

```
#!/bin/bash
redis-cli --scan > all.keys
while read -r key
do
    value=$(redis-cli get "$key")
    echo $key '|' $value
done < all.keys
```

```
[root@redis ~]# ./get_keys_and_values.sh | grep ^key:5
key:58 | value:58
key:5 | value:5
key:55 | value:55
key:50 | value:50
key:54 | value:54
key:59 | value:59
key:53 | value:53
key:56 | value:56
key:52 | value:52
key:51 | value:51
key:57 | value:57
[root@redis ~]#
```



Clients Redis

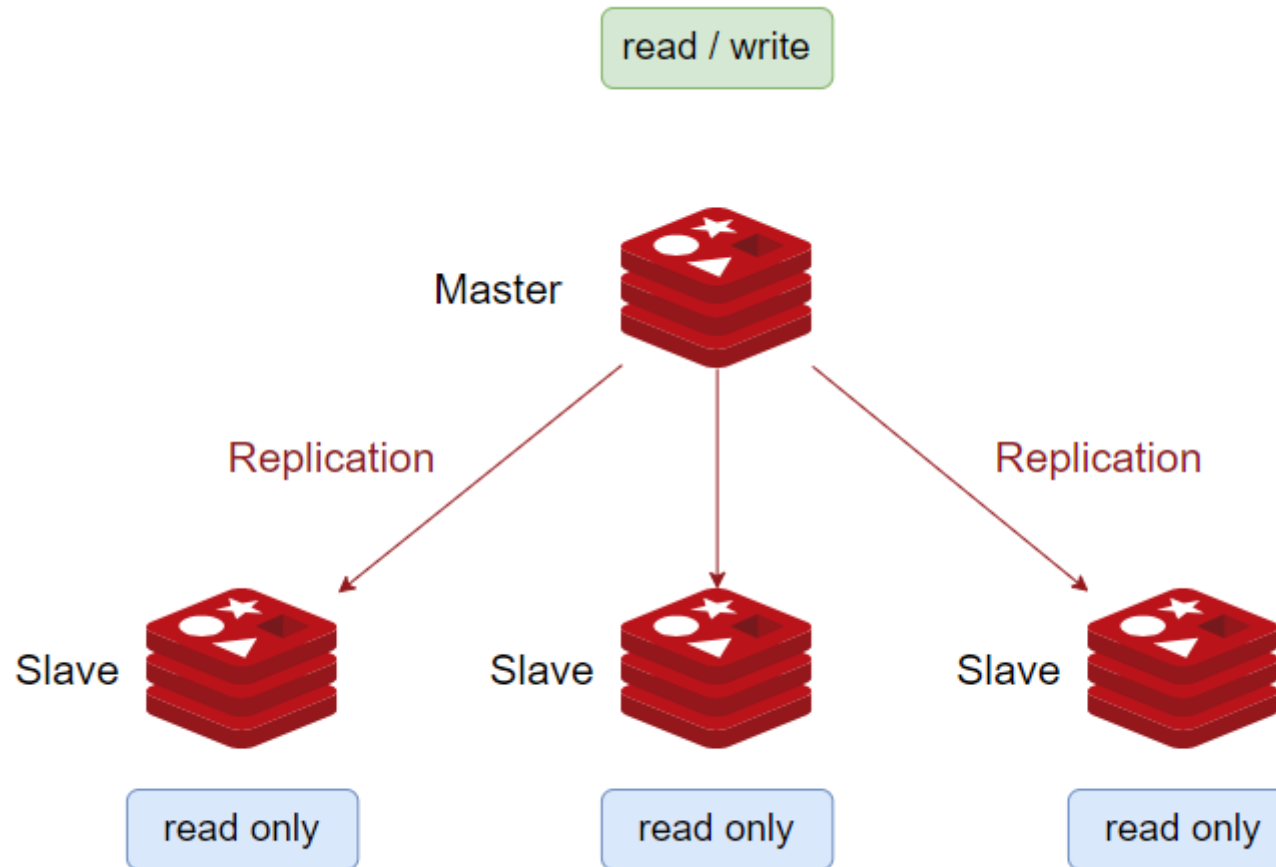
Clients Redis : Python

- Voir TP 3



Réplication

Réplication : principe



Réplication : Mise en place

- Voir TP 4

Commandes à connaître :

