

# Haute disponibilité de la base Mongo

## Introduction

De nombreux composants de Shinken Entreprise peuvent être installés et configurés de manière à former une architecture hautement disponible. Dans le page [Haute disponibilité des démons de Shinken Entreprise](#) sont décrites les procédures permettant de mettre en place des démons Shinken de remplacement pour améliorer la disponibilité.

Cependant, un certain nombre de ces démons et leurs modules associés s'appuient sur une base Mongo 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 Mongo.

## Remarques préliminaires

### Lexique

Dans cette documentation est décrite de manière détaillée comment mettre en place des mécanismes de haute disponibilité pour la base Mongo.

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:

- **Primaire**: nom de Mongo pour désigner un serveur maître
- **Secondaire**: nom de Mongo pour désigner un Spare
- **Replicat Set**: nom de Mongo pour désigner la haute disponibilité
- **Sharding**: nom de Mongo pour désigner le load balancing
- **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  $N/2 + 1$  nœuds dans un cluster de  $N$  nœuds.

## Commandes à lancer

Dans la procédure d'installation, des commandes doivent être lancées. On distingue ces commandes en 2 types:

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

Introduction
Remarques préliminaires
Lexique
Commandes à lancer
Nuance importante
Architecture mise en place
Détails sur les démons Mongo
Prérequis pour l'installation
Procédure de configuration
Etape 1: Installation de Shinken
Etape 2: Sécurisation des communications entre les daemons mongo
Etape 3: Mise en place des démons de stockage des données
Etape 4: Déclaration du replicaset dans Mongo
Etape 5: Mise en place des démons de gestion de la configuration
Etape 6: Mise en place des démons de routage des requêtes Mongo
Etape 7: Vérification du bon fonctionnement du cluster
Comportement de Shinken avec un cluster Mongo
Supervision du cluster Mongo
Introduction
Remarques préliminaires
Lexique
Commandes à lancer
Nuance importante
Architecture mise en place
Détails sur les démons Mongo
Prérequis pour l'installation
Procédure de configuration
Etape 1: Installation de Shinken
Etape 2: Sécurisation des communications entre les daemons mongo
Etape 3: Mise en place des démons de stockage des données
Etape 4: Déclaration du replicaset dans Mongo
Etape 5: Mise en place des démons de gestion de la configuration
Etape 6: Mise en place des démons de routage des requêtes Mongo
Etape 7: Vérification du bon fonctionnement du cluster
Comportement de Shinken avec un cluster Mongo
Supervision du cluster Mongo

(commande shell)

```
echo "commande shell"
```

- Les commandes Mongo doivent elles être lancées dans un shell Mongo (les instructions pour ouvrir le bon shell Mongo seront présentes avant ces commandes). Elles seront présentées comme suivant:

(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)**.

Un load balancing entre plusieurs nœuds Mongo permet d'améliorer la disponibilité de la base lorsqu'elle est soumise à des contraintes importantes (de nombreux utilisateurs en parallèle) en répartissant la charge mise sur chaque nœud Mongo.

On se concentre ici plutôt 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 **reparties** sur plusieurs nœuds.

Cette nuance a la conséquence suivante sur l'architecture mise en place:

- Puisque chaque nœud Mongo contient l'ensemble des données, il faut que **chacun des nœuds** puisse supporter la **charge complète** puisque la base est répliquée entièrement sur chaque nœud.



Il est donc conseillé d'avoir des machines identiques pour chaque nœud du cluster Mongo

## Architecture mise en place

### Détails sur les démons Mongo

Dans une installation Mongo classique, on possède sur un machine 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 Mongo sur plusieurs machines, qu'on appelle "nœuds" dans la suite de cette documentation.

Chaque nœud Mongo possède plusieurs démons Mongo qui permettent de gérer la base de données. Dans cette installation, on a 3 nœuds:

- Un nœud primaire, qui ordonnance la réplication et gère la configuration du cluster Mongo
- 2 noeuds 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:

