Mise en place de l'architecture Primaire - Secondaire - Arbiter

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

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

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	Désigne l'instance MongoDB maître, sur laquelle il est possible de faire les requêtes d'écriture
Secondaire	Désigne la ou les instances MongoDB contenant les données répliquées de l'instance Primaire. Il est possible via l'option "rs. slaveOk()" d'autoriser la lecture sur ces instances.
Replica Set *(abréviation rs)*	Un groupe de serveurs MongoDB qui met en œuvre la réplication de donnée écrites sur un serveur Primaire.
Nœuds	Désigne un serveur avec une instance MongoDB du "Replica Set".
Arbiter *(MongoDB)*	Désigne une instance MongoDB sans donnée, mais qui peut choisir un nouveau membre Primaire. Utile si l'on veut répliquer ses données que deux fois, car le "Replica Set" a besoin d'au moins trois instances pour fonctionner.
Sharding *(abréviation sh)*	Une architecture de base de données qui partitionne les données par plages de clés (*shard*) et distribue les données entre une ou plusieurs instances de la base de données. Le sharding permet une mise à l'échelle horizontale. Dans cette documentation, le Sharding va servir à mettre en place des démons pour router les requêtes sur le bon nœud maitre du Replica Set.
Élection	Processus de Mongo pour choisir un nouveau membre Primaire si le membre Primaire devient inaccessible
mongod	Démon en charge du stockage des données.
mongos	Démon en charge du routage des requêtes vers le démon **mongod** adéquat. Il est ici utilisé pour transférer les requêtes sur l'instance Primaire.
mongo-configsrv	Démon en charge du stockage de la configuration du Sharding. Les démons **mongos** s'appuient sur lui pour avoir la configuration du Sharding.

