

COURSE OUTCOMES

On completion of this course, the students shall be able to

COs	Statements	Bloom's Level
CO3	Use JavaScript for designing dynamic web pages and client side validation.	L3
CO4	Make Docker Account, deploy an application on Docker and install Virtual Docker on local machine and access files from Docker account	L4

SYLLABUS:

Unit-2	Server Side NodeJS	CONTACT HOURS: 4 hours
Server side NodeJS:-	Key features of NodeJS, Installation and Configuration, NodeJS Command Line, Sample Project using Node Express command prompt, Nodeclipse plugin, Sample Project using Nodeclipse, Performing CRUD Operations, Key features of MongoDB, Connection Pooling using NodeJS Mongo driver, Docker architecture, Virtual machines versus containers, about containers, Docker: A shipping container for code, Benefits of using containers, Docker basic concepts, Docker shared and layered file systems technology	
Unit-2	Contact Hours: 4 HOURS (Practical's)	
2.1	To run <i>JavaScript</i> application using Docker and manage the Docker volume.	
2.2	To Setup i. Container to WWW Communication, ii. Container to Local Host Machine Communication, iii. Container to Container Communication, iv. Creating a Container & Communicating to the Web (WWW) v. Container to Host Communication Work, vi. Container to Container Communication using Docker Desktop.	
2.3	To Understand Kubernetes architecture, building blocks and container orchestration.	

SUGGESTIVE READINGS

Text Books

T1. Web Programming with HTML5, CSS and Java Script, John Dean, First, Jones And Bartlett Learning, 2019

T2. Building Microservices, Sam Newman, First, O'Reilly Media, Inc., 2015

T3. Microservices Architecture, Ajay Kumar, First, Shroff / O'Reilly, 2014

Reference Books

R1. Front-End Web Development with Modern HTML, CSS, and JavaScript, Laurence Lars Svekis, First, Packt, 2022

R2. Microservice Architecture, Irakli Nadareishvili, Ronnie Mitra, Matt McLarty,, First, O'Reilly Media, Inc, 2016

Video Links

1. [Node.js Tutorial for Beginners](#)
2. [Create a simple Node.js Docker Container from scratch](#)
3. [MongoDB inside Docker Container](#)

4. [Connect Node to MongoDB with Docker and Docker Compose](#)
5. [What Is Docker? | What Is Docker And How It Works? | Docker Tutorial For Beginners | Simplilearn](#)
6. [Build and Dockerize a CRUD RESTful API using NodeJS and MongoDB from Scratch](#)
7. [Docker Installation In Ubuntu | How To Install Docker In Ubuntu? | Docker Installation | Simplilearn](#)

What is Node.js

Summary: in this tutorial, you will learn about what Node.js is and why and when you should use Node.js.

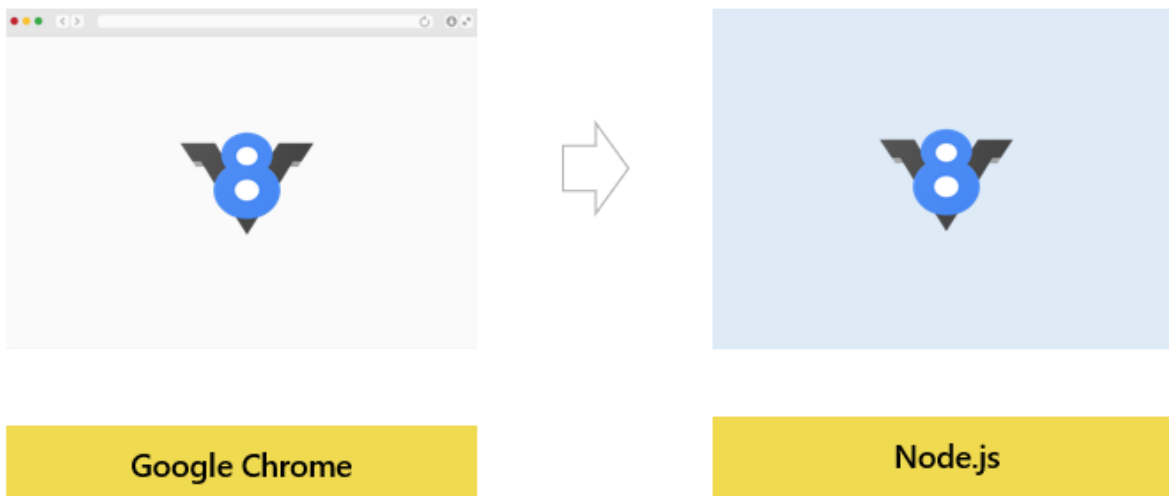
Introduction to Node.js

Node.js is an open-source cross-platform runtime environment that allows you to use JavaScript to develop server-side applications.

