Collecteur de type cfg-file-import (format Shinken ou nagios)

Sommaire

Introduction

De nombreux composants de Shinken Entreprise peuvent être installés et configurés de manière à former une architecture hautement disponible. Il est notamment possible de mettre en place des démons Shinken de remplacement pour améliorer la disponibilité (*voir la page* Haute disponibilité des démons de Shinken Entreprise).

Cependant, un certain nombre de ces démons et leurs modules associés s'appuient sur une base MongoDB pour fonctionner. Pour améliorer la résistance aux pannes de la plateforme Shinken dans son ensemble, on peut également s'assurer que la base Mongo soit plus robuste.

L'objectif de cette page de documentation est d'expliquer pas à pas la mise en place d'un cluster MongoDB.

Remarques préliminaires

Lexique

Cette documentation est décrite de manière détaillée pour expliquer la mise en place des mécanismes de haute disponibilité pour la base MongoDB.

Avant d'aborder les étapes de configuration et les détails techniques, cette section présente quelques points importants à savoir avant de continuer.

Voici un lexique décrivant certains termes utilisés dans la suite de cette documentation :

Terme	Explication
Primaire	Nom du lexique MongoDB pour désigner un serveur maître, le serveur sur lequel il est possible de faire des requêtes d'écriture dans la base.
Secondaire	Nom du lexique MongoDB pour désigner une base de données Spare
Replicat Set *(abréviation rs)*	Un groupe de serveurs MongoDB qui met en œuvre la réplication de donnée et le basculement automatique
Sharding	Une architecture de base de données qui partitionne les données par plages de clés (shard) et distribue les données entre deux ou plusieurs instances de la base de données. Le sharding permet une mise à l'échelle horizontale.
Quorum	Dans une assemblée, le quorum est un chiffre représentant un nombre minimal de personnes en dessous duquel une délibération ne peut pas être considérée comme valide. Dans le cas de Mongo, avoir un quorum suffisant signifie avoir suffisamment de nœuds qui arrivent à communiquer pour être sûr d'être la partie principale du cluster en cas de problème réseau. En général, on définit ce quorum à N/2 + 1 nœud dans un cluster de N nœuds.

Election	Processus de Mongo pour choisir un nouveau membre Primaire si le membre Primaire devient inaccessible

Commandes à lancer

Durant la procédure d'installation, deux types de commandes seront exécutés :

- Les commandes **shell Linux** qui, sauf mention contraire, doivent être lancées en tant que **root.**
  Elles seront présentées comme suit :

  (commande shell)

  ```
  echo "commande shell"
  ```

- Les commandes **MongoDB** doivent être lancées dans un shell Mongo (*les instructions pour ouvrir le bon shell Mongo seront présentées avant chaque commande*).
  Elles seront présentées comme suit :

  (commande mongo)

  ```
  commande mongo
  ```

Nuance importante

Ce qui est mis en place dans cette documentation est la **haute disponibilité (*réplication*) de la base Mongo, et non une répartition de charge entre plusieurs nœuds Mongo (*sharding*)**.

On se concentre ici sur la haute disponibilité, qui essaye de garantir l'intégrité des données et leur accès. Les données sont donc **répliquées** entre plusieurs nœuds Mongo, au lieu d'être **réparties** sur plusieurs nœuds.

⚠ La réplication implique que chaque nœud MongoDB contient l'ensemble des données ce qui signifie que **chacun des nœuds** doit pouvoir supporter la **charge complète**.
Il est donc conseillé d'avoir des machines identiques pour chaque nœud du cluster MongoDB

Architecture mise en place

Détails sur les démons Mongo

Dans une installation MongoDB classique, une machine fait fonctionner un démon qui se charge du stockage des données (*mongod*). Ce démon représente dans ce cas-là, la base Mongo en elle-même.

Pour obtenir une architecture distribuée permettant de mettre en place de la haute disponibilité, il faut répartir l'installation de MongoDB sur plusieurs machines, qu'on appellera "nœuds" dans la suite de cette documentation.

Chaque nœud Mongo fait fonctionner plusieurs démons Mongo qui permettent de gérer la base de données. Dans l'exemple décrit dans ce document, on a 3 nœuds:

- Un nœud primaire, qui ordonnance la réplication et gère la configuration du cluster Mongo
- 2 nœuds secondaires, qui obéissent aux ordres du nœud primaire et s'occupent d'enregistrer les données qu'on leur envoie

Dans une architecture distribuée hautement disponible, il faut des démons supplémentaires pour que Mongo puisse gérer la réplication. Sur chaque nœud, les démons utilisés sont les suivants :