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
```

Architecture mise en place

Présentation des démons Mongo

Dans une installation MongoDB classique, un serveur 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 serveurs (*appelé nœuds dans la suite de la documentation*).
- Chaque nœud Mongo fait fonctionner plusieurs démons Mongo qui permettent de gérer la base de données.

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

- **mongod**
  - Comme dans une installation MongoDB classique, le démon mongod se charge du stockage des données.
  - Il traite les requêtes d'écriture et de lecture.
  - Il contient toutes les données.
  - Par défaut :
    - sa configuration est dans le fichier /etc/mongod.conf
    - écoute sur le port 27018
    - ses données sont stockées dans :
      - /var/lib/mongodb sous Debian
      - /var/lib/mongo sous RHEL / CentOS / Alma / Rocky
    - ses logs sont stockés dans /var/log/mongodb/mongod.log
- **mongo-configsrv**
  - Ce démon gère la configuration du cluster MongoDB (*Sharding*) sur chaque nœud.
  - Les démons **mongos** s'appuient sur lui pour récupérer la configuration du cluster.
  - Il est obligatoire d'avoir trois démons **mongo-configsrv**.
    - Dans le cas standard, il y a un démon sur les trois nœuds du Cluster.
  - Par défaut :
    - sa configuration est dans le fichier /etc/mongo-configsrv.conf
    - écoute sur le port 27019
    - ses données sont stockées dans :
      - /var/lib/mongodb/configdb sous Debian
      - /var/lib/mongo/configdb sous RHEL / CentOS / Alma / Rocky
    - ses logs sont stockés dans /var/log/mongodb/mongo-configsrv.log
- **mongos**
  - Ce démon s'occupe du routage des requêtes vers le démon **mongod** Primaire.
  - En s'appuyant sur la configuration donnée par le démon **mongo-configsrv**, il possède la liste des **mongod** et peut ainsi trouver le serveur Primaire.
  - Ce démon écoute uniquement les requêtes locales.
  - Par défaut :
    - sa configuration est dans le fichier /etc/mongos.conf
    - écoute sur le port 27017 (*utilisé dans une architecture classique par le démon mongod*)
    - ses logs sont stockés dans /var/log/mongodb/mongos.log

Bien que cette documentation ne permette pas la mise en place d'un Sharding répartissant la charge entre plusieurs nœuds Mongo, les démons **mongos** et d'autres composants du Sharding sont utilisés.
Ces composants sont utiles pour trouver quel **mongod** a été élu Primaire.

Architecture Primaire / Secondaire / Arbiter (PSA)

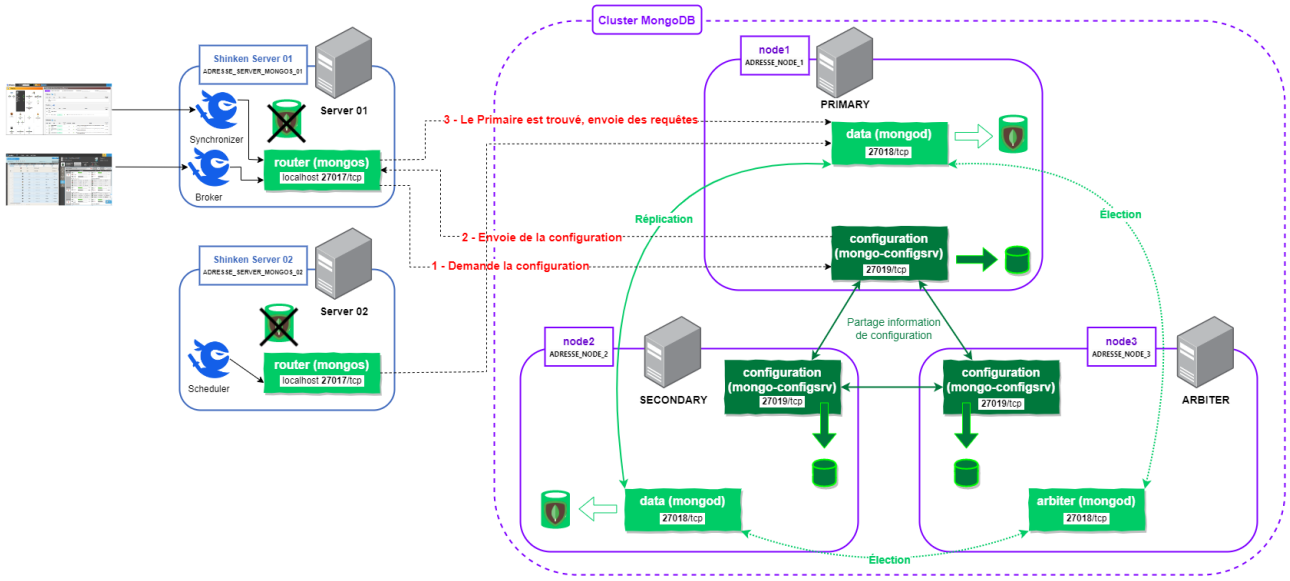
Voici la composition de l'architecture PSA avec trois nœuds :

- Un nœud Primaire, qui réceptionne et traite les requêtes d'écriture et de lecture. Il contient toutes les données.
- Un nœud Secondaire, qui réplique les données du nœud Primaire. Il contient aussi toutes les données.
- Un nœud Arbiter *(MongoDB)*, qui permet d'élire un nouveau nœud Primaire. Il ne contient pas de 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 deux systèmes plus indépendants et leur association plus facilement configurable.

Architecture PSA



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 ce 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.
  - Il faut noter que la réplication des nœuds va entrainer une charge réseau, que l'infrastructure doit pouvoir absorber.
- Les serveurs Secondaires ne doivent pas avoir de données.
- 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
  - Cela peut être utile s'il faut déplacer les mongo-configsrv de machine sans interruption de service, sinon il faudra refaire la configuration du Sharding.

⚠ **Serveurs Mongo et charge**

Comme indiqué précédemment, il faut que chaque serveur Primaire et Secondaire puisse supporter la charge CPU, RAM, Disque (*IO et espace*) et réseau des requêtes Mongo à lui seul, et cela, en permanence.

Dans cette architecture, l'Arbiter ne contiendra pas de donnée et peut-être une machine très légère. Voici la configuration minimale pour gérer uniquement un Arbiter Mongo et un mongo-configsrv:

- 2 CPU
- 20 Go de disque
- 2 Go de RAM

⚠ **Nœuds sur des machines virtuel**

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.

Sécurisation du cluster

