COMP-308 Summer 2023

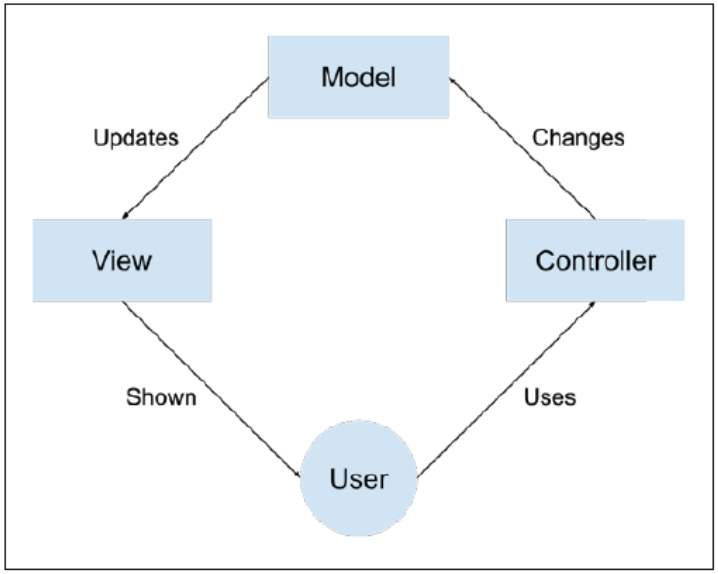
Introduction to MEAN

Objectives:

- ❑ Describe MEAN and MERN stack architecture.
- ❑ Install and run MongoDB.
- ❑ Install and run Node.js
- ❑ Create simple Node.js apps in Visual Studio Code

Three-tier web application development

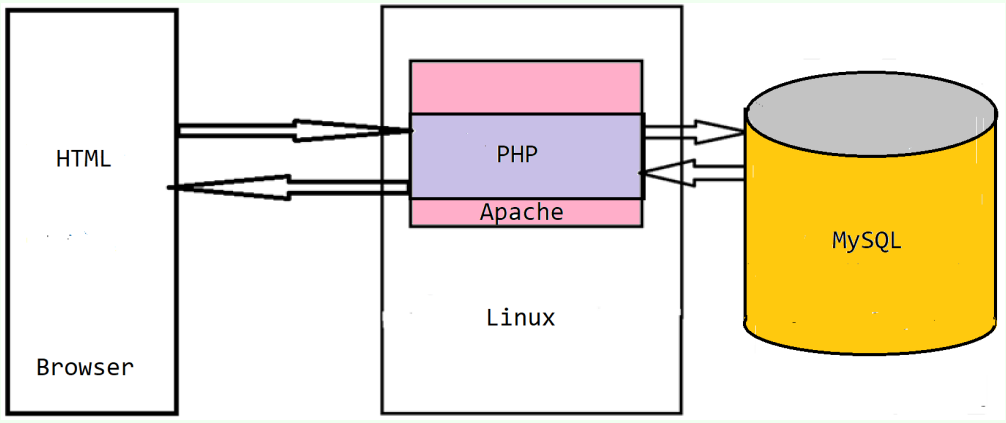
- The **three-tier architecture** consists of three important layers: **data services**, **business logic**, and **presentation**.
- In the MVC paradigm, the logic, data, and visualization are separated into three types of objects:
  - **View** - handles UI
  - **Model** - handles the data manipulation
  - **Controller** - controls the model and view

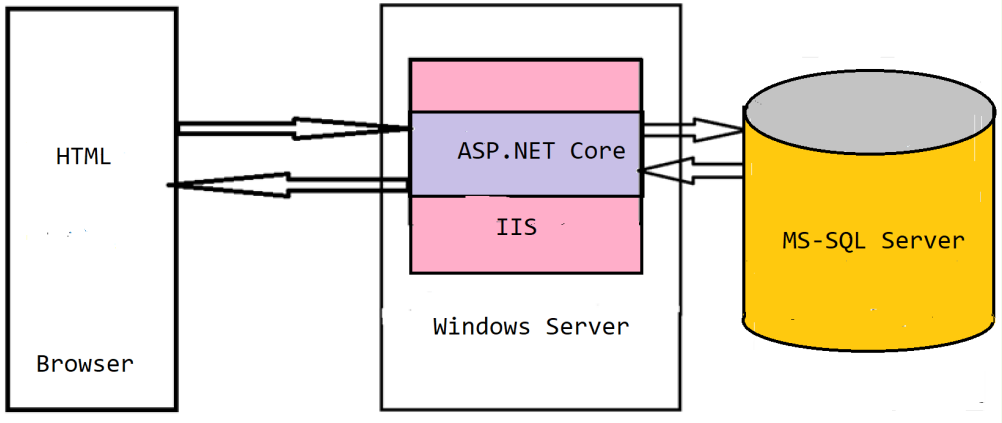


Common MVC architecture communication

Web technology stacks - recap

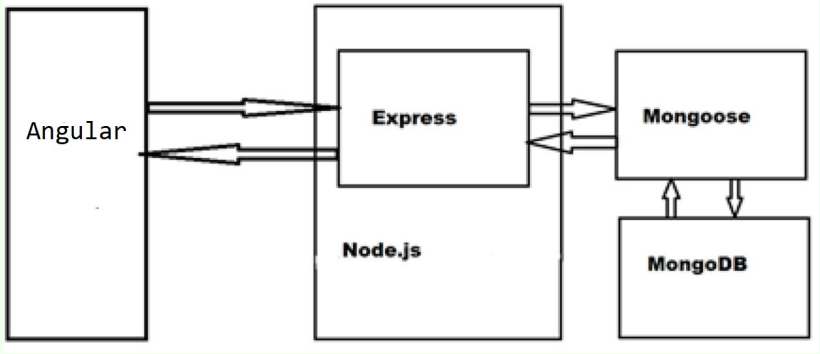
- **LAMP** stack:
  - Linux
  - Apache
  - MySQL
  - PHP/Python/Perl
- **.NET** (Core) stack:
  - .NET (Core)
  - IIS
  - ASP.NET (Core)
  - Web API and WCF
  - MS-SQL Server
- Other frameworks and tools (**Django**, Flask, etc.)
- Each layer uses a **different knowledge base**!





MEAN Stack - recap

- MEAN is an abbreviation for **MongoDB**, **Express**, **AngularJS**, and **Node.js**.
- Uses **only JavaScript - driven solutions** to implement the different parts of a web application.
- Has the following advantages:
  - A **single language** is used throughout the application
  - All the parts of the application can **support and often enforce** the use of the **MVC architecture**
  - Serialization and deserialization of data structures is no longer needed because **data marshaling is done using JSON objects**.



MEAN Stack - recap

- **MongoDB** is a scalable NoSQL database that used a JSON-like data model with dynamic schemas.
- **Express** is **a lightweight node.js web application framework**, providing a robust set of features for building single and multi-page, and hybrid web application.
- **Node.js** is a **server side JavaScript execution environment** built on Google Chrome's V8 JavaScript runtime - helps in building highly scalable and concurrent applications rapidly.
- **Angular** is a JavaScript framework developed by Google - a complete solution for rapid **front end development**.

LAMP versus MEAN - recap

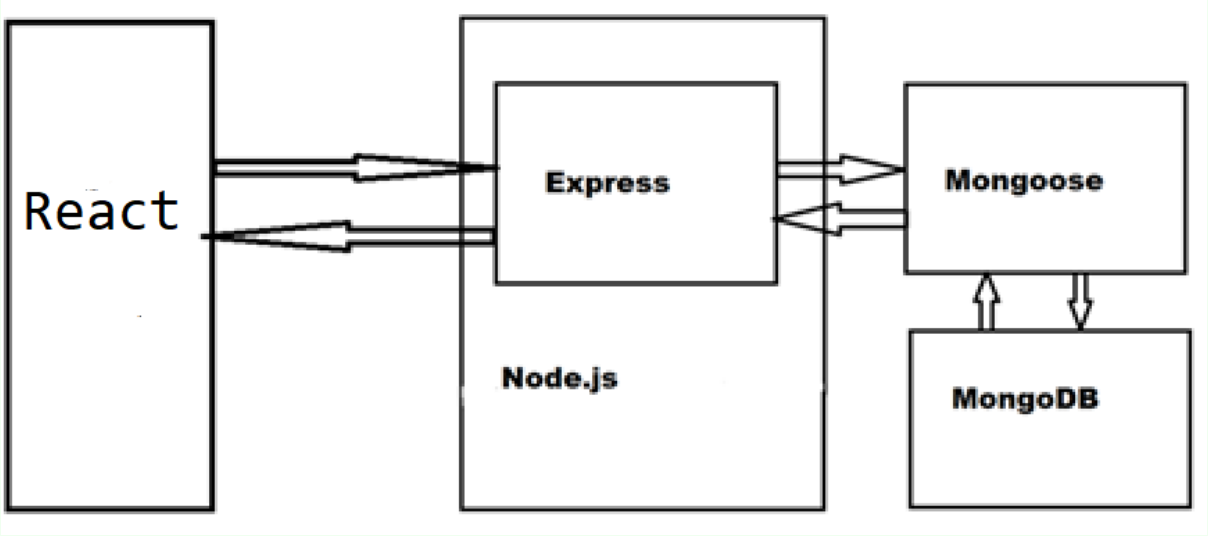
- Linux ➔ Node.js (platform)
- Apache ➔ Express.js (web server framework)
- MySQL ➔ MongoDB (persistence layer)
- PHP or Python or Perl ➔ Angular (User Interface)

Full-Stack JavaScript Advantages - recap

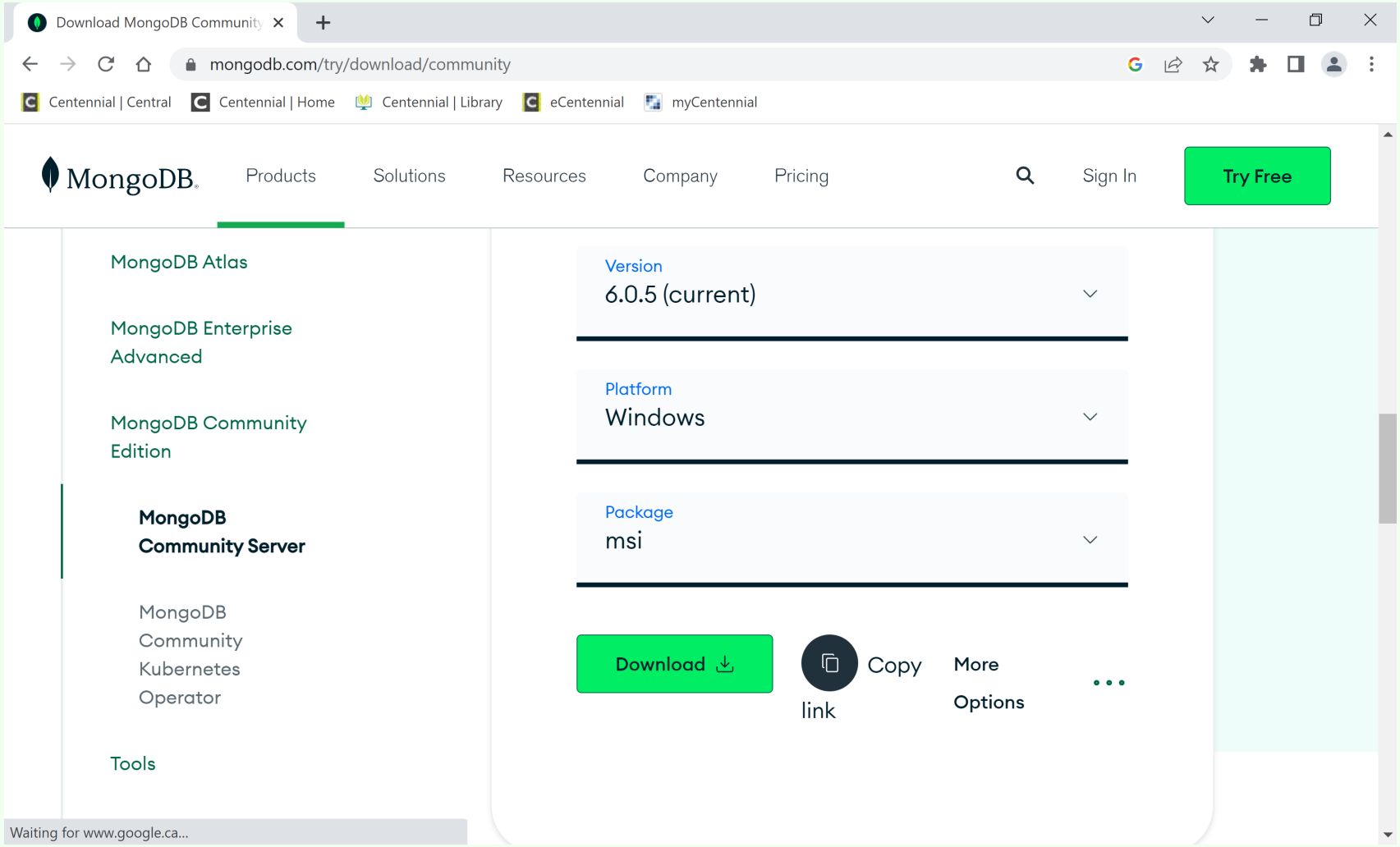
- High Performance
  - Increasing number of requests
  - Reducing response time
  - Non-blocking I/O - allows web server to handle more concurrent requests without requiring additional hardware or configuration
  - Cross-platform (Windows, Linux, MacOS)
  - One single programming language for the entire project

MERN Stack - recap

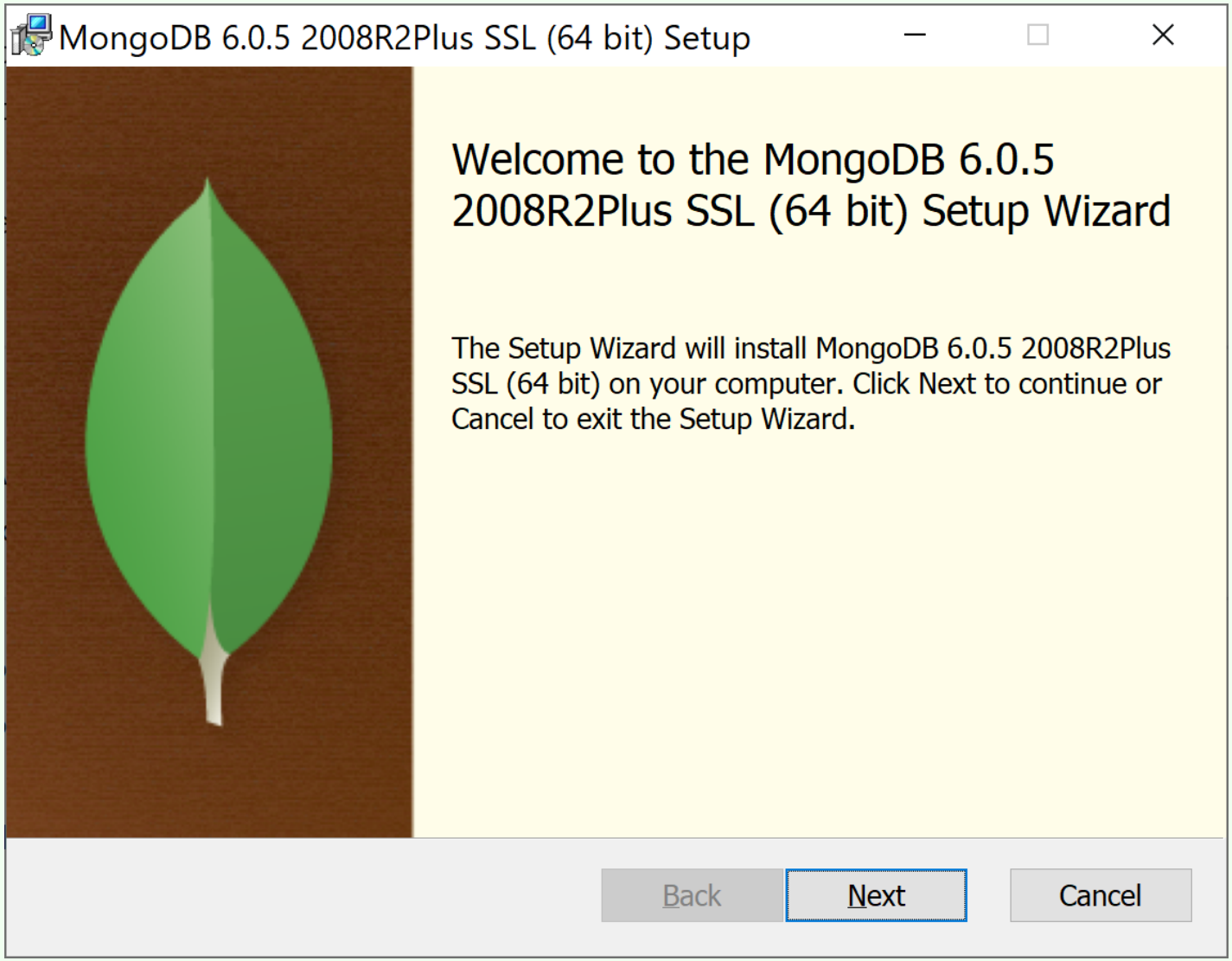
- **MongoDB**
- **Express**
- **Node.js**
- **ReactJS** is a JavaScript component-based **front-end library** maintained by Facebook – most popular for **front-end development**.



