Maintenance et résolution des problèmes dans un cluster Mongodb

Compte tenu de l'architecture mise en place et du nombre plus élevé de démons qui entrent en compte dans le fonctionnement d'un cluster Mongodb que dans une installation classique, la maintenance et la résolution des problèmes peut être plus complexes.

Cette page recense et détaille les opérations de maintenance et des recherches/résolutions de problèmes courants

- Vérification de l'état d'un cluster Mongodb
  - Services Linux
  - Console Mongodb
  - Checks Shinken
- Manipulation d'un cluster Mongodb
  - Redémarrage du cluster
- Résolution des problèmes
  - Erreurs lors du redémarrage des démons
  - Resynchronisation manuelle d'un noeud Mongodb
    - Symptomes
    - Explications et résolution
      - Méthode 1: Resynchronisation manuelle par copie de fichiers
      - Méthode 2: Resynchronisation automatique par réinitialisation du nœud

Vérification de l'état d'un cluster Mongodb

La vérification de l'état d'un cluster Mongo peut se faire de plusieurs manières:

- En vérifiant l'état des services Linux et les logs
- Directement via la console Mongodb
- Par les checks Mongodb livrés avec Shinken Entreprise

Services Linux

Avant de vérifier l'état détaillé du cluster Mongodb avec la console Mongodb ou les checks Shinken, on peut d'abord commencer par vérifier sur chaque nœud, l'état des services Linux associés aux différents démons Mongo.

La syntaxe à utiliser pour la vérification de ces services dépend de la version de CentOS utilisée:

CentOS 6

```
$ service mongod status
mongod (pid 2498) is
running...

$ service mongo-configsrv
status
mongod (pid 2593 2498) is
running...

$ service mongos status
mongos (pid 2690) is
running...
```

CentOS 7

```
$ systemctl status mongod
? mongod.service - SYSV: Mongo is a scalable, document-oriented database.
   Loaded: loaded (/etc/rc.d/init.d/mongod; bad; vendor preset: disabled)
   Active: active (running) since Tue 2019-10-22 16:39:52 CEST; 6min ago
     Docs: man:systemd-sysv-generator(8)
  Process: 908 ExecStart=/etc/rc.d/init.d/mongod start (code=exited,
status=0/SUCCESS)
 Main PID: 1037 (mongod)
   CGroup: /system.slice/mongod.service
           ??1037 /usr/bin/mongod -f /etc/mongod.conf --smallfiles

Oct 22 16:39:48 lab-mongo1 systemd[1]: Starting SYSV: Mongo is a scalable,
document-oriented database....
Oct 22 16:39:48 lab-mongo1 runuser[979]: pam_unix(runuser:session):
session opened for user mongod by (uid=0)
Oct 22 16:39:52 lab-mongo1 runuser[979]: pam_unix(runuser:session):
session closed for user mongod
Oct 22 16:39:52 lab-mongo1 mongod[908]: Starting mongod: [  OK  ]
Oct 22 16:39:52 lab-mongo1 systemd[1]: Started SYSV: Mongo is a scalable,
document-oriented database..

$ systemctl status mongo-configsrv
? mongo-configsrv.service - SYSV: Mongo is a scalable, document-oriented
database.
   Loaded: loaded (/etc/rc.d/init.d/mongo-configsrv; bad; vendor preset:
disabled)
   Active: active (running) since Tue 2019-10-22 16:39:51 CEST; 6min ago
     Docs: man:systemd-sysv-generator(8)
  Process: 906 ExecStart=/etc/rc.d/init.d/mongo-configsrv start
(code=exited, status=0/SUCCESS)
 Main PID: 1034 (mongod)
   CGroup: /system.slice/mongo-configsrv.service
           ??1034 /usr/bin/mongod -f /etc/mongo-configsrv.conf --smallfiles

Oct 22 16:39:48 lab-mongo1 systemd[1]: Starting SYSV: Mongo is a scalable,
document-oriented database....
Oct 22 16:39:48 lab-mongo1 runuser[975]: pam_unix(runuser:session):
session opened for user mongod by (uid=0)
Oct 22 16:39:51 lab-mongo1 runuser[975]: pam_unix(runuser:session):
session closed for user mongod
Oct 22 16:39:51 lab-mongo1 mongo-configsrv[906]: Starting mongod: [  OK  ]
Oct 22 16:39:51 lab-mongo1 systemd[1]: Started SYSV: Mongo is a scalable,
document-oriented database..

$ systemctl status mongos
? mongos.service - SYSV: Mongo is a scalable, document-oriented database.
   Loaded: loaded (/etc/rc.d/init.d/mongos; bad; vendor preset: disabled)
   Active: active (running) since Tue 2019-10-22 16:47:29 CEST; 28s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 2110 ExecStart=/etc/rc.d/init.d/mongos start (code=exited,
status=0/SUCCESS)
 Main PID: 2127 (mongos)
   CGroup: /system.slice/mongos.service
           ??2127 /usr/bin/mongos -f /etc/mongos.conf

Oct 22 16:46:47 lab-mongo1 systemd[1]: Starting SYSV: Mongo is a scalable,
document-oriented database....
Oct 22 16:46:47 lab-mongo1 runuser[2123]: pam_unix(runuser:session):
session opened for user mongod by (uid=0)
Oct 22 16:47:29 lab-mongo1 mongos[2110]: Starting mongos: [  OK  ]
Oct 22 16:47:29 lab-mongo1 systemd[1]: Started SYSV: Mongo is a scalable,
document-oriented database..
```