- **mongod**
  - Comme dans une installation MongoDB classique, le démon mongod se charge du stockage des données.
  - Il s'assure que les données sont bien répliquées sur les autres nœuds du cluster.
- **mongo-configsrv**
  - Ce démon gère la configuration du cluster MongoDB sur chaque nœud. Les autres démons MongoDB (*mongod et mongos*) s'appuient sur lui pour avoir des informations sur le cluster et sa configuration pour fonctionner.
  - Il est obligatoire d'avoir ce démon sur tous les serveurs du cluster. Les démons mongo-configsrv contiennent les métadonnées du cluster, telles que l'emplacement du shard des données.
- **mongos**
  - Ce démon s'occupe du routage des requêtes vers le démon mongod adéquat. En s'appuyant sur la configuration donnée par le démon mongo-configsrv, il sait sur quel démon mongod du cluster effectuer la requête.
    - En écriture : Effectue l'écriture sur le démon mongod du nœud primaire

        - En lecture : Effectue la lecture sur un serveur secondaire (*si autorisé*), sur le serveur primaire sinon
    - Ce démon écoute uniquement les requêtes locales.

Le schéma ci-contre présente l'architecture d'un nœud Mongo

Dans ce schéma, on a donc les 3 démons présents :

- **Démon mongod (data)**:
    - Ecoute sur le port 27018
    - et s'occupe de stocker les données (*dans* ***/var/lib/mongo*** *par défaut*)
- **Démon mongo-configsrv**:
    - Ecoute sur le port 27019
    - et s'occupe de stocker la configuration (**/var/lib/mongo/configdb** par défaut)
- **Démon mongos (routeur)**:
    - Ecoute sur le port 27017 (*utilisé dans une architecture classique par le démon mongod*).
    - Cette configuration permet à Shinken d'effectuer ses requêtes Mongo sur le port 27017 sans avoir à changer sa configuration.
    - Le démon mongos va ensuite effectuer le routage de la requête pour sa bonne exécution dans le cluster, en lecture ou en écriture.

? Unknown Attachment

L'interaction des nœuds du cluster est décrite sur le schéma ci-dessous.

Les démons présentés sont donc démarrés sur chacun des nœuds du cluster.

Shinken effectue ses requêtes de manière locale sur le démon mongos, qui ensuite s'occupe de l'aspect cluster de MongoDB et du bon traitement et acheminement des données.

Du point de vue de Shinken, la transformation de MongoDB en un cluster MongoDB est transparente.

Les requêtes sont toujours faites sur le même port et Shinken n'a pas conscience de la haute disponibilité de Mongo. Cela permet d'avoir 2 systèmes plus indépendants et leur association plus facilement configurable.

? Unknown Attachment

1. Le mongos local demande au mongo-configsrv qui est le serveur primaire du cluster
2. Le mongo-configsrv envoi l'information au mongos
3. Le mongos s'adresse directement au mongod

Il est obligatoire d'avoir le démon mongos en écoute local sur tous les serveurs, car c'est ce démon qui gère le sharding.

Chaque démon ayant besoin d'accéder à la base de donnée doit avoir un mongos d'accessible via son adresse locale (127.0.0.1 ou localhost)

Dans le cas où le mongos interrogé serait celui d'un serveur secondaire, le mongo-configsrv donnerait l'identité du serveur primaire de la même manière. Ensuite, le mongos s'adresserait toujours au serveur primaire.

Prérequis pour l'installation

Pour la mise en place du cluster Mongo, il faut que les conditions suivantes soient réunies :

- Il faut 3 serveurs distincts
    - Il peut s'agir de VM ou de serveurs physiques
        - Si ceux sont des VMs, il faut alors qu'elles soient hébergées par des serveurs physiques différents, pour la pertinence de la redondance.
- Il faut que chaque serveur du cluster puisse joindre tous les autres serveurs du cluster
- Pour des raisons de sécurité et de fiabilité des communications, on préfère que les serveurs du cluster soient dans le même réseau privé.
- Étant dédié à Shinken, les 2 serveurs secondaires du cluster Mongo doivent être vidés de toutes leurs données Mongo.
- Pour une configuration et une maintenance plus facile, on se réfère aux machines via leur nom DNS (*via un serveur DNS ou* ***/etc/hosts***) au lieu de leur adresse IP qui peut potentiellement changer

Serveurs Mongo et charge

Comme indiqué précédemment, il faut que chaque serveur puisse supporter la charge de requêtes Mongo à lui seul, et cela en permanence

Optimisation de l'architecture

Pour tirer au maximum avantage de l'architecture haute disponibilité, il est plus judicieux, dans le cas où les serveurs du cluster sont des VMs de ne pas les mettre sur le même hyperviseur. En effet, un problème sur l'hyperviseur pourrait alors affecter toutes les VMs en même temps et rendre l'architecture haute disponibilité inutile.