Dans une architecture mono-serveur, la sécurisation des communications avec la base est assuré par le fait que la base (*mongod*) est en écoute uniquement sur la boucle locale (*localhost*) via le paramètre "bind_id" de sa configuration. Dans ce cas, même s'il y a plusieurs serveurs qui ont besoin d'avoir accès à la base, on utilisera des tunnels SSH (*voir la page :* Sécurisation des connexions aux bases MongoDB).

La sécurisation d'un cluster Mongo est différente :

- Les démons mongod et mongo-configsrv vont devoir échanger des données (*entre eux et avec les mongos*) et ne peuvent pas utiliser de tunnel SSH. Il faudra donc que ces démons soient en écoute sur une interface publique (*c'est-à-dire que n'importe qui pourrait leur envoyer des requêtes*)
  - Pour sécuriser les connexions à ces démons, il faut utiliser un firewall.
  - Le firewall va limiter les accès aux démons mongod et mongo-configsrv aux machines du cluster et aux serveurs Shinken ayant besoin de se connecter aux clusters (*Synchonizer, Scheduler avec une retention Mongo, Broker*) .
- Les démons mongos, eux écouteront sur la boucle locale (*comme les mongod dans une architecture mono-serveur*) ce qui protègera l'accès aux données par ce démon.
- 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 peuvent 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).

Quelle que soit l'architecture de MongoDB (*mono-serveur ou cluster*), il est possible d'activer l'authentification par mot de passe afin de restreindre l' accès aux données à Shinken uniquement (*voir la page* MongoDB - activation de l'authentification par mot de passe *et la page* Activer l'authentification par mot de passe dans un cluster MongoDB).

Cette documentation fourni la configuration pour deux firewall :

- iptables : c'est le firewall utilisé par défaut sur CentOS 7 / RedHat 7.
  - Firewall très complet, avec une gestion fine des règles. Il est présenté comme plus ancien et plus compliqué que firewalld.
  - Il est tout à fait possible de l'utiliser sur RHEL / Alma / Rocky 8 ou RHEL / Alma / Rocky 9.
- firewalld : c'est le firewall utilisé par défaut sur RHEL / Alma / Rocky 8 ou RHEL / Alma / Rocky 9.
  - Firewall plus simple que iptables (*il couvre plus facilement les cas standards*).
  - Il est tout à fait possible de l'utiliser sur CentOS 7 / RedHat 7.

Par défaut sous Debian, il n'y a pas de firewall installer.

À moins d'avoir une recommandation de l'équipe de sécurité, il est conseillé d'utiliser :

- iptables sur CentOS 7 / RedHat 7 et
- firewalld sur RHEL / Alma / Rocky 8 ou RHEL / Alma / Rocky 9 ou Debian 13

Un cluster Mongo en écoute sur une interface publique avec un accès internet sans protection, se fera pirater en quelques minutes !

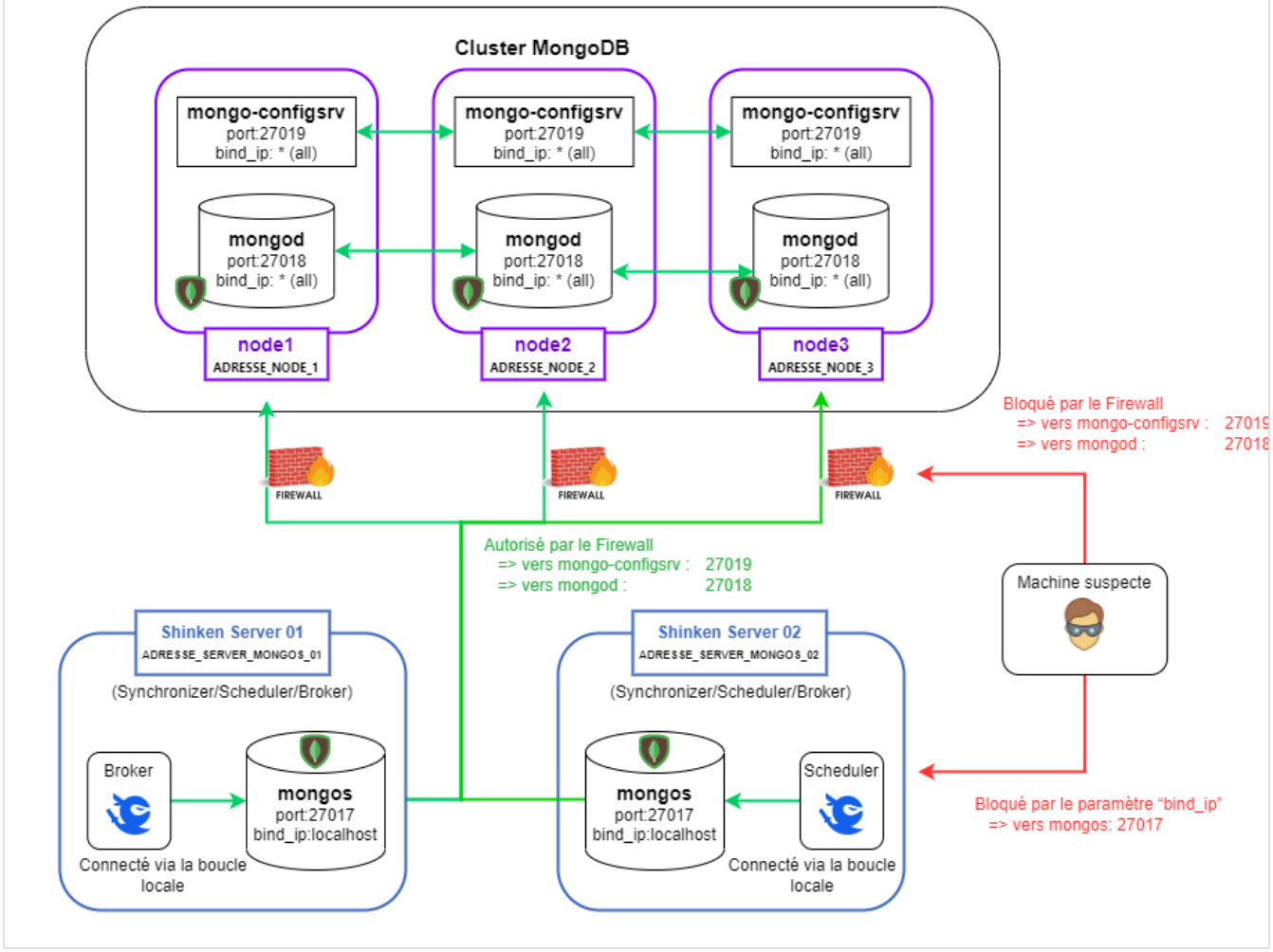
Il n'est pas possible d'utiliser les deux firewall, il faut choisir soit iptables soit firewalld.

⚠️ Avec firewalld, il n'est pas possible d'utiliser les noms DNS des différents serveurs, il faudra indiquer les adresses IP.

Même avec iptable, il est déconseillé d'utiliser les noms DNS des différents serveurs pour définir des règles de firewall et plutôt préférer l'utilisation d'adresses IP.

- Une règle avec un nom DNS est traduite avec une IP **au moment de la définition de la règle** (*le nom n'est pas enregistré tel quel*).
- Lors du filtrage, il n'y a **pas de résolution DNS live** sur chaque règle enregistrée dans le firewall lors d'une réception d'un paquet à filtrer.
- Un changement de configuration d'un hôte dans le DNS n'a pas d'impact sur la configuration live du firewall (*il faut réappliquer les règles du firewall pour qu'un changement NOM<->IP soit répercuté dans les règles, cela peut être trompeur*).

Sécurisation du cluster



Configuration du firewall avec iptables (firewall par défaut de CentOS 7 / RedHat 7)

Les commandes qui suivent sont à passer **sur chaque nœud du cluster** (*le Primaire et les Secondaires, node 1 à 3 dans le schéma "Sécurisation du cluster" ci-dessus*). Il n'est pas utile de les passer sur serveurs ayant seulement un mongos (*"Shinken Serveur 01" et "Shinken Serveur 02" dans le schéma "Sécurisation du cluster" ci-dessus*)

Mise en place du service *iptables*

RHEL / CentOS 7 ou RHEL / Alma / Rocky 8 ou RHEL / Alma / Rocky 9

- **(commande shell)**

