

# RéPLICATION ET REPRISE SUR PANNE (Partie 1)

Dans une base de données NoSQL telle que **MongoDB**, l'un des enjeux majeurs est d'assurer une **haute disponibilité** et une **tolérance aux pannes**. Pour répondre à ces besoins, MongoDB utilise un mécanisme appelé **Replica Set**, basé sur une architecture **Maître–Esclave** (plus exactement *Primary – Secondary*).

Un **Replica Set** est un groupe de serveurs MongoDB contenant les mêmes données.

Il comprend :

- **Un serveur Primary** → Par défaut, le serveur primaire reçoit toutes les lectures et écritures pour assurer la réconciliation des données et garantir une **cohérence forte (accéder toujours à la version finale)**
- **Un ou plusieurs serveurs Secondary** → répliquent automatiquement les données du Primary.

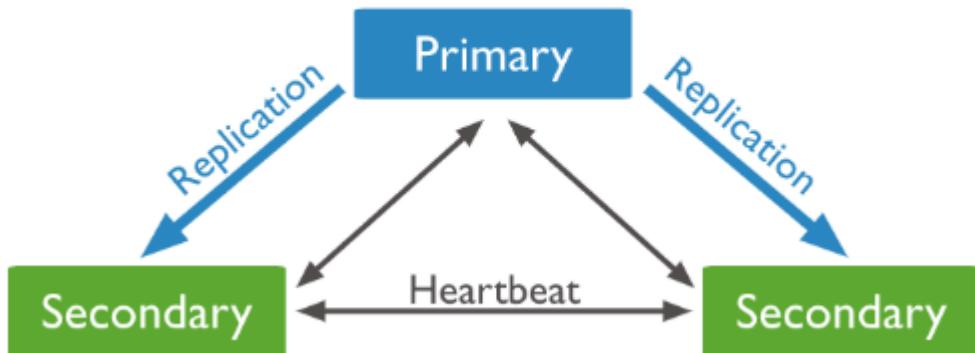
Remarque : Pour passer à l'échelle et répartir la charge, on peut configurer MongoDB pour autoriser les lectures sur les réplicas secondaires. Mais comme la réPLICATION n'est pas instantanée, ça peut présenter un risque (accéder à une version obsolète)

## La réPLICATION de MongodB est asynchrone

La réPLICATION n'est pas la stratégie de MongoDB pour monter en charge, il utilise le partitionnement (voir plus tard), en revanche elle assure la **tolérance en panne**

- **Optionnel : un Arbitre** → participe à l'élection d'un nouveau Primary, mais ne stocke pas de données.

Ce mécanisme garantit que le système continue de fonctionner même si une machine tombe en panne.



## Fonctionnement de la réPLICATION

**Le Primary reçoit :**

- toutes les écritures,
- la majorité des lectures (sauf configuration spéciale).

✓ **Chaque Secondary :**

- copie en temps réel le journal d'opérations du Primary,
- applique les changements localement.

Ainsi, toutes les données sont présentes sur plusieurs machines.

