

Introduction to Databases

Topics:

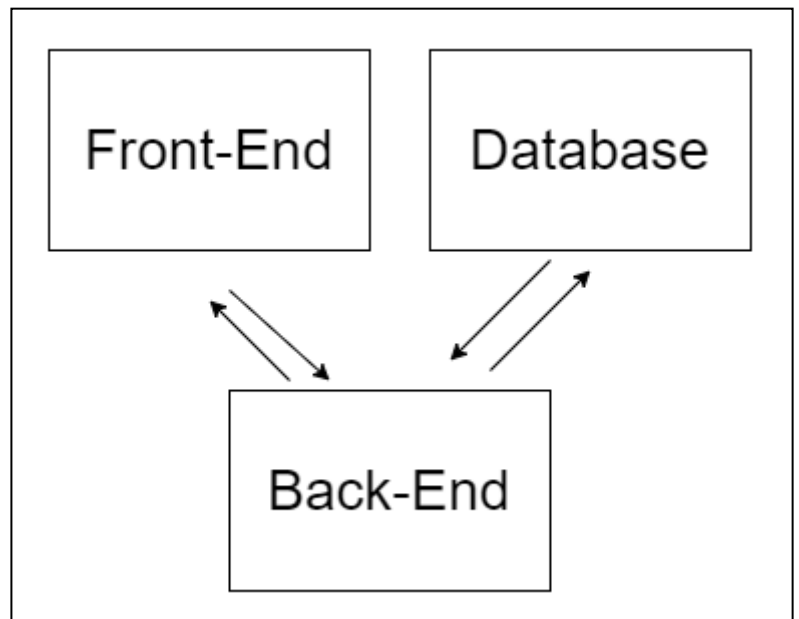
- *Introduction*
- *SQL and NoSQL*
- *MongoDB*
- *Atlas*
- *Installation*

In all our applications that are built till now, the most common problem is we are not able to save the data persistently. Once the server restarts the data in the memory is lost. We can't store everything in the file. If we did that, then we would have to manage it and as the users grow, the application itself would become large and it will take up space. We cannot put a gigabyte file into the memory and think it would work. When we host an application, we would be charged even for 100 MB as well. The basic Heroku tier has 500MB of RAM and 1-core processor for free. If we go for a 2-core processor, it will charge us \$ 14/month. It gets expensive.

To solve this, we have something called Databases. They have one single or multiple files depending on how the database wants to store data. The entire software running inside the database is very fast to optimize everything. If we have a huge number of files and we want to search for a one particular file, then it would take a long time to search through all of it. But database software manages this and is very fast in searching; it depends on the algorithm that the databases use. The database solves our problem of saving the data persistently.

Another good part is we have that separation.

In logical separation, our frontend is already separated and it works on its own. Then we have application/backend servers which figure out how to run the application and it runs on its own entity. Finally, having the database, we will be separating the data. All the data will be handled somewhere else and our



application logic has to communicate with the database and tell what should be done (store, delete, change etc.). the crud operations would be off-loaded to the databases. the user will request some pages and it will be communicated to the backend server and it will get that data from the database and send it the front end. Everything in its own entity.

Logical separation is necessary, because if everything is smacked into one package, organization and maintainability issues would rise. Ex: Amazon has all its data in a single warehouse and if some disaster occurs it would lose all its data. Similarly, any other enterprise level application does not store data in one single place. These databases are stored in different places having replicas of data. This maintenance of data in replicas is called 'replica sets' It costs more, but the data is precious and cannot be lost at any cost.

In our local computer, as we are developing, we have a browser which is in the system itself. The application server or server itself through NodeJS or Express JS is also in the system. Databases are also another process. But there is something else called a database server to which we do not have direct access. We get access, but in this case, it would be my application server and will get access from it.

Just as a browser is the client for the application server. Application server was running on the localhost, on our computer, which is 127.0.0.1. Similarly, when we install a database in our system, the database would be running inside our system with the same IP address, but at a different port.

NodeJS – 127.0.0.1:3000

MongoDB – 127.0.0.1:8873

MySQL – 127.0.0.1:3303

Everything is running in the same system but at different **ports**.

There are cloud providers (AWS, Azure, Heroku, GCP etc), who can externalize the local database. We can off load this server into the cloud (someone else's computer, hosted on the internet). So, if we move the database to any of the cloud providers, and can push the database inside this. Based on that, the cloud providers will give us some IP addresses. This is also a tedious task to upload the database server. So, for this, MongoDB created atlas which would take care of the setup. So, the entire database would be getting served from MongoDB. We do not have everything in one system at production level. Atlas or MongoDB will be setting up the entire infrastructure to maintain our database. We have one area of interaction i.e., atlas.

Difference between SQL and NoSQL:

<i>SQL Databases</i>	<i>NoSQL Databases</i>
<i>Tables with fixed rows and columns</i>	Document: JSON documents, Key-value: key-value pairs, Wide-column: tables with rows and dynamic columns, Graph: nodes and edges
<i>Oracle, MySQL, Microsoft SQL Server, and PostgreSQL</i>	Document: MongoDB and CouchDB, Key-value: Redis and DynamoDB, Wide-column: Cassandra and HBase, Graph: Neo4j and Amazon Neptune
<i>General purpose</i>	Document: general purpose, Key-value: large amounts of data with simple lookup queries, Wide-column: large amounts of data with predictable query patterns, Graph: analysing and traversing relationships between connected data

SQL databases like MySQL, PostgreSQL, SQLite etc, store data in the form of tables. SO, users' data would be saved in a tabular form.