Procédure de configuration

Avant de commencer, voici un résumé des différentes étapes nécessaires pour la configuration du cluster Mongo :

- Installation de Shinken et Mongo.
  - L'installation de Shinken installe également ses dépendances, et donc Mongo.
  - Cette procédure d'installation est spécifique à la version de Mongo installée avec Shinken et l'utilisation d'une version différente de Mongo (*y compris sans utilisation d'un cluster*) n'est pas supportée et peut entraîner de nombreux bugs.
- Mise en place des démons de stockage des données
- Déclaration du replicaset dans Mongo: une fois les démons de stockage des données mis en place, on dit à Mongo qu'ils font partie d'un même ensemble de réplications de données .
- Mise en place des démons de gestion de la configuration Mongo
- Mise en place des démons de routage Mongo
- Vérification du bon fonctionnement du cluster : une fois l'installation terminée, on vérifie l'état du cluster Mongo et son bon fonctionnement.

Etape 1: Installation de Shinken

La première étape dans l'installation d'un cluster MongoDB est avant tout l'installation de MongoDB. Cette documentation s'appuie sur la version de Mongo installée par Shinken (*v3.0.15*).

- Le bon fonctionnement de Shinken n'est garanti qu'avec la version de Mongo installée automatiquement lors de l'installation de Shinken.
- Toute autre version n'est pas supportée par Shinken et peut entraîner de nombreux bugs.

L'installation de Shinken est décrite dans la page de documentation dédiée (*voir la page* Guide d'installation et de mise à jour).

Etape 2: Sécurisation des communications entre les démons mongoDB

Schéma global de sécurisation

Pour sécuriser la communication entre les serveurs MongoDB du cluster, on met en place des règles de firewall afin que seuls les serveurs mongo puissent échanger sur leurs ports de communications.

Sécurisation du cluster
? Unknown Attachment

Entre les nœuds du cluster MongoDB (Restriction des communications réseau)

Sur tous les serveurs du cluster, nous n'allons autoriser que les communications que vers les ports **27018** et **27019**, et ce, de manière restreinte.

Nous activons le firewall (*iptables*) et créons une nouvelle zone qui gère nos règles MongoDB.

(commande shell)

```
# on installe iptables en tant que service et on l'active
yum -y install iptables-services
systemctl enable iptables


# On crée une table iptables mongo pour y mettre nos règles
iptables -N MONGO # création d'une nouvelle chaîne

##
# Attention à bien changer les ADRESSE 1 à 3 avec les adresses des serveurs mongo du cluster.
##
iptables -A MONGO --src ADRESSE_1 -j ACCEPT
iptables -A MONGO --src ADRESSE_2 -j ACCEPT
iptables -A MONGO --src ADRESSE_3 -j ACCEPT
iptables -A MONGO --src 127.0.0.1 -j ACCEPT
iptables -A MONGO -j DROP  # drop everyone else
iptables -I INPUT -m tcp -p tcp --dport 27018 -j MONGO
iptables -I INPUT -m tcp -p tcp --dport 27019 -j MONGO

# On sauvegarde nos règles pour le redémarrage
service iptables save
```

IMPORTANT !!!

Attention à bien changer ADRESSE_1, ADRESS_2, ADRESSE_3 avec les vraies adresses des serveurs

À cette étape, avec ces règles, seuls les serveurs du cluster entre eux peuvent accéder aux données de mongo.

Etape 3: Mise en place des démons de stockage des données

Le premier démon mis en place est le démon **mongod**, qui est responsable du stockage des données.

Avant de commencer, on arrête le démon **mongod** sur tous les serveurs du cluster.
Sur tous les serveurs:

(commande shell)

```
/etc/init.d/mongod stop
```

On change aussi la configuration du démon mongod pour qu'il écoute sur le bon port et toutes les interfaces réseau et déclare son appartenance au replicaset.
Sur tous les serveurs:

(commande shell)

```
vi /etc/mongod.conf
```

/etc/mongod.conf

```
# network interfaces
net:
  port: 27018
  unixDomainSocket:
    enabled: false
  #bindIp: 127.0.0.1  # Listen to local interface only, comment to listen on all interfaces.

replication:
  replSetName: rs-shinken
```

IMPORTANT !!!

Attention à bien vérifier que les parties "**unixDomainSocket**" et "**enabled**" soient présentes, sinon les ajouter **en faisant attention à garder la même indentation**.

Le paramètre bindIp doit être COMMENTÉ.

- Commenter la ligne permet d'autoriser au niveau du mongod des connexions de n'importe quel IP.
- Mais pour rappel, ceci n'est pas un problème, car les connexions au mongod sont protégées par le firewall.

Mongo ne peut démarrer que si la base est vide sur les serveurs secondaires. On vide donc la base **sur les serveurs secondaires seulement**.