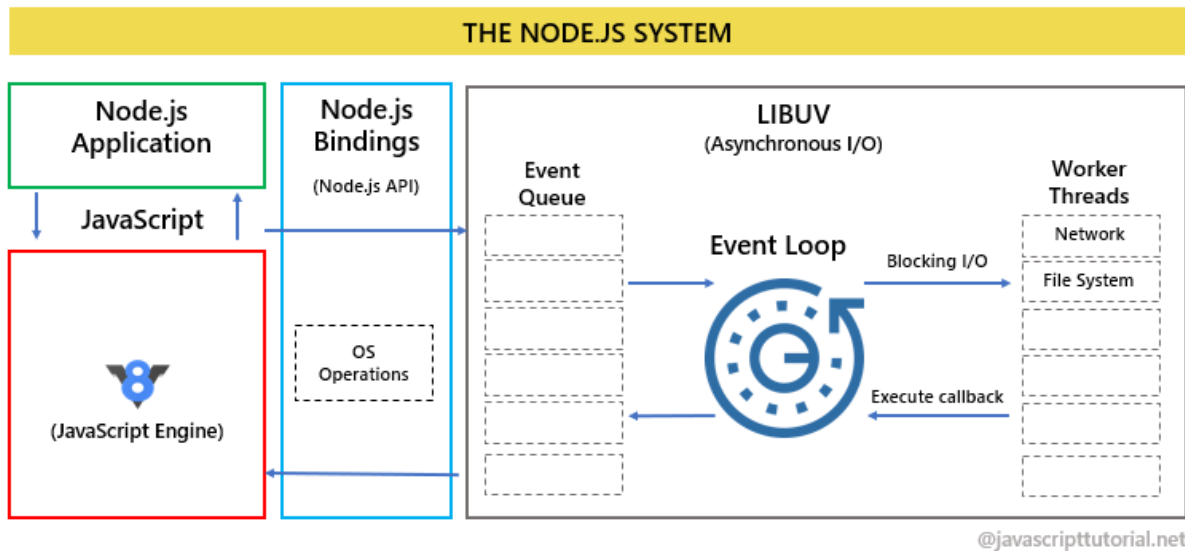
Every web browser has a JavaScript engine that compiles JavaScript code to machine code. For example, [Firefox uses SpiderMonkey](#), and [Google Chrome uses V8](#).

Because browsers use different JavaScript engines, sometimes, you will see that JavaScript behaves differently between the browsers.

In 2009, [Ryan Dahl](#), the creator of Node.js, took the V8 engine and embedded it in an application that could execute JavaScript on the server.



The following picture illustrates the Node.js system:



Node.js uses the single-threaded, non-blocking, and event-driven execution model, which is similar to the execution model of JavaScript in the web browser.

Node.js is single-threaded

Node.js is single-threaded, meaning each process runs only one thread of execution.

The single-threaded execution model allows Node.js to handle more concurrent requests easily via the event loop. As a result, Node.js applications typically consume less memory compared to others.

If you are unfamiliar with the [event loop](#), check out this tutorial. It's important to note that the event loop in Node.js works similarly to the event loop in web browsers.

Node.js uses Non-blocking I/O

I/O stands for input/output, which can involve disk access, a network request, or a database connection. I/O requests are expensive and slow, leading to the potential blocking of other operations.

Node.js addresses blocking I/O issues by utilizing non-blocking I/O operations.

In the non-blocking model, you can initiate a request while doing other tasks. Once the request is completed, a callback function is executed to handle the result.

Node.js is event-driven

Node.js uses event-driven architecture, meaning that Node.js waits for events to happen and responds to them when they occur, instead of following a linear, top-to-bottom execution model.

The following describes how Node.js event-driven architecture works:

1. An event, such as a client sending a request to a server, triggers an event handler function.
2. Node.js then adds the event handler (a callback function) to an Event Queue.
3. The Event Loop, which is a continuous loop running in Node.js, checks the Event Queue and processes the events one by one. It executes the corresponding event handler for each event.
4. While the Event Loop is processing these events, Node.js can continue handling other tasks, such as listening for new events.

Why node.js

Node.js is good for prototyping and agile development, allowing you to build fast and highly scalable applications.

Node.js has a large ecosystem of open-source libraries so that you can reuse existing ones and spend more time focusing on the business logic.

If you already use JavaScript for front-end development, you can leverage your existing skills for server-side development.

Using the same JavaScript for both client-side and server-side development makes your codebase cleaner and more consistent.

Node.js real-world use cases

Node.js has a wide range of real-world applications:

- **Scalable network applications:** Node.js is used by big companies like LinkedIn, Uber, and Netflix to develop scalable applications.
- **Real-time applications** – Since Node.js uses an event-driven, non-blocking I/O model, it is ideal for real-time applications including chat and online gaming.
- **API and Microservices** – Node.js is commonly used to develop Restful APIs and microservices.
- **IoT devices:** Node.js powers various Internet of Things (IoT) applications.