## Que se passe-t-il si le Primary tombe en panne ?

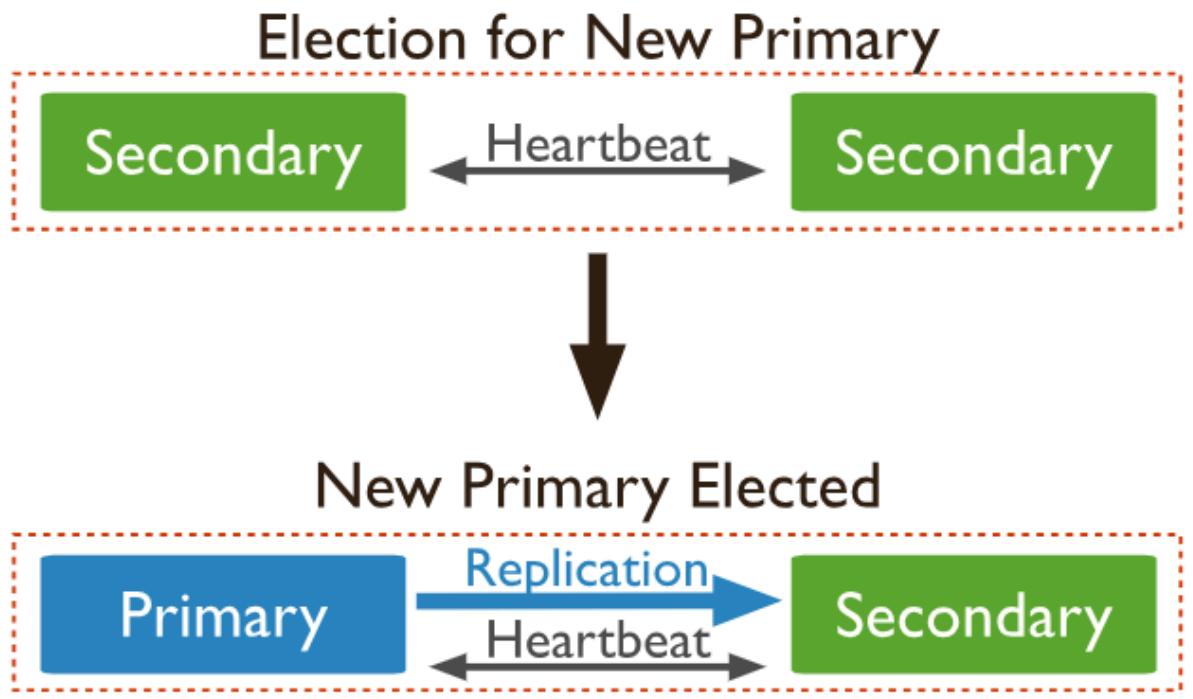
MongoDB lance automatiquement une **élection** pour élire un nouveau Primary.

### Règle fondamentale : la classe majoritaire (majority)

Un serveur ne peut être élu Primary que si **la majorité des nœuds** (machines) vote pour lui.

## Pourquoi ?

Cela empêche qu'un sous-réseau isolé crée un second Primary → évite les incohérences.



## Cas pratique :

### Cas 1 : Deux sous-réseaux avec un nombre de machines différent

Sous-réseau 1 : **5 machines**

Sous-réseau 2 : **4 machines**

La majorité = 5 nœuds → située dans le sous-réseau 1

→ Le Primary **doit** être dans ce sous-réseau

Si le réseau se coupe en deux :

- Sous-réseau 1 → 5 nœuds → **majorité**, peut élire un nouveau Primary
- Sous-réseau 2 → 4 nœuds → **pas de majorité**, reste en Secondary

→ Résultat : le système continue de fonctionner sans corruption des données.

## Cas 2 : Deux sous-réseaux avec le même nombre de machines

Sous-réseau 1 : 4 machines

Sous-réseau 2 : 4 machines

Problème : aucune partie ne détient la majorité ( $4 + 4 \rightarrow \text{majority} = 5$ )

### Solution : Ajouter un arbitre (arbitre)

Cet arbitre vote pour un des deux sous-réseaux → donne la majorité.

L'arbitre ne stocke PAS de données, il ne sert qu'aux élections.