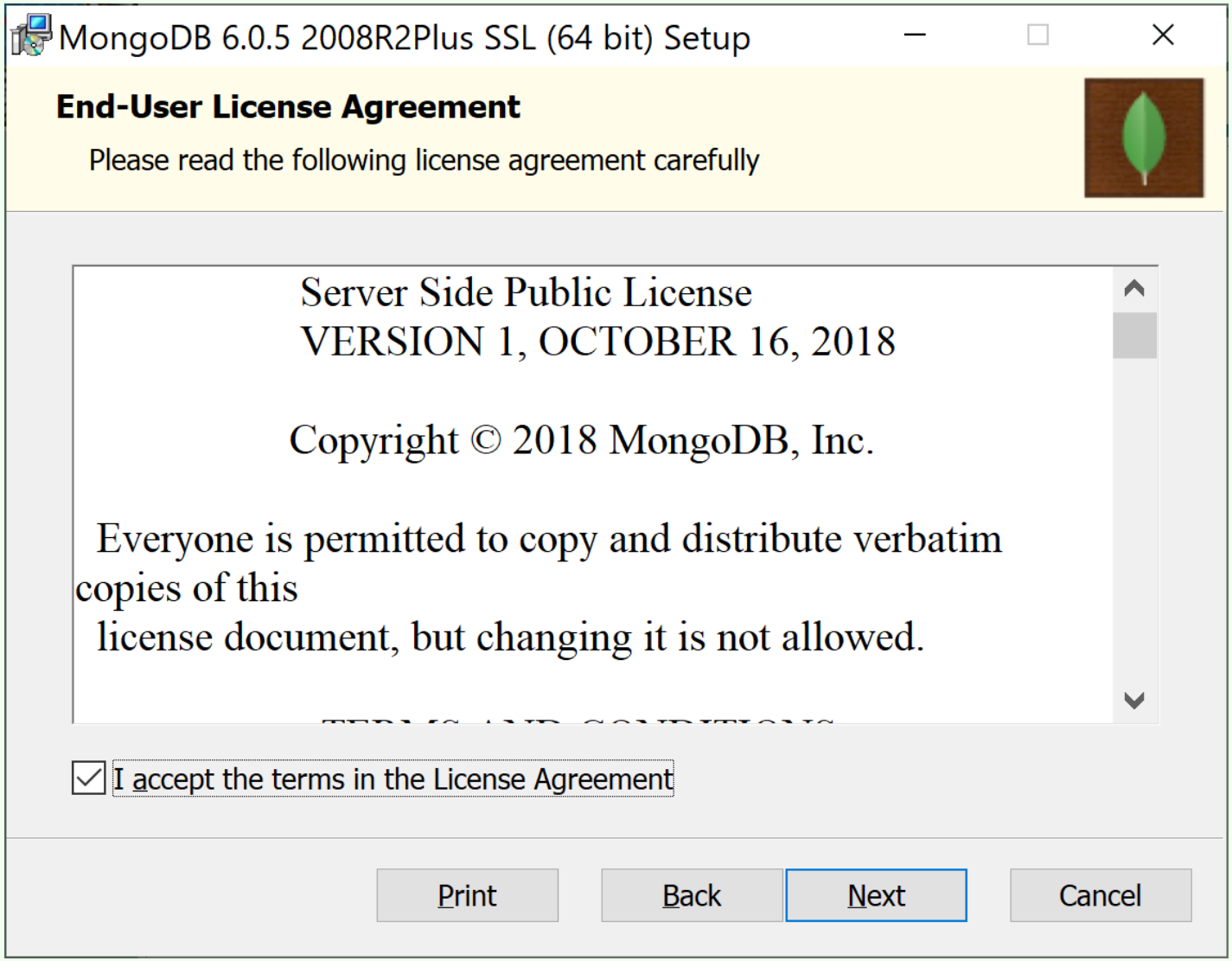
Downloading MongoDB



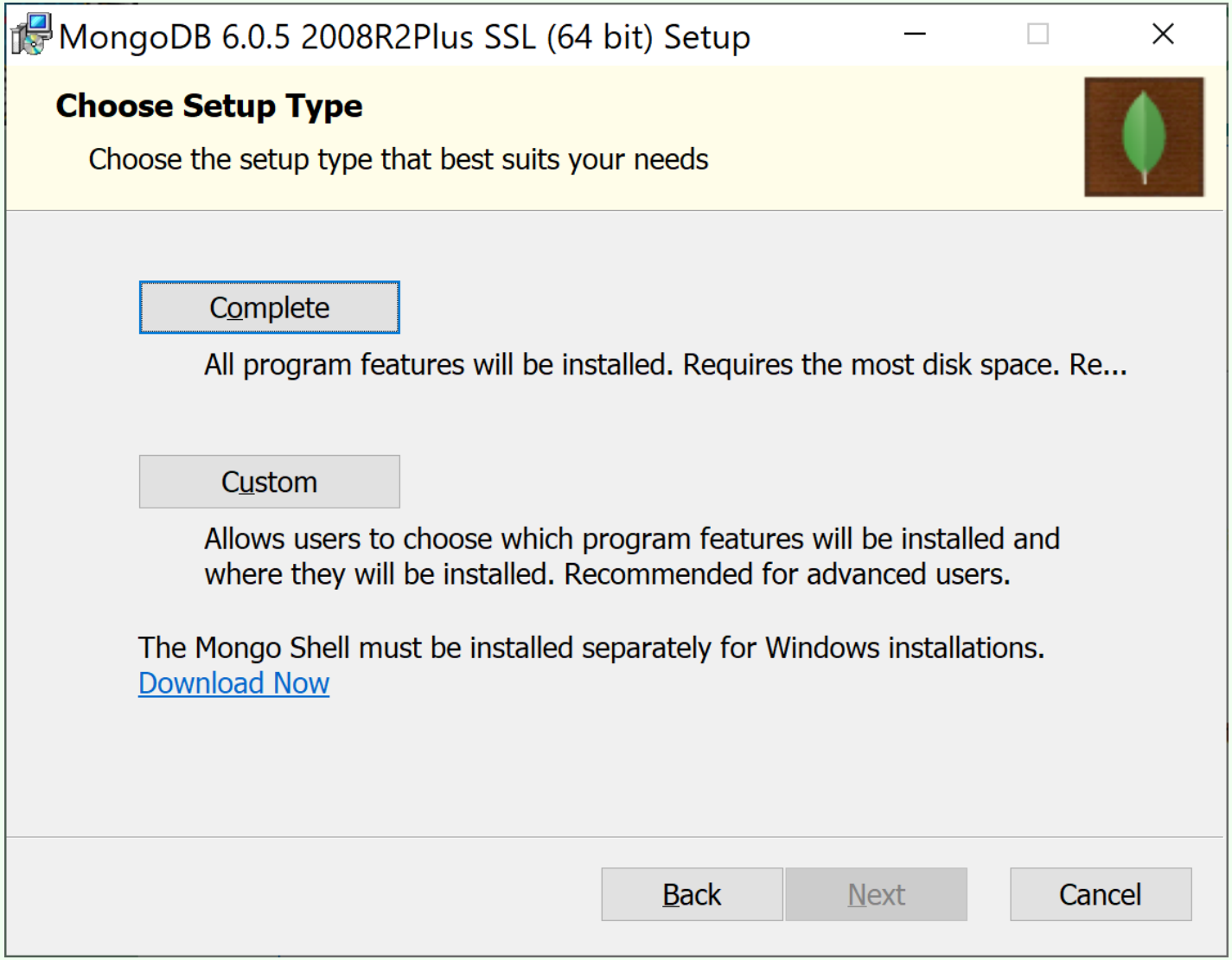
Installing MongoDB



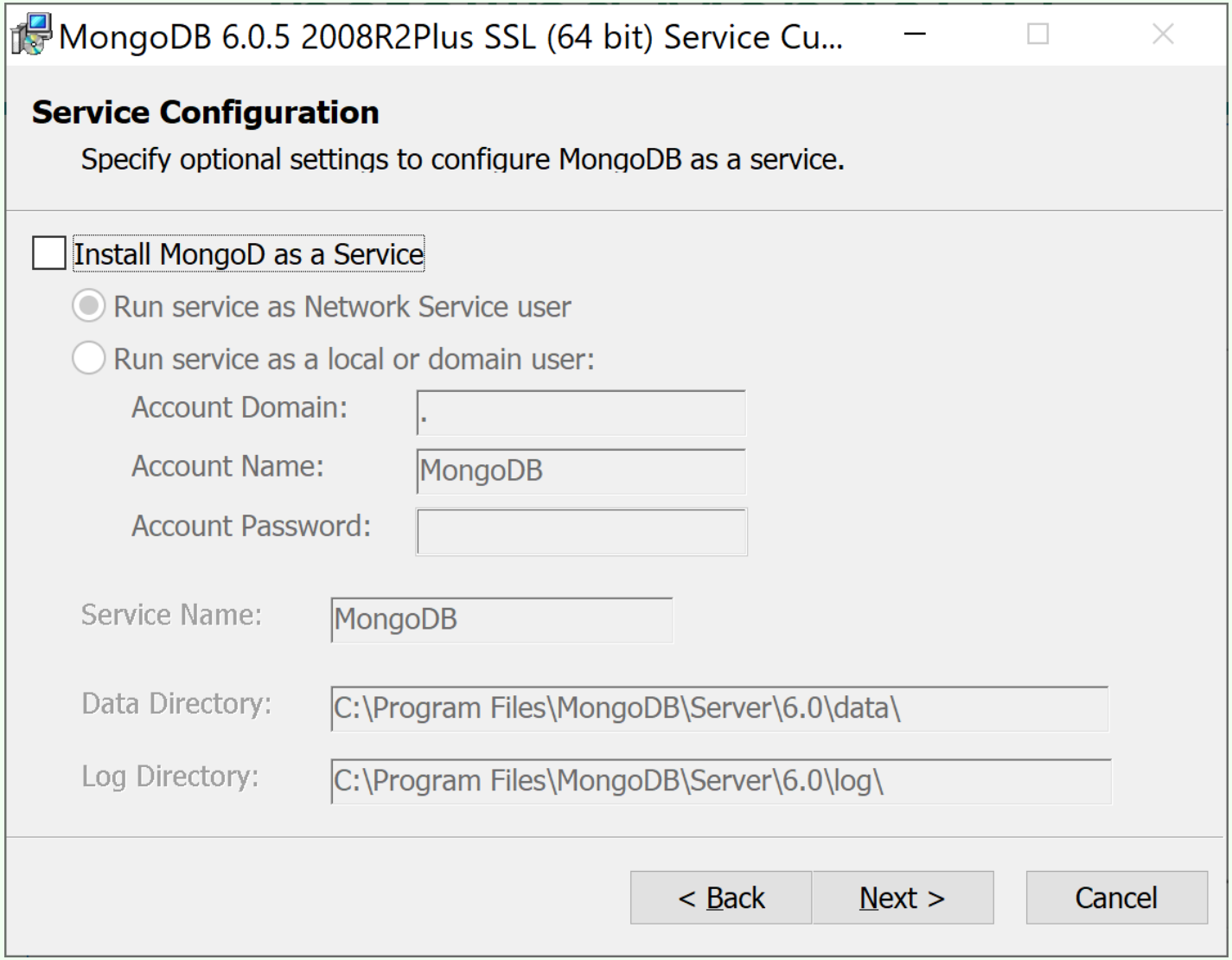
Installing MongoDB



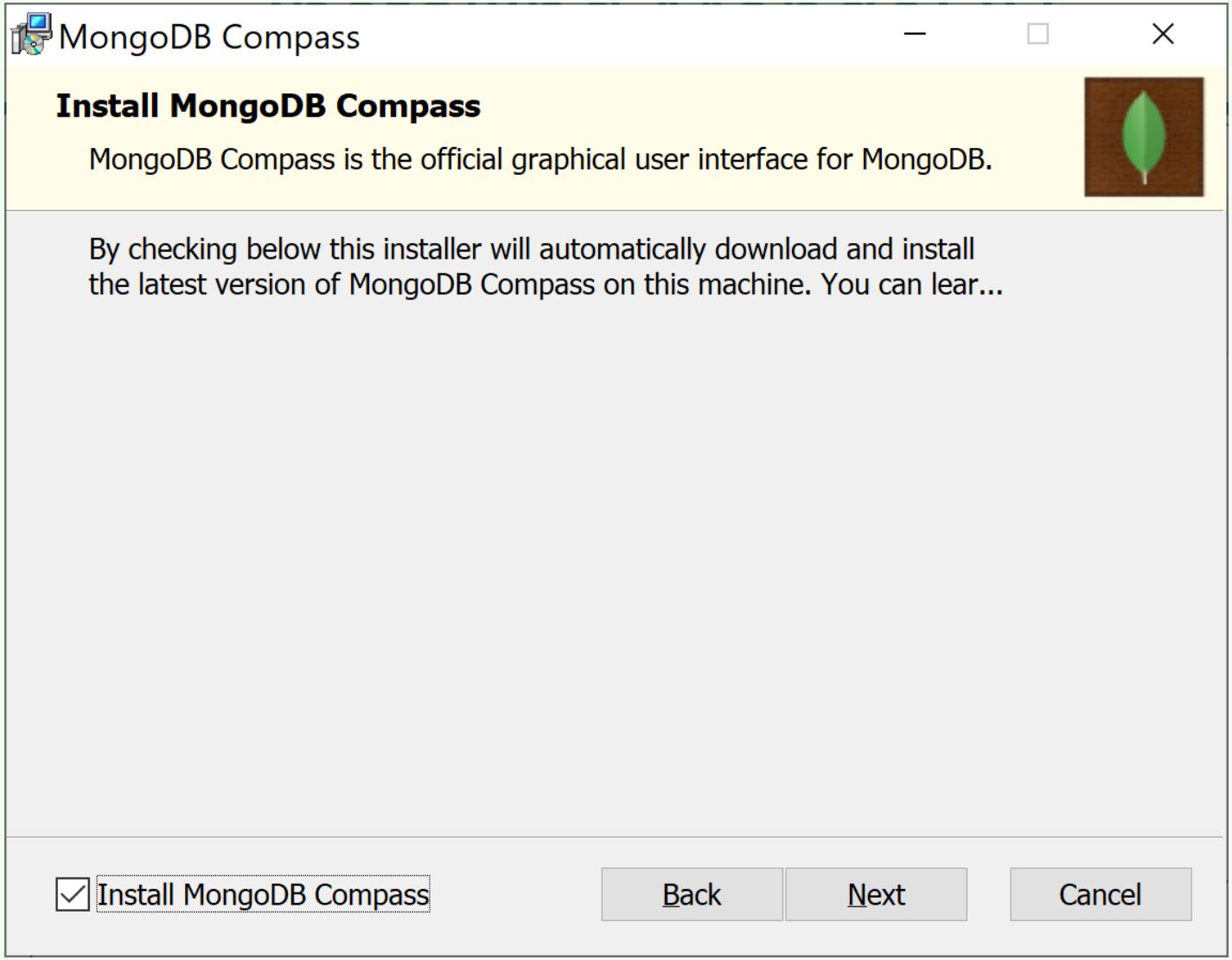
Installing MongoDB



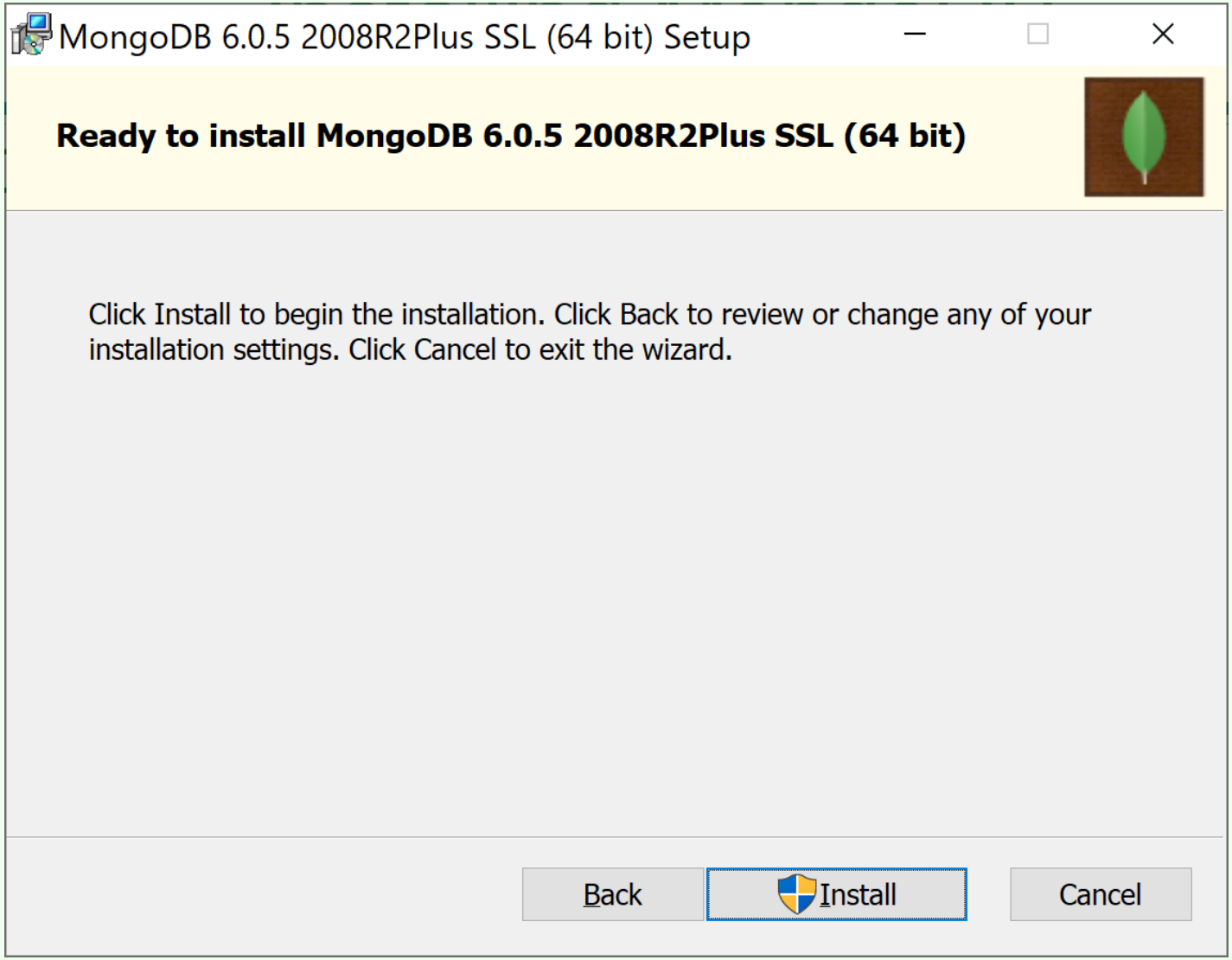
Installing MongoDB



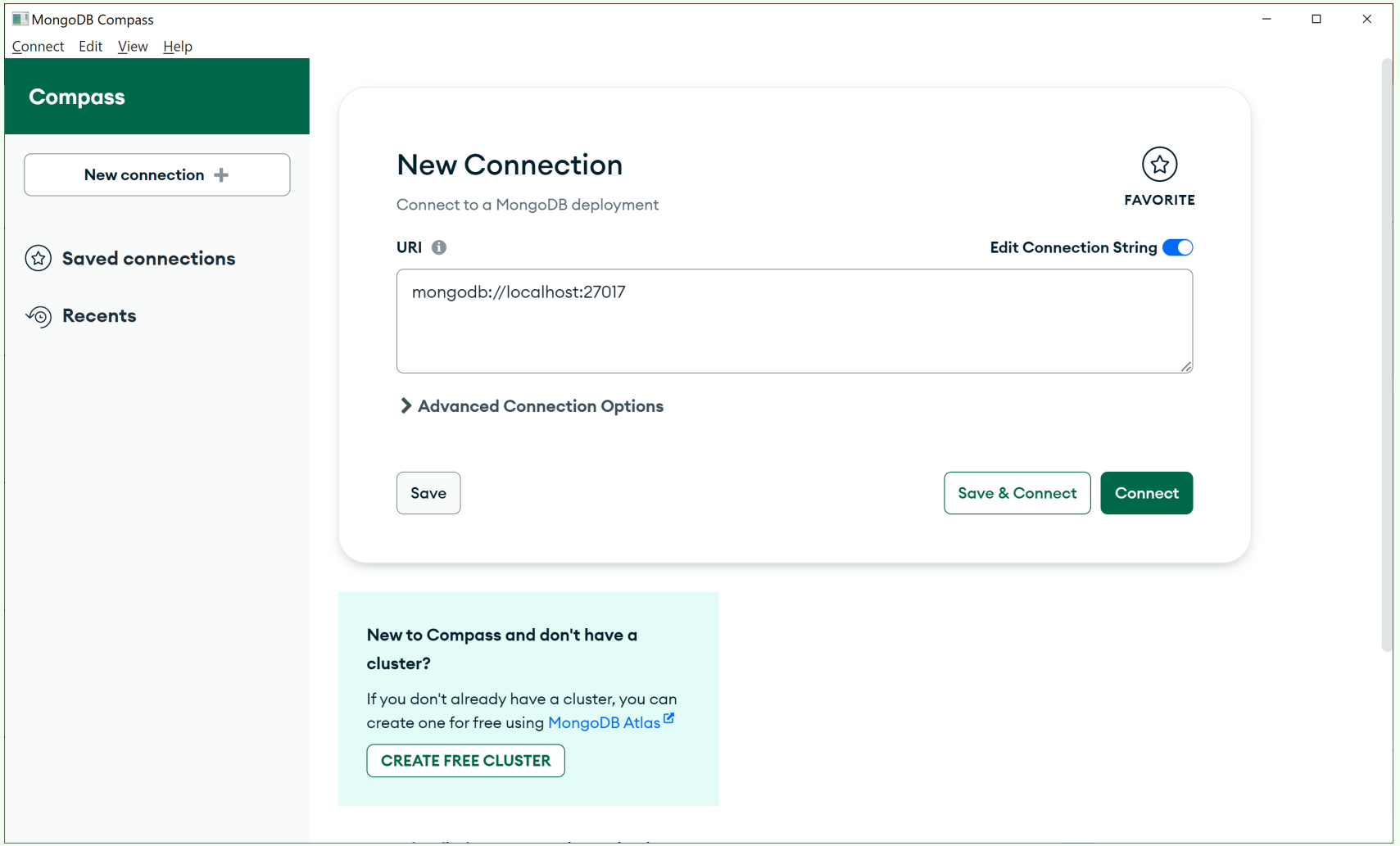
Installing MongoDB



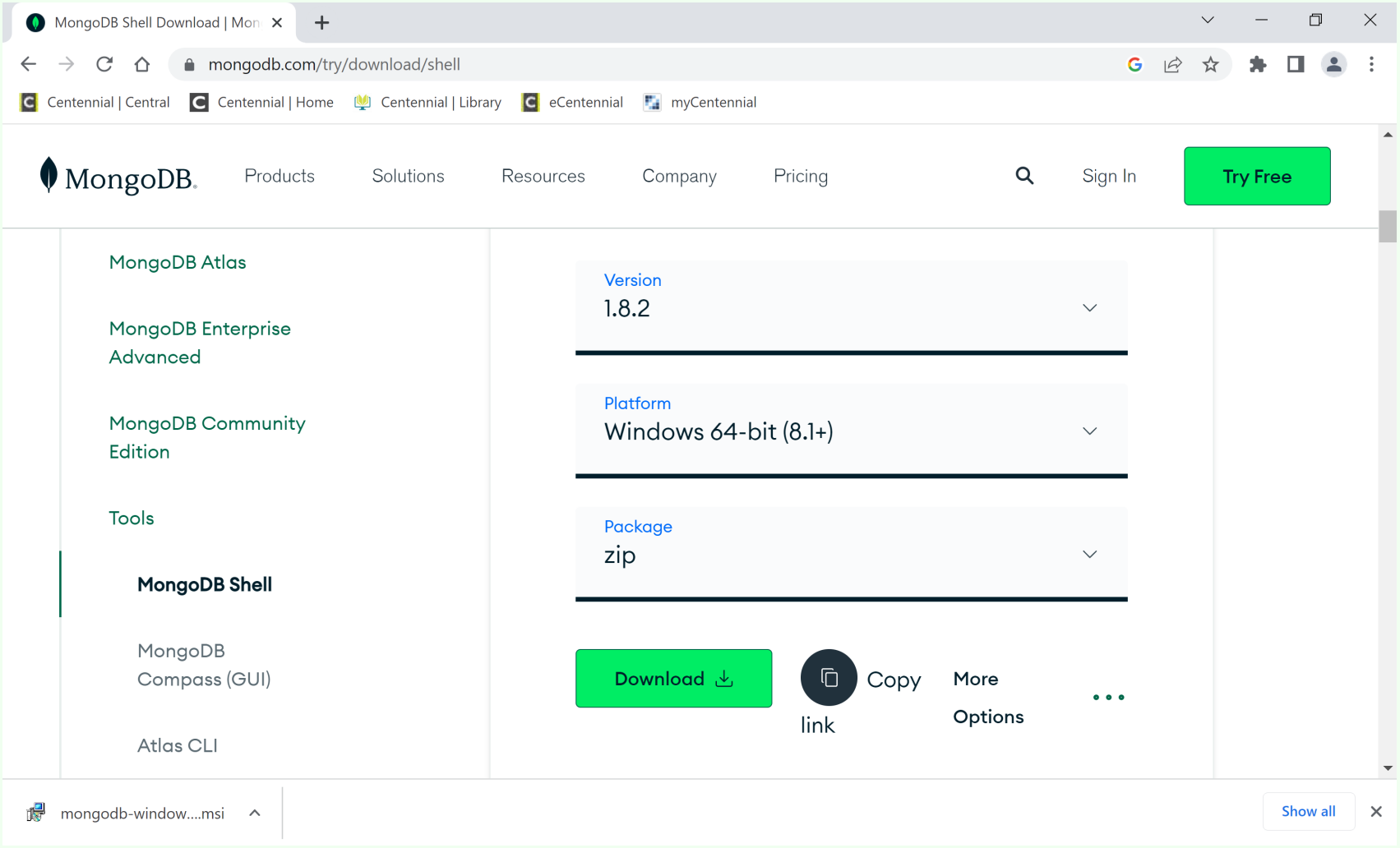
Installing MongoDB



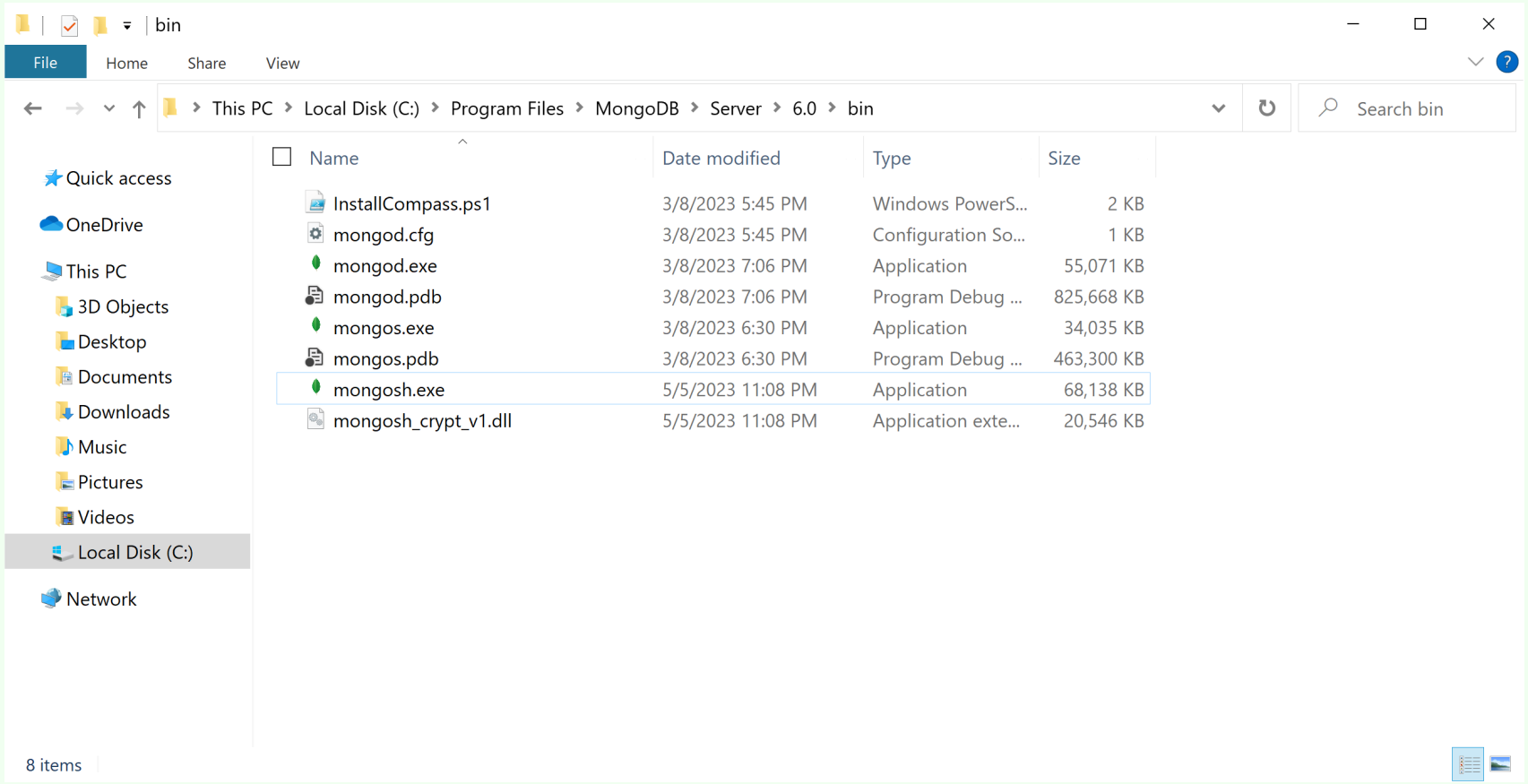
Installing MongoDB



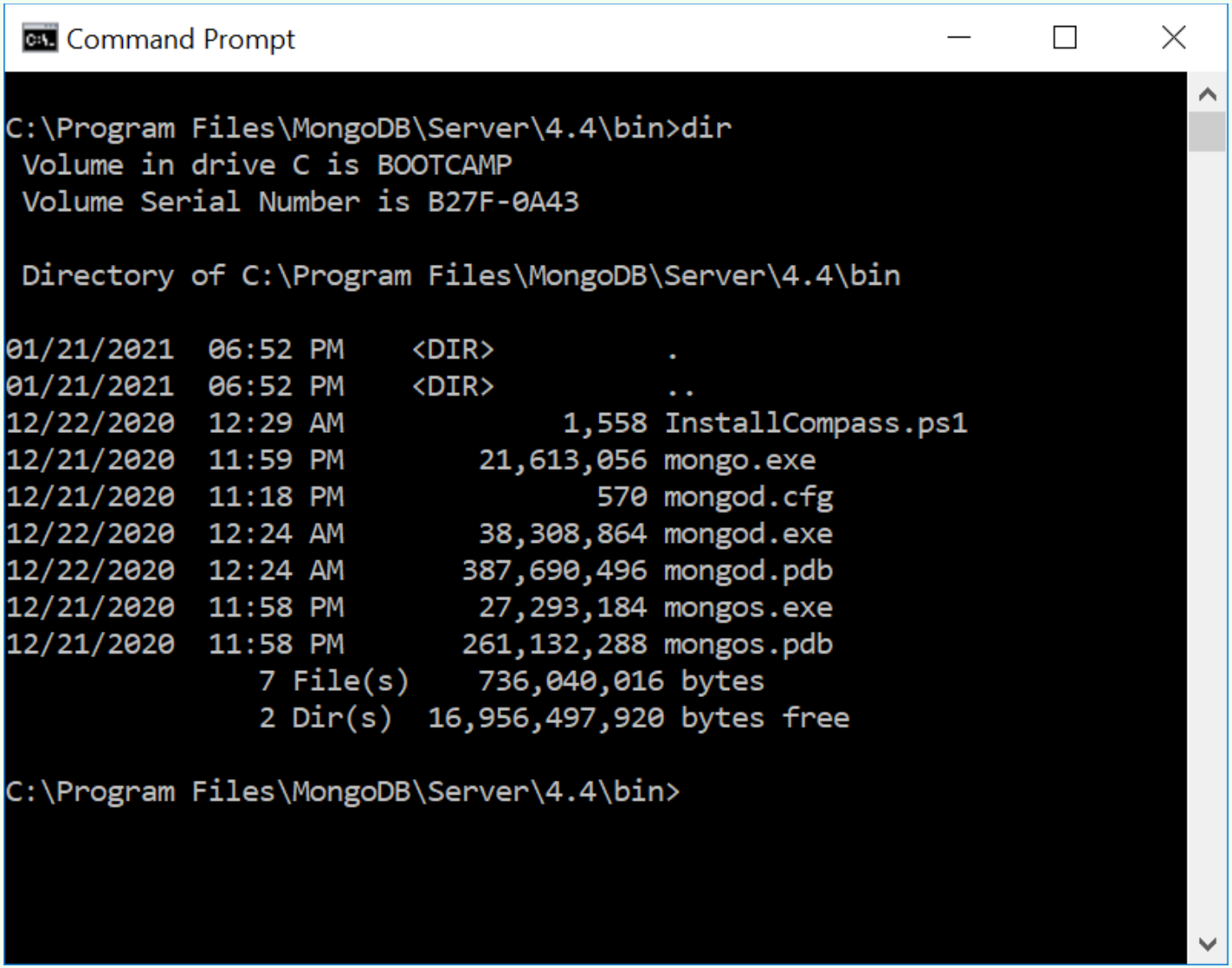