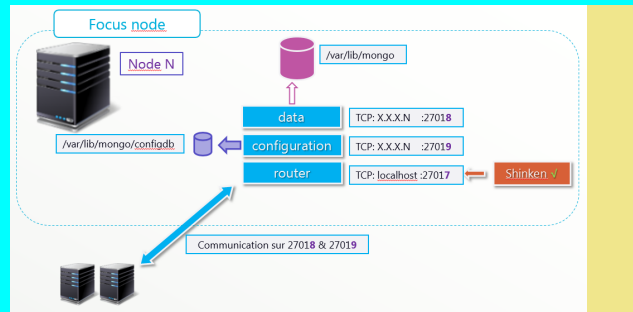
- **mongod**
  - Comme dans une installation Mongo classique, le démon mongod se charge du stockage des données
  - Il s'assure que les données sont bien répliquées sur les autres nœuds du cluster.
- **mongo-configsrv**
  - Ce démon gère la configuration du cluster Mongo sur chaque nœud. Les autres démons Mongo s'appuient sur ce démon pour avoir des informations sur le cluster et sa configuration pour fonctionner.
- **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 seulement les requêtes locales

Un nœud du cluster Mongo a alors l'architecture décrite dans le schéma ci-contre.

Dans ce schéma, qui représente un seul nœud du cluster, 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 qui est utilisé dans une architecture classique par le démon mongod. Cette configuration permet à Shinken d'effectuer ses requêtes Mongo de manière locale sur le port 27017 comme d'habitude. Le démon mongos va ensuite s'occuper du routage de la requête pour que cette donnée soit stockée correctement dans le cluster.

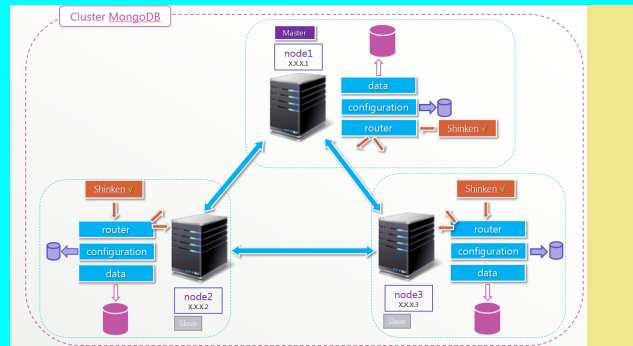


Chaque nœud du cluster interagit comme décrit sur le schéma ci-contre.

Dans ce schéma, chaque nœud constituant du cluster possède les démons décrits précédemment. Shinken effectue ses requêtes de manière locale sur le démon mongos, qui ensuite s'occupe de l'aspect cluster de Mongo et du bon traitement et acheminement des données.

**Du point de vue de Shinken, la transformation de Mongo en un cluster Mongo 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.



## 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
- 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.
- Les 2 serveurs secondaires du cluster Mongo doivent être vidés de toutes 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 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 eux seuls, 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 VM de ne pas les mettre sur le même hyperviseur. En effet, un problème sur l'hyperviseur pourrait alors affecter toutes les VM 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.
- Avant de commencer la mise en place des démons Mongo, on crée une clé d'authentification qui sera utilisée par les démons Mongo.
- 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 de la même architecture.
- 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 Mongo est avant tout l'installation de Mongo. Cette documentation s'appuie sur la version de Mongo installée par Shinken (v2.6.9).

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: [Guide d'installation et de mise à jour](#)

### Etape 2: Sécurisation des communications entre les daemons mongo

Pour sécuriser la communication entre les serveurs Mongo du cluster, deux solutions existent:

- on crée une clé qui sera partagée et utilisée par les différents démons de Mongo
- on mets en place des règles de firewall afin que seuls les serveurs mongo puissent échanger sur leurs port de communications

#### Mise en place d'une clé partagée entre les daemons mongo

- **Sur le serveur primaire**, on commence par créer la clé:

(commande shell)

```
openssl rand -base64 756 > /etc/mongod.keyfile
```

- A l'aide de votre outil de copie favori (scp par exemple), copier cette clé sur les autres serveurs du cluster

(commande shell)

```
scp -p /etc/mongod.keyfile root@node2:/etc/mongod.keyfile  
scp -p /etc/mongod.keyfile root@node3:/etc/mongod.keyfile
```

- **Sur tous les serveurs**, donner les droits adéquats au fichier de clé:

(commande shell)

```
chown mongod:mongod /etc/mongod.keyfile  
chmod 400 /etc/mongod.keyfile
```

## Mise en place de règles de firewall

Sur tous les serveurs, nous n'allons autoriser les communications vers les ports **27018** et **27019** (les seuls à être exposés sur le réseau dans une installation avec mongos) qu'entre les 3 serveurs composants notre cluster.

- Nous activons le firewall et créons une nouvelle zone qui gère nos règles mongo

(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 # create a new chain

##
# Attention à bien changer les ADRESSE 1 à 3
##
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 les ADRESSE 1 à 3

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

Une fois la clé d'authentification créée et copiée sur tous les serveurs du cluster, on peut commencer la mise en place des démons Mongo.