In NoSQL, data is stored in document forms. It will not use the tabular form. Ex: MongoDB, Cloud Fire Store (Google), Cassandra, CouchDB, Redis etc. It can store the entire data as a string that can be used by Redis and it also stores data in json format as well.

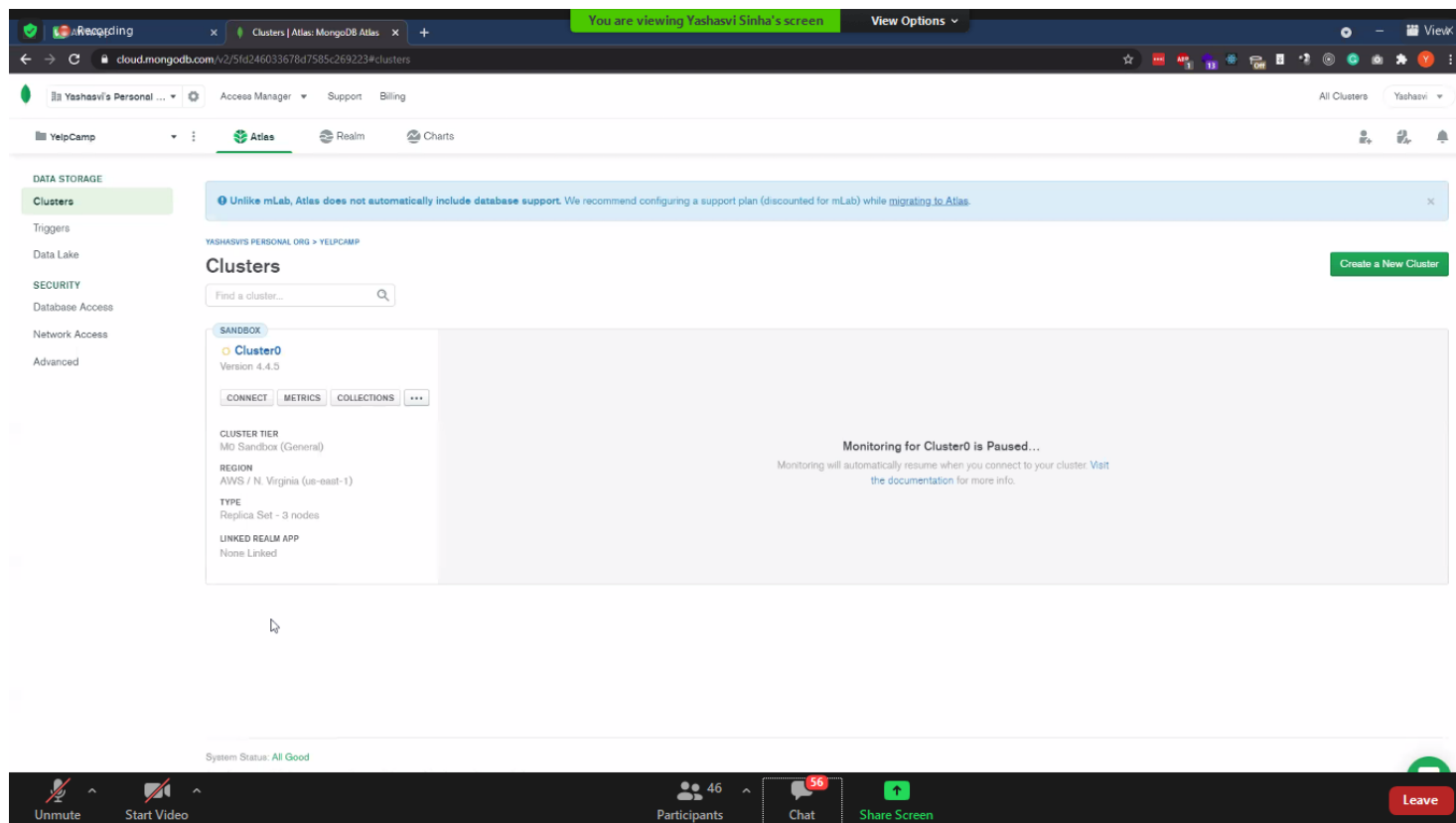
General idea is SQL databases are relational, meaning, we have users and orders. In orders we have order id, products, cost and other details along with user id. This will allow us to know which user has created which order. This relation can be maintained in SQL, but it doesn't mean that we cannot have the same in NoSQL.

Both have their own merits and demerits. SQL is very rigid, but NoSQL is very flexible and can add fields dynamically.

Based on what use case we have or the type of application we are trying to build we can either opt for SQL or NoSQL. NoSQL is generally suited for unstructured data. Example would be SMS – it does not have a particular format. We can add and remove fields dynamically depending on application growth. Both have their own merits and demerits.