Lorsqu'un démon du cluster n'est pas correctement démarré, on peut commencer les recherches en analysant les logs de chaque démon du cluster Mongodb.

Sur chaque noeud du cluster, on trouve les logs suivants:

- Démon mongod: */var/log/mongodb/mongod.log*
- Démon mongo-configsrv: */var/log/mongodb/mongo-configsrv.log*
- Démon mongos: */var/log/mongodb/mongos.log*

Console Mongodb

La console Mongodb permet de récupérer des informations détaillées sur l'état de chaque démon ainsi que leur statut au sein du cluster (primaire, secondaire, en récupération, injoignable etc...).

Pour accéder à la console Mongodb, il se connecter depuis n'importe quel noeud du cluster sur le démon "mongod" avec la commande suivante:

(commande shell)

```
mongo --port 27018
```

Une fois la console Mongo lancée, on se retrouve avec un invite de commande différent, qui nous permet de vérifier directement l'état du cluster:

(console mongodb)

```
rs.status()
```

On obtient le résultat suivant:

```
rs-shinken:SECONDARY> rs.status();
{
    "set" : "rs-shinken",
     "date" : ISODate("2018-05-04T07:29:15Z"),
    "myState" : 2,
    "syncingTo" : "node2:27018",
    "members" : [
         {
              "_id" : 0,
              "name" : "node1:27018",
              "health" : 0,
               "state" : 8,
               "stateStr" : "(not reachable/healthy)",
               "uptime" : 0,
               "optime" : Timestamp(0, 0),
               "optimeDate" : ISODate("1970-01-01T00:00:00Z"),
               "lastHeartbeat" : ISODate("2018-05-04T07:29:13Z"),
               "lastHeartbeatRecv" : ISODate("1970-01-01T00:00:00Z"),
               "pingMs" : 0
        },
         {
               "_id" : 1,
               "name" : "node2:27018",
               "health" : 1,
               "state" : 1,
              "stateStr" : "PRIMARY",
              "uptime" : 81988,
               "optime" : Timestamp(1525250222, 106),
               "optimeDate" : ISODate("2018-05-02T08:37:02Z"),
               "lastHeartbeat" : ISODate("2018-05-04T07:29:14Z"),
               "lastHeartbeatRecv" : ISODate("2018-05-04T07:29:14Z"),
               "pingMs" : 1,
               "electionTime" : Timestamp(1525336975, 1),
              "electionDate" : ISODate("2018-05-03T08:42:55Z")
         },
        {
               "_id" : 2,
               "name" : "node3:27018",
               "health" : 1,
               "state" : 2,
               "stateStr" : "SECONDARY",
              "uptime" : 81988,
               "optime" : Timestamp(1525250222, 106),
               "optimeDate" : ISODate("2018-05-02T08:37:02Z"),
              "self" : true
        }
        ],
        "ok" : 1
}
```