```
yum -y install iptables-services
systemctl enable iptables
```

Debian 13

- **(commande shell)**

```
apt install -y iptables iptables-persistent
systemctl enable netfilter-persistent
```

Sécurisation des communications entre les nœuds du cluster (entre mongod et mongo-configsrv)

Créer les règles iptables pour filtrer le trafic à destination des ports de MongoDB :

(commande shell)

```
# 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
##
iptables -A MONGO --src ADRESSE_NODE_1 -j ACCEPT
iptables -A MONGO --src ADRESSE_NODE_2 -j ACCEPT
iptables -A MONGO --src ADRESSE_NODE_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
```

Dans ces commandes :

- **ADRESSE_NODE_1** fait référence à l'adresse du nœud Primaire
- **ADRESSE_NODE_2** fait référence à l'adresse du nœud Secondaire
- **ADRESSE_NODE_3** fait référence à l'adresse du nœud Arbiter

Sécurisation des communications entre les nœuds du cluster et les servers Shinken (entre mongod/mongo-configsrv et mongos)

Il faudra définir une règle pour chaque serveur avec un "mongos" (*et ne faisant pas partie du cluster*).

Par exemple, dans le cas du schéma "Sécurisation du cluster" il faudra exécuter deux fois la commande ci dessous (*et toujours sur chaque nœud du cluster*) :

- une fois pour "Shinken Serveur 01"
- une autre fois pour "Shinken Serveur 02".

(commande shell)

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

Dans ces commandes :

- **ADRESSE_SERVER_MONGOS_01** fait référence à l'adresse du serveur avec les démons mongos et Shinken.

Sauvegarde des règles de firewall

RHEL / CentOS 7 ou RHEL / Alma / Rocky 8 ou RHEL / Alma / Rocky 9

- **(commande shell)**

```
service iptables save
```

Debian 13

- **(commande shell)**

```
service netfilter-persistent save
```

Configuration du firewall avec firewalld (firewall par défaut de RHEL / Alma / Rocky 8 ou RHEL / Alma / Rocky 9)

Les commandes qui suivent sont à passer **sur chaque nœud du cluster** (*le Primaire et les Secondaires, node 1 à 3 dans le schéma "Sécurisation du cluster" ci-dessus*). Il n'est pas utile de les passer sur serveurs ayant seulement un mongos (*"Shinken Serveur 01" et "Shinken Serveur 02" dans le schéma "Sécurisation du cluster" ci-dessus*)

Mise en place du service firewalld

RHEL / CentOS 7 ou RHEL / Alma / Rocky 8 ou RHEL / Alma / Rocky 9

- **(commande shell)**

```
yum -y install firewalld
systemctl enable firewalld
systemctl start firewalld
```

Debian 13

- **(commande shell)**

```
apt install -y firewalld
systemctl enable firewalld
systemctl start firewalld
```

Sécurisation des communications entre les nœuds du cluster (entre mongod et mongo-configsrv)

Définition des règles de filtrage du trafic à destination des ports MongoDB :

(commande shell)

```
# On crée une zone mongo pour y mettre nos règles
firewall-cmd --permanent --new-zone=mongo
firewall-cmd --reload


# On vérifie que la zone a bien été créée.
firewall-cmd --list-all --zone=mongo

# On ajoute les règles pour autoriser les communications des serveurs du cluster sur les ports 27018 et
27019.
##
# Attention à bien changer les ADRESSE 1 à 3
##
firewall-cmd --zone=mongo --add-source=ADRESSE_NODE_1
firewall-cmd --zone=mongo --add-source=ADRESSE_NODE_2
firewall-cmd --zone=mongo --add-source=ADRESSE_NODE_3
firewall-cmd --zone=mongo --add-port=27018/tcp
firewall-cmd --zone=mongo --add-port=27019/tcp

# On ajoute aussi les règles par défaut pour ssh, cockpit et dhcp.
# Ne pas faire cette commande sur CentOS 7. Le service "cockpit" n'existe pas sur CentOS 7
firewall-cmd --zone=mongo --add-service=cockpit
firewall-cmd --zone=mongo --add-service=dhcpv6-client
firewall-cmd --zone=mongo --add-service=ssh