---

### Manipulations :

**Étape 1 – Créer les dossiers de données : disque1, disque2, disque3**

**Étape 2 – Lancer les 3 serveurs MongoDB**

```
ing@ing:~/Bureau/NOSQL_TPS/TP_réPLICATION_MongoDB$ mongod --replSet monreplicaset --port 27018 --dbpath ~/Bureau/NOSQL_TPS/TP_réPLICATION_MongoDB/disque1
{"t": {"$date": "2025-12-04T14:48:38.516+01:00"}, "s": "I", "c": "NETWORK", "id": 4915701, "ctx": "main", "msg": "Initialized wire specification", "attr": {"spec": {"incomingExternalClient": {"minWireVersion": 0, "maxWireVersion": 21}, "incomingInternalClient": {"minWireVersion": 0, "maxWireVersion": 21}, "outgoing": {"minWireVersion": 6, "maxWireVersion": 21}, "isInternalClient": true}}}
{"t": {"$date": "2025-12-04T14:48:38.516+01:00"}, "s": "I", "c": "CONTROL", "id": 23285, "ctx": "main", "msg": "Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}
{"t": {"$date": "2025-12-04T14:48:38.517+01:00"}, "s": "I", "c": "NETWORK", "id": 4648601, "ctx": "main", "msg": "Implicit TCP FastOpen unavailable. If TCP FastOpen is required, set tcpFastOpenServer, tcpFastOpenClient, and tcpFastOpenQueueSize."}
{"t": {"$date": "2025-12-04T14:48:38.519+01:00"}, "s": "I", "c": "REPL", "id": 5123008, "ctx": "main", "msg": "Successfully registered PrimaryOnlyService", "attr": {"service": "TenantMigrationDonorService", "namespace": "config.tenantMigrationDonors"}}
{"t": {"$date": "2025-12-04T14:48:38.519+01:00"}, "s": "I", "c": "REPL", "id": 5123008, "ctx": "main", "msg": "Successfully registered PrimaryOnlyService", "attr": {"service": "TenantMigrationRecipientService", "namespace": "config.tenantMigrationRecipients"}}
...[redacted]
```

```
ing@ing:~/Bureau/NOSQL_TPS/TP_réPLICATION_MongoDB$ mongod --replSet monreplicaset
--port 27019 --dbpath ~/Bureau/NOSQL_TPS/TP_réPLICATION_MongoDB/disque2
{"t":{"$date":"2025-12-04T14:50:25.630+01:00"}, "s": "I", "c": "CONTROL", "id": 232
85, "ctx": "main", "msg": "Automatically disabling TLS 1.0, to force-enable TLS 1.
0 specify --sslDisabledProtocols 'none'"}
{"t":{"$date":"2025-12-04T14:50:25.631+01:00"}, "s": "I", "c": "NETWORK", "id": 491
5701, "ctx": "main", "msg": "Initialized wire specification", "attr": {"spec": {"incomi
ngExternalClient": {"minWireVersion": 0, "maxWireVersion": 21}, "incomingInternalClien
t": {"minWireVersion": 0, "maxWireVersion": 21}, "outgoing": {"minWireVersion": 6, "maxWi
reVersion": 21}, "isInternalClient": true}}}
{"t":{"$date":"2025-12-04T14:50:25.632+01:00"}, "s": "I", "c": "NETWORK", "id": 464
8601, "ctx": "main", "msg": "Implicit TCP FastOpen unavailable. If TCP FastOpen is r
equired, set tcpFastOpenServer, tcpFastOpenClient, and tcpFastOpenQueueSize."}
{"t":{"$date":"2025-12-04T14:50:25.632+01:00"}, "s": "I", "c": "REPL", "id": 512
3008, "ctx": "main", "msg": "Successfully registered PrimaryOnlyService", "attr": {"se
rvice": "TenantMigrationDonorService", "namespace": "config.tenantMigrationDonors"}}
{"t":{"$date":"2025-12-04T14:50:25.632+01:00"}, "s": "I", "c": "REPL", "id": 512
3008, "ctx": "main", "msg": "Successfully registered PrimaryOnlyService", "attr": {"se
rvice": "TenantMigrationRecipientService", "namespace": "config.tenantMigrationRecip
ients"}}
exit status 0
```

```
ing@ing:~/Bureau/NOSQL_TPS/TP_réPLICATION_MongoDB$ mongod --replSet monreplicaset
--port 27020 --dbpath ~/Bureau/NOSQL_TPS/TP_réPLICATION_MongoDB/disque3
{"t":{"$date":"2025-12-04T14:52:00.850+01:00"}, "s": "I", "c": "NETWORK", "id": 491
5701, "ctx": "main", "msg": "Initialized wire specification", "attr": {"spec": {"incomi
ngExternalClient": {"minWireVersion": 0, "maxWireVersion": 21}, "incomingInternalClien
t": {"minWireVersion": 0, "maxWireVersion": 21}, "outgoing": {"minWireVersion": 6, "maxWi
reVersion": 21}, "isInternalClient": true}}}
{"t":{"$date":"2025-12-04T14:52:00.850+01:00"}, "s": "I", "c": "CONTROL", "id": 232
85, "ctx": "main", "msg": "Automatically disabling TLS 1.0, to force-enable TLS 1.
0 specify --sslDisabledProtocols 'none'"}
{"t":{"$date":"2025-12-04T14:52:00.851+01:00"}, "s": "I", "c": "NETWORK", "id": 464
8601, "ctx": "main", "msg": "Implicit TCP FastOpen unavailable. If TCP FastOpen is r
equired, set tcpFastOpenServer, tcpFastOpenClient, and tcpFastOpenQueueSize."}
{"t":{"$date":"2025-12-04T14:52:00.852+01:00"}, "s": "I", "c": "REPL", "id": 512
3008, "ctx": "main", "msg": "Successfully registered PrimaryOnlyService", "attr": {"se
rvice": "TenantMigrationDonorService", "namespace": "config.tenantMigrationDonors"}}
{"t":{"$date":"2025-12-04T14:52:00.852+01:00"}, "s": "I", "c": "REPL", "id": 512
3008, "ctx": "main", "msg": "Successfully registered PrimaryOnlyService", "attr": {"se
rvice": "TenantMigrationRecipientService", "namespace": "config.tenantMigrationRecip
ients"}}
{"t":{"$date":"2025-12-04T14:52:00.852+01:00"}, "s": "I", "c": "CONTROL", "id": 594
5603, "ctx": "main", "msg": "Multi threading initialized"}
{"t":{"$date":"2025-12-04T14:52:00.852+01:00"}, "s": "I", "c": "TENANT_M", "id": 709
exit status 0
```