Server-Side Node.js

Node.js is a server-side platform built on Google Chrome's JavaScript Engine (V8 Engine). Node.js was developed by Ryan Dahl in 2009 and its latest version is v0.10.36. The definition of Node.js as supplied by its official documentation is as follows –

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

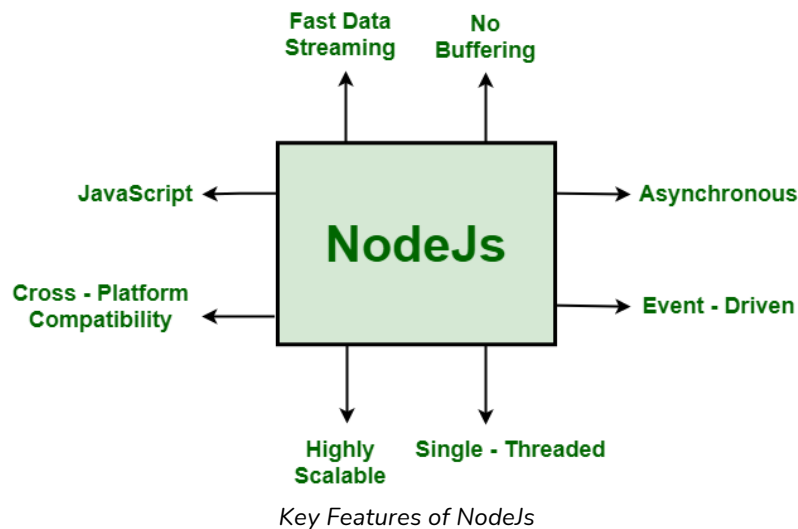
Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.

Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent.

Node.js = Runtime Environment + JavaScript Library

Features of Node.js

Following are some of the important features that make Node.js the first choice of software architects.



- **Asynchronous and Event-Driven:** The Node.js library's APIs are all asynchronous (non-blocking) in nature. A server built with Node.JS never waits for data from an API. After accessing an API, the server moves on to the next one. In order to receive and track responses of previous API requests, it uses a notification mechanism called Events.
- **Single-Threaded:** Node.js employs a single-threaded architecture with event looping, making it very scalable. In contrast to typical servers, which create limited threads to process requests, the event mechanism allows the node.js server to reply in a non-blocking manner and makes it more scalable. When compared to traditional servers like Apache HTTP Server, Node.js uses a single-threaded program that can handle a considerably larger number of requests.
- **Scalable:** NodeJs addresses one of the most pressing concerns in software development: scalability. Nowadays, most organizations demand scalable software. NodeJs can also handle concurrent requests efficiently. It has a cluster module that manages load balancing for all CPU cores that are active. The capability of NodeJs to partition applications horizontally is its most appealing feature. It achieves this through the use of child processes. This allows the organizations to provide distinct app versions to different target audiences, allowing them to cater to client preferences for customization.
- **Quick execution of code:** Node.js makes use of the V8 JavaScript Runtime motor, which is also used by Google Chrome. Hub provides a wrapper for the JavaScript motor, which makes the runtime motor faster. As a result, the preparation of requests inside Node.js becomes faster as well.
- **Cross-platform compatibility:** NodeJS may be used on a variety of systems, including Windows, Unix, Linux, Mac OS X, and mobile devices. It can be paired with the appropriate package to generate a self-sufficient executable.
- **Uses JavaScript:** JavaScript is used by the Node.js library, which is another important aspect of Node.js from the engineer's perspective. Most of the engineers are already familiar with JavaScript. As a result, a designer who is familiar with JavaScript will find that working with Node.js is much easier.
- **Fast data streaming:** When data is transmitted in multiple streams, processing them takes a long time. Node.js processes data at a very fast rate. It processes and uploads a file simultaneously, thereby saving a lot of time. As a result, NodeJs improves the overall speed of data and video streaming.
- **No Buffering:** In a Node.js application, data is never buffered.

Applications of Node.js

The following are some of the areas where Node.js is proving to be an effective technology partner-

1. Single Page Applications
2. Data-Intensive Real-time Applications
3. I/O bound Applications
4. JSON APIs based Applications
5. Data Streaming Applications

Installation of Node JS on Windows

Node.js can be installed in multiple ways on a computer. The approach used by you depends on the existing Node.js development environment in the system. There are different package installers for different environments. You can install Node.js by grabbing a copy of the source code and compiling the application. Another way of installing Node.js is by cloning the Node.js GIT repository in all three environments and then installing it on the system.