# On vérifie que notre configuration a bien été prise en compte.
firewall-cmd --list-all --zone=mongo
# On sauvegarde notre configuration pour le redémarrage
firewall-cmd --runtime-to-permanent
```

Dans ces commandes :

- **ADRESSE_NODE_1** fait référence à l'adresse du nœud Primaire
- **ADRESSE_NODE_2** fait référence à l'adresse du nœud Secondaire
- **ADRESSE_NODE_3** fait référence à l'adresse du nœud Arbiter

Pour la configuration de firewalld, il n'est pas possible d'utiliser les noms DNS, il faudra indiquer les adresses IP des différents serveurs.

Sécurisation des communications entre les nœuds du cluster et les servers Shinken (entre mongod/mongo-configsrv et mongos)

Il faudra définir une règle pour chaque serveur avec un "mongos" (*et ne faisant pas partie du cluster*).

Par exemple, dans le cas du schéma "Sécurisation du cluster" il faudra exécuter deux fois les commandes qui suivent (*et toujours sur chaque nœud du cluster*) :

- une fois pour "Shinken Serveur 01"
- une autre fois pour "Shinken Serveur 02".

(commande shell)

```
firewall-cmd --zone=mongo --add-source=ADRESSE_SERVER_MONGOS_01
firewall-cmd --runtime-to-permanent
```

Dans ces commandes :

- **ADRESSE_SERVER_MONGOS_01** fait référence à l'adresse du serveur avec les démons mongos et Shinken.

Pour la configuration de firewalld, il n'est pas possible d'utiliser les noms DNS, il faudra indiquer les adresses IP des différents serveurs.

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 (*mongod*).
- Déclaration du Replica Set 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 réplication de données.
- Mise en place des démons de gestion de la configuration Mongo (*mongo-configsrv*).
- Mise en place des démons de routage Mongo (*mongos*).
- 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). Il faut installer Shinken sur chaque nœud du Cluster MongoDB. Il peut être nécessaire ensuite de désactiver les démons Shinken qui ne seront pas nécessaires sur la machine.

Etape 2: Sécurisation des communications entre les nœuds du cluster (entre mongod et mongo-configsrv)

Appliquer le chapitre "Sécurisation des communications entre les nœuds du cluster (entre mongod et mongo-configsrv)" pour le firewall choisi (*iptables ou firewalld*).

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)

```
service 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 Replica Set.
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
```

Modifications

1. **Modifier** le **port** (*les mongod vont écouter sur le port 27018 pour laisser le port 27017 aux mongos*).
2. **Vérifier** que les parties **"unixDomainSocket**" et "**enabled**" **soit présents**,
   - sinon **les ajouter** en faisant attention à garder la même indentation que dans l'exemple ci-dessus.
3. **Mettre en commentaire** la ligne **bindIp** (*permet aux nœuds du cluster de communiquer entre eux. Le contrôle d'accès à la base se fera par la mise en place du firewall, voir le chapitre* Sécurisation du cluster).
4. **Ajouter** le bloc **replication** (*permet d'indiquer que cette instance de mongod fait partie d'un Replica Set*), comme dans l'exemple ci-dessus.

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

Sur les serveurs Secondaires et Arbiter :

Purge des données sur les serveurs Secondaires et l'Arbiter

RHEL / CentOS 7 ou RHEL / Alma / Rocky 8 ou RHEL / Alma / Rocky 9

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 ce serveur avant de continuer.

- **(commande shell)**

```
service mongod stop
rm -rf /var/lib/mongo/*
```

Debian 13

> **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 ce serveur avant de continuer.

- **(commande shell)**

```
service mongod stop
rm -rf /var/lib/mongodb/*
```

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

Sur tous les serveurs :

(commande shell)

```
service mongod start
```

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

On peut vérifier que ces démons ont bien été démarrés avec la commande 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 Replica Set dans Mongo

Il faut ensuite déclarer dans Mongo que ces démons font partie du même Replica Set 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 Replica Set.

Sur le serveur Primaire :

(commande shell)

```
mongo --port 27018
```

Cette commande lance le Shell Mongo afin de lancer la commande suivante :

(commande mongo)

```
rs.initiate({
    _id : "rs-shinken",
    members: [
        { _id: 0, host: "ADRESSE_NODE_1:27018", priority: 2 },
        { _id: 1, host: "ADRESSE_NODE_2:27018" }
    ]
});
```

Dans cette commande :

- **ADRESSE_NODE_1** fait référence à l'adresse du nœud Primaire
- **ADRESSE_NODE_2** fait référence à l'adresse du nœud Secondaire

On ajoute l'Arbiter

(commande mongo)

