

# How to Create a MongoDB Replica Set

Learn how to create a MongoDB replica set for high availability and data redundancy. Follow our step-by-step guide to set up and configure your replica set properly.

# Contents

01	Introduction	3
02	Prerequisites	3
03	1. Configure the Hosts File	3
04	2. Set Up the Replication Key	4
05	3. Configure MongoDB	5
06	4. Bootstrap the Replica Set	7
07	More Resources	10

# Introduction

---

When you deploy a MongoDB database in a mission-critical application, you can configure a replica set for high availability. A replica set offers redundancy and high availability, which reduces downtime during a disaster because there is no single point of failure. The minimum recommended configuration for a MongoDB replica set is one primary node and two secondary nodes, but replica sets can have up to 50 members. Your application writes only to the primary node, and the secondary nodes replicate the data. If the primary node fails, the replica set holds an election to choose a new primary. Applications can read data from secondaries but cannot write to them.

This guide explains how to create a MongoDB replica set. It was tested on Ubuntu 20.04, but the steps are similar for other Linux distributions.

## Prerequisites

---

You need three servers attached to the same [Vultr VPC](#). Each of the servers should have:

- A [non-root user](#) configured with sudo privileges
- [MongoDB installed](#) and secured with a password
- For clarity, this guide uses the following hostnames and private IP addresses for the servers. You should substitute these with your values.
  - **Primary:** server-1 - 10.0.0.1
  - **Secondary:** server-2 - 10.0.0.2
  - **Secondary:** server-3 - 10.0.0.3

## 1. Configure the Hosts File

---

MongoDB recommends using DNS hostnames instead of IP addresses for the replica members. Because the Vultr VPC is a private network without DNS, add

the private IP addresses and hostnames to the hosts file. **Repeat these steps on each server.**

1. SSH to the server as a non-root user.
2. Edit the hosts file.

```
$ sudo nano /etc/hosts
```

3. Locate the line below.

```
127.0.0.1 localhost
```

4. Enter the IP addresses and hostnames as shown under that line.

```
127.0.0.1 localhost
10.0.0.1 server-1
10.0.0.2 server-2
10.0.0.3 server-3
```

5. Save and close the file.

## 2. Set Up the Replication Key

All servers in the replica set share a base64 key. Follow these steps to install the key.

Use the `openssl` command to generate a new key on one of the servers.

```
$ openssl rand -base64 756
```

You should get a block like this. Copy this base64 key. You'll use it in the following steps.

```
Yga80PbkHKptRRo0NFcPaPzOCFpySgpwNHMA3JS179wyGC0I0Yg/FUnDyiIhGe5D
YVQF3o+S1iscBiKftsPZ5WBojRREcefAUH0qK7pVB0jT+oYuH6ltMGiDtH26XjVB
... truncated ...
```

```
yxJm+UjpN0n8V1pH1LrMJT4FC4Bw3L7vqSnxVbLRnQi02Y0ECfyPgepCCNIyuaP  
mMSUJ8mm1q4jdf0AKvCspeliSQ/cqaxKfqaTWjzhsLk8eHbU
```

## Repeat these steps on each server.

1. Create a new `auth_key` file.

```
$ sudo nano /var/lib/mongodb/auth_key
```

2. Paste your base64 key.
3. Save and close the file.
4. Set the permissions to 400, making the file read-only for the file owner and access denied for all others.

```
$ sudo chmod 400 /var/lib/mongodb/auth_key
```

5. Change the owner and group to **mongodb**.

```
$ sudo chown mongodb:mongodb /var/lib/mongodb/auth_key
```

## 3. Configure MongoDB

In this section, you'll configure the shared key, network interface, and replica set name. **Repeat these sub-sections on each server.**

### 3.1. Configure the Shared Key

1. Open **mongod.conf** in an editor for the following steps.

```
$ sudo nano /etc/mongod.conf
```

2. Find the **security** section.

```
security:
  authorization: enabled
#operationProfiling:
```

3. Below the **authorization:** line, add the **keyFile** value as shown.

```
security:
  authorization: enabled
  keyFile: /var/lib/mongodb/auth_key
#operationProfiling:
```