In google search → MongoDB atlas

We need to create an account in MongoDB cloud. Once you have the account, login and you will have a view similar to



We can go to organizations (top-left). Initially we will have one created on the account name. Once inside a particular organization, we will create a new project. Click on the **New Project** and name it as e-commerce (hypnotically). We can add members as well. By default, we will be already added to the particular project. Atlas

Atlas, depending on the which cloud provider is being used, will create a virtual computer, install MongoDB on it and will give you the access. We don't have to install anything in our system. It is a manage database server. We don't have install anything in our system or elsewhere manually.

Clusters:

By default, even in the free tier, atlas provides you with a free replica set. there will be one database server but internally, they will be 3 database servers. We will directly access one, but the software inside it will be replicating it. It is like one big server has 3-servers internally. So, one change made in a big server will be replicated to the internal 3-servers as well. All are not in the same location (physically) but will be located at different places. The way cloud works is, one is hosted in the US, another

in Europe, and the last one in India. They will work in such a way; it will give a logical exposure of one server. In short there will be one logical exposure to your application. One server which will be the primary server with which the backend would be directly communicating with. This primary server internally will have clusters of databases meaning three individual databases would be running parallelly even though they are in 3 different locations but the main or primary server through which the application is talking through, will give us API's in such a way, that the application will think that every database is one database.

SO, if a particular country has a restriction or a natural calamity occurs and one cluster got failed, then the other 2-clusters which are active will be used to get the data from and once the cluster that was not working is repaired, then using these 2-clusters we can copy all the data again into it.

If one replica fails, then it can be rebuilt and once it is rebuilt, the data is copied again and now it is back to be the replica again. data is parallelly saved in all the three nodes.

Once we click on '**Build a Cluster**', then we can use **AWS**, **Google Cloud** and **Azure** as our cloud storage. It will take us to the cluster tier. There is an option for backup which is not automatic for free and finally we give the name of the cluster.

It will now create a cluster and it will take type.

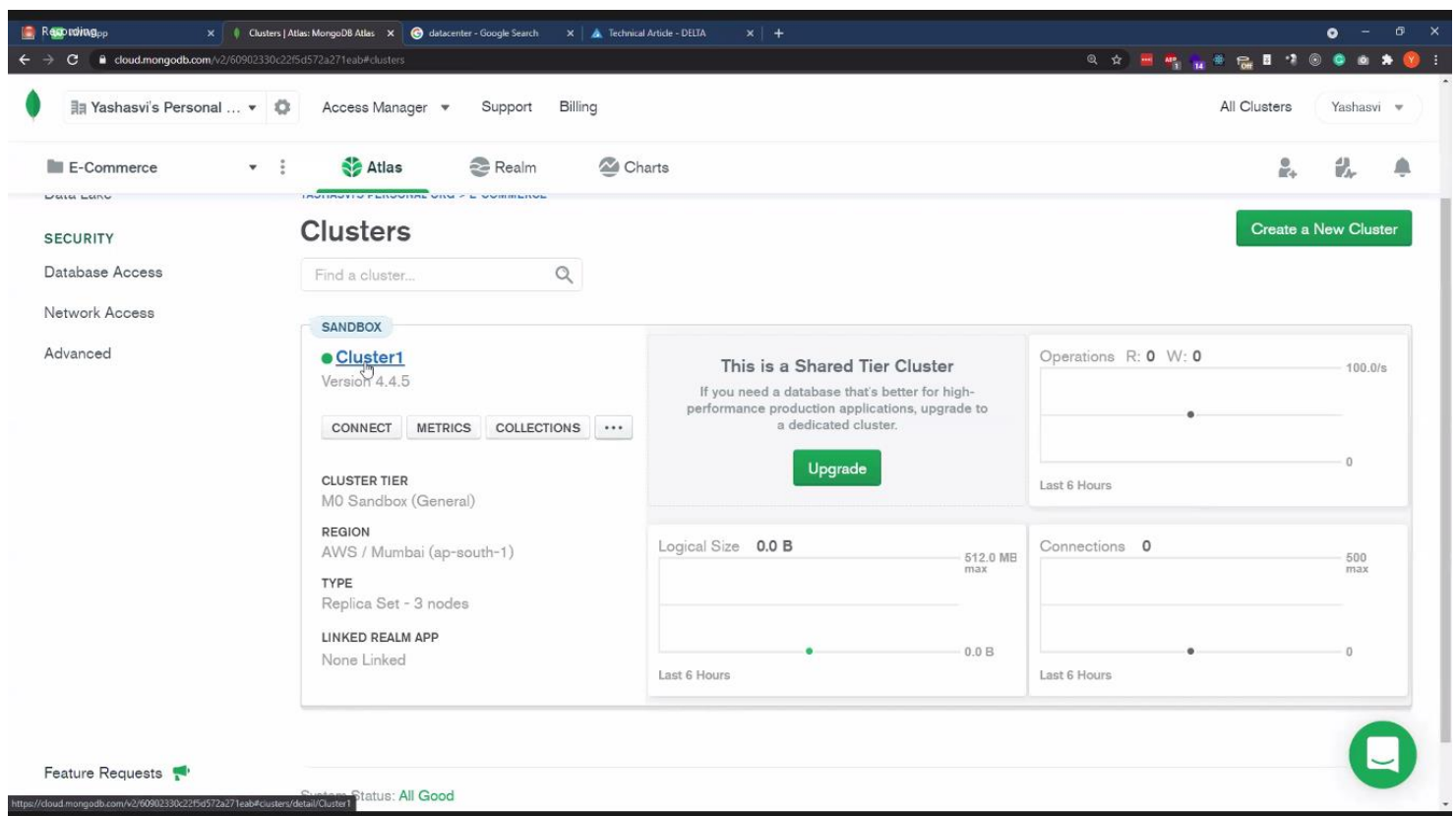
The screenshot displays the MongoDB Atlas 'Build a Cluster' wizard. The interface is divided into a left sidebar with navigation options like 'E-Commerce', 'Database Access', 'Network Access', and 'Advanced'. The main content area shows the 'Clusters' section with a search bar and a 'Create a New Cluster' button. The cluster configuration is shown in a 'Sandbox' mode for 'Cluster1' (Version 4.4). The configuration details include: 'CONNECT', 'METRICS', 'COLLECTIONS', and '...' buttons; 'CLUSTER TIER' set to 'M0 Sandbox (General)'; 'REGION' set to 'AWS / Mumbai (ap-south-1)'; 'TYPE' set to 'Replica Set - 3 nodes'; and 'LINKED REALM APP' set to 'None Linked'. A large message on the right states 'Your cluster is being created. New clusters take between 1-3 minutes to provision.' The bottom of the screen shows a video call interface with participants, chat, and a 'Leave' button.