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, écoute sur toutes les interfaces réseau, déclare son appartenance au replicaset et qu'il utilise la clé définie précédemment.

Sur tous les serveurs:

(commande shell)

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

/etc/mongod.conf

```
# Changer le port d'écoute pour écouter sur 27018
port=27018
```

```
# Commenter la ligne bind_ip pour écouter sur toutes les interfaces réseau et pas seulement l'interface locale (faire commencer la ligne par #)
#bind_ip=127.0.0.1

# Déclarer que le démon mongo appartient au replicaset, qu'on va appeler "rs-shinken"
replSet=rs-shinken

# Utiliser la clé créée dans l'étape 2
keyFile=/etc/mongod.keyfile
```

- 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** ont été configurés et démarrés. Il 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 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

Une fois les démons mongod configurés et démarrés sur chaque serveur du cluster Mongo, il faut 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
```

(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 la commande précédente, *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 de l'algorithme d'élection.

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

(commande mongo)

```
rs.status()
```

Le prompt du shell Mongo devrait également avoir changé pour afficher les mentions "PRIMARY" ou "SECONDARY" dans les shells Mongo. 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()
```

A 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 Mongo.

## Étape 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 qui vont permettre à d'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 de 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 est désigné par "tarball\_shinken".  
**Sur tous les serveurs:**

(commande shell)

```
cp tarball_shinken/tools/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  
#bind_ip=127.0.0.1
```



### Piège

Le fichier de configuration */etc/mongo-configsrv.conf* comporte également une option *replSet*. 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 *rep/Set* des fichiers de configuration des démons *mongo-configsrv* soit bien commentée et non utilisée.

- On copie ensuite le script d'init du démon et on ajoute 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)

```
cp tarball_shinken/tools/mongo-cluster/mongo-configsrv /etc/init.d/mongo-configsrv
chmod a+x /etc/init.d/mongo-configsrv
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 Mongo

Sur toutes les machines, 2 démons (*mongod* et *mongo-configsrv*) ont été mis en place. Les données peuvent correctement être stockées dans le cluster, et la configuration du cluster va pouvoir être gérée par les démons *mongo-configsrv*. 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 Mongo.

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

**Sur tous les serveurs:**

(commande shell)

```
cp tarball_shinken/tools/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
configdb=node1:27019,node2:27019,node3:27019
```

- On copie ensuite le script d'init du démon et on ajoute 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)

```
cp tarball_shinken/tools/mongo-cluster/mongos /etc/init.d/mongos
chmod a+x /etc/init.d/mongos
chkconfig --add mongos
chkconfig mongos on
```

- On démarre ensuite le *mongos* sur chaque machine.

**Sur tous les serveurs:**

(commande shell)

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

Les démons *mongos* sont maintenant en place sur chaque machine du cluster Mongo. Il reste à déclarer à Mongo les serveurs de données qui constituent le cluster (c'est 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 Mongo est maintenant terminée !

La section suivante permet de vérifier que la communication entre les démons du cluster Mongo s'effectue correctement et de visualiser l'état du cluster

## 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 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("5aeb6d7e4edc5bd313ba7b7") }
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, quel nœud est actuellement le nœud primaire

Sur n'importe quel serveur:

(commande shell)

```
mongo --port 27018
```

(commande mongo)

