SÉCURITÉ DES DONNÉES

Sécurité MongoDB

Authentification, autorisation et gestion des droits d'accès

Principes et exemples pratiques — RBAC, rôles intégrés et personnalisés, chiffrement, isolation réseau et bonnes pratiques de production

Sommaire

1 Pourquoi sécuriser MongoDB ?

Une base non protégée est une porte ouverte

Par défaut, une instance MongoDB fraîchement installée n'active ni authentification ni contrôle d'accès. N'importe quel client pouvant atteindre le port 27017 peut lire, modifier ou supprimer toutes les données — et effacer les traces derrière lui.

Rappel historique

Des vagues d'attaques ont ciblé des milliers d'instances MongoDB exposées sans authentification sur Internet : données effacées et rançon exigée pour leur restitution.

Le réflexe à avoir

Considérer la sécurité dès l'installation, pas comme une étape ajoutée après coup en production.

Ce module couvre les trois piliers : authentification (qui se connecte), autorisation (ce qu'il a le droit de faire), et protection du canal (réseau, chiffrement).

2 Les couches de sécurité (defense in depth)

Aucune mesure seule ne suffit — elles se cumulent

1	**Authentification**	Vérifier l'identité : qui se connecte au serveur ?
2	**Autorisation (RBAC)**	Vérifier les droits : que peut faire cet utilisateur ?
3	**Chiffrement**	Protéger les données en transit (TLS) et au repos
4	**Isolation réseau**	Limiter qui peut même atteindre le serveur
5	**Audit**	Tracer qui a fait quoi, pour investiguer après coup

3 Activer l'authentification

Aucun contrôle d'accès n'existe tant qu'elle n'est pas activée

Option 1 — fichier de configuration

```
# mongod.conf
security:
  authorization: enabled
```

Option 2 — ligne de commande

```
mongod --auth --dbpath /data/db
```

Ce que ça change concrètement

Avant : n'importe quel client connecté au port peut tout faire.

Après : chaque connexion doit s'authentifier avec des identifiants valides, et chaque action est ensuite vérifiée contre les rôles de l'utilisateur (module suivant).

⚠️ *Redémarrer mongod après modification — l'authentification n'est pas rétroactive sur une session déjà ouverte.*

4 Le premier utilisateur administrateur

L'exception localhost pour amorcer la sécurité

L'exception localhost

Une fois --auth activé mais avant qu'aucun utilisateur n'existe, MongoDB autorise une connexion locale (localhost) sans identifiants — le temps de créer le premier compte administrateur. Cette exception se ferme automatiquement dès qu'un utilisateur existe.

```
// connecté en local, avant tout utilisateur créé
use admin
db.createUser({
  user: "admin",
  pwd: "un_mot_de_passe_tres_fort",
  roles: [{ role: "root", db: "admin" }]
})
```

Le rôle « root » donne un accès total — à réserver strictement à ce compte d'amorçage, jamais aux comptes applicatifs (voir module 10).

5 Le modèle RBAC

Role-Based Access Control : users, roles, privileges

User

Un compte identifié par des identifiants (utilisateur/mot de passe ou certificat), rattaché à une base.

Role

Un ensemble nommé de privilèges. Un utilisateur peut avoir plusieurs rôles, sur plusieurs bases.

Privilege

Une action autorisée (find, insert, update…) sur une ressource précise (collection, base, cluster).

Un utilisateur possède un ou plusieurs rôles → chaque rôle regroupe un ou plusieurs privilèges → chaque privilège autorise des actions précises sur une ressource précise.

6 Rôles intégrés — base de données

Les rôles prêts à l'emploi les plus courants

Rôle	Portée
`read`	Lecture seule sur une base
`readWrite`	Lecture et écriture sur une base
`dbAdmin`	Administration d'une base (index, statistiques) — pas de lecture des données
`userAdmin`	Créer/modifier des utilisateurs et rôles sur une base
`dbOwner`	Combine readWrite + dbAdmin + userAdmin sur une base

Chaque rôle « base » peut être accordé sur une base précise, ou sur toutes les bases avec la variante readAnyDatabase / readWriteAnyDatabase.