How to Install Node.js on Windows

The first step in using Node.js is the installation of the Node.js libraries on the client system. Below are the steps to download and install Node.js in Windows:

Step 1) Download Node.js Installer for Windows

Go to the site <https://nodejs.org/en/download/> and download the necessary binary files.

In our example, we are going to Download Node.js on Windows with the 32-bit setup files.

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

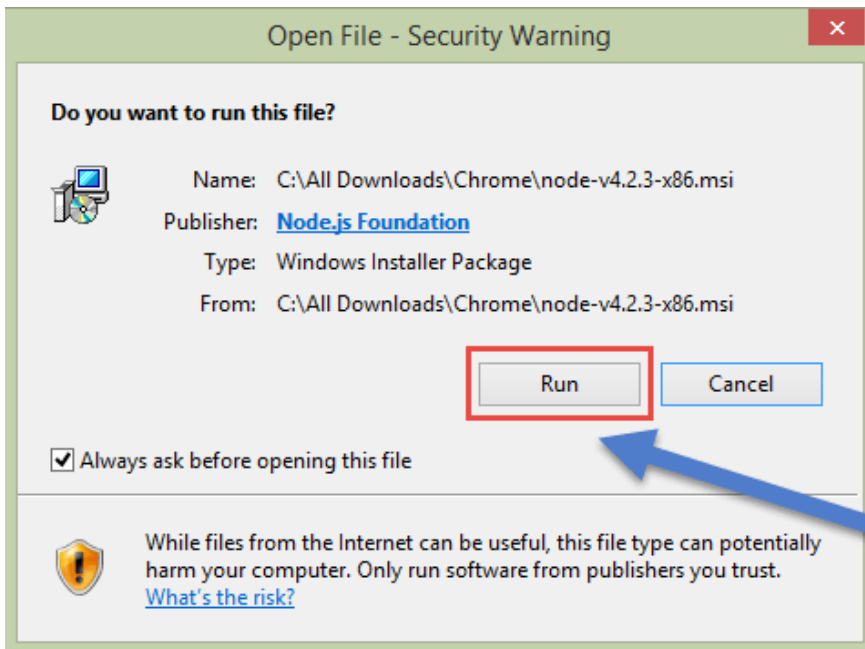
	Windows Installer <small>node-v4.2.3-x86.msi</small>	Macintosh Installer <small>node-v4.2.3.pkg</small>	Source Code <small>node-v4.2.3.tar.gz</small>
Windows Installer (.msi)	32-bit	64-bit	
Windows Binary (.exe)	32-bit	64-bit	
Mac OS X Installer (.pkg)		64-bit	
Mac OS X Binaries (.tar.gz)		64-bit	

Download the 32-bit installer

Step 2) Run the installation

Double click on the downloaded .msi file to start the installation.

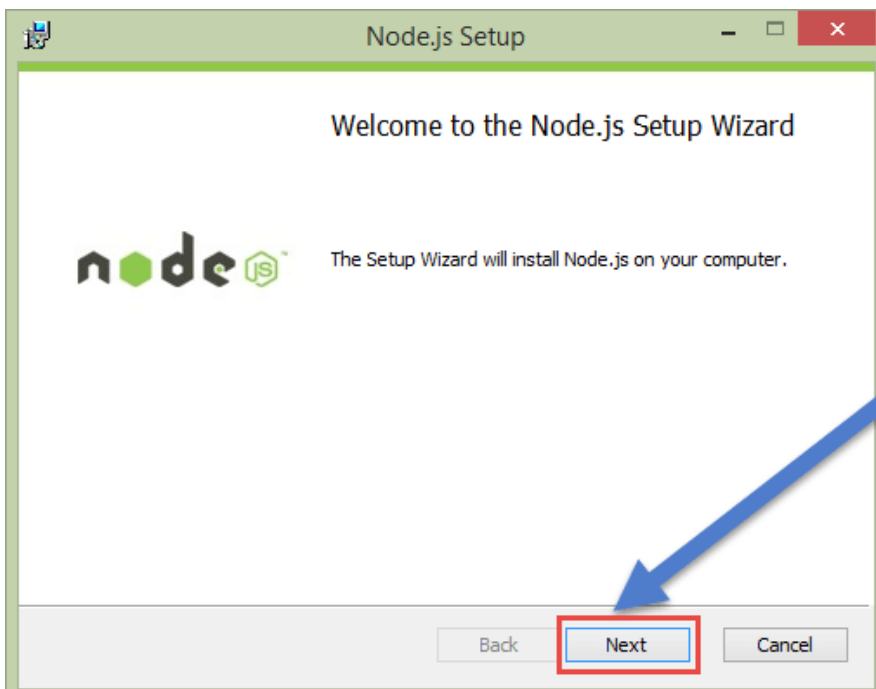
Click the Run button on the first screen to begin the installation.