```
sh.status()
```

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

```
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
}
```

## Comportement de Shinken avec un cluster Mongo

L'utilisation d'un cluster Mongo permet d'améliorer la stabilité de Shinken Entreprise. Lorsque le nœud principal du cluster Mongo entre en erreur, Mongo procède à l'élection d'un nouveau nœud principal dans le cluster. A 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 Mongo, 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 Mongo 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 30s.
- Le module SLA du Broker redémarre tant que la base Mongo est indisponible et reprend ses tâches dès que Mongo redevient disponible.

## Supervision du cluster Mongo

Un cluster Mongo peut être supervisé avec Shinken.

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 **7018** 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	Shinken-local-mongo-2-ALL-localhost	Mongodb-replicaset	Mongodb-replicaset	OK - State: 1 (Primary)
✓	Check	Shinken-local-mongo-1-ALL-localhost	Mongodb-replication-lag	Mongodb-replication-lag	OK - This is the primary.
✓	Check	local-mongo2	Mongodb-replicaset	Mongodb-replicaset	OK - State: 2 (Secondary)
✓	Check	local-mongo2	Mongodb-replication-lag	Mongodb-replication-lag	OK - Lag is 0.0 seconds

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 et préciser 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.

## Introduction

De nombreux composants de Shinken Entreprise peuvent être installés et configurés de manière à former une architecture hautement disponible. Dans le page [Haute disponibilité des démons de Shinken Entreprise](#) sont décrites les procédures permettant de mettre en place des démons Shinken de remplacement pour améliorer la disponibilité.

Cependant, un certain nombre de ces démons et leurs modules associés s'appuient sur une base Mongo 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 Mongo.

Introduction
Remarques préliminaires
Lexique
Commandes à lancer
Nuance importante
Architecture mise en place
Détails sur les démons Mongo
Prérequis pour l'installation
Procédure de configuration
Etape 1: Installation de Shinken
Etape 2: Sécurisation des communications entre les daemons mongo
Etape 3: Mise en place des démons de stockage des données
Etape 4: Déclaration du replicaset dans Mongo
Etape 5: Mise en place des démons de gestion de la configuration
Etape 6: Mise en place des démons de routage des requêtes Mongo
Etape 7: Vérification du bon fonctionnement du cluster
Comportement de Shinken avec un cluster Mongo
Supervision du cluster Mongo
Introduction
Remarques préliminaires
Lexique
Commandes à lancer
Nuance importante
Architecture mise en place
Détails sur les démons Mongo
Prérequis pour l'installation
Procédure de configuration
Etape 1: Installation de Shinken
Etape 2: Sécurisation des communications entre les daemons mongo
Etape 3: Mise en place des démons de stockage des données
Etape 4: Déclaration du replicaset dans Mongo
Etape 5: Mise en place des démons de gestion de la configuration
Etape 6: Mise en place des démons de routage des requêtes Mongo
Etape 7: Vérification du bon fonctionnement du cluster
Comportement de Shinken avec un cluster Mongo
Supervision du cluster Mongo

## Remarques préliminaires

## Lexique

Dans cette documentation est décrite de manière détaillée comment mettre en place des mécanismes de haute disponibilité pour la base Mongo.

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:

- **Primaire:** nom de Mongo pour désigner un serveur maître
- **Secondaire:** nom de Mongo pour désigner un Spare
- **Replicat Set:** nom de Mongo pour désigner la haute disponibilité
- **Sharding:** nom de Mongo pour désigner le load balacing
- **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  $\lfloor \frac{N}{2} + 1 \rfloor$  nœuds dans un cluster de  $N$  nœuds.

## Commandes à lancer

Dans la procédure d'installation, des commandes doivent être lancées. On distingue ces commandes en 2 types:

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

(commande shell)