## 3.2. Configure the Network Interface

1. Find the **network interfaces** section.

```
# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1
```

2. Add the respective server name after the loopback interface (127.0.0.1) to each server. For example:

### On server-1:

```
# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1, server-1
```

### On server-2:

```
# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1, server-2
```

### On server-3:

```
# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1, server-3
```

### 3.3. Configure the Replica Set Name

1. Find the **replication** section.

```
#replication:
```

2. Remove the Pound comment from the replication line. Below that, add `replSetName: "rs0"` as shown.

```
replication:
  replSetName: "rs0"
```

3. Restart MongoDB on the primary node.

```
$ sudo systemctl restart mongod
```

4. Restart MongoDB on the secondary nodes.

```
$ sudo systemctl restart mongod
```

## 4. Bootstrap the Replica Set

In this section, you'll add the nodes to the replica set and bootstrap the replication process.

1. **On the primary node**, log in to MongoDB.

```
$ mongosh -u your_admin_name -p --authenticationDatabase admin
```

2. Enter the password for your admin account and press Enter to proceed.

3. Run the following command to add the replica set members.

```
test> rs.initiate(  
  {  
    _id: "rs0",  
    members: [  
      { _id: 0, host: "server-1" },  
      { _id: 1, host: "server-2" },  
      { _id: 2, host: "server-3" }  
    ]  
  })
```

4. You should get the following response when the replica set starts. Notice the prompt changes to `rs0 [direct: secondary] test>`.

```
{ ok: 1 }  
rs0 [direct: secondary] test>
```

5. Create a sample **company\_db** database.

```
rs0 [direct: secondary] test> use company_db
```

6. You should get the following response and the prompt changes to `rs0 [direct: primary] company_db>`. This member is now the primary node.

```
switched to db company_db  
rs0 [direct: primary] company_db>
```

7. Insert a sample record in a new **employees** collection in the **company\_db** database.

```
rs0 [direct: primary] company_db> db.employees.insertOne({  
  "staff_id" : 1,  
  "staff_name" : "JOHN DOE",  
  "phone" : "11111111"  
})
```

8. You should get the output below.

```
{
  acknowledged: true,
  insertedId: ObjectId("621dcf1abdb5b0c5e59294d9")
}
```

9. **On each secondary node**, log in to MongoDB.

```
$ mongosh -u your_admin_name -p --authenticationDatabase admin
```

10. Enter the password for your admin account and press Enter to proceed.

11. You should see the prompt below, showing that the members are secondary nodes.

```
rs0 [direct: secondary] test>
```

12. On each secondary node, switch to the **company\_db**.

```
rs0 [direct: secondary] test> use company_db
```

13. You should get the following output.

```
switched to db company_db
```

14. Run the following command on each secondary node, which allows them to accept read commands.

```
rs0 [direct: secondary] company_db>
db.getMongo().setReadPref('primaryPreferred')
```

15. List the document from the **employees** collection.

```
rs0 [direct: secondary] company_db> db.employees.find()
```

16. You should get the following output on each secondary node, which shows that the replica set replicated the data to each node.

```
[
  {
    _id: ObjectId("621dcf1abdb5b0c5e59294d9"),
    staff_id: 1,
    staff_name: 'JOHN DOE',
    phone: '11111111'
  }
]
```

17. Try adding a new employee record on any secondary node.

```
rs0 [direct: secondary] company_db> db.employees.insertOne({
    "staff_id" : 2,
    "staff_name" : "MARY ROE",
    "phone" : "22222222"
  })
```

18. The command should fail. Secondary nodes are read-only.

```
MongoServerError: not primary
```

19. If you stop the primary server or it goes offline, the replica set elects one of the secondary nodes to be the new primary node.

```
$ sudo systemctl stop mongod
```

## More Resources

For application reliability and data recovery, always consider replication for MongoDB instances. If your MongoDB replica set is working as expected, you can learn more about MongoDB and MongoDB replica sets in the [MongoDB documentation Replication section](#).

You also need to consider how to handle failover in your client application. As explained [in the documentation](#):

If an operation fails because of a network error, `ConnectionFailure` is raised, and the client reconnects in the background.



VULTR