Click the
Run button

Step 3) Continue with the installation steps

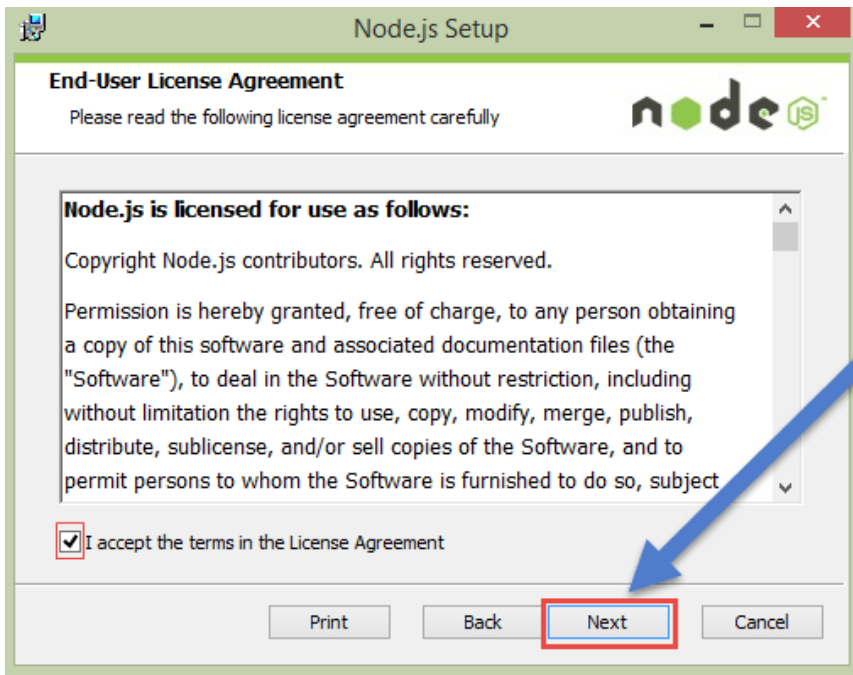
In the next screen, click the “Next” button to continue with the installation



Click the
Next button

Step 4) Accept the terms and conditions

In the next screen, Accept the license agreement and click on the Next button.

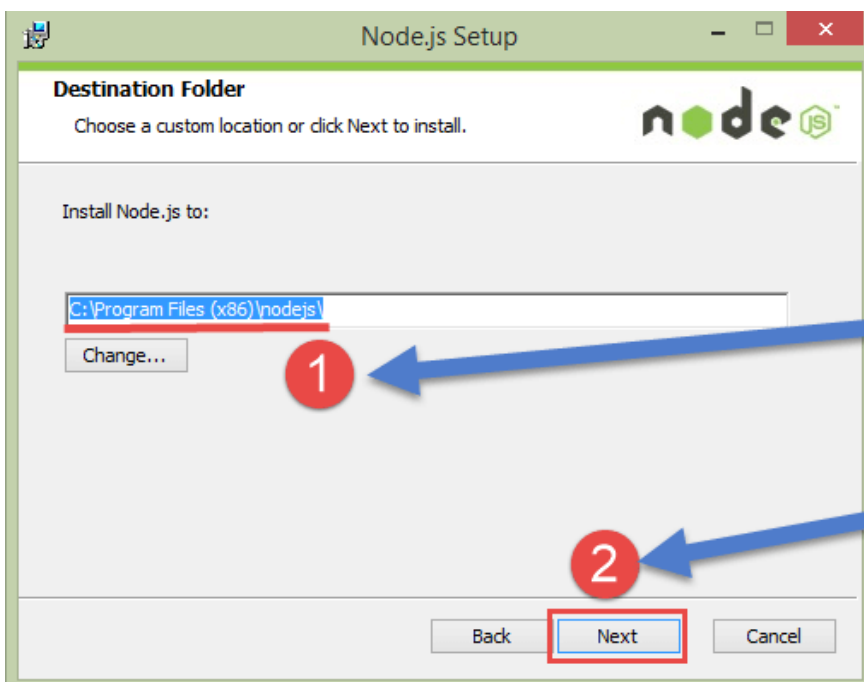


Accept the
license
agreement
and click
the Next

Step 5) Set up the path

In the next screen, choose the location where Node.js needs to be installed and then click on the Next button.

1. First, enter the file location for the installation of Node.js. This is where the files for Node.js will be stored after the installation.
2. Click on the Next button to proceed ahead with the installation.