Avec l'affichage ci-dessus, on tire par exemple les informations suivantes sur notre cluster:

- Le cluster comporte 3 nœuds (*node1*, *node2*, *node3*)
- Le nœud *node2* est le nœud primaire, et le nœuds *node3* est un nœud secondaire
- Le noeud *node1* n'est pas joignable
- Le noeud *node2* a été élu comme noeud primaire le 03 Mar 2018 à 07h29 (GMT)

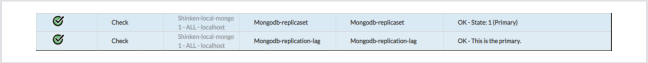
Checks Shinken

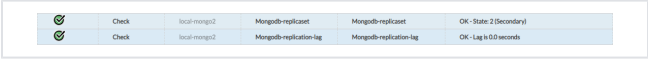
Un cluster Mongo peut être supervisé avec Shinken. Shinken Entreprise met à votre disposition un Pack MongoDB

Le modèle d'hôte "**mongodb**" prend également en compte les aspects de réplication de la base avec les checks "*Mongodb-replicaset*" et "*Mongodb-replication-lag*".

En pratique, pour superviser un cluster Mongo avec Shinken Entreprise, on associe un hôte Shinken à un nœud du cluster. Chaque hôte aura le modèle "**mongodb**" accroché, ce qui permet de superviser ces 3 nœuds de manière indépendante. On effectue également la supervision sur le port **27018** au lieu du port 27017 utilisé par défaut.

Voici un aperçu du résultat des checks concernant la réplication de Mongo, pour un nœud primaire et pour un nœud secondaire:





Le modèle **mongodb** permet d'utiliser un tunnel SSH pour la connexion aux serveurs et l'exécution des checks. L'utilisation du tunnel SSH se fait en modifiant la donnée MONGO_CONNECTION_METHOD. Précisez la valeur "ssh" au lieu de "direct".

L'utilisateur et la clé SSH utilisés pour créer ce tunnel peuvent se configurer en modifiant les données MONGO_SSH_USER et MONGO_SSH_KEY sur l'hôte.

Le modèle **mongodb** se connecte aux démons mongod pour effectuer les vérifications sur le cluster Mongo. Dans une installation classique, ce démon utilise le port 27017.

Dans le cas du cluster, on a modifié le port que ce démon utilise pour utiliser le port 27018. Il faut donc également modifier le port utilisé dans Shinken en modifiant la donnée MONGO_PORT en 27018 sur l'hôte.

La configuration et le fonctionnement du pack MongoDB sont décrits en détail sur la page de documentation du Pack MongoDB.

Manipulation d'un cluster Mongodb

Redémarrage du cluster

Les démons du cluster Mongo sont dépendants les uns des autres et doivent être démarrés dans un ordre particulier pour pouvoir fonctionner.

Pour rappel, chaque noeud du cluster contient les démons Mongo suivants:

- **mongod**: responsable du stockage des données
- **mongo-configsrv**: gère la configuration des éléments du cluster
- **mongos**: Sert d'interface à Shinken. Route les requêtes reçues à l'intérieur du cluster Mongo

L'ordre de démarrage des démons est donc le suivant: mongod mongo-configsrv mongos

- On démarre le démon mongod en premier, qui s'occupe de stocker les données.
- On démarre ensuite le démon mongo-configsrv
- On démarre enfin le démon mongos, qui a besoin du démon mongo-configsrv pour pouvoir démarrer

La séquence de démarrage est donc la suivante:

- Sur tous les noeuds du cluster, on démarre **mongod**:

```
# CentOS 6
service mongod start

# CentOS 7
systemctl start mongod
```

- Sur tous les noeuds, on démarre ensuite **mongo-configsrv**:

```
# CentOS 6
service mongo-configsrv start

# CentOS 7
systemctl start mongo-configsrv
```

- Enfin, on démarre sur chaque noeud le démon **mongos**:

```
# CentOS 6
service mongos start

# CentOS 7
systemctl start mongos
```

L'extinction des démons du cluster peut se faire dans n'importe quel ordre. Le redémarrage des démons sur un seul nœud suit également le même ordre de démarrage.