7 Rôles intégrés — cluster et sauvegarde

Pour les tâches d'administration transverses

Rôle	Portée
`clusterAdmin`	Administration du cluster entier (réplication, sharding)
`backup`	Autorise les opérations de sauvegarde (mongodump et équivalents)
`restore`	Autorise les opérations de restauration
`root`	Accès total — combine tous les rôles administrateurs

À retenir

Ces rôles à large portée ne doivent être accordés qu'à un nombre restreint de comptes d'administration humains, jamais à un compte applicatif — c'est l'objet du principe du moindre privilège (module 10).

8 Créer un utilisateur applicatif

Un compte dédié, aux droits limités à son besoin réel

```
use boutique
db.createUser({
  user: "app_boutique",
  pwd: "mot_de_passe_fort_dedie",
  roles: [
    { role: "readWrite", db: "boutique" }
  ]
})
```

Pourquoi un compte par application

Ce compte ne peut lire ni écrire que dans la base boutique — aucun accès aux autres bases du serveur, ni aux commandes d'administration.

Si les identifiants de l'application fuient, l'impact reste circonscrit à cette seule base.

```
Se connecter avec ce compte : mongosh --username app_boutique --password --authenticationDatabase boutique
```

9 Rôles personnalisés (custom roles)

Quand les rôles intégrés sont trop larges

```
use boutique
db.createRole({
  role: "lecteur_catalogue",
  privileges: [
    {
      resource: { db: "boutique", collection: "produits" },
      actions: ["find"]
    }
  ],
  roles: []
})
```

Ce rôle personnalisé n'autorise que la lecture (find) de la seule collection produits — même readWrite serait trop permissif pour un service qui ne fait que consulter le catalogue.

10 Principe du moindre privilège

Donner uniquement ce qui est nécessaire — ni plus, ni « au cas où »

❌ À éviter

Un compte applicatif unique avec le rôle root, réutilisé pour tous les services.

Un même mot de passe partagé entre l'équipe de développement et la production.

Des comptes créés « au cas où » et jamais révoqués.

✔️ À privilégier

Un compte par service, avec un rôle scopé à la base (voire à la collection) qu'il utilise réellement.

Des comptes distincts entre environnements (dev, staging, production).

Une revue périodique des comptes et de leurs rôles.

11 Gérer les utilisateurs existants

Lister, modifier, révoquer, supprimer

```
// lister les utilisateurs de la base courante
db.getUsers()

// ajouter un rôle à un utilisateur existant
db.grantRolesToUser("app_boutique", [{ role: "read", db: "logs" }])

// retirer un rôle
db.revokeRolesFromUser("app_boutique", [{ role: "read", db: "logs" }])

// supprimer un utilisateur
db.dropUser("ancien_compte")
```

12 Mécanismes d'authentification

Plusieurs méthodes selon le contexte

Mécanisme	Usage typique
`SCRAM-SHA-256`	Mécanisme par défaut — utilisateur / mot de passe
`x.509`	Authentification par certificat — adaptée aux communications inter-serveurs
`LDAP (Enterprise)`	Délégation à un annuaire d'entreprise existant
`Kerberos (Enterprise)`	Authentification unifiée dans un environnement d'entreprise

SCRAM-SHA-256 convient à la grande majorité des usages ; x.509 est recommandé pour sécuriser la communication entre nœuds d'un Replica Set ou Cluster.

13 Chiffrement en transit (TLS/SSL)

Protéger les données qui circulent sur le réseau

```
# mongod.conf
net:
  tls:
    mode: requireTLS
    certificateKeyFile: /etc/ssl/mongodb.pem
    CAFile: /etc/ssl/ca.pem
```

Pourquoi c'est indispensable

Sans TLS, identifiants et données transitent en clair sur le réseau — un tiers en écoute (réseau partagé, cloud mal isolé) peut les intercepter.

mode: requireTLS impose le chiffrement pour toutes les connexions ; preferTLS l'autorise sans l'imposer — utile en transition, à éviter en cible finale.