- **INFO** [section]
- **ROLE**



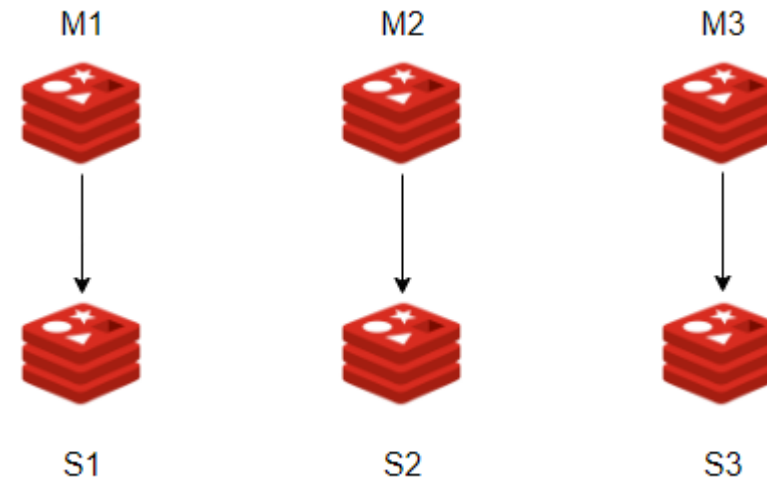
Redis Cluster

Cluster

- Redis Cluster est une configuration Redis dans laquelle les données sont automatiquement réparties sur plusieurs nœuds Redis.

Cluster = multiples Shards Redis

- Il permet de poursuivre les opérations lorsque certains nœuds tombent en panne ou sont incapables de communiquer.
- Il permet d'avoir une certaine disponibilité.
- Améliore les performances grâce au Sharding
- Scalabilité : jusqu'à 1000 nœuds
- Si un Master tombe son Slave devient automatiquement Master



+

•

Cluster : fonctionnement

- Chaque nœud utilise 2 ports :
 - 1 pour échanger avec les clients
 - 1 pour échanger avec les nœuds du cluster
- Chaque Master Node se voit attribuer un hash slot (une part des données) du cluster
- Un cluster contient 16384 hashes repartis sur les différents nodes utilisés
- **PI** : $\text{HASH_SLOT} = \text{CRC16}(\text{key}) \bmod 16384$
- Avec 3 nodes A, B, C:
 - A : slots de 0 à 5500.
 - B : slots de 5501 à 11000.
 - C : slots de 11001 à 16383.



Cluster : Réalisation

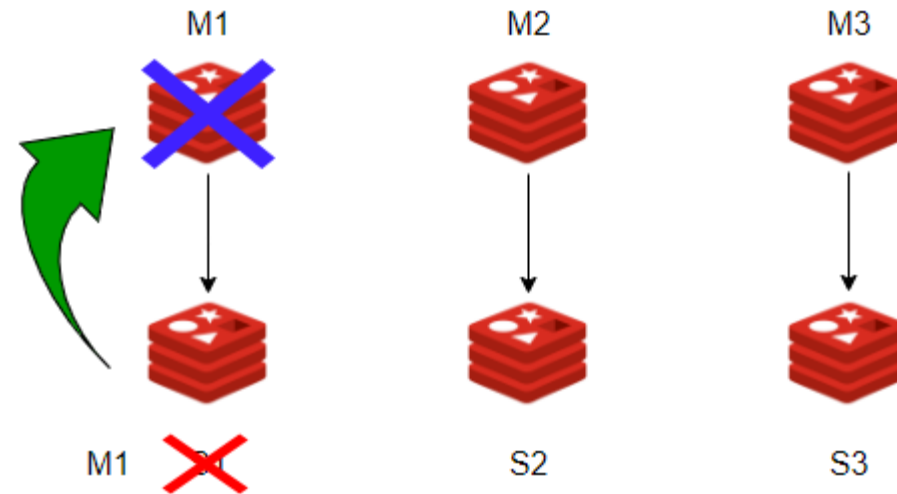
- Création du cluster :

```
redis-cli --cluster create IP1:PORT1 IP2:PORT2 IP3:PORT3 IP4:PORT4 IP5:PORT5 IP6:PORT6 --cluster-replicas 1
```