LA COMMANDE SUIVANTE VA SUPPRIMER TOUTE LA BASE MONGO SUR LE SERVEUR. Il faut bien s'assurer de ne pas avoir de données importantes sur ces serveurs avant de continuer.

Sur les serveurs secondaires :

(commande shell)

```
/etc/init.d/mongod stop
rm -rf /var/lib/mongo/*
```

La configuration du démon mongod est terminée. Ils peuvent être redémarrés.

Sur tous les serveurs :

(commande shell)

```
/etc/init.d/mongod start
```

Les démons **mongod** ne sont pour l'instant pas actif, mais sont configurés et prêts à recevoir des connexions.

On peut vérifier que ces démons ont bien été démarrés avec un netstat.

Sur tous les serveurs :

(commande shell)

```
netstat -laputen | grep 27018
```

Si le démon est démarré, cette commande affiche une ligne de résultat qui indique que le démon est démarré et écoute bien sur le port 27018 comme spécifié dans la configuration.

Etape 4: Déclaration du replicaset dans Mongo

Il faut ensuite déclarer dans Mongo que ces démons font partie du même replicaset et qu'il ne s'agit pas de démons mongod isolés.

Pour cela, on se connecte au démon mongod sur le serveur primaire pour déclarer le replicaset.
Sur le serveur primaire :

(commande shell)

```
mongo --port 27018
```

Cette commande lance le Shell Mongo dans lequel vous pouvez lancer la commande suivante :

(commande mongo)

```
rs.initiate({
    _id : "rs-shinken",
    members: [
        { _id: 0, host: "node1:27018", priority: 2 },
        { _id: 1, host: "node2:27018" },
        { _id: 2, host: "node3:27018" }
    ]
});
```

Dans cette commande, *node1*, *node2* et *node3* font référence aux noms DNS respectivement du nœud 1, du nœud 2 et du nœud 3.

Dans Mongo, lorsqu'un replicaset est défini, un algorithme d'élection est exécuté pour déterminer quel nœud du cluster sera le nœud primaire. Pour s'assurer que le nœud 1 soit bien le nœud primaire, on lui assigne une priorité supérieure pour qu'il sorte vainqueur.

On peut vérifier ensuite que le replicaset a bien été configuré via le shell Mongo ouvert précédemment :

(commande mongo)

```
rs.status()
```

Voilà ce que la commande au-dessus devrait afficher :

```
rs.status()
{
        "set" : "rs-shinken",
        "date" : ISODate("2020-02-11T16:59:55.734Z"),
        "myState" : 1,
        "members" : [
                {
                        "_id" : 0,
                        "name" : "HOSTMASTER:27018",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        "uptime" : 214,
                        "optime" : Timestamp(1581440386, 1),
                        "optimeDate" : ISODate("2020-02-11T16:59:46Z"),
                        "electionTime" : Timestamp(1581440388, 1),
                        "electionDate" : ISODate("2020-02-11T16:59:48Z"),
                        "configVersion" : 1,
                        "self" : true
                },
                {
                        "_id" : 1,
                        "name" : "HOSTSLAVE1:27018",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 8,
                        "optime" : Timestamp(1581440386, 1),
                        "optimeDate" : ISODate("2020-02-11T16:59:46Z"),
                        "lastHeartbeat" : ISODate("2020-02-11T16:59:54.832Z"),
                        "lastHeartbeatRecv" : ISODate("2020-02-11T16:59:55.013Z"),
                        "pingMs" : 1,
                        "configVersion" : 1
                },
                {
                        "_id" : 2,
                        "name" : "HOSTSLAVE2:27018",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 8,
                        "optime" : Timestamp(1581440386, 1),
                        "optimeDate" : ISODate("2020-02-11T16:59:46Z"),
                        "lastHeartbeat" : ISODate("2020-02-11T16:59:54.829Z"),
                        "lastHeartbeatRecv" : ISODate("2020-02-11T16:59:55.034Z"),
                        "pingMs" : 1,
                        "configVersion" : 1
                }
        ],
        "ok" : 1
}
```

- Le prompt du shell Mongo devrait également avoir changé pour afficher les mentions "PRIMARY" ou "SECONDARY".
- Par défaut, les lectures ne sont pas autorisées sur les Mongo secondaires. On peut autoriser ces lectures via un shell Mongo.
  Sur les serveurs secondaires:

(commande shell)
`mongo --port 27018`

(commande mongo)

```
rs.slaveOk()
```

À cette étape de la procédure de configuration, les données sont répliquées par les démons mongod.

Par contre, on ne possède pas encore de moyen facile pour accéder de manière automatique à Mongo via les applications, d'où le besoin d'un démon permettant de faire un routage des requêtes vers MongoDB.