The cluster has been created. In the overview we can see it has created 3 individual sets and one of them will be acting as primary node.

1 cluster – 3 nodes

The main area is the **Collections**. By default, there is no database created right now. We will come to collections and will add our own data. For now, we will click on ‘**Add My Own Data**’.

The cluster we have created is serving all the databases. In one cluster we can have multiple databases; one database for users, one database for products, depending on how big the project is.



We will go to the database and will name it as **e-commerce**. In the ‘**collection name**’, we will mention products. So now e-commerce is one database inside my cluster and inside e-commerce there is another database as products. We can click on **INSERT DOCUMENTS**, we will give

```
{  
  "name": "milk",  
  "price": 45,  
  "mfg": "Amul",  
  "expiry": 2021-08-21
```

```
}
```

Then we click on insert and it will become one entry like in an object structure.

```
{  
  name: "cheese",  
  price: 120,  
}
```

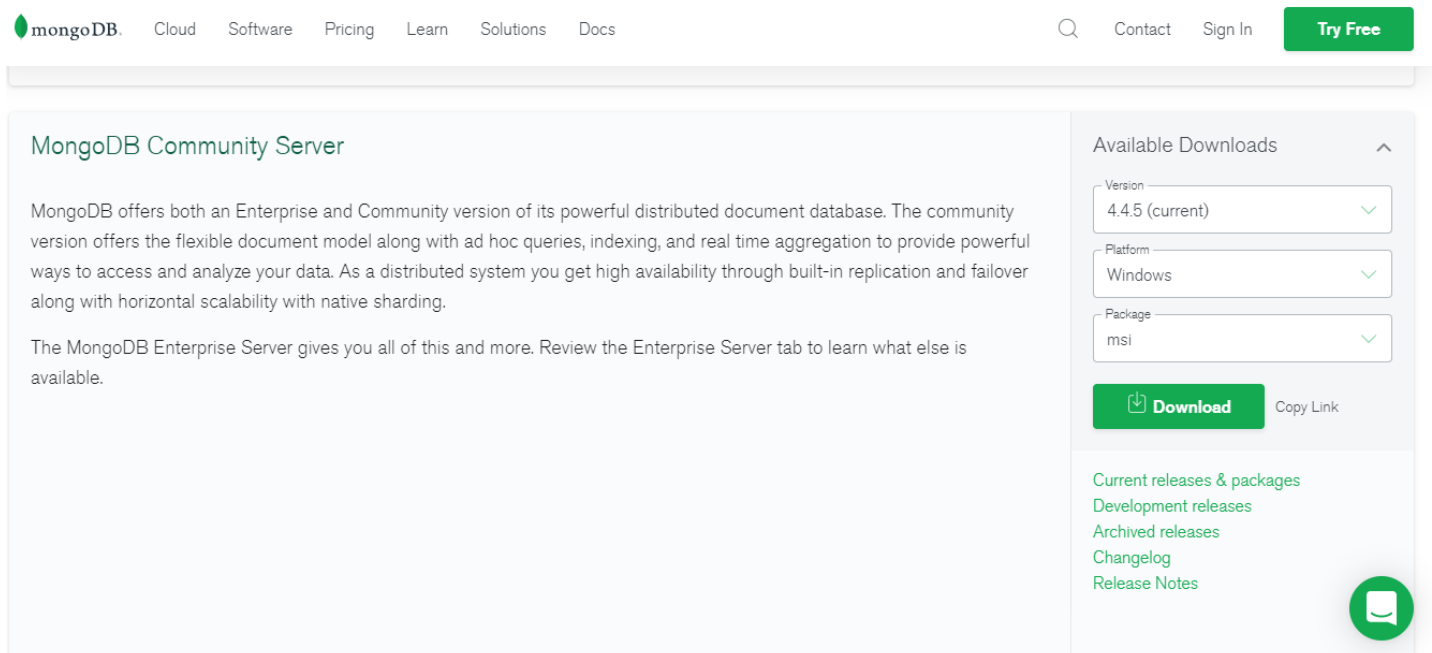
Now if we click on insert, it gets added with only two fields. We have added two objects inside our e-commerce database.