- Voir TP 5

Cluster : Haute disponibilité

- Si un master tombe, son Slave reprend sa place dans le cluster
- Voir TP6



Cluster : Haute disponibilité

```
127.0.0.1:7001> CLUSTER info
cluster_state:ok
cluster_slots_assigned:16384
cluster_slots_ok:16384
cluster_slots_pfail:0
cluster_slots_fail:0
cluster_known_nodes:6
cluster_size:3
cluster_current_epoch:9
cluster_my_epoch:8
cluster_stats_messages_ping_sent:966
cluster_stats_messages_pong_sent:986
cluster_stats_messages_fail_sent:6
cluster_stats_messages_update_sent:1
cluster_stats_messages_sent:1959
cluster_stats_messages_ping_received:986
cluster_stats_messages_pong_received:962
cluster_stats_messages_fail_received:6
cluster_stats_messages_received:1954
127.0.0.1:7001>
127.0.0.1:7001>
```

Quelques commandes

```
127.0.0.1:7001>
127.0.0.1:7001> cluster myid
"dfd393b9ceel302f27ac0dd443d4bc3f19859d35"
127.0.0.1:7001>
```

```
127.0.0.1:7001> CLUSTER SLOTS
1) 1) (integer) 0
   2) (integer) 5460
   3) 1) "127.0.0.1"
      2) (integer) 7001
      3) "dfd393b9ceel302f27ac0dd443d4bc3f19859d35"
   4) 1) "127.0.0.1"
      2) (integer) 7006
      3) "9ff6aa0f7c512470f4a28f823balfd1b9d07a479"
2) 1) (integer) 5461
   2) (integer) 10922
   3) 1) "127.0.0.1"
      2) (integer) 7004
      3) "6db399ada15c23b25d695de35d1d41d40e55e19b"
   4) 1) "127.0.0.1"
      2) (integer) 7002
      3) "a64ed5fac0691357a8a5b7f1987cf97bac20c040"
3) 1) (integer) 10923
   2) (integer) 16383
   3) 1) "127.0.0.1"
      2) (integer) 7003
      3) "6a09270e1494a0445d9cfb6460aa33258624e795"
   4) 1) "127.0.0.1"
      2) (integer) 7005
      3) "bfd1f9baa7d6fa8bcd0e2f49e670c6c557bd2d"
127.0.0.1:7001>
```

```
127.0.0.1:7001> cluster nodes
dfd393b9ceel302f27ac0dd443d4bc3f19859d35 127.0.0.1:7001@17001 myself,master - 0 1698412789000 8 connected 0-5460
bfd1f9baa7d6fa8bcd0e2f49e670c6c557bd2d 127.0.0.1:7005@17005 slave 6a09270e1494a0445d9cfb6460aa33258624e795 0 1698412791942 3 connected
6a09270e1494a0445d9cfb6460aa33258624e795 127.0.0.1:7003@17003 master - 0 1698412791000 3 connected 10923-16383
a64ed5fac0691357a8a5b7f1987cf97bac20c040 127.0.0.1:7002@17002 slave 6db399ada15c23b25d695de35d1d41d40e55e19b 0 1698412791000 9 connected
6db399ada15c23b25d695de35d1d41d40e55e19b 127.0.0.1:7004@17004 master - 0 1698412791000 9 connected 5461-10922
9ff6aa0f7c512470f4a28f823balfd1b9d07a479 127.0.0.1:7006@17006 slave dfd393b9ceel302f27ac0dd443d4bc3f19859d35 0 1698412791539 8 connected
127.0.0.1:7001>
```

```
127.0.0.1:7001>
127.0.0.1:7001> cluster replicas dfd393b9ceel302f27ac0dd443d4bc3f19859d35
1) "9ff6aa0f7c512470f4a28f823balfd1b9d07a479 127.0.0.1:7006@17006 slave dfd393b9ceel302f27ac0dd443d4bc3f19859d35 0 1698412908000 8 connected"
127.0.0.1:7001>
127.0.0.1:7001>
```



RediSearch

RediSearch

Avec Redis :

SET user1 value1 GET user1 OK

HSET user1 name John HGETALL user1 name OK

MAIS

GET users WHERE name = "John" KO!