Le temps limite pour qu'un membre du cluster soit vu comme défaillant est "heartbeatTimeoutSecs" donné par la commande rs.conf(); dans un shell de Mongo.

Par défaut cette limite est à 10 secs.

Note : Il s'agit d'un timeout réseau. Si un serveur du membre est coupé le replica set aura une erreur lors de la vérification de ce membre ce qui déclenchera une nouvelle élection de suite.

Etape 5: Mise en place des démons de gestion de la configuration

Avant de mettre en place le routage des requêtes pour les démons mongod de notre cluster, il faut mettre en place les démons serveurs de configuration qui vont permettre aux autres démons Mongo d'accéder facilement à la configuration du cluster Mongo.

Dans cette étape, on se contente de mettre en place le démon, mais pas son contenu. La configuration du cluster sera déclarée lors de la mise en place du démon de routage des requêtes Mongo.

On commence par créer le dossier qui contient la configuration du démon et lui attribuer les bons droits.
Sur tous les serveurs :

(commande shell)

```
mkdir /var/lib/mongo/configdb
chown mongod:mongod /var/lib/mongo/configdb
```

On copie ensuite le fichier de configuration par défaut du démon depuis l'archive d'installation de Shinken. Le chemin vers l'archive d'installation et désigné par "tarball_shinken".
Sur tous les serveurs :

- Pour Centos / Redhat 7 :

  (commande shell)

  ```
  cp tarball_shinken/tools/mongo-cluster/redhat_centos_7/mongo-configsrv.conf /etc/mongo-configsrv.conf
  vi /etc/mongo-configsrv.conf
  ```

- Pour Alma / Redhat 8 :

  (commande shell)

  ```
  cp tarball_shinken/tools/mongo-cluster/redhat_8/mongo-configsrv.conf /etc/mongo-configsrv.conf
  vi /etc/mongo-configsrv.conf
  ```

/etc/mongo-configsrv.conf

```
# Comme dans la configuration du démon mongod, on commente la ligne bind_ip pour que le démon écoute sur
toutes les interfaces
net:
  port: 27019
  unixDomainSocket:
    enabled: false
  #bindIp: 127.0.0.1  # Listen to local interface only, comment to listen on all interfaces.
```

IMPORTANT !!!

Attention à bien vérifier que les parties "**unixDomainSocket**" et "**enabled**" soient présentes, sinon les ajouter **en faisant attention à garder la même indentation**.

Le paramètre bindIp doit être COMMENTÉ.

- Commenter la ligne permet d'autoriser au niveau du mongod des connexions de n'importe quel IP.
- Mais pour rappel, ceci n'est pas un problème, car les connexions au mongod sont protégées par le firewall.

Piège

Le fichier de configuration **/etc/mongo-configsrv.conf** comporte également une option *replSetName*. Les démons *mongo-configsrv* gèrent la configuration du cluster d'un point de vue global, et doivent donc être démarrés en tant que démons indépendants et non en tant que partie du replicaset.

Il faut s'assurer que l'option *replSetName* des fichiers de configuration des démons *mongo-configsrv* soit bien commentée et non utilisée.

On ajoute ensuite le démon dans la liste des services gérés par le système, ainsi que dans la liste des services à lancer au démarrage de la machine.
Sur tous les serveurs :

(commande shell)

```
chkconfig --add mongo-configsrv
chkconfig mongo-configsrv on
```

On démarre ensuite le démon sur tous les serveurs.
Sur tous les serveurs :

(commande shell)

```
/etc/init.d/mongo-configsrv start
```

Etape 6: Mise en place des démons de routage des requêtes vers MongoDB

La dernière étape est de configurer le démon mongos qui sert de point d'accès pour Shinken, et permet de rediriger les requêtes vers le bon nœud MongoDB.

- Sera nécessaire pour les démons tels que le Broker, Synchronizer, ou les Schedulers (*dans le cas de rétention mongo*).
- Les 3 actions suivantes sont à faire pour chaque nouveau démon de routage nécessaire (*mongos*).

1 - Changement du port de mongod

Modifier le port d'écoute de mongod pour le passer à 27018

(commande shell)

```
vi /etc/mongod.conf
```

/etc/mongod.conf

```
# network interfaces
net:
  port: 27018
  unixDomainSocket:
    enabled: false
  bindIp: 127.0.0.1  # Listen to local interface only, comment to listen on all interfaces.
```

IMPORTANT !!!

Attention à bien vérifier que les parties "**unixDomainSocket**" et "**enabled**" soient présentes, sinon les ajouter **en faisant attention à garder la même indentation**.

Le paramètre bindIp doit être DE-commenté, et l'adresse doit être 127.0.0.1.