```
echo "commande shell"
```

- Les commandes Mongo doivent elles être lancées dans un shell Mongo (les instructions pour ouvrir le bon shell Mongo seront présentes avant ces commandes). Elles seront présentées comme suivant:

(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)**.

Un load balancing entre plusieurs nœuds Mongo permet d'améliorer la disponibilité de la base lorsqu'elle est soumise à des contraintes importantes (de nombreux utilisateurs en parallèle) en répartissant la charge mise sur chaque nœud Mongo.

On se concentre ici plutôt 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 **reparties** sur plusieurs nœuds.

Cette nuance a la conséquence suivante sur l'architecture mise en place:

- Puisque chaque nœud Mongo contient l'ensemble des données, il faut que **chacun des nœuds** puisse supporter la **charge complète** puisque la base est répliquée entièrement sur chaque nœud.



Il est donc conseillé d'avoir des machines identiques pour chaque nœud du cluster Mongo

## Architecture mise en place

### Détails sur les démons Mongo

Dans une installation Mongo classique, on possède sur une machine 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 Mongo sur plusieurs machines, qu'on appelle "nœuds" dans la suite de cette documentation.

Chaque nœud Mongo possède plusieurs démons Mongo qui permettent de gérer la base de données. Dans cette installation, 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:

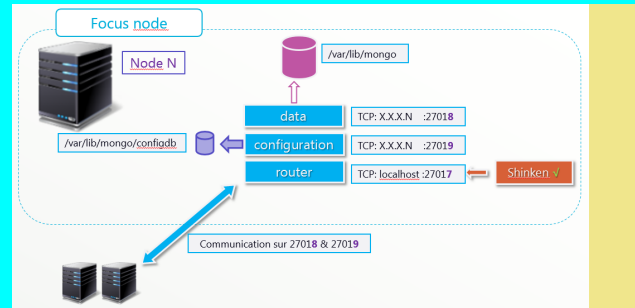
- **mongod**
  - Comme dans une installation Mongo classique, le démon mongod se charge du stockage des données

- Il s'assure que les données sont bien répliquées sur les autres nœuds du cluster.
- **mongo-configsrv**
  - Ce démon gère la configuration du cluster Mongo sur chaque nœud. Les autres démons Mongo s'appuient sur ce démon pour avoir des informations sur le cluster et sa configuration pour fonctionner.
- **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 seulement les requêtes locales

Un nœud du cluster Mongo a alors l'architecture décrite dans le schéma ci-contre.

Dans ce schéma, qui représente un seul nœud du cluster, 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 qui est utilisé dans une architecture classique par le démon mongod. Cette configuration permet à Shinken d'effectuer ses requêtes Mongo de manière locale sur le port 27017 comme d'habitude. Le démon mongos va ensuite s'occuper du routage de la requête pour que cette donnée soit stockée correctement dans le cluster.

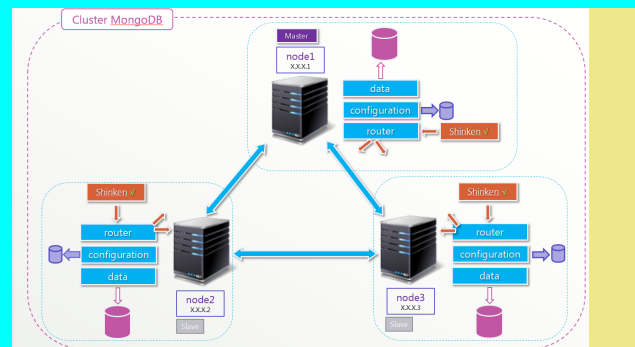


Chaque nœud du cluster interagit comme décrit sur le schéma ci-contre.

Dans ce schéma, chaque nœud constituant du cluster possède les démons décrits précédemment. Shinken effectue ses requêtes de manière locale sur le démon mongos, qui ensuite s'occupe de l'aspect cluster de Mongo et du bon traitement et acheminement des données.

**Du point de vue de Shinken, la transformation de Mongo en un cluster Mongo 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.



## 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
- 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.
- Les 2 serveurs secondaires du cluster Mongo doivent être vidés de toutes 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 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 eux seuls, 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 VM de ne pas les mettre sur le même hyperviseur. En effet, un problème sur l'hyperviseur pourrait alors affecter toutes les VM 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.
- Avant de commencer la mise en place des démons Mongo, on crée une clé d'authentification qui sera utilisée par les démons Mongo.
- 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 de la même architecture.
- 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 Mongo est avant tout l'installation de Mongo. Cette documentation s'appuie sur la version de Mongo installée par Shinken (v2.6.9).

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: [Guide d'installation et de mise à jour](#)

### Etape 2: Sécurisation des communications entre les daemons mongo

Pour sécuriser la communication entre les serveurs Mongo du cluster, deux solutions existent:

- on crée une clé qui sera partagée et utilisée par les différents démons de Mongo
- on met en place des règles de firewall afin que seuls les serveurs mongo puissent échanger sur leurs port de communications

### Mise en place d'une clé partagée entre les daemons mongo

- **Sur le serveur primaire**, on commence par créer la clé:

(commande shell)

```
openssl rand -base64 756 > /etc/mongod.keyfile
```

- A l'aide de votre outil de copie favori (scp par exemple), copier cette clé sur les autres serveurs du cluster

(commande shell)

```
scp -p /etc/mongod.keyfile root@node2:/etc/mongod.keyfile  
scp -p /etc/mongod.keyfile root@node3:/etc/mongod.keyfile
```

- **Sur tous les serveurs**, donner les droits adéquats au fichier de clé:

(commande shell)

```
chown mongod:mongod /etc/mongod.keyfile
chmod 400 /etc/mongod.keyfile
```

## Mise en place de règles de firewall

**Sur tous les serveurs**, nous n'allons autoriser les communications vers les ports **27018** et **27019** (les seuls à être exposés sur le réseau dans une installation avec mongos) qu'entre les 3 serveurs composants notre cluster.

- Nous activons le firewall et créons une nouvelle zone qui gère nos règles mongo (attention à bien changer les ADRESSE 1 à 3

(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 # create a new chain

##
# Attention à bien changer les ADRESSE 1 à 3
##
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
```

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

Une fois la clé d'authentification créée et copiée sur tous les serveurs du cluster, on peut commencer la mise en place des démons Mongo.