Résolution des problèmes

Erreurs lors du redémarrage des démons

Lorsqu'un démon Mongo démarre, il écrit dans /var/run/mongodb un fichier .pid qui contient le PID du processus associé (/var/run/mongodb/mongod.pid pour le démon mongod, /var/run/mongodb/mongos pour le démon mongos et /var/run/mongodb/mongo-configsrv pour le démon mongo-configsrv). Les script de démarrage des démons Mongo appelés lors de la gestion des démons avec "*service*" ou "*systemctl*" se basent sur ces fichiers PID pour la manipulation du démon.

Ce mécanisme permet aux script d'inits livrés avec Mongo de redémarrer ou arreter les démons Mongo concernés.

Cependant, ce mécanisme a quelques failles:

- Lorsqu'un démon refuse de démarrer ou qu'il s'arrête de manière imprévue (crash ou tué par le système), le fichier PID associé peut ne pas être nettoyé et être toujours présent alors que le démon n'est plus en fonctionnement
- Lors du démarrage d'un démon via les script d'init (ou commandes *service*/*systemctl*), la présence d'un fichier PID empêche le démarrage du démon.

Dans ce cas, on voit dans les logs donnés par journalctl (CentOS 7) le message suivant (on prend ici mongos pour l'exemple):

```
-- Unit mongos.service has begun starting up.
Oct 23 10:22:34 mongo-primaire mongos[5174]: Error starting mongos. /var/run/mongodb/mongos.pid exists.
Oct 23 10:22:34 mongo-primaire systemd[1]: mongos.service: control process exited, code=exited status=1
Oct 23 10:22:34 mongo-primaire systemd[1]: Failed to start SYSV: Mongo is a scalable, document-oriented
database..
-- Subject: Unit mongos.service has failed
-- Defined-By: systemd
-- Support: http://lists.freedesktop.org/mailman/listinfo/systemd-devel
--
-- Unit mongos.service has failed.
--
-- The result is failed.
Oct 23 10:22:34 mongo-primaire systemd[1]: Unit mongos.service entered failed state.
Oct 23 10:22:34 mongo-primaire systemd[1]: mongos.service failed.
Oct 23 10:22:34 mongo-primaire polkitd[631]: Unregistered Authentication Agent for unix-process:5168:1290264
(system bus name :1.1560, object path /org/freedesktop/PolicyKit1/AuthenticationA
```

Sous CentOS 6, l'affichage est plus sommaire mais remonte la même erreur (exemple pour mongod):

```
$ service mongod start
Error starting mongod. /var/run/mongodb/mongod.pid exists.
```

La résolution de ce problème se fait en 2 temps:

- La présence d'un fichier PID alors que le démon n'est pas démarré témoigne d'un problème au niveau du démon concerné. Il faut avant de pouvoir redémarrer le démon enquêter sur la cause d'extinction du démon. Pour cette recherche, on peut regarder aux endroits suivants:
    - Logs du démon, localisés dans /var/log/mongodb
    - Logs du système: /var/log/messages ou bien avec la commande 'dmesg'
- Une fois le problème identifié et géré, on peut alors supprimer le fichier correspondant dans /var/run/mongodb et redémarrer le démon Mongo

Cette procédure est identique pour l'ensemble des démons qui constituent le cluster (mongod, mongos, mongo-configsrv).

⚠ Un fichier PID est également créé et non supprimé par Mongo lorsqu'un démon refuse de démarrer correctement, comme par exemple dans le cas d'une configuration de Mongo incorrecte. On a dans ce cas le comportement suivant:

- Le démon Mongo en question démarre. Il écrit un fichier PID et commence son processus de démarrage.
- Il détecte une erreur (configuration incorrecte par exemple) et s'arrête, sans supprimer le fichier PID

Dans ce cas, il faudra donc supprimer le fichier PID généré par le démon pour redémarrer le démon.

Resynchronisation manuelle d'un noeud Mongodb

Symptomes

Suite à une interruption sur un des nœuds du cluster, on peut observer les erreurs suivantes.