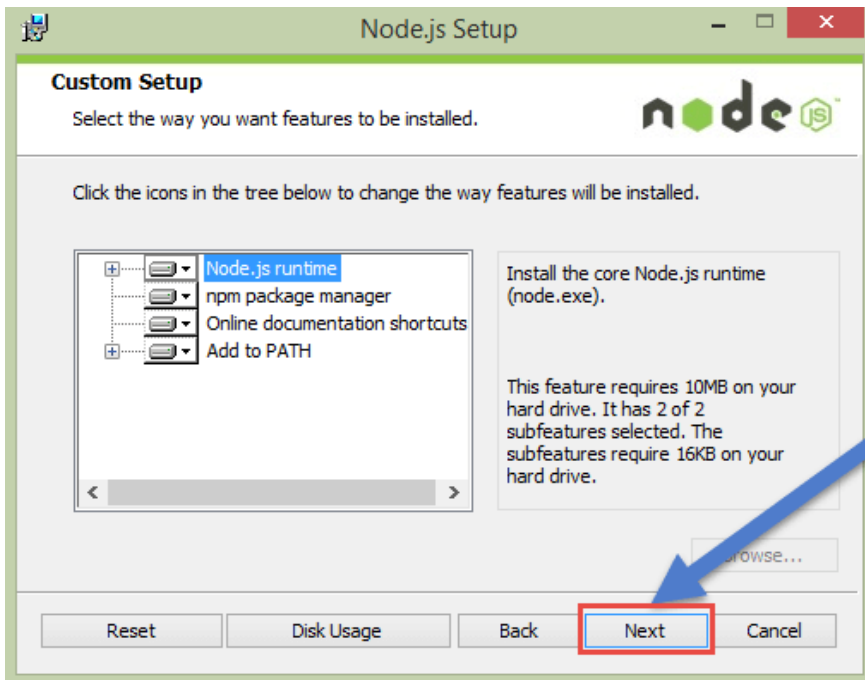


Enter the
file location
for the
installation

Click on the
Next button

Step 6) Select the default components to be installed

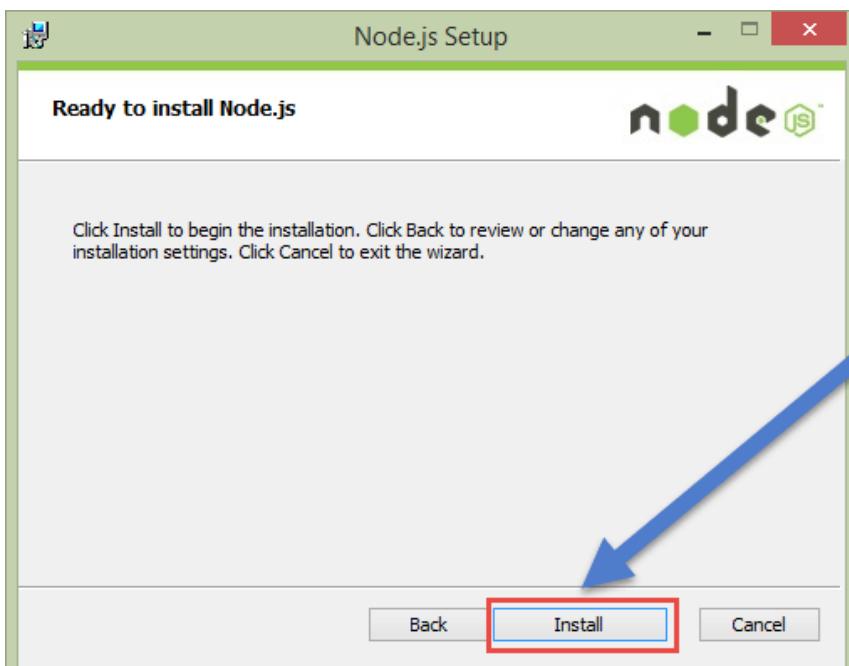
Accept the default components and click on the Next button.



Accept the default components and click on Next

Step 7) Start the installation

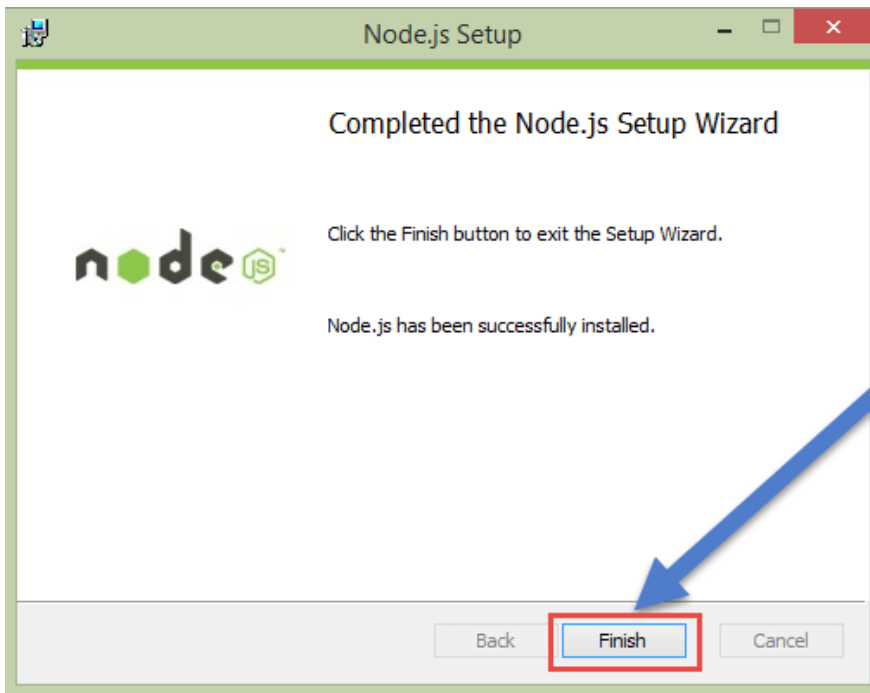
In the next screen, click the Install button to start installing Node.js on Windows.