Install mongosh



MongoDB bin folder

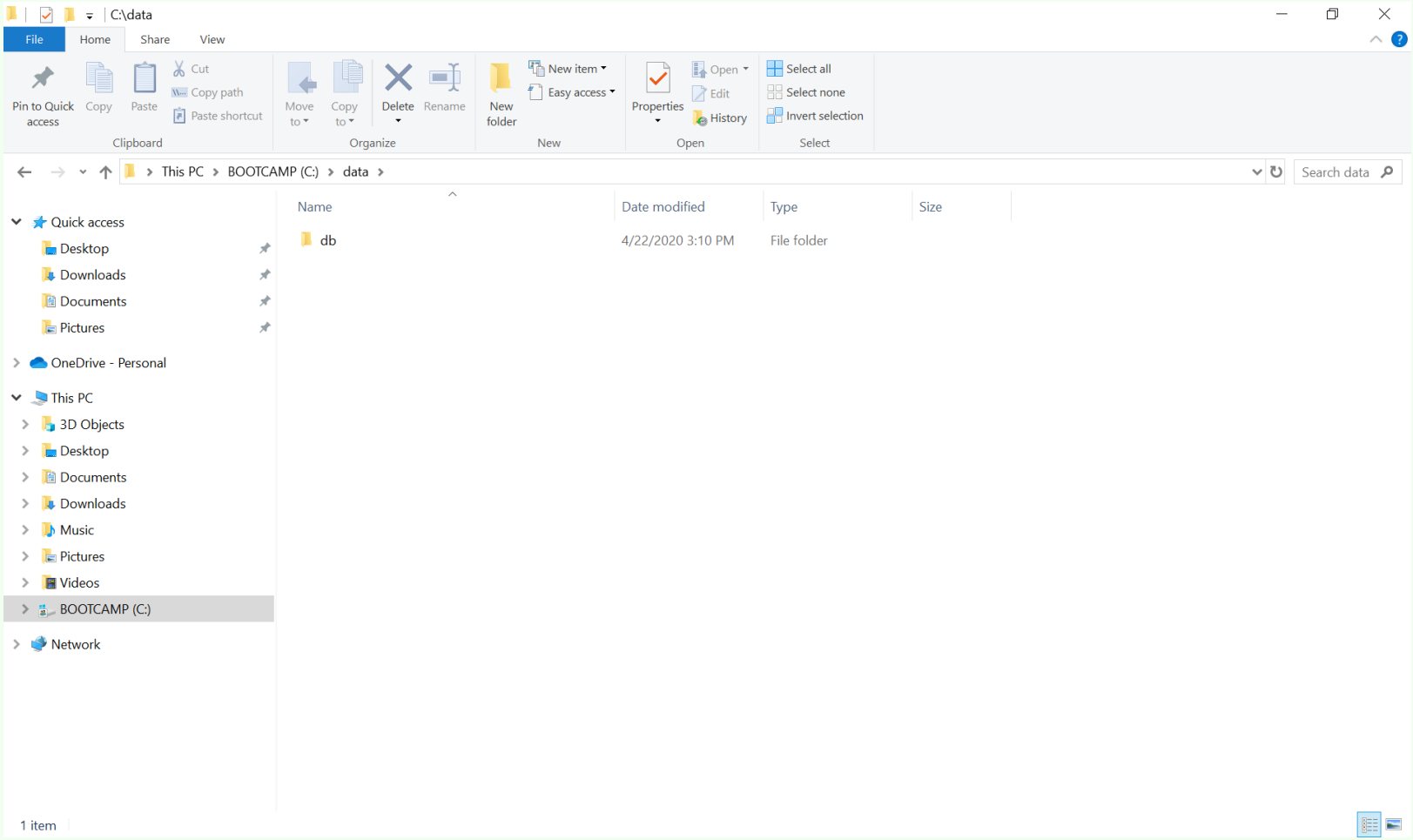


MongoDB bin folder



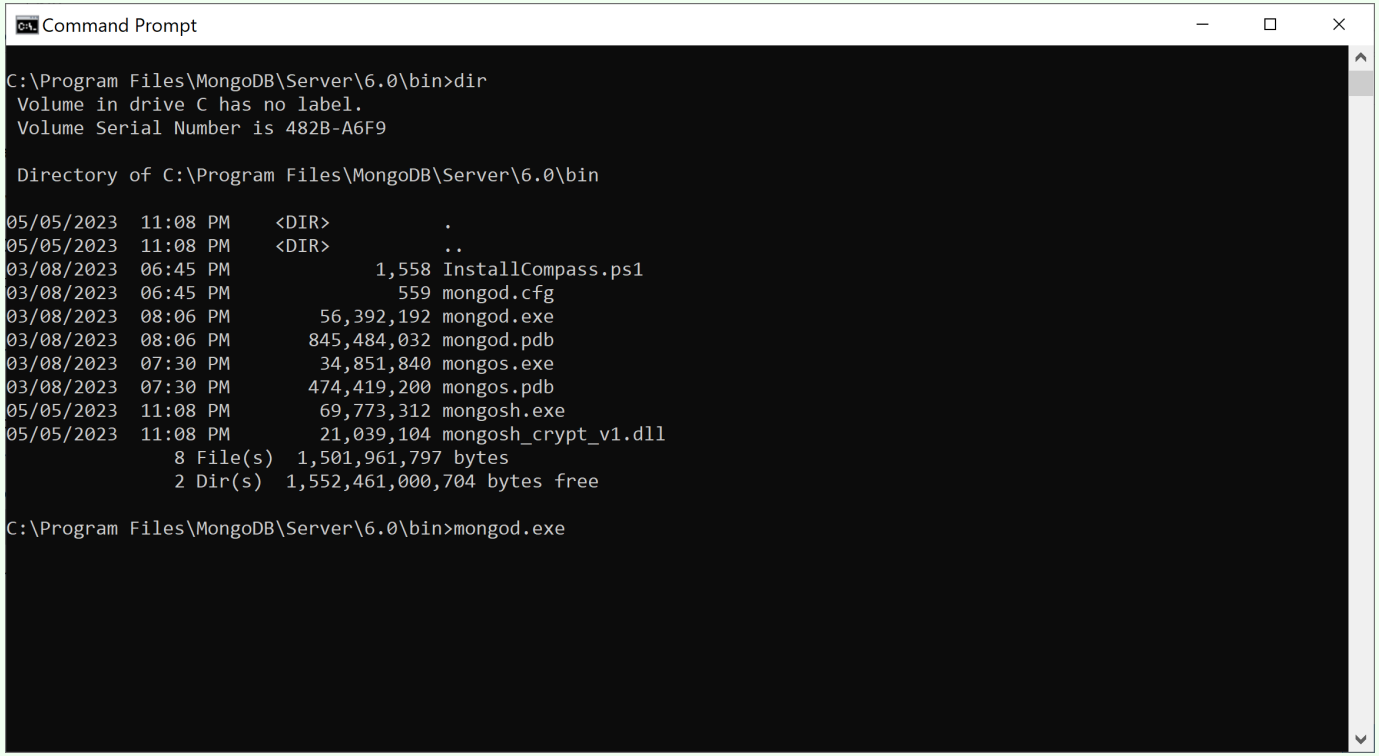
MongoDB default folder for files

- **Create MongoDB default folder** to store its files - on Windows, the default location is **C:\data\db:**



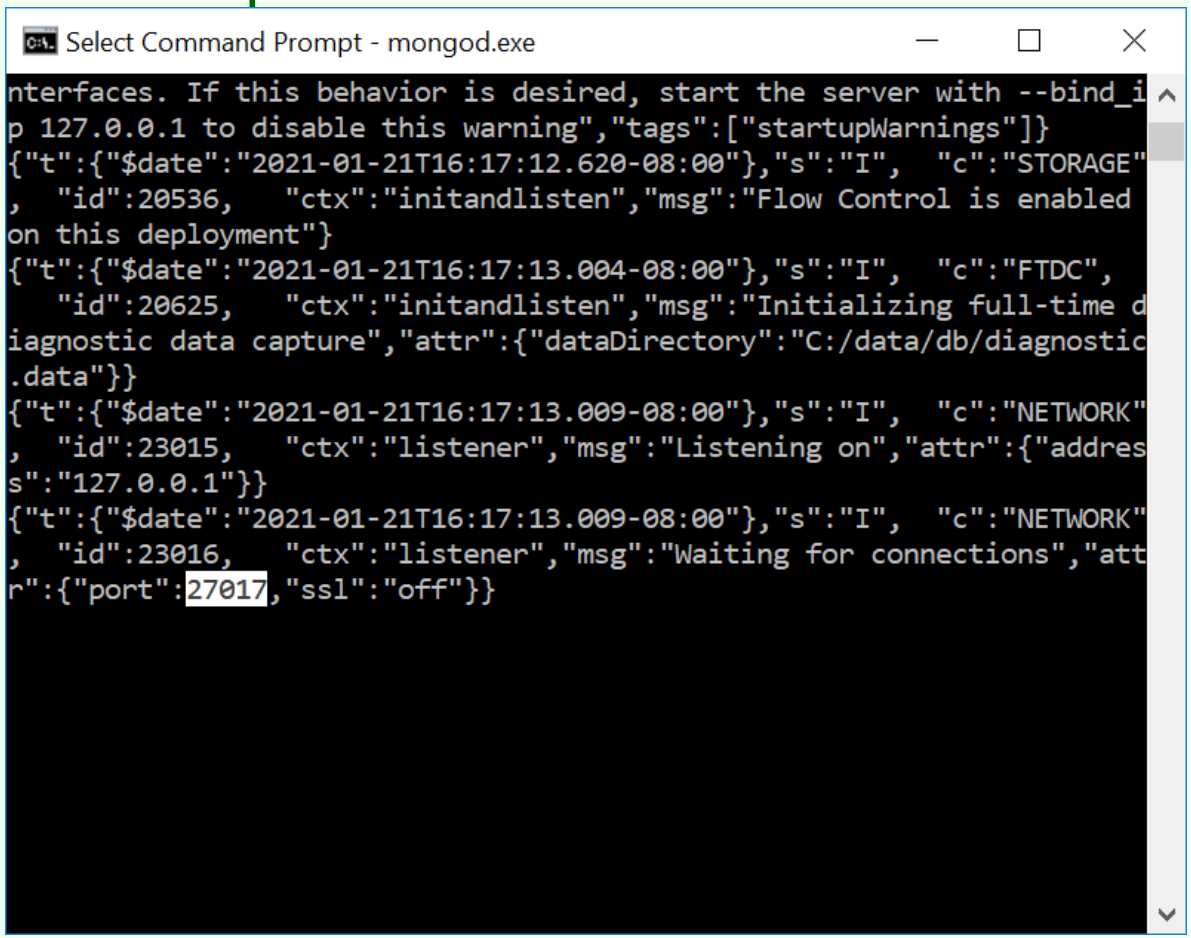
Running MongoDB

- Open the command prompt and execute the following command: **"C:\Program Files\MongoDB\Server\6.0\bin\mongod.exe",** or run **mongod** from the **bin** directory:



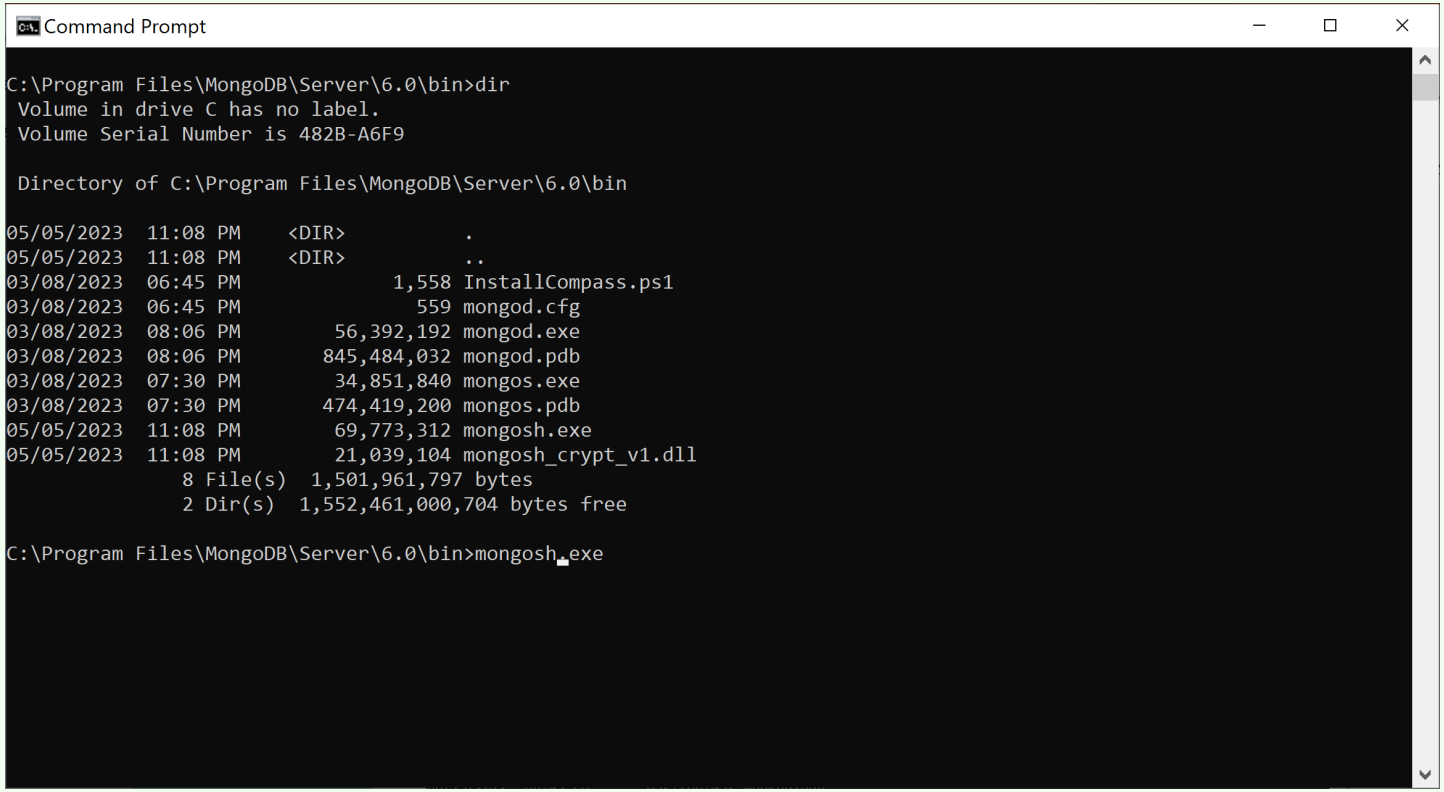
Running MongoDB

- The main **MongoDB service** will start listening to the default 27017 port:



Using the MongoDB shell

- MongoDB shell allows to you to interact with your server instance using the command line.
- To start the shell, navigate to the MongoDB bin folder and run the **mongosh** service as follows:



Using the MongoDB shell

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
Using MongoDB:          6.0.5
Using Mongosh:          1.8.2

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

------
   The server generated these startup warnings when booting
   2023-05-05T23:18:57.352-04:00: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
   2023-05-05T23:18:57.353-04:00: This server is bound to localhost. Remote systems will be unable to connect to this se
rver. 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
------

------
   Enable MongoDB's free cloud-based monitoring service, which will then receive and display
   metrics about your deployment (disk utilization, CPU, operation statistics, etc).

   The monitoring data will be available on a MongoDB website with a unique URL accessible to you
   and anyone you share the URL with. MongoDB may use this information to make product
   improvements and to suggest MongoDB products and deployment options to you.

   To enable free monitoring, run the following command: db.enableFreeMonitoring()
   To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
------