- Seuls les démons sur la machi

2 - Autoriser les connexions du mongos au cluster MongoDB

On autorise les connexions de mongos vers mongod et mongo-configsrv

- **À faire sur les serveurs du cluster mongo** (*qui portent le **mongod** et **mongo-configsrv***) :
  - **pour CHAQUE serveur shinken** qui va avoir besoin d'accéder au cluster MongoDB, on **rajoute UNE règle** d'accès dans **iptables.**

(commande shell)

```
iptables -I MONGO --src ADRESSE_SERVER_MONGOS_01 -j ACCEPT
service iptables save
```

3 - Activation des mongos

On commence par copier le fichier de configuration par défaut :

- Pour Centos / Redhat 7 :

  (commande shell)

  ```
  cp tarball_shinken/tools/mongo-cluster/redhat_centos_7/mongos.conf /etc/mongos.conf
  vi /etc/mongos.conf
  ```

- Pour Alma / Redhat 8 :

  (commande shell)

  ```
  cp tarball_shinken/tools/mongo-cluster/redhat_8/mongos.conf /etc/mongos.conf
  vi /etc/mongos.conf
  ```

/etc/mongos.conf

```
# Dans le paramètre configdb, on renseigne la liste des serveurs de configuration qui constituent le cluster
net:
  port: 27017
  unixDomainSocket:
    enabled: false
  bindIp: 127.0.0.1  # Listen to local interface only, comment to listen on all interfaces.

sharding:
   configDB: node1:27019,node2:27019,node3:27019
```

IMPORTANT !!!

Attention à bien vérifier que les parties "**unixDomainSocket**" et "**enabled**" soient présentes, sinon les ajouter **en faisant attention à garder la même indentation**.

Le paramètre bindIp doit être DE-commenté, et l'adresse doit être 127.0.0.1.

- Seuls les démons sur la machine pourront se connecter localement sur le mongos.

On ajoute ensuite le démon dans la liste des services gérés par le système, ainsi que dans la liste des services à lancer au démarrage de la machine.

(commande shell)

```
chkconfig --add mongos
chkconfig mongos on
```

On démarre ensuite le mongos sur chaque machine.

(commande shell)

```
/etc/init.d/mongos start
```

Il est possible de configurer certains paramètres de mongos (*voir la page* Cas spécial des mongos (démon de routage)).

Etape 7: Activation de la configuration du cluster MongoDB

Il reste à déclarer à Mongo les serveurs de données qui constituent le cluster (*ce sont ces informations qui seront stockées par le démon mongo-configsrv*).

Sur le serveur primaire :

(commande shell)

```
mongo
```

(commande mongo)

```
sh.addShard('rs-shinken/node1:27018,node2:27018,node3:27018')
```

La configuration du cluster MongoDB est maintenant terminée !

Etape 8: Vérification du bon fonctionnement du cluster

Une fois la procédure complétée, on peut ensuite vérifier à l'aide d'un shell Mongo l'état du replicaset.

Vérification de l'état des connexions

Via le démon mongos, on peut vérifier l'ensemble des connexions du cluster.

Sur n'importe quel serveur :

(commande shell)

```
mongo
```

(commande mongo)

```
sh.status()
```

On obtient alors un résumé des machines du cluster et l'ensemble des bases de données gérées par ce cluster :

```
--- Sharding Status ---
sharding version: {
    "_id" : 1,
    "version" : 4,
    "minCompatibleVersion" : 4,
    "currentVersion" : 5,
    "clusterId" : ObjectId("5ae1b6d7e4edc5bd313ba7b7") }
shards: {
     "_id" : "rs-shinken",
    "host" : "rs-shinken/node1:27018,node2:27018,node3:27018" }
databases: {
    "_id" : "admin",
    "partitioned" : false,
    "primary" : "config" }
{
    "_id" : "shinken",
    "partitioned" : false,
    "primary" : "rs-shinken" }
{
     "_id" : "synchronizer",
    "partitioned" : false,
    "primary" : "rs-shinken" }
{
     "_id" : "test_db",
    "partitioned" : false,
    "primary" : "rs-shinken"
}
```

Vérification de l'état du replicaset

Via le démon mongod, on peut vérifier l'état du replicaset. On peut alors voir l'état de chaque nœud, en particulier le nœud qui est actuellement le nœud primaire.