Click the Next button to begin the installation

Step 8) Complete the installation

Click the Finish button to complete the installation.



Click the
Finish
button to
complete
the
installation

Windows is now recommending that developers use Node.js with [WSL2](https://docs.microsoft.com/en-us/windows/wsl/) (the Windows subsystem for Linux)

How to Install NPM on Windows 10/8/7

The other way to install Node.js on any client machine is to use a “package manager.”

On Windows, the NPM (Node Package Manager) download is known as Chocolatey. It was designed to be a decentralized framework for quickly installing applications and tools that you need.

For installing NPM on Windows via Chocolatey, the following steps need to be performed.

Step 1) Installing Chocolatey – The Chocolatey website (<https://chocolatey.org/>) has very clear instructions on how this framework needs to be installed.

- The first step is to run the below command in the command prompt windows. This command is taken from the Chocolatey web site and is the standard command for installing Node.js via Chocolatey.
- The below command is a PowerShell command which calls the remote PowerShell script on the Chocolatey website. This command needs to be run in a PowerShell command window.
- This PowerShell script does all the necessary work of downloading the required components and installing them accordingly.

```
@powershell -NoProfile -ExecutionPolicy Bypass -Command "iex ((new-object  
wet.webclient).DownloadString('https://chocolatey.org/install.ps1'))" && SET  
PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin
```

```
WARNING: You can safely ignore errors related to missing log files when
upgrading from a version of Chocolatey less than 0.9.9.
'Batch file could not be found' is also safe to ignore.
'The system cannot find the file specified' - also safe.
chocolatey.nupkg file not installed in lib.
Attempting to locate it from bootstrapper.
PATH environment variable does not have D:\ProgramData\chocolatey\bin in it.
ing...
Chocolatey (choco.exe) is now ready.
You can call choco from anywhere, command line or powershell by typing choco.
Run choco /? for a list of functions.
You may need to shut down and restart powershell and/or consoles
first prior to using choco.
Ensuring chocolatey commands are on the path
Ensuring chocolatey.nupkg is in the lib folder
```

D:\windows\System32>

You will see the above messages at the end of the installation

Step 2) The next step is to install Node.js to your local machine using the Chocolatey, package manager. This can be done by running the below command in the command prompt.

cinst nodejs install

```
nodejs.install v5.2.0
The package nodejs.install wants to run 'chocolateyInstall.ps1'.
Note: If you don't run this script, the installation will fail.
Note: To confirm automatically next time, use '-y' or consider setting
'allowGlobalConfirmation'. Run 'choco feature -h' for more details.
Do you want to run the script?
1) yes
2) no
3) print
yes
Downloading nodejs.install 64 bit
from 'https://nodejs.org/dist/v5.2.0/node-v5.2.0-x64.msi'
Installing nodejs.install...
nodejs.install has been installed.
The install of nodejs.install was successful.
```

Notify that the installation was successful

If the installation is successful, you will get the message of the successful installation of Node.js.

Note: If you get an error like

“C:\ProgramData\chocolatey\lib\libreoffice\tools\chocolateyInstall.ps1” Then manually create the folder in the path

Step 5: Updating the Local npm version

You can run the following command, to quickly update the npm
npm install npm --global // Updates the ‘CLI’ client

Installation of Node JS on Linux

Node JS is a JavaScript runtime built on Chrome's V8 engine, and on Ubuntu Linux, it can be installed via the NodeSource repository for the latest version. [Hostinger's](#) VPS provides a robust environment for Node.js apps, offering control and flexibility. They have templates for easy Node.js setup on Ubuntu 22.04 with Node.js, along with an OpenLiteSpeed server. Additionally, a CloudPanel template simplifies Node.js app creation and management with a user-friendly interface, suitable for those with little VPS experience.

It has 4 active plans tailored to meet different requirements:

KVM1, KVM2, KVM4, KVM8 ranging from ₹499/mo to ₹1829/mo.