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, écoute sur toutes les interfaces réseau, déclare son appartenance au replicaset et qu'il utilise la clé définie précédemment.  
**Sur tous les serveurs:**

(commande shell)

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

/etc/mongod.conf

```
# Changer le port d'écoute pour écouter sur 27018
port=27018

# Commenter la ligne bind_ip pour écouter sur toutes les interfaces réseau et pas seulement l'interface
locale (faire commencer la ligne par #)
#bind_ip=127.0.0.1

# Déclarer que le démon mongo appartient au replicaset, qu'on va appeler "rs-shinken"
replSet=rs-shinken

# Utiliser la clé créée dans l'étape 2
keyFile=/etc/mongod.keyfile
```

- 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** ont été configurés et démarrés. Il 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 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.

## Étape 4: Déclaration du replicaset dans Mongo

Une fois les démons mongod configurés et démarrés sur chaque serveur du cluster Mongo, il faut 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
```

(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 la commande précédente, *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 de l'algorithme d'élection.

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

(commande mongo)

```
rs.status()
```

Le prompt du shell Mongo devrait également avoir changé pour afficher les mentions "PRIMARY" ou "SECONDARY" dans les shells Mongo.

- 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()
```

A 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 Mongo.

## 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 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 de 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 est désigné par "tarball\_shinken".

**Sur tous les serveurs:**

(commande shell)

```
cp tarball_shinken/tools/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
#bind_ip=127.0.0.1
```



### Piège

Le fichier de configuration `/etc/mongo-configsrv.conf` comporte également une option `rep/Set`. 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 `rep/Set` des fichiers de configuration des démons `mongo-configsrv` soit bien commentée et non utilisée.**

- On copie ensuite le script d'init du démon et on ajoute 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)

```
cp tarball_shinken/tools/mongo-cluster/mongo-configsrv /etc/init.d/mongo-configsrv
chmod a+x /etc/init.d/mongo-configsrv
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 Mongo

Sur toutes les machines, 2 démons (`mongod` et `mongo-configsrv`) ont été mis en place. Les données peuvent correctement être stockées dans le cluster, et la configuration du cluster va pouvoir être gérée par les démons `mongo-configsrv`. 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 Mongo.

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

**Sur tous les serveurs:**

(commande shell)

```
cp tarball_shinken/tools/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
configdb=node1:27019,node2:27019,node3:27019
```

- On copie ensuite le script d'init du démon et on ajoute 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)

```
cp tarball_shinken/tools/mongo-cluster/mongos /etc/init.d/mongos
chmod a+x /etc/init.d/mongos
chkconfig --add mongos
chkconfig mongos on
```

- On démarre ensuite le `mongos` sur chaque machine.

**Sur tous les serveurs:**

(commande shell)

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

Les démons mongos sont maintenant en place sur chaque machine du cluster Mongo. Il reste à déclarer à Mongo les serveurs de données qui constituent le cluster (c'est ces informations qui seront stockées par le démon mongo-configsvr).

**Sur le serveur primaire:**

(commande shell)

```
mongo
```

(commande mongo)

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

La configuration du cluster Mongo est maintenant terminée !

La section suivante permet de vérifier que la communication entre les démons du cluster Mongo s'effectue correctement et de visualiser l'état du cluster

## 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 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, quel nœud est actuellement le nœud primaire

Sur n'importe quel serveur:

(commande shell)

```
mongo --port 27018
```

(commande mongo)

```
sh.status()
```

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

```
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
}

```

## Comportement de Shinken avec un cluster Mongo

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

Cette coupure à 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 Mongo, 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 Mongo 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 30s.
- Le module SLA du Broker redémarre tant que la base Mongo est indisponible et reprend ses tâches dès que Mongo redevient disponible.

## Supervision du cluster Mongo

Un cluster Mongo peut être supervisé avec Shinken.

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 **7018** 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	Shinken-local-mongo 1-ALL-localhost	Mongodb-replicaset	Mongodb-replicaset	OK - State: 1 (Primary)
✓	Check	Shinken-local-mongo 1-ALL-localhost	Mongodb-replication-lag	Mongodb-replication-lag	OK - This is the primary.

✓	Check	local-mongo-2	Mongodb-replicaset	Mongodb-replicaset	OK - State: 2 (Secondary)
✓	Check	local-mongo-2	Mongodb-replication-lag	Mongodb-replication-lag	OK - Lag is 0.0 seconds

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 et préciser 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.