La première est qu'un des nœuds reste bloqué dans un état "RECOVERING", comme on peut le voir via les checks du pack Mongodb, ou bien via la console Mongodb avec la commande "rs.status()'. L'affichage qu'on peut avoir ressemblerait au suivant pour le nœud en erreur:

```
{
        "_id" : 4,
        "name" : "55.55.55.55:27017",
        "health" : 1,
        "state" : 3,
        "stateStr" : "RECOVERING",
        "uptime" : 502511,
        "optime" : {
                "t" : 1340841938000,
                "i" : 5028
        },
        "optimeDate" : ISODate("2012-06-28T00:05:38Z"),
        "lastHeartbeat" : ISODate("2012-08-22T22:47:00Z"),
        "pingMs" : 0,
        "errmsg" : "error RS102 too stale to catch up"
},
```

On peut également observer une erreur similaire dans les logs du démon **mongod** (/var/log/mongodb/mongod.log) sur le noeud en état RECOVERING:

```
replSet error RS102 too stale to catch up
```

Explications et résolution

Lors de l'utilisation de la réplication de données dans un cluster Mongodb, Mongo enregistre dans un log d'opérations (oplog) les différentes requêtes effectuées dans le cluster. Pour s'assurer que les données enregistrées soient bien synchronisées, chaque entrée du log d'opération est rejoué sur tous les autres nœuds. De cette manière, on s'assure que chaque nœud possède bien les mêmes données.

Lors d'une interruption sur un des nœuds du cluster, ce log d'opérations s'agrandit jusqu'à atteindre sa taille maximale. A ce moment la, les nouvelles requêtes effectuées sur le cluster sont écrites dans le log d'opérations mais un réécrivant les requêtes les plus anciennes.

Quand le nœud en erreur est remis en état et redémarré correctement, il ne peut pas se resynchroniser au niveau des données stockées puisqu'une partie des requêtes les plus anciennes a été écrasée. On observe dans ce cas l'erreur et les logs mentionnés précédemment:

- Le nœud démarre correctement mais reste bloqué en état "RECOVERING"
- L'erreur "*too stale to catch up*" apparaît dans les logs du démon mongod du nœud Mongo en question

Il faut donc arriver à resynchroniser manuellement les données stockées sur le nœud Mongo en erreur.

Pour ça, on a 2 solutions:

- Effectuer une copie manuelle des données depuis le nœud primaire (ou un autre nœud en état PRIMARY ou SECONDARY)
- Vider les données sur le nœud en erreur. Mongodb va considérer ce nœud comme un nouveau nœud venant d'être ajouté au cluster et va procéder automatiquement à la resynchronisation des données stockées sur le nœud.

Méthode 1: Resynchronisation manuelle par copie de fichiers

La procédure suivante correspond à la première solution, qui consiste à copier les données manuellement pour mettre à jour les données du nœud en RECOVERING.

> Par la suite, on appellera *noeud_reference* n'importe quel nœud du cluster qui contient des données à jour, et *noeud_recovery* le nœud avec les données à resynchroniser.

Cette procédure se décompose en 2 grandes étapes:

- On sauvegarde les données sur *noeud_reference*, qu'on copie sur *noeud_recovery*
- Sur *noeud_recovery*, on remplace les données existantes par celles du *noeud_reference* copiées précédemment

Cette mise à jour de données se fait de la manière suivante:

- **Sur le noeud *noeud_reference***: On commence par éteindre le démon Mongod qui s'occupe du stockage des données. On éteint le démon mongod pour éviter que les données soient modifiées pendant la copie.

```
# CentOS 6
service mongod stop
# CentOS 7
systemctl stop mongod
```

- **Sur le noeud *noeud_reference***: Une fois mongod éteint, on peut sauvegarder les données de mongod et les envoyer sur le nœud en RECOVERY.

```
# On crée une archive contenant les données de Mongo
tar -zcvf donnees_mongo_reference.tar.gz /var/lib/mongo/
# On envoie cette archive sur le noeud secondaire
scp donnees_mongo_reference.tar.gz noeud_recovery:~
```

- **Sur le nœud *noeud_recovery***: On éteint également mongod sur ce nœud pendant la copie des données

```
# CentOS 6
service mongod stop
# CentOS 7
systemctl stop mongod
```