Sur n'importe quel serveur :

(commande shell)

```
mongo --port 27018
```

(commande mongo)

```
rs.status()
```

On obtient alors un détail de l'état des différentes machines du cluster. On peut aussi facilement identifier quel nœud est primaire.

```
rs-shinken276:PRIMARY> rs.status()
{
        "set" : "rs-shinken",
        "date" : ISODate("2020-11-24T11:01:42.638Z"),
        "myState" : 1,
        "members" : [
                {
                        "_id" : 0,
                        "name" : "node1:27018",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        "uptime" : 325692,
                        "optime" : Timestamp(1606215702, 217),
                        "optimeDate" : ISODate("2020-11-24T11:01:42Z"),
                        "electionTime" : Timestamp(1605890035, 1),
                        "electionDate" : ISODate("2020-11-20T16:33:55Z"),
                        "configVersion" : 646914,
                        "self" : true
                },
                {
                        "_id" : 1,
                        "name" : "node2:27018",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 325690,
                        "optime" : Timestamp(1606215701, 155),
                        "optimeDate" : ISODate("2020-11-24T11:01:41Z"),
                        "lastHeartbeat" : ISODate("2020-11-24T11:01:41.343Z"),
                        "lastHeartbeatRecv" : ISODate("2020-11-24T11:01:40.698Z"),
                        "pingMs" : 1,
                        "syncingTo" : "node3:27018",
                        "configVersion" : 646914
                },
                {
                        "_id" : 2,
                        "name" : "node3:27018",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 325690,
                        "optime" : Timestamp(1606215701, 155),
                        "optimeDate" : ISODate("2020-11-24T11:01:41Z"),
                        "lastHeartbeat" : ISODate("2020-11-24T11:01:41.617Z"),
                        "lastHeartbeatRecv" : ISODate("2020-11-24T11:01:41.078Z"),
                        "pingMs" : 1,
                        "syncingTo" : "node1:27018",
                        "configVersion" : 646914
                }
        ],
        "ok" : 1,
        "$gleStats" : {
                "lastOpTime" : Timestamp(0, 0),
                "electionId" : ObjectId("5fb7eff330ad536e074d03df")
        }
}
```

Etape 9 : Activer le chiffrement (SSL) pour les communications d'un cluster MongoDB

Actuellement, seules les machines du cluster ou celles équipées d'un mongos peuvent accéder à la base de données. N'importe quelle connexion provenant de ses serveurs est acceptée par la base et les paquets qui transitent sur le réseau sont en clair.

Activer le chiffrement SSL permet d'authentifier les clients qui se connectent à la base de données, lesquels doivent présenter un certificat reconnu par celle-ci. Cela assure de plus que les paquets transitant sur le réseau sont chiffrés et ne puissent pas être lus par un attaquant. L'activation du chiffrement se fait après l'installation du Cluster (*voir la page :* Activer le chiffrement (SSL) pour les communications d'un cluster MongoDB).

Cas spécial des mongos (démon de routage)

Gestion des longues requêtes

Notre version de mongos est livrée avec une fonctionnalité de gestion des longues requêtes.

Si le nœud primaire du cluster mongo perd la connexion pendant l'envoi d'une requête, mongos attendra 15 minutes avant de considérer la requête comme perdue.

La fonctionnalité de gestion des longues requêtes permet d'éviter ça en envoyant des sondes vers le nœud primaire. Ceci permettant de vérifier que la machine distante est bien toujours accessible.

Configuration

Nous livrons notre version de mongos avec un fichier de configuration permettant de gérer cette fonctionnalité.

> Le fichier est disponible dans : /etc/shinken/tools_used_by_shinken/shinken_mongo/mongos_socket_keep_alive.cfg

> Ces paramètres changeront la façon dont votre mongos réagira vis-à-vis des nœuds du cluster. Il est totalement déconseillé de changer ces valeurs sans votre support Shinken

Nom de la clé	Unité	Valeur par défaut	Description
tools_used_by_shinken__mongo__tcp_keep_alive__enabled	Booléen	1 / True	Permet d'activer la fonction "tcp_keep_alive". Cette fonction permet aux longues requêtes mongo de vérifier si la machine vers laquelle la requête a été envoyée est toujours accessible ou non. Dans le cas contraire, la requête sera avortée. > Si ce paramètre est désactivé, tous les paramètres suivants seront ignorés.
tools_used_by_shinken__mongo__tcp_keep_alive__time_before_idle	Seconde	15	Défini le nombre de secondes à partir duquel une requête est considérée comme longue. Lorsqu'une requête est considérée comme longue, des sondes sont envoyées vers la machine distante pour vérifier si elle est bien toujours disponible.
tools_used_by_shinken__mongo__tcp_keep_alive__number_of_probes	---	3	Nombre de sondes à envoyer vers la machine distante avant de considérer celle-ci comme inaccessible et donc d'avorter la requête. Une sonde est une requête de type "ping".
tools_used_by_shinken__mongo__tcp_keep_alive__time_interval_between_probes	Seconde	5	Temps d'attente en secondes entre chaque envoi de sonde vers la machine distante. > Ce temps n'impactera PAS le temps de la requête principale.
tools_used_by_shinken__mongo__tcp_keep_alive__enable_debug	Booléen	1 / True	Active le mode DEBUG de la fonctionnalité "tcp_keep_alive". Le mode DEBUG permet d'afficher toutes les erreurs des envois de sondes dans les logs.