```
rs.addArb("ADRESSE_NODE_3:27018")
```

Dans ces commandes :

- **ADRESSE_NODE_3** fait référence à l'adresse du nœud Arbiter

Dans Mongo, lorsqu'un Replica Set 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 Replica Set 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" : "ADRESSE_NODE_1:27018",
                        "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" : "ADRESSE_NODE_2:27018",
                        "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" : "ADRESSE_NODE_3:27018",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "ARBITER",
                        "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" / "SECONDARY" / "ARBITER".
- 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 d'accéder de manière automatique au bon nœud qui va traiter les requêtes. En effet, si le serveur Primaire est indisponible, alors le Secondaire va prendre le relai et il faudra donc rediriger toutes les requêtes sur ce serveur.

Pour résoudre cela, il faut mettre en place les démons mongos et le Sharding permettant de faire un routage des requêtes vers le bon nœud du Replica Set.

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

Note :

- Par défaut cette limite est à 10 secs.
- Il s'agit d'un timeout réseau.
- Bien que non recommandé, il est possible de changer ce timeout.
  - Il faudra alors changer les paramètres d'auto-reconnexion à la base de Shinken (*voir le chapitre* Comportement de Shinken avec un cluster MongoDB)
  - Il n'est pas recommandé de changer ce paramètre, car l'utilité de changer ce paramètre est de permettre une plus grande tolérance de la latence réseau entre les éléments du cluster. Or mettre en place un cluster MongoDB dans un réseau avec une grande latence n'est pas optimal, à cause du délai que cela provoque :
    - sur la réplication,
    - sur le temps d'indisponibilité de la base lorsque le nœud Primaire est indisponible (*le temps que le Secondaire et l'Arbiter trouve que le Primaire est indisponible*)
    - sur la charge réseau que va ajouter la réplication sur un réseau déjà lent.

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

Avant d'activer le routage des requêtes pour les démons mongod de notre cluster, il faut mettre en place les démons "serveurs de configuration" qui vont permettre de stocker et distribuer la configuration du Sharding au mongos.

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 configuration du démon de routage des requêtes Mongo (*mongos*).

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

RHEL / CentOS 7 ou RHEL / Alma / Rocky 8 ou RHEL / Alma / Rocky 9

- **(commande shell)**

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

Debian 13

- **(commande shell)**

```
mkdir /var/lib/mongodb/configdb
chown mongodb:mongodb /var/lib/mongodb/configdb
```

On copie le fichier de configuration par défaut.

Sur tous les serveurs :

(commande shell)

```
cp /etc/shinken-user-example/configuration/mongo/mongo-cluster/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.
```

Modification

1. **Vérifier** que le **port** est bien 27019.
2. **Vérifier** que les parties **"unixDomainSocket"** et **"enabled"** soient présentes,
   - sinon **les ajouter** en faisant attention à garder la même indentation.
3. Mettre **en commentaire** la ligne **bindIp** (*permet aux nœuds du cluster de communiquer entre eux. Le contrôle d'accès à la base se fera par la mise en place du firewall voir chapitre* Sécurisation du cluster).
4. **S'assurer** que l'option **replSetName** est **commentée** :
   - 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 Replica Set.

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 :

Activation au démarrage du système

RHEL / CentOS 7 ou RHEL / Alma / Rocky 8 ou RHEL / Alma / Rocky 9

- **(commande shell)**

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

Debian 13

- **(commande shell)**

```
systemctl enable mongo-configsrv
```

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

(commande shell)

```
service mongo-configsrv start
```

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

Il faut ajouter un démon mongos sur tous les serveurs avec démons Shinken nécessitant un accès à la base de donnée (*Synchronizer, Scheduler, Broker*).

1 - Sécurisation des communications entre les nœuds du cluster et les serveurs Shinken (entre mongod/mongo-configsrv et mongos)

Appliquer le chapitre "Sécurisation des communications entre les nœuds du cluster et les servers Shinken (entre mongod/mongo-configsrv et mongos)" pour le firewall choisi (*iptables ou firewalld*).

Si cela n'est pas fait, le démon mongos n'aura pas accès aux nœuds du cluster.

2 - Désactivation du mongod local

> Ne pas faire cette étape si les démons mongos et shinken, sont sur un nœud du cluster (*Primaire / Secondaire / Arbiter*)

Il faut désactiver le démon mongod local sur cette machine. En effet, les requêtes vont maintenant passer par le démon mongos sur le même port réseau.

RHEL / CentOS 7 ou RHEL / Alma / Rocky 8 ou RHEL / Alma / Rocky 9

- **(commande shell)**

```
service mongod stop
chkconfig mongod off
```

Debian 13

- **(commande shell)**

```
service mongod stop
systemctl disable mongod
```

3 - Mise en place du démon

Comme pour le démon mongo-configsrv, on commence par copier le fichier de configuration par défaut

Sur tous les serveurs :

(commande shell)

```
cp /etc/shinken-user-example/configuration/mongo/mongo-cluster/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: ADRESSE_NODE_1:27019,ADRESSE_NODE_2:27019,ADRESSE_NODE_3:27019
```

Modification

1. **Vérifier** que le **port** est bien 27017.
2. **Vérifier** que les parties **"unixDomainSocket**" et "**enabled**" **soit présents**,
   - sinon **les ajouter** en faisant attention à garder la même indentation que dans l'exemple ci-dessus.
3. **Vérifier** que **bindIp** vaut bien 127.0.0.1 et n'est **PAS** en commentaire (*permet limiter l'accès au mongos au démon de la machine. voir chapitre* Sécurisation du cluster).
4. **Ajouter** le bloc sharding pour lister les adresses des machines avec mongo-configsrv, comment dans l'exemple ci-dessus.
   - **ADRESSE_NODE_1 / ADRESSE_NODE_2 / ADRESSE_NODE_3** fait référence aux adresses des machines avec mongo-configsrv.

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 :

RHEL / CentOS 7 ou RHEL / Alma / Rocky 8 ou RHEL / Alma / Rocky 9

- **(commande shell)**

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

Debian 13

- **(commande shell)**