- **Sur le nœud *noeud_recovery***: On extrait l'archive contenant les données copiées depuis *noeud_reference* puis on procède à la copie des données. Puisqu'on ne veut pas remplacer le contenu de */var/lib/mongo/configdb* (qui contient la configuration et les données du démon configsrv), on supprime ce dossier de l'archive extraite **avant** la copie

```
tar -xf donnes_mongo_reference.tar.gz
cd <dossier_extrait_par_larchive>
rm -rf configdb
cp -p ./* /var/lib/mongo
```

  Après copie des données, le dossier /var/lib/mongo devrait une structure similaire à la suivante (si le format de données est MMAPV1):

```
/var/lib/mongo
??? configdb/
??? journal/
??? local.0
??? local.1
??? local.2
??? local.3
??? local.4
??? local.ns
??? mongod.lock
??? shinken.0
??? shinken.ns
??? storage.bson
??? synchronizer.0
??? synchronizer.1
??? synchronizer.ns
```

Si le format des données dans mongod est WiredTiger, le dossier /var/lib/mongo devra avoir une structure similaire à celle ci:

```
/var/lib/mongo
??? collection-0-733002362619731958.wt
??? collection-1003--7335448947504265621.wt
??? collection-1010--5410065806332554216.wt
??? collection-1012--5410065806332554216.wt
??? collection-1014--5410065806332554216.wt
??? index-1004--7335448947504265621.wt
??? index-1011--5410065806332554216.wt
??? index-1013--5410065806332554216.wt
??? index-1015--5410065806332554216.wt
??? index-996--7335448947504265621.wt
??? journal/
??? _mdb_catalog.wt
??? mongod.lock
??? sizeStorer.wt
??? storage.bson
??? configdb/
??? WiredTiger
??? WiredTiger.basecfg
??? WiredTiger.lock
??? WiredTiger.turtle
??? WiredTiger.wt
```

- **Sur les noeuds *noeud_reference* et *noeud_recovery***: Une fois les données copiées, on peut redémarrer le démon mongod sur les 2 nœuds

```
# CentOS 6
service mongod start
# CentOS 7
systemctl start mongod
```

- On peut maintenant vérifier l'état du cluster Mongo via la console mongo (mongo --port 27018) avec la commande suivante:

```
rs.status()
```

Le nœud en recovery devrait maintenant être repassé dans sont état habituel (PRIMARY ou SECONDARY)

Méthode 2: Resynchronisation automatique par réinitialisation du nœud

La première méthode de resynchronisation consiste à copier manuellement les fichiers pour permettre au nœud en retard d'avoir des données à jour.

Le deuxième méthode consiste à vider les données du noeud noeud_recovery. Mongo va alors détecter ce noeud comme un nouveau noeud venant d'être ajouté au cluster et procède automatiquement à la synchronisation des données.

Cette méthode resynchronise l'ensemble du nœud depuis zéro. Elle peut par conséquent générer du trafic réseau et une utilisation de ressources systèmes plus importantes pendant la resynchronisation.

La procédure pour cette méthode est la suivante:

- **Sur le noeud *noeud_recovery***: On commence par éteindre le démon Mongod qui s'occupe du stockage des données. On éteint le démon mongod pour éviter que les données soient modifiées pendant la copie.

```
# CentOS 6
service mongod stop
# CentOS 7
systemctl stop mongod
```

- **Sur le noeud *noeud_recovery***: On supprime l'ensemble des données dans /var/lib/mongo, **à l'exception du dossier /var/lib/mongo/configdb** qui contient la configuration et données du démon mongo-configsrv.
- **Sur le noeud *noeud_recovery***: Une fois les données supprimées, on peut redémarrer le démon mongod

```
# CentOS 6
service mongod start
# CentOS 7
systemctl start mongod
```

- On peut maintenant vérifier l'état du cluster Mongo via la console mongo (mongo --port 27018) avec la commande suivante:

```
rs.status()
```

  Le nœud en RECOVERY va commencer sa resynchronisation. Il devrait à terme repasser dans sont état habituel (PRIMARY ou SECONDARY)