Warning: Found ~/.mongorc.js, but not ~/.mongoshrc.js. ~/.mongorc.js will not be loaded.
  You may want to copy or rename ~/.mongorc.js to ~/.mongoshrc.js.
test>
```

Testing the database

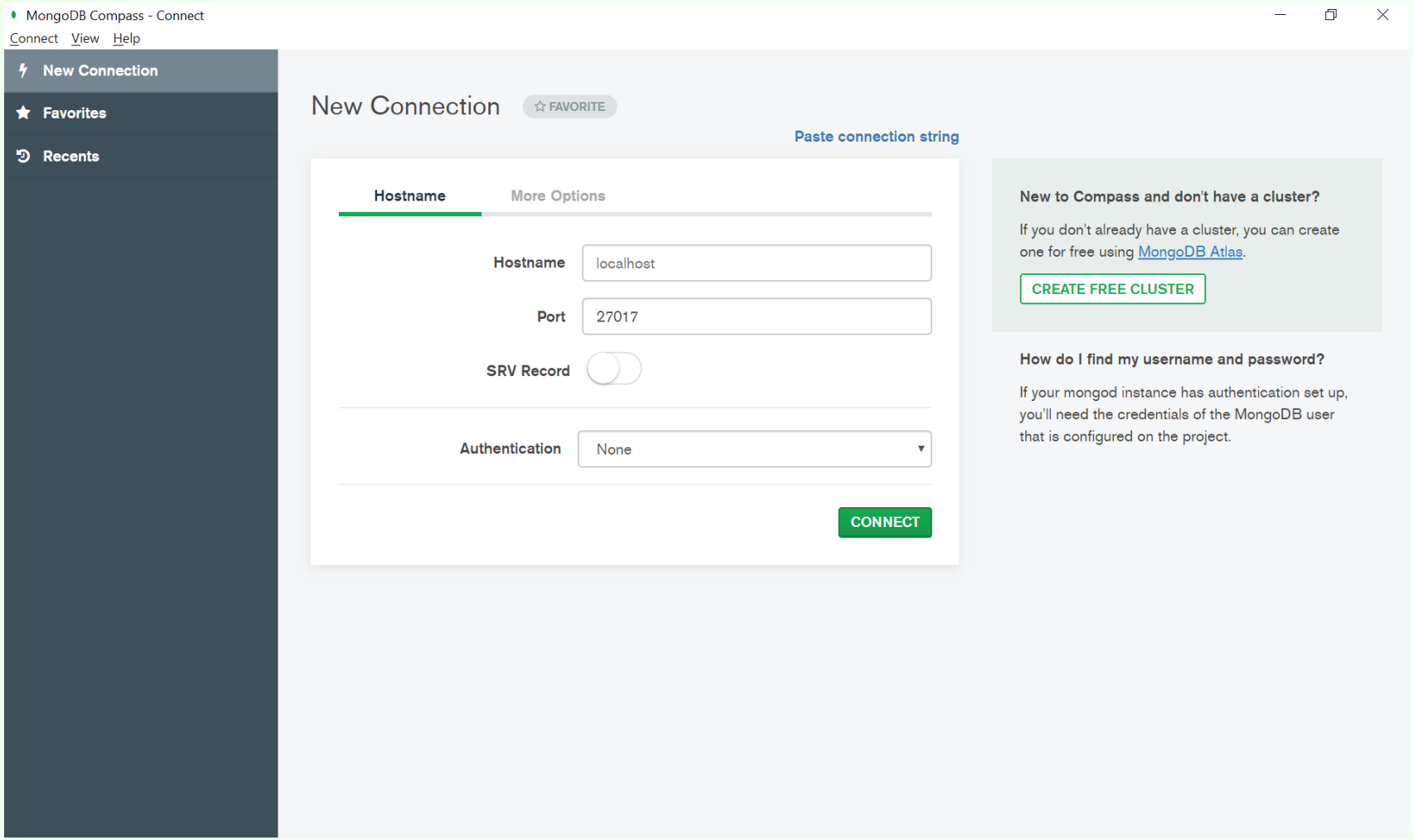
- To test your database, run the following command:

> db.courses.insert({title: "COMP-308 Emerging Technologies"}).

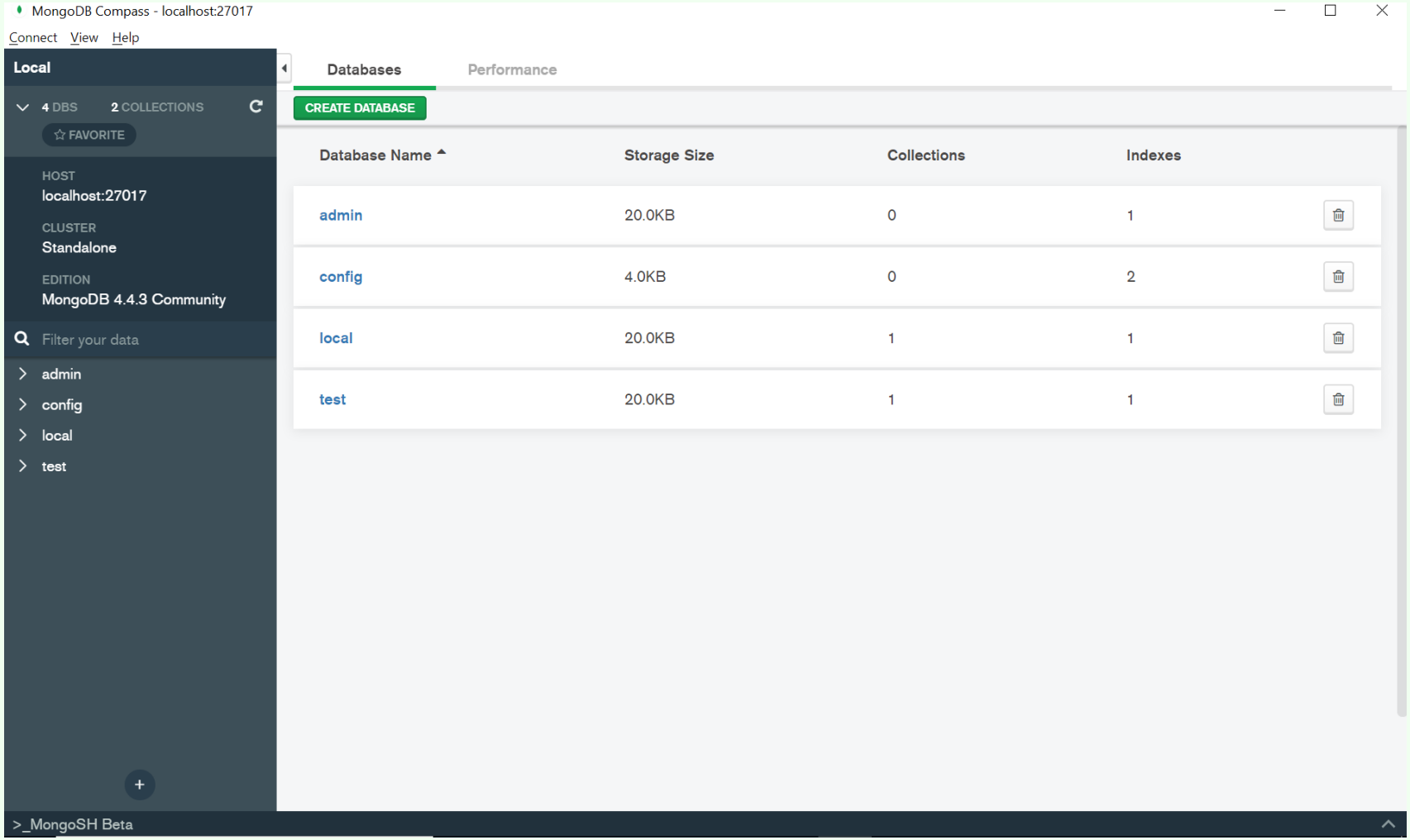
- To retrieve the course object, execute the following command:

> db.courses.find()

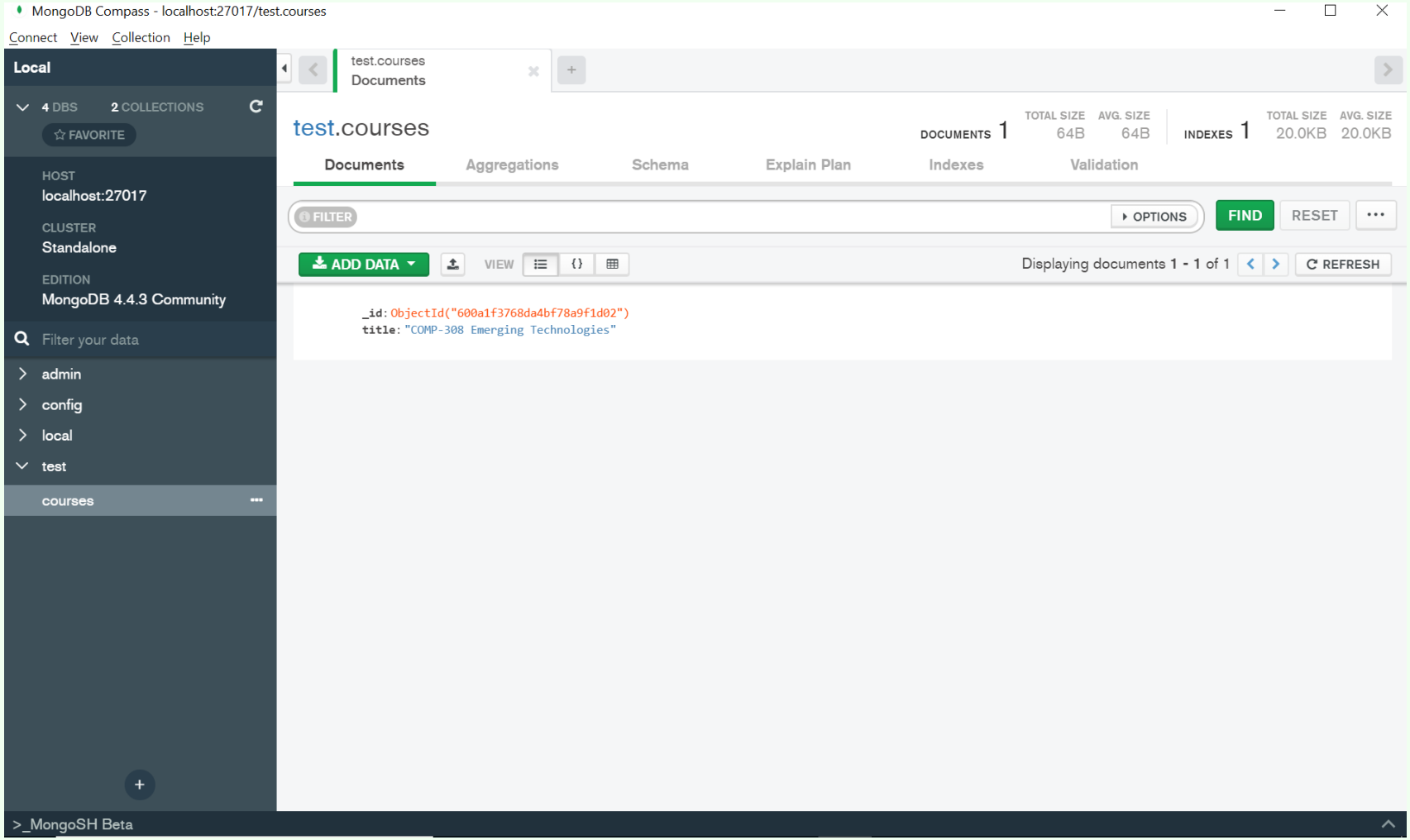
MongoDB Compass



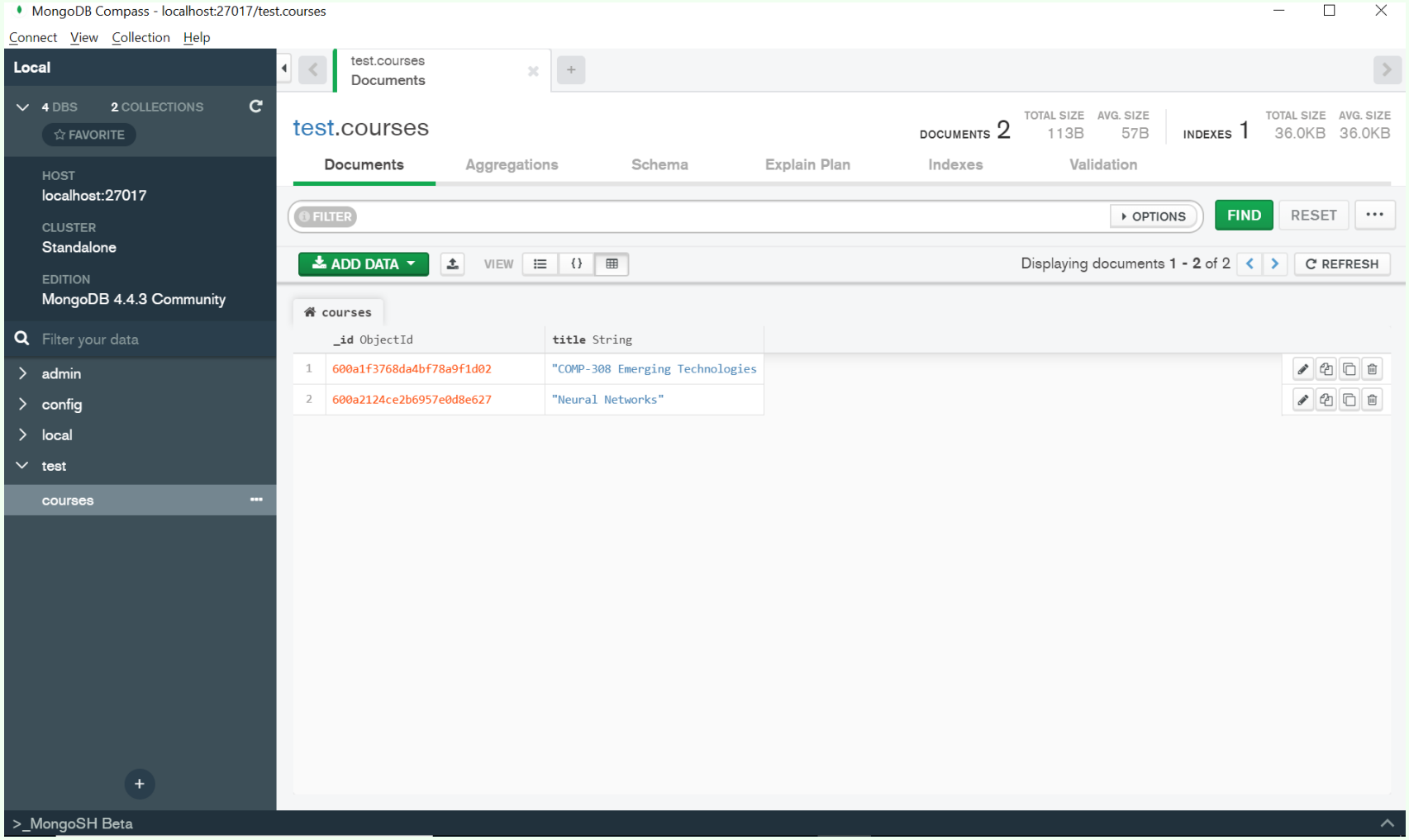
MongoDB Compass



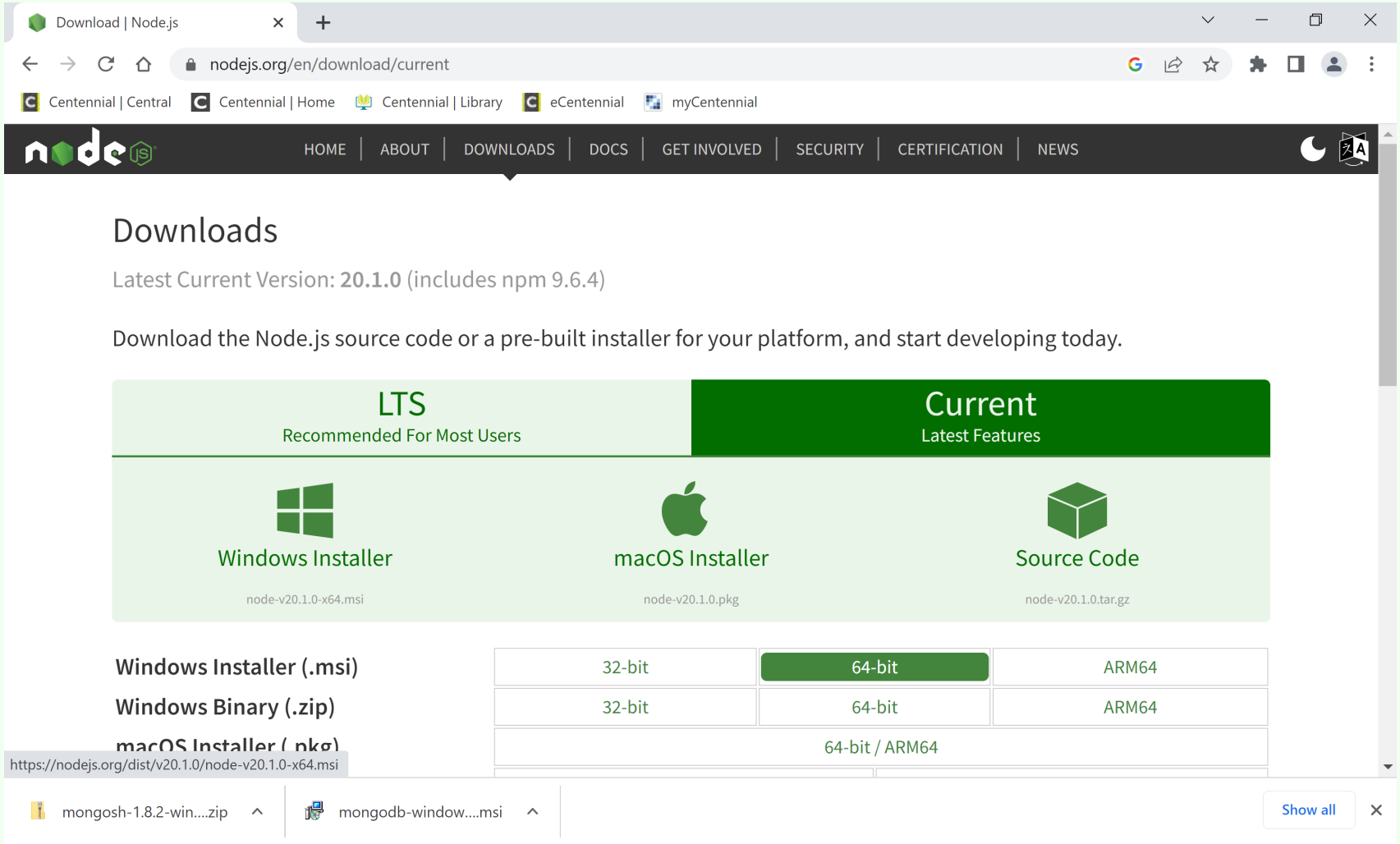
Add Documents



Add Documents

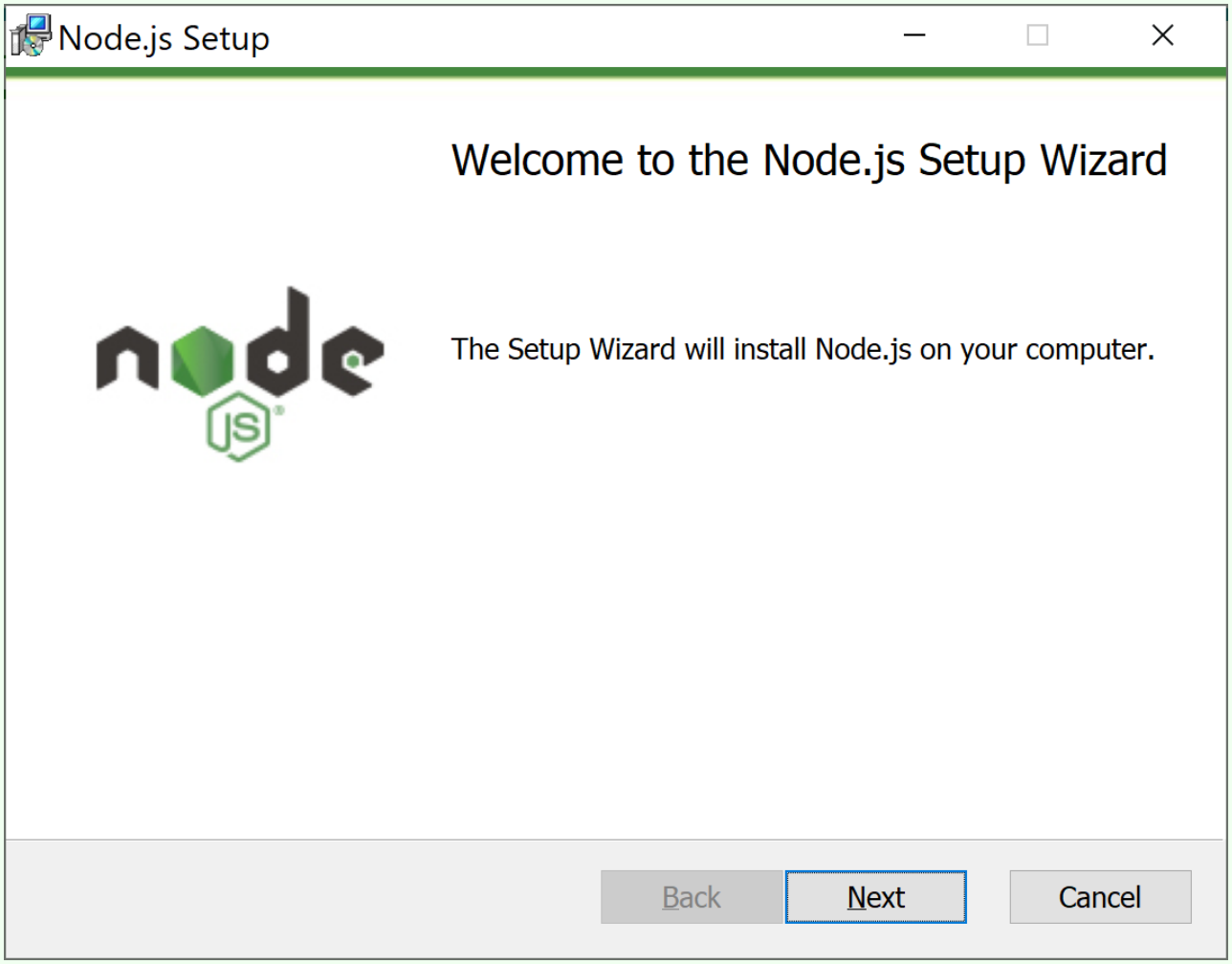