In the NoSQL world, **collections** are the **logical name** that we would give. If we are storing products, we will give the **collection name** as product and if we are saving user data, then we would name the **collection name** as users.

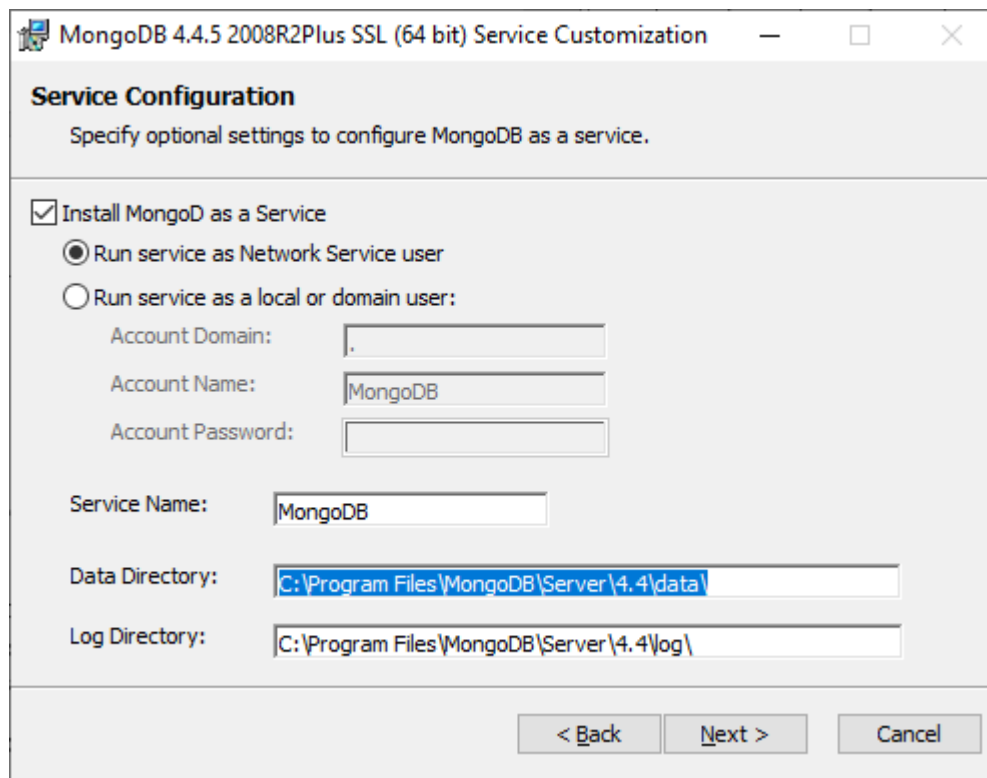
Products is a collection inside which we have two objects or two documents. When creating the cluster, we are giving the space as 500 MB. This will run as long as the space does not go beyond 500 MB.

Google → install MongoDB

We use the community edition. We will go to the MongoDB download and click on MongoDB community Server; it's free. Select windows and download it in **.msi** extension. This will install MongoDB Server which creates a server and will run it behind the scene.



The screenshot shows the MongoDB website's download page for the Community Server. The header includes the MongoDB logo and navigation links: Cloud, Software, Pricing, Learn, Solutions, and Docs. On the right, there are links for Contact, Sign In, and a prominent green 'Try Free' button. The main content area is titled 'MongoDB Community Server' and contains descriptive text about the community edition's features, such as flexible document model, indexing, and real-time aggregation. It also mentions the Enterprise version. To the right of the text is a sidebar titled 'Available Downloads' which allows users to select the version (4.4.5 current), platform (Windows), and package type (msi). Below these selections is a green 'Download' button and a 'Copy Link' option. At the bottom of the sidebar, there are links for 'Current releases & packages', 'Development releases', 'Archived releases', 'Changelog', and 'Release Notes'. A green chat icon is visible in the bottom right corner of the page.



We don't need to pile-up documents in our C drive, so we would create a drive in other drive.

MongoDB Compass: it is a graphical user interface to see all the data running in our system. It is one of the many clients which allows us to access our database.

Now that the compass is installed, in the MongoDBLogs, there is a file created. It has details of what is being logged into the server. In the MongoDBFiles folder, there are many folders and files created.

