### 🧩 **Two Ways to Run MongoDB on AWS**

#### 1. **Managed MongoDB**

* Examples: MongoDB Atlas, Amazon DocumentDB (MongoDB-compatible)
* What it means: You don’t manage servers, backups, scaling, or patching — it’s all handled for you
* Pros:
  + Easy to set up and scale
  + Built-in security, backups, monitoring
  + High availability and global replication
* Cons:
  + Less control over internals
  + Can be more expensive at scale

#### 2. **Self-Hosted MongoDB**

* Where it runs: On EC2 instances that you manage
* What it means: You install and configure MongoDB yourself
* Pros:
  + Full control over setup, tuning, and security
  + Can be cost-effective for large, stable workloads
* Cons:
  + You manage everything: replication, backups, scaling, patching
  + Higher operational overhead

### 🧠 Which Should You Choose?

| Criteria | Managed MongoDB | Self-Hosted MongoDB |
| --- | --- | --- |
| Setup & Maintenance | Minimal (hands-off) | Manual (you manage all) |
| Scaling | Auto-scaling available | Manual or scripted |
| Cost | Higher at scale | Lower if optimized |
| Control & Customization | Limited | Full control |
| Use Case Fit | Fast-moving projects | Custom or legacy setups |

✅ Alright — let’s expand each one into detailed, real-world steps for your Rentify project:

## 📌 **1️⃣ Managed MongoDB**

Example: MongoDB Atlas (most popular)  
Alternative: Amazon DocumentDB (if you want 100% AWS)

✅ 🔹 How to set up MongoDB Atlas

👉 Goal: Fully managed cluster you can connect your Node.js API to.

Step-by-step

1️⃣ Create an Atlas account

* Go to [MongoDB Atlas](https://www.mongodb.com/cloud/atlas/register).
* Sign up (free tier is fine for development).

2️⃣ Create a project & cluster

* Create a Project (e.g., Rentify-Dev).
* Click Build a Cluster.
* Pick Shared Cluster → free M0 cluster is enough to start.
* Pick a cloud provider & region (e.g., AWS + closest region to your users).

3️⃣ Configure security

* Add your IP address to the IP Whitelist → this allows your EC2/localhost to connect.
  + Example: Add 0.0.0.0/0 to allow all (only for testing — restrict later).
* Create a DB user with username/password for your app.

4️⃣ Create a database

* Inside the cluster, create a new database & collection, e.g., rentify > properties.

5️⃣ Get the connection string

* Click Connect → Connect your application → Node.js.
* Copy the connection URI — looks like:

mongodb+srv://<username>:<password>@<clustername>.mongodb.net/rentify?retryWrites=true&w=majority

6️⃣ Update your Node.js app

* In your .env:

MONGODB\_URI=mongodb+srv://username:password@cluster.mongodb.net/rentify?retryWrites=true&w=majority

* In your Node.js code:

import mongoose from 'mongoose';

mongoose.connect(process.env.MONGODB\_URI)

.then(() => console.log('MongoDB Atlas connected'))

.catch(err => console.error(err));

7️⃣ Test your connection

* Start your API → verify connection → make CRUD requests → check data in Atlas UI.

✅ Managed MongoDB: Done!

🔗 Official Docs:

* [Atlas Quick Start](https://www.mongodb.com/docs/atlas/getting-started/)
* [Atlas Security](https://www.mongodb.com/docs/atlas/security/)

## ✅ **🔹 Amazon DocumentDB alternative**

How it works: DocumentDB is API-compatible with MongoDB, but is fully managed inside AWS.

Steps:

1. Go to RDS → Create DocumentDB Cluster.
2. Choose instance class, replica count, VPC, etc.
3. Set up parameter groups to match MongoDB driver compatibility.
4. Get the cluster endpoint.
5. Use the MongoDB Node.js driver with the DocumentDB endpoint in your connection string.
6. Create a VPC security group to allow inbound/outbound traffic.

✅ It’s more work than Atlas and has some driver version quirks — most startups prefer Atlas unless you need tight AWS-only architecture.

## ✅ **Managed Pros & Cons**

✅ Pros:

* No admin hassle.
* Automated backups and scaling.
* Built-in monitoring and role-based security.

❌ Cons:

* Can cost more as you scale.
* Less control if you want custom server tuning.

## 📌 **2️⃣ Self-Hosted MongoDB**

Example: Install MongoDB yourself on an EC2 instance.

### ✅ **🔹 How to self-host MongoDB on EC2**

### **Step-by-step**

1️⃣ Launch an EC2 instance

* Choose Ubuntu 22.04 LTS (or your preferred Linux).
* Pick an instance type (t3.micro for dev).
* Configure security group → allow SSH (22) & MongoDB port (27017).

2️⃣ SSH into the server

ssh -i your-key.pem ubuntu@your-ec2-public-ip

3️⃣ Install MongoDB

# Import the public key

wget -qO - https://www.mongodb.org/static/pgp/server-7.0.asc | sudo apt-key add -

# Create the source list

echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/7.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-7.0.list

# Update and install

sudo apt-get update

sudo apt-get install -y mongodb-org

4️⃣ Start MongoDB

sudo systemctl start mongod

sudo systemctl enable mongod

5️⃣ Secure MongoDB

* Bind IP: Edit /etc/mongod.conf → change bindIp to 0.0.0.0 if you want remote access (be careful — only allow trusted IPs in your security group).
* Create admin user:

mongo

use admin

db.createUser({user: "admin", pwd: "password", roles:[{role: "root", db: "admin"}]})

6️⃣ Set up auth

* In mongod.conf, enable:

security:

authorization: enabled

* Restart:

sudo systemctl restart mongod

7️⃣ Connect your Node.js app

* Connection URI for self-hosted:

mongodb://admin:password@your-ec2-ip:27017/rentify

8️⃣ Backups

* Use mongodump and mongorestore for manual backups.
* Or automate with cron + S3.

✅ Self-hosted: Done!

## ✅ **Self-Hosted Pros & Cons**

✅ Pros:

* Complete control.
* Can be cheaper for massive workloads with consistent traffic.

❌ Cons:

* You must handle backups, replication, monitoring.
* Must patch and secure manually.
* Single point of failure if you don’t run replica sets.

## 🎯 **Which should you pick for Rentify?**

👉 For 99% of modern apps:  
✅ Use MongoDB Atlas → fast, secure, best for dev & prod.

## 🚀 **Next:**

* ✅ Choose your DB strategy (Managed vs Self-hosted)
* ✅ Launch your backend server (EC2 or ECS)
* ✅ Hook up the connection string
* ✅ Deploy & test

If you’d like, I can prepare:

* Terraform or CloudFormation examples
* EC2 + NGINX setup scripts
* S3 file upload guide
* Complete .env example

Just say: "Yes, give me the full backend deploy playbook!" and I’ll build it out for you. Want it? 🔥