```
systemctl enable mongos
```

On démarre ensuite le mongos sur chaque machine.
Sur tous les serveurs :

(commande shell)

```
service mongos start
```

4 - Activation du routage (Sharding)

Une fois que les démons mongos sont en place, il faut activer le Sharding.

Sur un seul des serveurs Shinken avec un mongos faire :

(commande shell)

```
mongo
```

(commande mongo)

```
use admin
db.runCommand({addShard: "rs-shinken/ADRESSE_NODE_1:27018,ADRESSE_NODE_2:27018,ADRESSE_NODE_3:27018",
name:'shard-shinken'})
```

Dans cette commande :

- **ADRESSE_NODE_1 / ADRESSE_NODE_2 / ADRESSE_NODE_3** fait référence aux adresses des machines avec mongod (*Nœuds et Arbiter MongoDB*).

La configuration du cluster MongoDB est maintenant terminée !

La section suivante permet :

- de vérifier que la communication entre les démons du cluster MongoDB s'effectue correctement
- de visualiser l'état du cluster.

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

Etape 7: 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 Replica Set.

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/ADRESSE_NODE_1:27018,ADRESSE_NODE_2:27018,ADRESSE_NODE_3:27018" }
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 Replica Set

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

Sur n'importe quel nœud du cluster :

(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" : "ADRESSE_NODE_1:27018",
                        "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" : "ADRESSE_NODE_2:27018",
                        "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" : "ADRESSE_NODE_3:27018",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "ARBITER",
                        "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 8: 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).

Etape 9: Activer l'authentification par mot de passe au cluster MongoDB

L'authentification par mot de passe peut être activée sur MongoDB afin de restreindre l'accès aux données uniquement à Shinken (*voir la page* Activer l'authentification par mot de passe dans un cluster MongoDB).

Cas spécial des mongos (démon de routage)

Gestion des requêtes longues

Shinken livre une version de mongos avec une fonctionnalité de gestion des requêtes longues.

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 requêtes longues permet d'éviter cela en envoyant des sondes vers le nœud primaire, ce qui permet de vérifier que la machine distante est toujours accessible.

Configuration

Shinken livre sa propre 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 mongos réagira vis-à-vis des nœuds du cluster. Il est fortement déconseillé de changer ces valeurs sans le support Shinken

Nom	Type	Unité	Défaut	Description
`tools_used_by_shinken__mongo__tcp_keep_alive__enabled`	Booléen	---	**1**	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. Valeur possible : • 0 (*désactivé*) • 1 (*activé*) Si ce paramètre est désactivé, tous les paramètres suivants seront ignorés.
`tools_used_by_shinken__mongo__tcp_keep_alive__time_before_idle`	Entier	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`	Entier	---	**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`	Entier	Seconde	**5**	Temps d'attente 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**	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. Valeur possible : • 0 (*désactivé*) • 1 (*activé*)

Comportement de Shinken avec un cluster MongoDB

L'utilisation d'un cluster MongoDB renforce la stabilité de Shinken Entreprise.