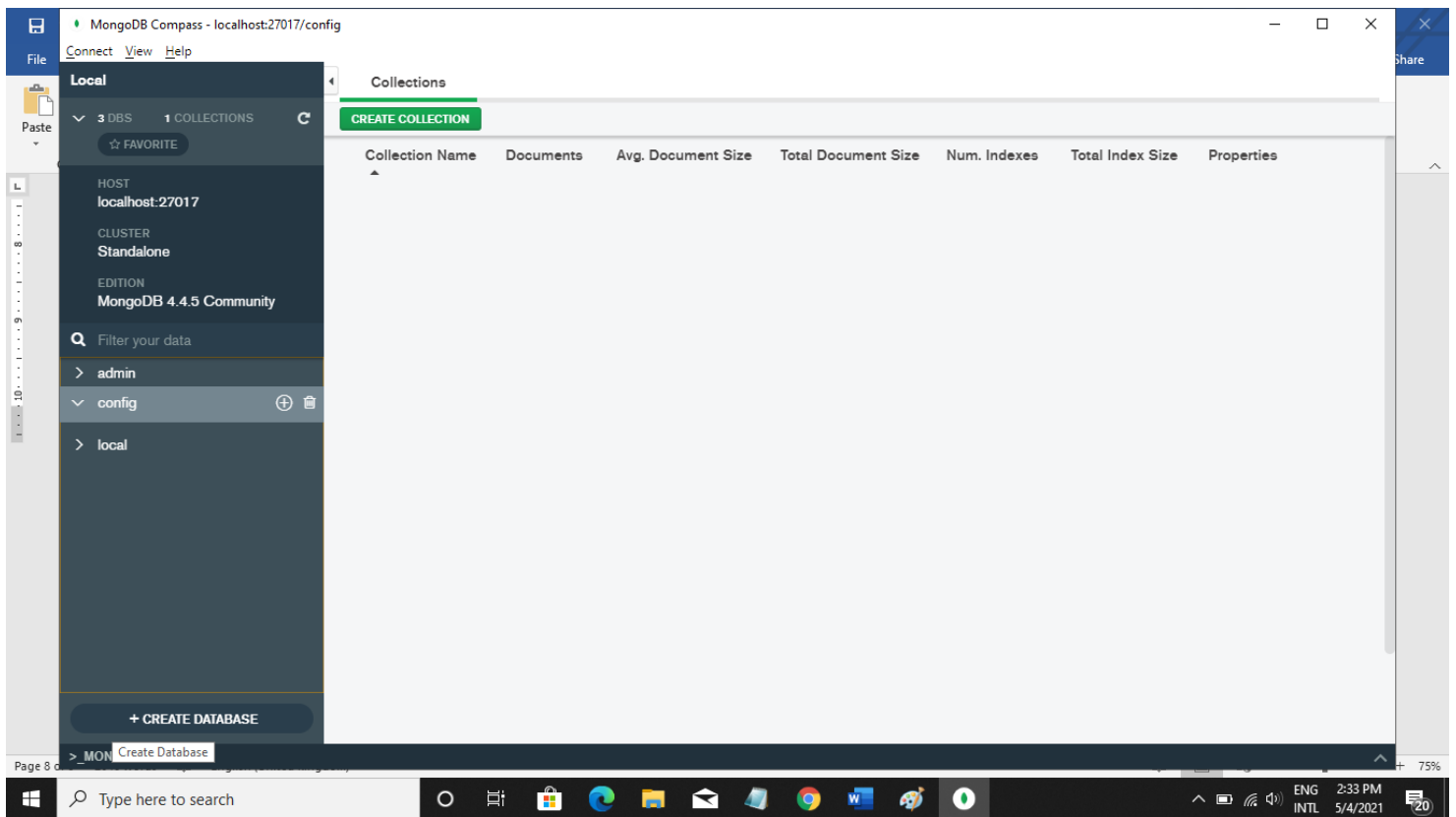
In MongoDB Compass, we need to create a connection. It needs a string to connect to the local database. It will open-up a connection window.

Hostname: localhost

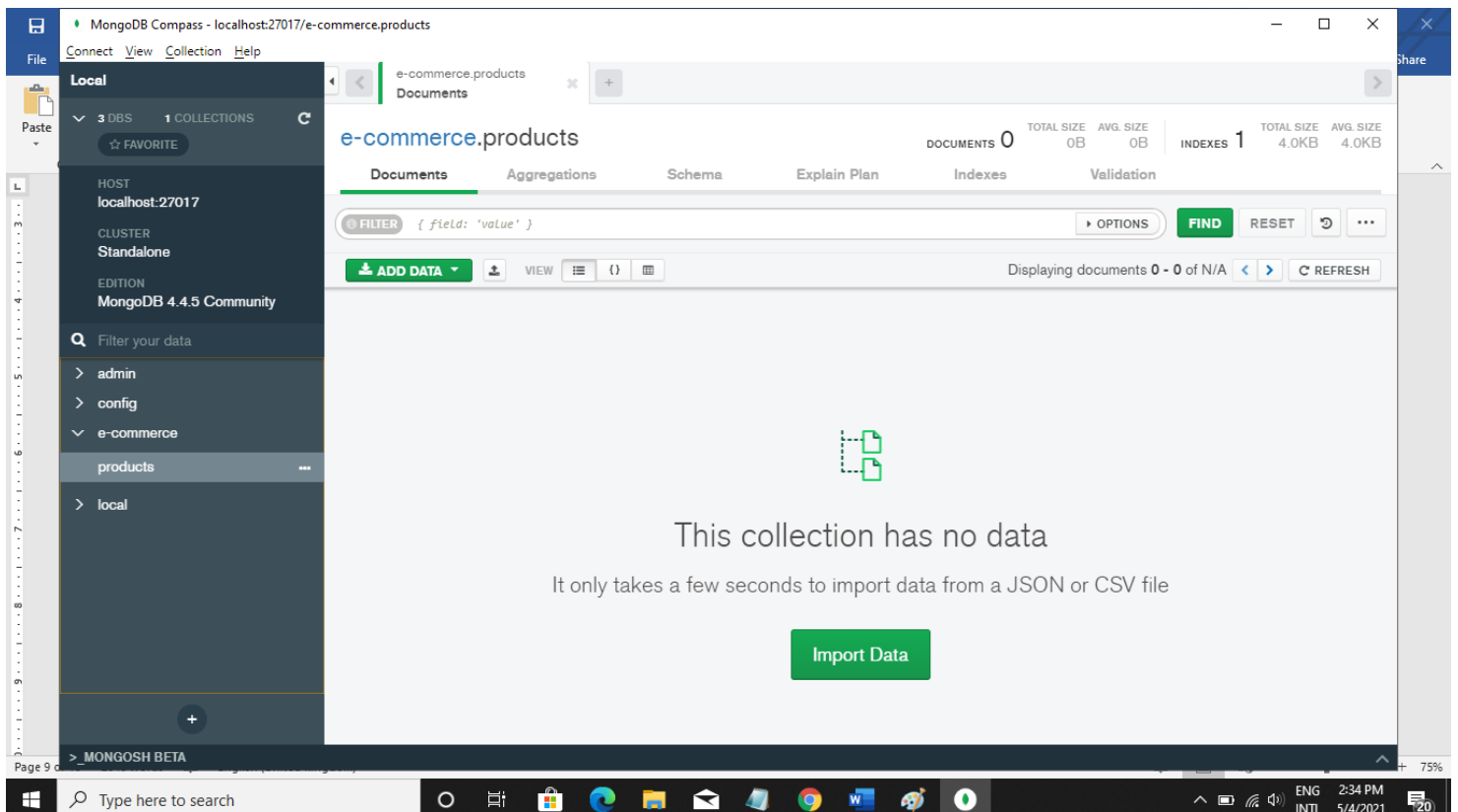
Port: by default, 27017 on which our database is running.