Installing Node.js

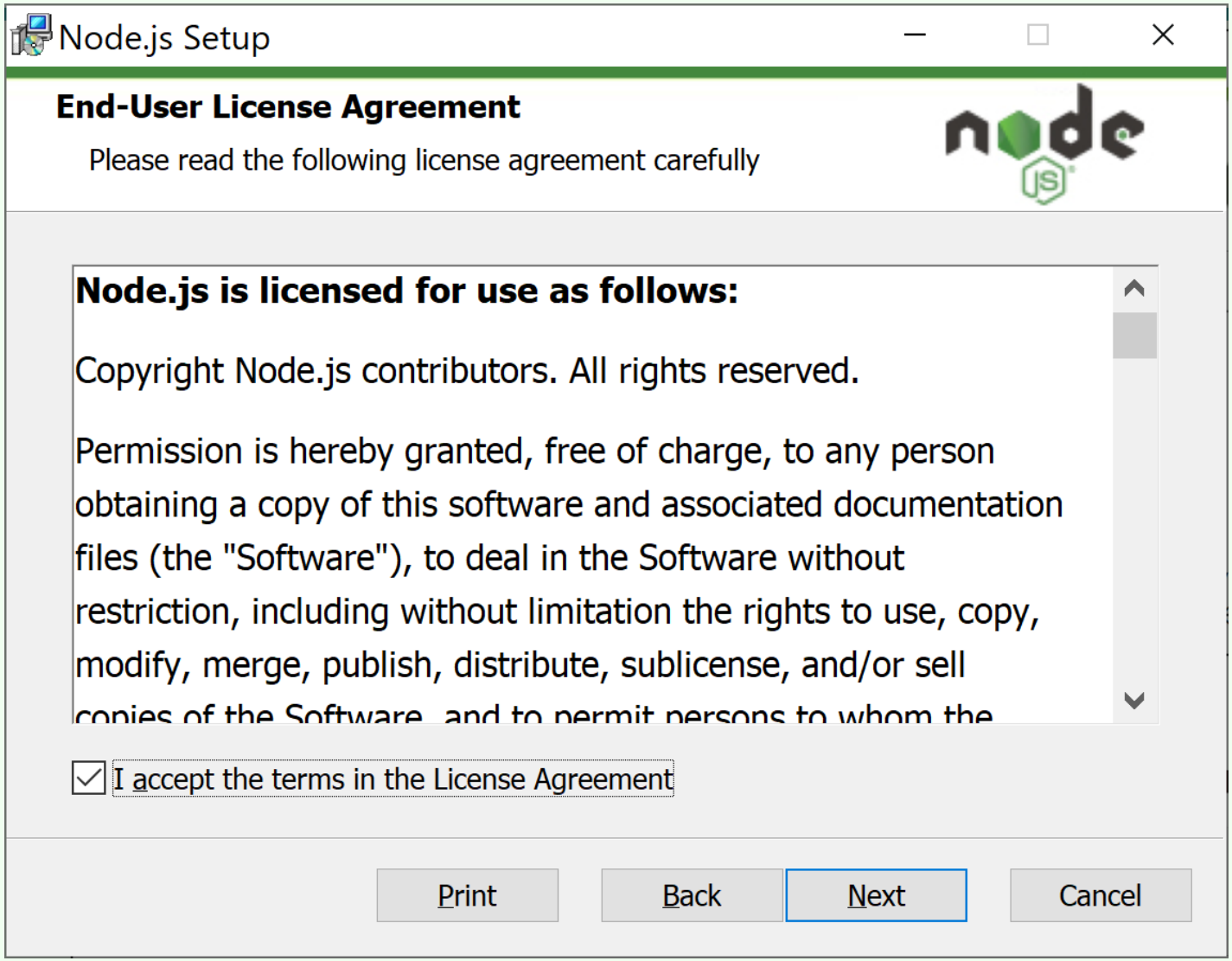


Installing Node.js

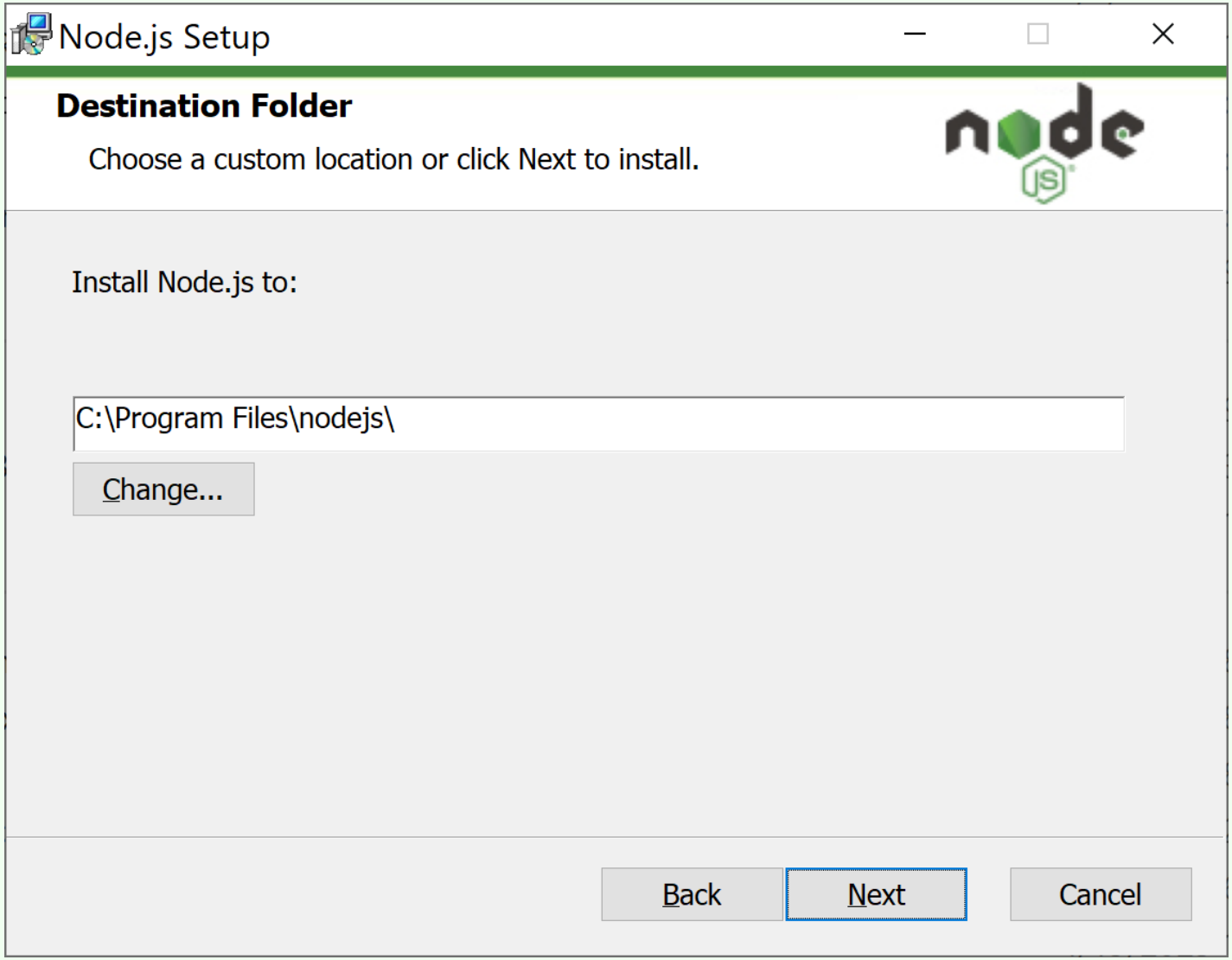
- Install 64bit node-v20.1.0-x64.msi file.



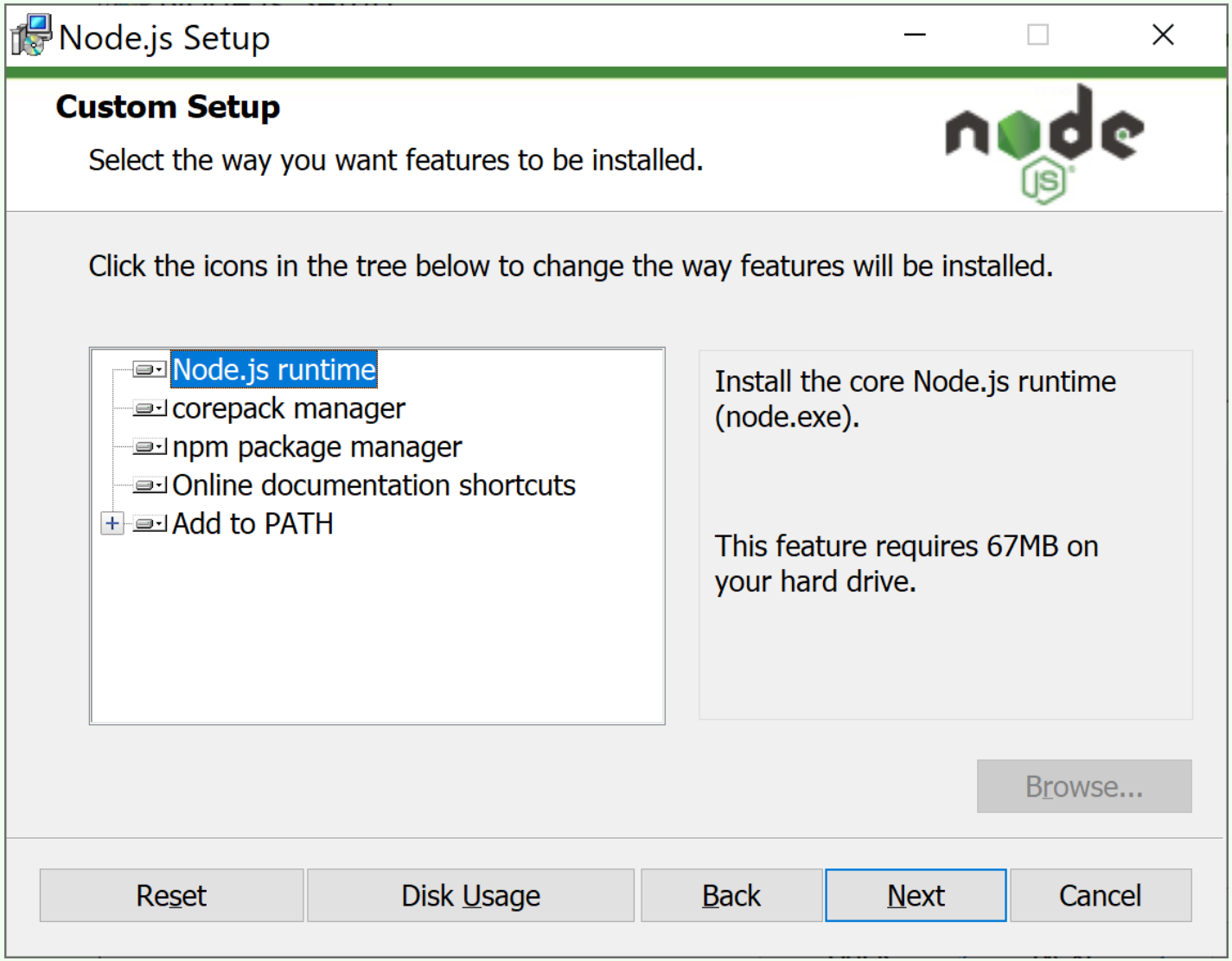
Installing Node.js



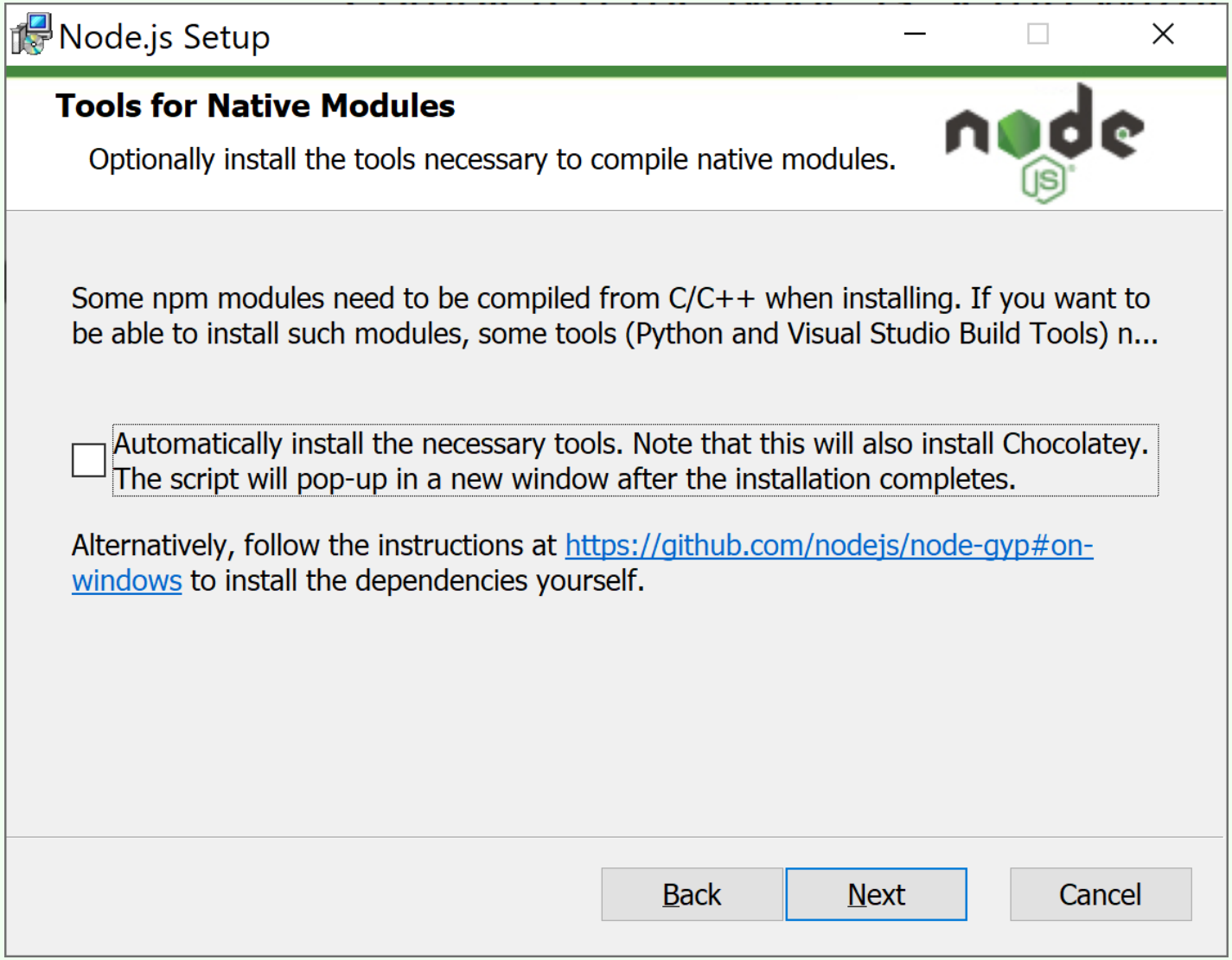
Installing Node.js



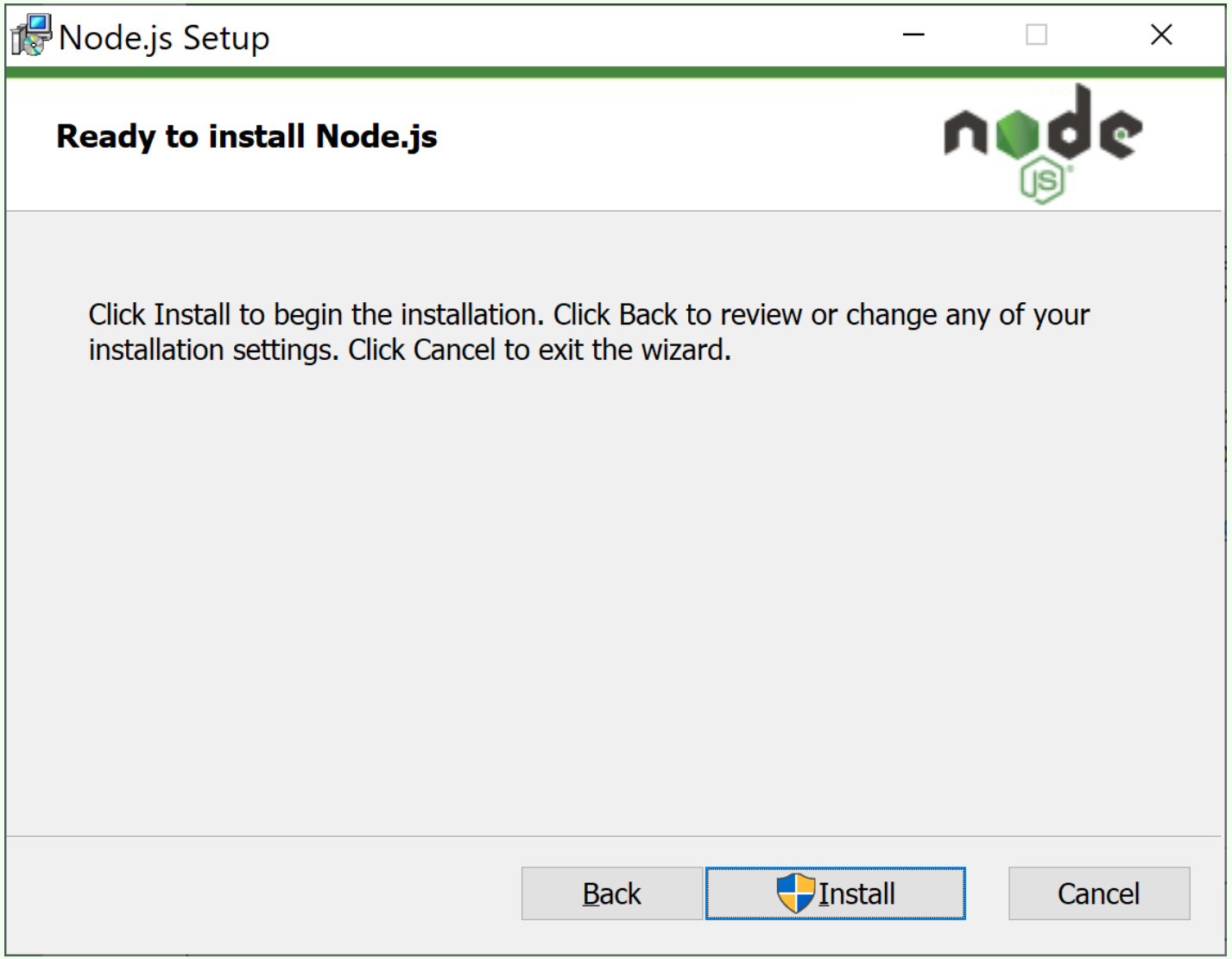
Installing Node.js



Installing Node.js

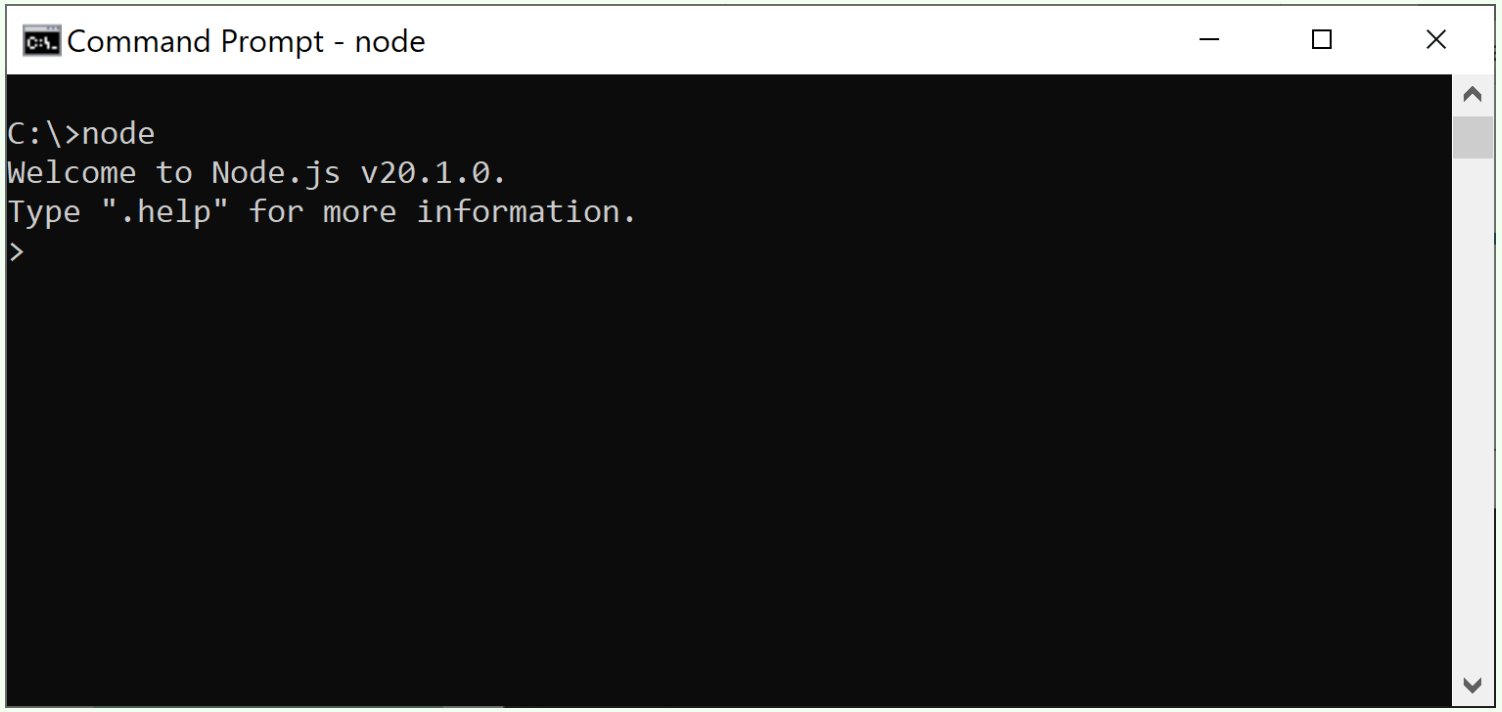


Installing Node.js



Running Node.js

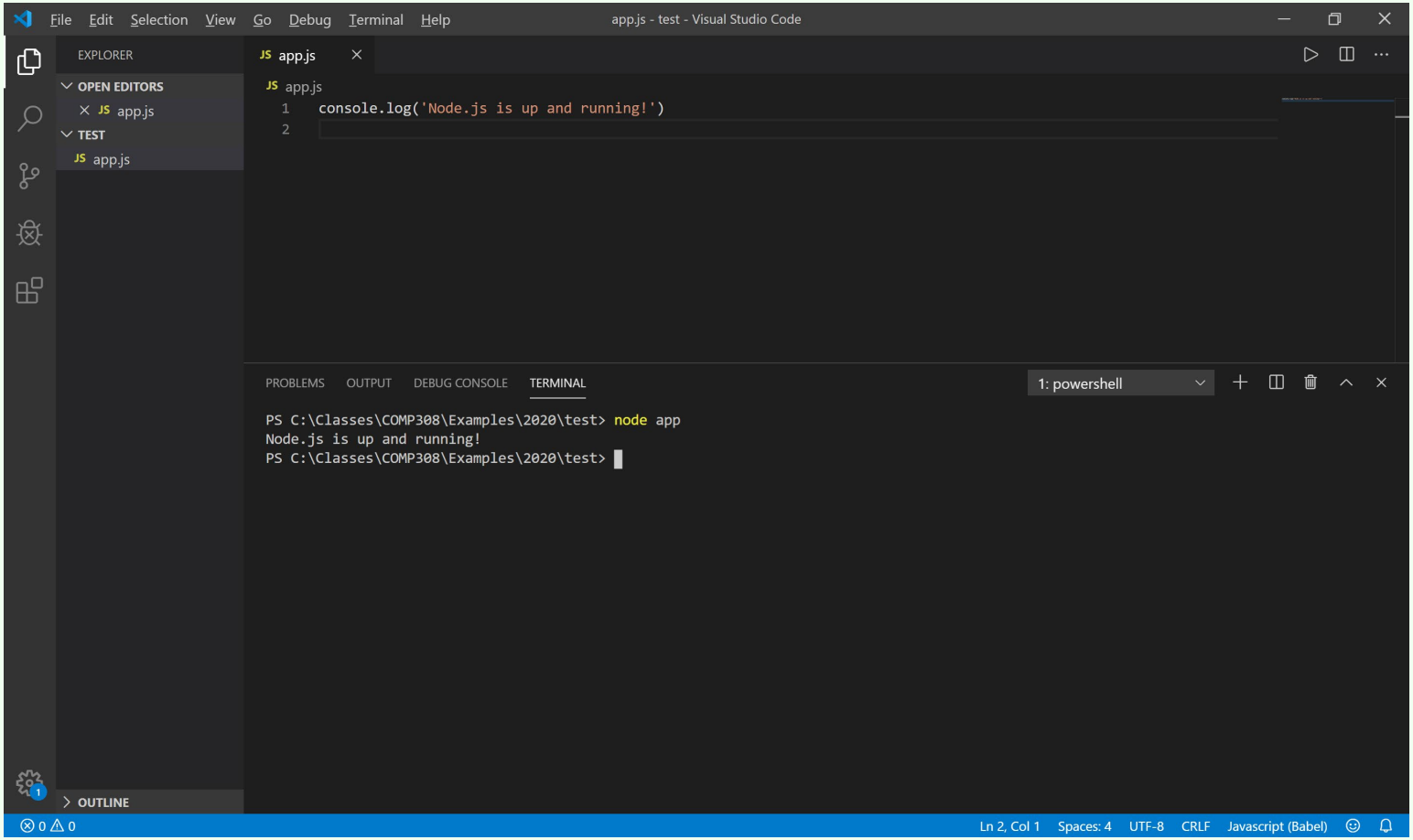
- Just run node from command prompt:



Running Node.js

- Create a JavaScript file app.js with this line:

  console.log('Node.js is up and running!');



Node.js prompt commands

Command Prompt - node