14 Isolation réseau

Limiter qui peut même atteindre le serveur

```
# mongod.conf — n'écouter que sur le réseau interne
net:
  bindIp: 127.0.0.1,10.0.1.5
  port: 27017
```

bindIp

N'écouter que sur les interfaces réseau nécessaires — jamais 0.0.0.0 en production sans pare-feu strict en complément.

Pare-feu / VPC

Restreindre l'accès au port 27017 aux seules machines applicatives autorisées, via groupes de sécurité ou règles de pare-feu.

Rappel du module 1 : la majorité des incidents MongoDB proviennent d'instances exposées directement sur Internet, avec ou sans authentification.

15 Chiffrement au repos et audit

Protéger les données stockées, tracer les actions

Chiffrement au repos

Chiffre les fichiers de données sur disque (WiredTiger encryption at rest), pour qu'une copie physique du disque reste illisible sans la clé.

Disponible en édition Enterprise et sur MongoDB Atlas.

Audit

Journalise les opérations (connexions, requêtes, modifications de rôles) dans un log dédié, filtrable par utilisateur ou par action.

Disponible en édition Enterprise et sur MongoDB Atlas.

16 Rotation des identifiants

Un mot de passe ne devrait pas vivre indéfiniment

```
// changer le mot de passe d'un utilisateur
db.changeUserPassword("app_boutique", "nouveau_mot_de_passe_fort")
```

Bonnes pratiques

Faire tourner les mots de passe applicatifs périodiquement, et immédiatement après le départ d'une personne y ayant eu accès.

Secrets managers

En production, stocker les identifiants dans un gestionnaire de secrets (Vault, AWS Secrets Manager…) plutôt qu'en clair dans un fichier de configuration versionné.

17

Checklist sécurité production

Neuf points à vérifier avant la mise en production

1. Authentification activée (security.authorization: enabled)
2. Compte root réservé à l'amorçage, jamais utilisé en usage courant
3. Un compte applicatif par service, avec des rôles scopés
4. Rôles personnalisés pour les besoins fins (lecture seule sur une collection…)
5. TLS activé (requireTLS) sur toutes les connexions
6. bindIp restreint aux interfaces nécessaires, pas d'exposition Internet directe
7. Pare-feu / groupes de sécurité limitant l'accès au port MongoDB
8. Mots de passe forts, gérés via un secrets manager, avec rotation périodique
9. Chiffrement au repos et audit activés si édition Enterprise/Atlas

18 Cas pratique de synthèse

Mise en place des accès pour une application e-commerce

Compte	Rôle attribué
`admin (amorçage)`	root sur admin — utilisé une seule fois puis mis de côté
`dba_boutique`	dbOwner sur boutique — gestion courante par l'équipe DBA
`app_boutique_web`	readWrite sur boutique — utilisé par le serveur applicatif
`app_boutique_stats`	lecteur_catalogue (custom) — service de reporting, lecture seule
`backup_boutique`	backup sur admin — utilisé uniquement par le job de sauvegarde

Chaque compte n'a que les droits nécessaires à son rôle exact — une fuite d'identifiants sur le service de reporting ne permet ni écriture, ni accès aux autres collections.

19

Erreurs fréquentes à éviter

Ce qui revient le plus souvent en audit de sécurité

Instance sans authentification exposée sur Internet

La cause n°1 des incidents MongoDB documentés publiquement.

Un seul compte root partagé par toute l'équipe

Aucune traçabilité de qui a fait quoi, révocation impossible sans tout casser.

bindIp laissé à 0.0.0.0 sans pare-feu

Le serveur écoute sur toutes les interfaces, y compris publiques.

Mots de passe en clair dans le code versionné

Visibles par quiconque a accès au dépôt, y compris dans l'historique Git.

Pour aller plus loin

- **Documentation officielle** — mongodb.com/docs/manual/security
- **Référence RBAC** — Liste complète des rôles intégrés et de leurs privilèges
- **MongoDB Atlas** — Chiffrement au repos, audit et isolation réseau gérés nativement
- **Pour continuer** — Un TP dédié : mise en place pas à pas des comptes de ce cas pratique

Merci — la sécurité se construit en couches, pas en une seule mesure.