Click on connect.

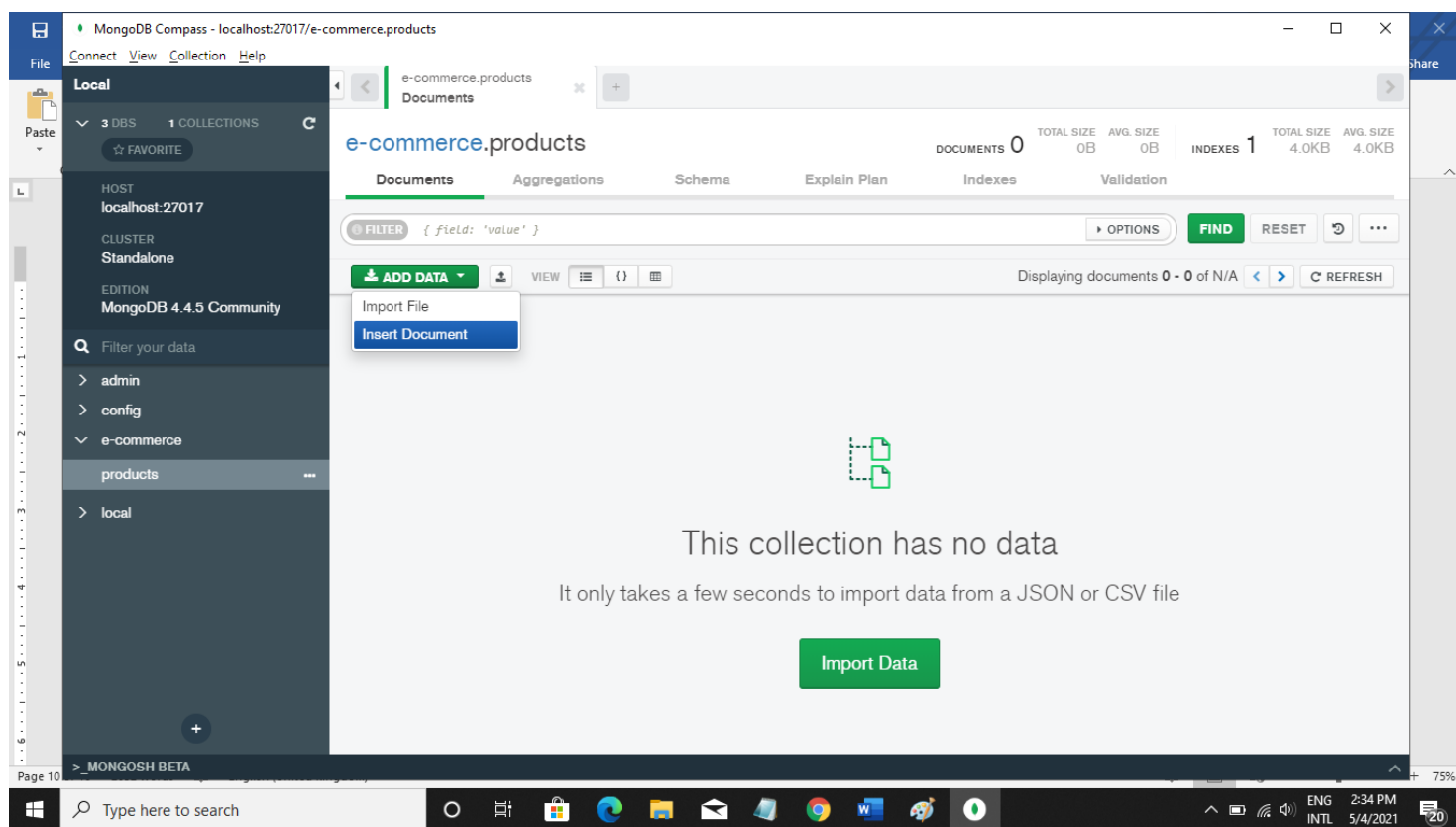
We can see three databases are created automatically to store everything which it needs to run the database properly.