Comportement de Shinken avec un cluster MongoDB

L'utilisation d'un cluster MongoDB permet d'améliorer la stabilité de Shinken Entreprise. Lorsque le nœud principal du cluster MongoDB entre en erreur, MongoDB procède à l'élection d'un nouveau nœud principal dans le cluster. À ce moment, on voit alors une coupure dans Shinken pendant quelques secondes le temps qu'un nouveau nœud primaire soit élu.

Cette coupure a les incidences suivantes :

- Le démon le plus impacté est le **Synchronizer**. Puisque ce démon a besoin en permanence d'une connexion à la base Mongo, il s'arrête lorsque la base n'est pas disponible. Avec un cluster MongoDB, une coupure de connexion s'effectue quand aucun nœud primaire n'est disponible, c'est-à-dire quand le nœud primaire devient injoignable, ou bien quand il devient à nouveau joignable. (quand la disponibilité des nœuds change, Mongo procède à l'élection d'un nouveau nœud primaire, d'où la coupure)
- Le module de rétention MongoDB du Scheduler ne sera que très peu probablement impacté par une coupure temporaire. En effet, la sauvegarde de la rétention Mongo s'effectue à intervalles réguliers (par défaut toutes les heures) et prend quelques secondes tout au plus.

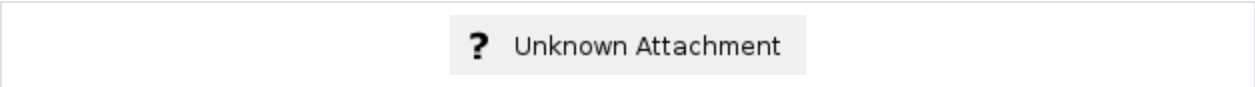
En cas d'indisponibilité de la base, plusieurs essais sont faits avant de déclarer la sauvegarde de la rétention comme échouée.
De plus, l'absence de sauvegarde de rétention n'a d'incidence que lors du redémarrage du Scheduler. Pour qu'un problème se produise lors de la sauvegarde de la rétention, il faut donc que le choix par Mongo d'un nouveau nœud primaire s'effectue pendant les quelques secondes par heure de sauvegarde de la rétention et prenne plus de 30 secs.

- Le module SLA du Broker redémarre tant que la base Mongo est indisponible et reprend ses tâches dès que MongoDB redevient disponible.

Configuration de Shinken pour utiliser le cluster Mongo

Pour que Shinken comprenne qu'il doit utiliser le cluster mongo, il faut modifier chaque module et fichier de configuration utilisant mongo :

- L'URI mongo doit pointer vers l'interface locale (127.0.0.1 ou localhost). Il faut que ça soit également le cas du démon mongo**S** et qui répartira les requêtes en fonction des serveurs accessible.



Liste non-exhaustive des modules utilisant mongo :

- Mongodb
- MongodbRetention
- sla
- synchronizer-collector-linker
- synchronizer-module-database-backup

Liste des fichiers de configuration nécessitant mongo

- synchronizer.cfg
  - discovery.cfg

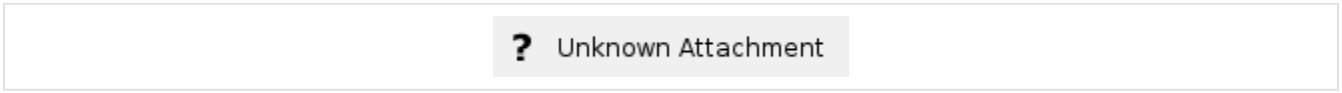
Supervision du cluster Mongo

Un cluster Mongo peut être supervisé avec Shinken. Shinken Entreprise met à votre disposition un pack MongoDB (*voir la page* 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 d'hôte **mongodb** utilise par défaut une connexion directe aux démons Mongo. Suite à l'étape 1 qui consiste à mettre en place une clé d'authentification pour la communication entre les démons, les opérations possibles et vérifications effectuées par les checks échouent.

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 dans l'étape 3 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 disponible dans la documentation (*voir la page* Pack MongoDB.)

Maintenance et résolution des problèmes

L'installation en cluster de MongoDB apporte des avantages au niveau de l'accessibilité et de la redondance des données. Cependant, ces avantages nécessitent un système plus complexe au niveau de MongoDB et la manipulation de MongoDB présente quelques différences par rapport à une architecture classique.

Les différentes opérations de maintenance et résolutions de maintenance qui peuvent apparaître sur MongoDB sont présentées dans la documentation (*voir la page* Maintenance et résolution des problèmes dans un cluster Mongodb).