```
C:\nodejs>node
> .help
.break    Sometimes you get stuck, this gets you out
.clear    Alias for .break
.editor   Enter editor mode
.exit     Exit the repl
.help     Print this help message
.load     Load JS from a file into the REPL session
.save     Save all evaluated commands in this REPL session to a file
>
```

Node Package Manager

- NPM is the best way to install, update, and remove Node.js modules
- NPM has the following main features:
  - A registry of packages to browse, download, and install third-party modules
  - A CLI tool to manage local and global packages
- NPM is installed during the Node.js installation process.

Using NPM

- NPM has two installation modes: **local** and **global**.
- The **default local mode** is used more often and installs the third-party packages in a local **node_modules** folder placed inside your application folder.
- It has no effect system-wise, and is used to install the packages your application needs, without polluting your system with unnecessary global files.
- The global mode is used to install packages you want Node.js **to use globally**.
- Will demonstrate the use of NPM by installing Express

Using NPM

- The global mode will usually install the packages in the C:\Users\%USERNAME%\AppData\Roaming\npm\node_modules folder, **making it available to any Node.js application running on the system**.
- To install a package using the npm run the following command:

  npm install <Package Unique Name>

- Installing a module globally is similar to its local counterpart, but you'll have to add the –g flag as follows:

  npm install –g <Package Unique Name>

Using NPM

- For example, to locally install Express, you'll need to **navigate to your application folder** and issue the following command:

  npm install express

- NPM supports a wide range of semantic versioning, so to install a specific version of a package, you can use the **npm** install command as follows:

  npm install <Package Unique Name>@<Package Version>

- For instance, to install the second major version of the Express package, you'll need to issue the following command:

  npm install express@4.x

Using NPM

- **Removing a package using NPM**
- To remove an installed package, you'll have to navigate to your application folder and run the following command:

  npm uninstall < Package Unique Name>
- NPM will then look for the package and try to remove it from the local node_modules folder.
- To remove a global package, you'll need to use the -g flag as follows:

  npm uninstall –g < Package Unique Name>

Using NPM

- **Updating a package using NPM**
- To update a package to its latest version, issue the following command:

  npm update < Package Unique Name>
- NPM will download and install the latest version of this package even if it doesn't exist yet.
- To update a global package, use the following command:

  npm update –g < Package Unique Name>

Managing dependencies using the package.json file

- NPM allows you to use a **configuration file** named **package.json** in the root folder of your application.
- In your package.json file, you'll be able to **define various metadata properties of your application**, including properties such as the **name**, **version**, and **author** of your application.
- This is also where you **define your application dependencies**.

Managing dependencies using the package.json file

- The **package.json** file is basically a JSON file that contains the **different attributes you'll need to describe your application properties**.
- An application using the latest Express and Grunt packages will have a package**. json** file as follows:

```
{
"name" : "MERN",
"version" : "0.0.1",
"dependencies" : {
"express" : "latest",
"grunt" : "latest"
}
}
```

Creating a package.json file

- While you can manually create a package.json file, an easier approach would be to use the **npm init** command.
- To do so, use your command-line tool and issue the following command:

$ npm init

- NPM will ask you a few questions about your application and will automatically **create a new package.json file** for you.
- A sample process should look similar to the following screenshot:

Creating a package.json file

Command Prompt

```
C:\Classes\COMP308\Examples\Node>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (Node) MEAN
Sorry, name can no longer contain capital letters.
name: (Node) mean
version: (1.0.0) 0.0.1
description: my mean app
entry point: (index.js) server.js
test command:
git repository:
keywords: MongoDB, Express, AngularJS, NodeJS
author: ilia nika
license: (ISC) MIT
About to write to C:\Classes\COMP308\Examples\Node\package.json:

{
  "name": "mean",
  "version": "0.0.1",
  "description": "my mean app",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "MongoDB",
    "Express",
    "AngularJS",
    "NodeJS"
  ],
  "author": "ilia nika",
  "license": "MIT"
}


Is this ok? (yes) yes
```

Installing the package.json dependencies

- After creating your **package.json** file, you'll be able to install your application dependencies by navigating to your application's root folder and using the **npm** install command as follows:

npm install

- NPM will automatically detect your package.json file and will install all your application dependencies, placing them under a local **node_modules** folder.
- An alternative and sometimes better approach to install your dependencies is to use the following **npm** update command:

npm update

Updating the package.json file

- Another robust feature of the **npm** install command is the ability to **install a new package and save the package information as a dependency in your package.json file**.
- This can be accomplished using the **--save** optional flag when installing a specific package.
- For example, to install the latest version of Express and save it as a dependency, you can issue the following command:

npm install express --save

Creating Node.js Apps in VS Code

- Write a simple Node.js app in VS Code.
- See the video on eCentennial.
- Test Week 1 Examples
- Complete the Lab Session 1
- Complete Interactive Exercise 1

References

- Reference textbook
- https://nodejs.org/en/
- http://expressjs.com/
- https://angular.io/
- https://www.mongodb.org/
- https://code.visualstudio.com/