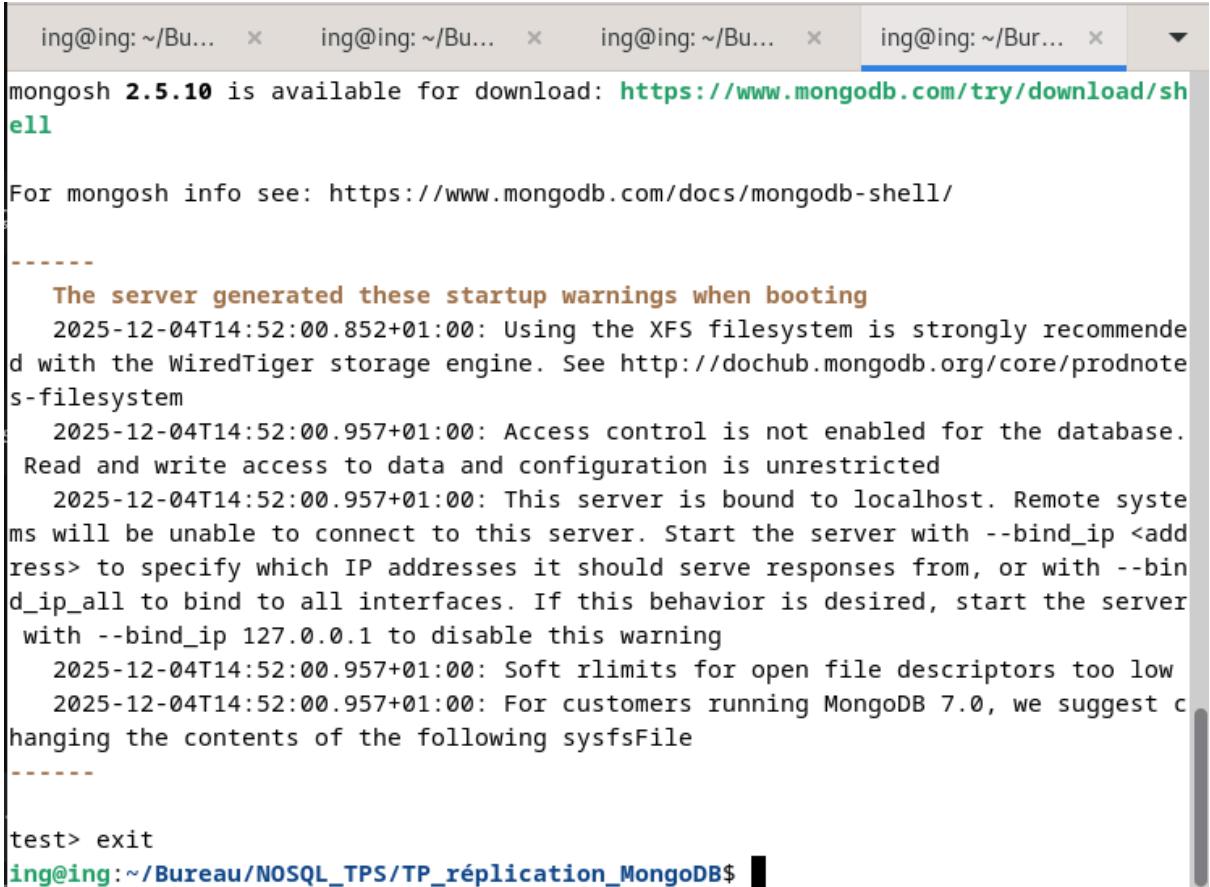


---

### Étape 3 – Se connecter à un nœud et initialiser le replica set

Ouvre un 4<sup>e</sup> terminal :