GET users WHERE name LIKE "Jo*" KO!

Redis n'est pas adapté pour faire ce genre de requêtes

La solution : RediSearch

RediSearch

- Moteur :
 - ☐ d'indexation
 - ☐ de recherche Full-Text
- Prend en charge
 - ☐ les requêtes multi-champs
 - ☐ les requêtes avec conditions booliennes : AND, OR, NOT
 - ☐ filtres numériques et ranges
 - ☐ agrégation des données

RediSearch offre des fonctionnalités SQL like à Redis

RediSearch : Installation

- Voir TP 7

RediSearch : Création d'index

- **Prérequis** : keys déjà insérées dans Redis

FT.CREATE index **ON HASH PREFIX COUNT** "key" **SCHEMA** field_name type [field_name type ...]

- Index : nom de l'index
- ON HASH : format des keys
- PREFIX : des noms des keys à indexer
- COUNT : nombre de PREFIX
- SCHEMA : les champs à indexer avec leurs types : text, numeric, ...

Exemple : index de "titre", "date_publication" et "categories" d'un blog dont les "keys" commencent par blog:post:

FT.CREATE idx **ON HASH PREFIX** 1 blog:post: **SCHEMA** titre **TEXT SORTABLE** date_publication **NUMERIC SORTABLE** categories **TAG SORTABLE**

RediSearch : Gestion d'index

- **FT._LIST** : Liste des indexes existants
- **FT.INFO** index : informations sur l'index
- **FT.ALTER** index ... : Modification d'une propriété de l'index

Exemple : **FT.ALTER** idx:movie **SCHEMA ADD** plot **TEXT WEIGHT 0.5**

- **FT.DROPINDEX** index : suppression de l'index

La suppression de l'index n'affecte pas les données dans la base.

RediSearch : Requêtes de recherche

FT.SEARCH index "query"

index : nom de l'index concerné

query : requête de recherches

- Voir TP 8



RedisInsight



RedisInsight

- Interface utilisateur (GUI) d'interaction avec Redis
- Application desktop
- Support natif des modules redis
- Interface client intuitive
- Command helper
- Outils d'administration de Redis

+



RedisInsight : Installation & utilisation

- Téléchargement : <https://redis.com/fr/redis-enterprise/redisinsight/>
- Installation :
- Utilisation : voir TP 9



Sécurité

Sécurité :

- Un environnement sécurisé : Pas d'accès direct depuis l'extérieur sans intermédiaire
- « Protected mode » : Lorsqu'il est activé et que l'utilisateur par défaut n'a pas de mot de passe, le serveur n'accepte que les connexions locales depuis 127.0.0.1
- Sécurité réseau : par défaut seules les connexions locales sont autorisées
- Renommer les commandes sensibles dans redis.conf :
- Authentification : activée par la directive « **requirepass** motdepasse »

```
bind 127.0.0.1 -:::1
```

```
rename-command CONFIG ""
```

```
requirepass foobared
```


Sécurité : Authentification user/password

Authentification activée

```
[root@formation redis]# redis-cli
127.0.0.1:6379>
127.0.0.1:6379> keys *
(error) NOAUTH Authentication required.
127.0.0.1:6379>
127.0.0.1:6379> auth foobared
OK
127.0.0.1:6379> keys *
1) "key1"
127.0.0.1:6379>
127.0.0.1:6379>
[root@formation redis]#
[root@formation redis]# redis-cli -a foobared
Warning: Using a password with '-a' or '-u' option on the command line interface may not be safe.
127.0.0.1:6379>
127.0.0.1:6379> keys *
1) "key1"
127.0.0.1:6379>
127.0.0.1:6379>
[root@formation redis]#
[root@formation redis]# redis-cli --askpass
Please input password: *****
127.0.0.1:6379>
127.0.0.1:6379> KEYS *
1) "key1"
127.0.0.1:6379>
127.0.0.1:6379> █
```

+

•

**Merci
et
bonne continuation**