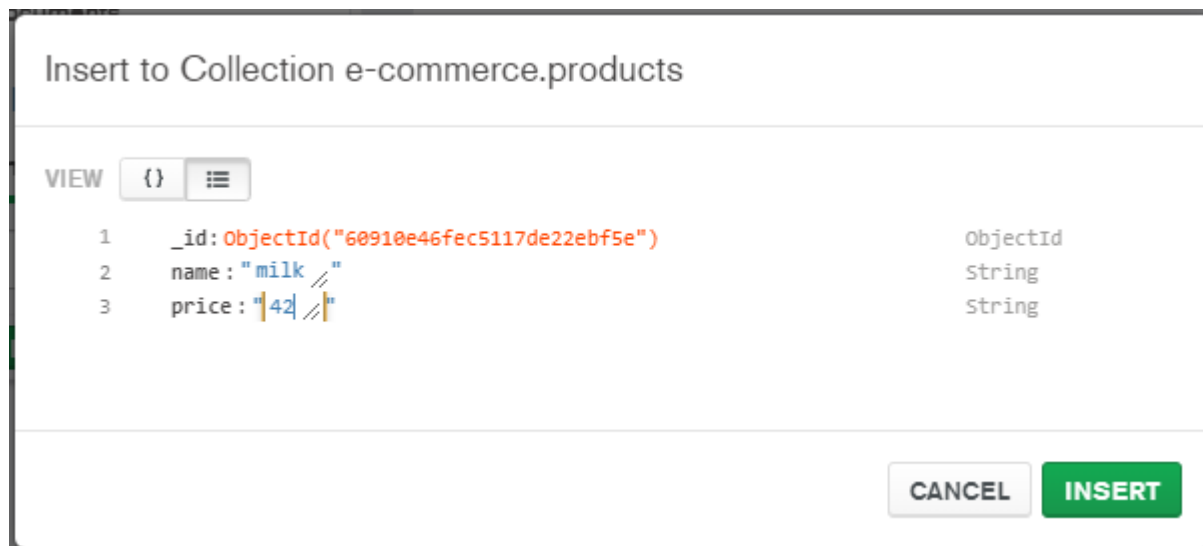
We will click on create a Database and give the database name as e-commerce and collection name as products. Click on create and e-commerce database is created and inside it products collection is also created and we have data inside it. So now, click on INSERT DOCUMENT, and we'll give few fields to it



We will click on ADD DATA,



Once we click on it,



The object is inserted inside the products collection. This server is running on the port: 27017.

e-commerce.products

DOCUMENTS 1 TOTAL SIZE 51B AVG. SIZE 51B INDEXES 1 TOTAL SIZE 4.0KB AVG. SIZE 4.0KB

Documents Aggregations Schema Explain Plan Indexes Validation

FILTER { field: 'value' } **OPTIONS** **FIND** **RESET** **REFRESH**

ADD DATA **VIEW** **{} {} {} {}** Displaying documents 0 - 1 of 1 **REFRESH**

```

_id: ObjectId("60910e46fec5117de22ebf5e")
name: "milk"
price: "42"

```

In our MongoDBFiles folder, if we open storage.bson file, it would not show the data to us. The weird symbols we see are binary symbols.

Name	Date modified	Type
diagnostic.data	5/4/2021 2:40 PM	File folder
journal	5/4/2021 2:27 PM	File folder
_mdb_catalog.wt	5/4/2021 2:33 PM	WT File
collection-0--5063251782234133201.wt	5/4/2021 2:28 PM	WT File
collection-2--5063251782234133201.wt	5/4/2021 2:28 PM	WT File
collection-4--5063251782234133201.wt	5/4/2021 2:38 PM	WT File
collection-7--5063251782234133201.wt	5/4/2021 2:37 PM	WT File
index-1--5063251782234133201.wt	5/4/2021 2:28 PM	WT File
index-3--5063251782234133201.wt	5/4/2021 2:28 PM	WT File
index-5--5063251782234133201.wt	5/4/2021 2:33 PM	WT File
index-6--5063251782234133201.wt	5/4/2021 2:38 PM	WT File
index-8--5063251782234133201.wt	5/4/2021 2:35 PM	WT File
mongod.lock	5/4/2021 2:27 PM	LOCK File
sizeStorer.wt	5/4/2021 2:39 PM	WT File
storage.bson	5/4/2021 2:27 PM	BSON File
WiredTiger	5/4/2021 2:27 PM	File
WiredTiger.lock	5/4/2021 2:27 PM	LOCK File
WiredTiger.turtle	5/4/2021 2:40 PM	TURTLE File
WiredTiger.wt	5/4/2021 2:40 PM	WT File
WiredTigerHS.wt	5/4/2021 2:27 PM	WT File

We have created e-commerce and inside it we can have multiple collections. All these collections are being managed by MongoDB.