Se connecter au serveur primaire :



```
ing@ing: ~/Bu... x ing@ing: ~/Bu... x ing@ing: ~/Bu... x ing@ing: ~/Bur... x
mongosh 2.5.10 is available for download: https://www.mongodb.com/try/download/shell

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

-----
The server generated these startup warnings when booting
2025-12-04T14:52:00.852+01:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
2025-12-04T14:52:00.957+01:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
2025-12-04T14:52:00.957+01:00: This server is bound to localhost. Remote systems will be unable to connect to this server. Start the server with --bind_ip <address> to specify which IP addresses it should serve responses from, or with --bind_ip_all to bind to all interfaces. If this behavior is desired, start the server with --bind_ip 127.0.0.1 to disable this warning
2025-12-04T14:52:00.957+01:00: Soft rlimits for open file descriptors too low
2025-12-04T14:52:00.957+01:00: For customers running MongoDB 7.0, we suggest changing the contents of the following sysfsFile
-----

test> exit
ing@ing:~/Bureau/NOSQL_TPS/TP_réPLICATION_MongoDB$
```

Initialiser le replica set :

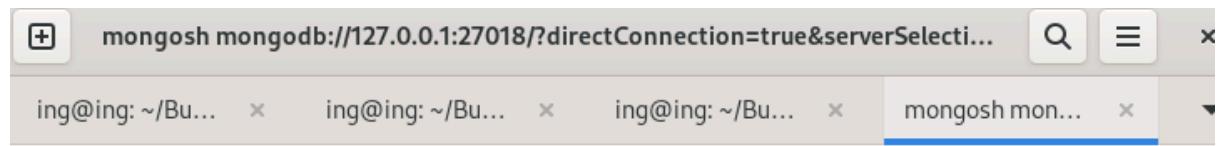
```
test> rs.initiate()
{
  info2: 'no configuration specified. Using a default configuration for the set
  me: 'localhost:27018',
  ok: 1
}
monreplicaset [direct] other1 test> exit
```

## Ajouter les serveurs 2 et 3 au replicaset :

```
ing@ing:~/Bu...  x  ing@ing:~/Bu...  x  ing@ing:~/Bu...  x  mongosh mon...  x
monreplicaset [direct: primary] test> rs.add("localhost:27019")
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1764858515, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAA='),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1764858515, i: 1 })
}
monreplicaset [direct: primary] test> rs.add("localhost:27020")
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1764858526, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAA='),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1764858526, i: 1 })
}
```

## Afficher la configuration actuelle du replicaset :

```
monreplicaset [direct: primary] test> rs.config()
{
  _id: 'monreplicaset',
  version: 5,
  term: 1,
  members: [
    {
      _id: 0,
      host: 'localhost:27018',
      arbiterOnly: false,
      buildIndexes: true,
      hidden: false,
      priority: 1,
      tags: {},
      secondaryDelaySecs: Long('0'),
      votes: 1
    },
    {
      _id: 1,
      host: 'localhost:27019',
      arbiterOnly: false,
      buildIndexes: true,
      hidden: false,
```



```
mongosh mongodb://127.0.0.1:27018/?directConnection=true&serverSelectionTimeoutMS=30000
```

```
nonreplicaset [direct: secondary] test> rs.isMaster()
{
  topologyVersion: {
    processId: ObjectId('6931979a7307a48c80ba42e7'),
    counter: Long('12')
  },
  hosts: [ 'localhost:27018', 'localhost:27019', 'localhost:27020' ],
  setName: 'monreplicaset',
  setVersion: 5,
  ismaster: false,
  secondary: true,
  primary: 'localhost:27020',
  me: 'localhost:27018',
  lastWrite: {
    opTime: { ts: Timestamp({ t: 1764874866, i: 1 }), t: Long('2') },
    lastWriteDate: ISODate('2025-12-04T19:01:06.000Z'),
    majorityOpTime: { ts: Timestamp({ t: 1764874866, i: 1 }), t: Long('2') },
    majorityWriteDate: ISODate('2025-12-04T19:01:06.000Z')
  },
  maxBsonObjectSize: 16777216,
  maxMessageSizeBytes: 48000000,
  maxWriteBatchSize: 100000,
  localTime: ISODate('2025-12-04T19:01:14.381Z'),
```