Lorsque le nœud principal du cluster MongoDB devient indisponible, MongoDB déclenche l'élection d'un nouveau nœud principal. Cette opération entraîne une brève indisponibilité de la base de données, le temps qu'un nouveau nœud primaire soit élu. Pour minimiser l'impact de cette période d'indisponibilité sur le fonctionnement de Shinken, un mécanisme automatique de reconnexion à la base est intégré.

Ce mécanisme est contrôlé par les paramètres **auto_reconnect_max_try** et **auto_reconnect_sleep_between_try**, disponibles pour tous les modules et démons se connectant à la base de données (*voir le chapitre suivant*).

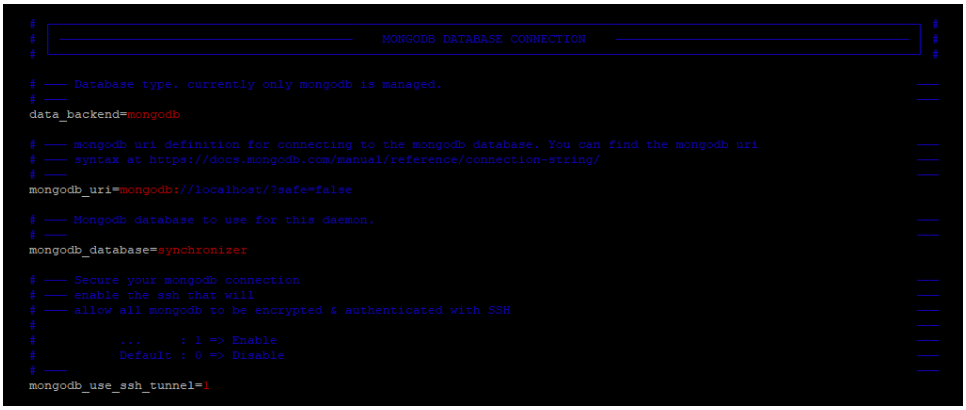
Il est recommandé de modifier ces valeurs uniquement si le paramètre **heartbeatTimeoutSecs**, fixé par défaut à 10 secondes, a été ajusté. Ce paramètre détermine le délai avant qu'un nœud du cluster MongoDB soit considéré comme indisponible, ce qui peut être utile en cas de latence élevée entre les éléments du cluster.

Pour garantir une reconnexion efficace, la valeur de **auto_reconnect_max_try * auto_reconnect_sleep_between_try** doit toujours être supérieure au paramètre **heartbeatTimeoutSecs**.

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*).



Liste non exhaustive des modules utilisant mongo :

- Module de Broker
  - event-manager-writer
  - sla
  - broker-module-livedata
    - livedata-module-sla-provider
  - WebUI
    - MongoDB
    - SLA (WebUI)
    - event-manager-reader
- Module de Scheduler
  - MongodbRetention
- Synchronizer
  - le démon lui-même (*dans synchronizer.cfg*)
  - synchronizer-collector-linker
  - discovery-import
  - synchronizer-module-database-backup

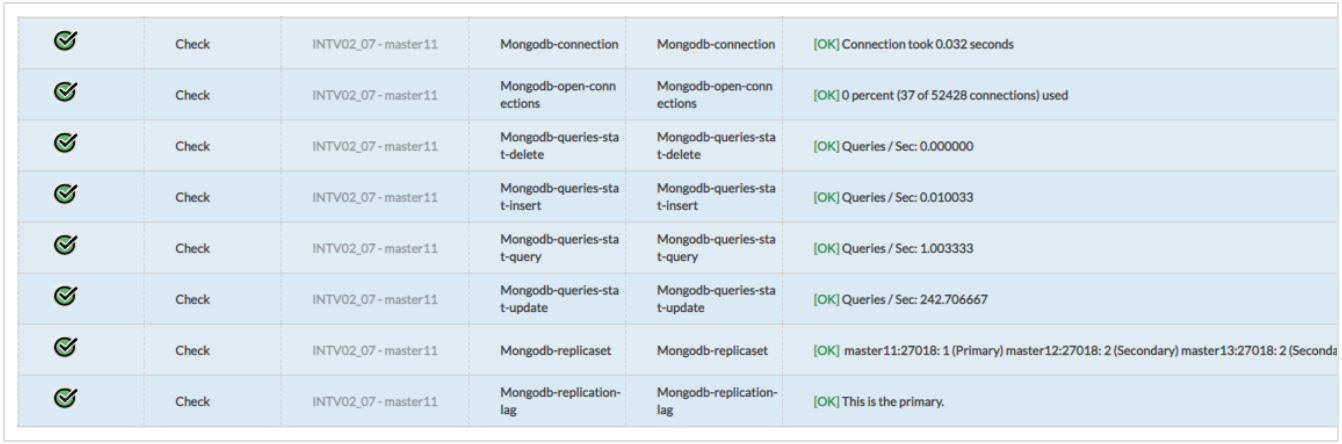
Supervision du cluster Mongo

Un cluster Mongo peut être supervisé avec Shinken. Shinken Entreprise met à disposition un pack MongoDB (*voir la page* Pack MongoDB)

Le modèle d'hôte "**mongodb3**" 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 à chaque nœud du cluster. Chaque hôte aura le modèle "**mongodb3**" 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 :



	Check	INTV02_07 - master11	Mongodb-connection	Mongodb-connection	[OK] Connection took 0.032 seconds
	Check	INTV02_07 - master11	Mongodb-open-connections	Mongodb-open-connections	[OK] 0 percent (37 of 52428 connections) used
	Check	INTV02_07 - master11	Mongodb-queries-stat-delete	Mongodb-queries-stat-delete	[OK] Queries / Sec: 0.000000
	Check	INTV02_07 - master11	Mongodb-queries-stat-insert	Mongodb-queries-stat-insert	[OK] Queries / Sec: 0.010033
	Check	INTV02_07 - master11	Mongodb-queries-stat-query	Mongodb-queries-stat-query	[OK] Queries / Sec: 1.003333
	Check	INTV02_07 - master11	Mongodb-queries-stat-update	Mongodb-queries-stat-update	[OK] Queries / Sec: 242.706667
	Check	INTV02_07 - master11	Mongodb-replicaset	Mongodb-replicaset	[OK] master11:27018: 1 (Primary) master12:27018: 2 (Secondary) master13:27018: 2 (Seconda
	Check	INTV02_07 - master11	Mongodb-replication-lag	Mongodb-replication-lag	[OK] This is the primary.

Le modèle d'hôte **mongodb3** utilise par défaut une connexion directe aux démons Mongo. Suite à la configuration d'un firewall pour sécuriser le cluster, les requêtes depuis les Poller risquent d'être refusées. Il faut ajouter les serveurs des Poller dans les règles de firewall, ou changer la méthode de connexion. Le modèle 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 être configurés en modifiant les données **MONGO_SSH_USER** et **MONGO_SSH_KEY** sur l'hôte.

Le modèle **mongodb3** 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 d'un cluster, on modifie le port utilisé par ce démon pour le port 27018. Pour pouvoir le superviser dans Shinken, il est nécessaire de mettre à jour la donnée **MONGO_PORT** sur l'hôte.

La configuration et le fonctionnement du pack MongoDB sont disponibles 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).