Its KVM2 plan is cheapest and most popular amongst those running small applications.

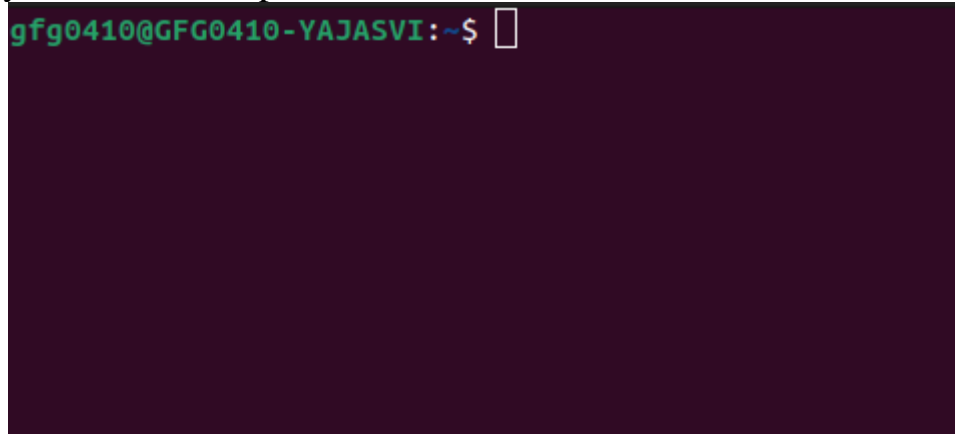
As we discussed, **Node.js, powered by Chrome's V8 engine**, is crucial for **server-side JavaScript applications**. As you explore setting up Node.js on Ubuntu via the NodeSource repository, you're stepping into the world of backend development. Once you've mastered installing Node.js and understanding its ecosystem on Ubuntu, you might be eager to expand your skills further into full-stack web development.

We offer/suggest a comprehensive [Full Stack Development with React & Node JS](#) designed to build upon your Node.js knowledge. This course not only covers advanced frontend design principles but also dives deep into backend technologies like Node.js, databases, and server management. There are two methods to officially install **Node.js on Ubuntu** which are as follows:

Using Ubuntu's official repository

Node.js is available in Ubuntu's repository and you can easily install it using a few commands. Follow the steps below to install Node.js on your Ubuntu operating system.

Step 1: Open your terminal or press Ctrl + Alt + T.



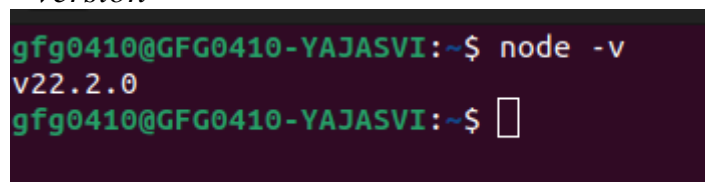
Terminal

Step 2: To install node.js use the following command:

sudo apt install nodejs

Step 3: Once installed, verify it by checking the installed version using the following command:

node -v or node --version



node version

Note: It is recommended to install Node Package Manager(NPM) with Node.js. NPM is an open source library of Node.js packages. To install NPM, use the following commands:

```
sudo apt install npm
```

```
npm -v or npm --version
```

Node and NPM will be successfully installed on your Ubuntu machine.

Using NodeSource repository

The latest version of Node JS can be installed from [NodeSource repository](#)

Follow the steps below to install the Node.js on your Ubuntu.

Step 1: Open your terminal or press Ctrl + Alt + T and use the following commands:

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

Step 2: Install Python software libraries using the following command:

```
sudo apt-get install python-software-properties
```

Step 3: Add Node.js PPA to the system.

```
curl -sL https://deb.nodesource.com/setup_220.x | sudo -E bash -
```

Note:

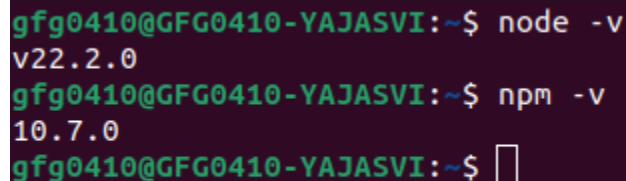
Here, we are installing node.js version 22, if you want to install version 23, you can replace setup_22.x with setup_23.x.

Step 4: To Install Node.js and NPM to your Ubuntu machine, use the command given below:

```
sudo apt-get install nodejs
```

Step 5: Once installed, verify it by checking the installed version using the following command:

```
node -v or node --version npm -v or npm --version
```



```
gfg0410@GFG0410-YAJASVI:~$ node -v
v22.2.0
gfg0410@GFG0410-YAJASVI:~$ npm -v
10.7.0
gfg0410@GFG0410-YAJASVI:~$
```

npm version

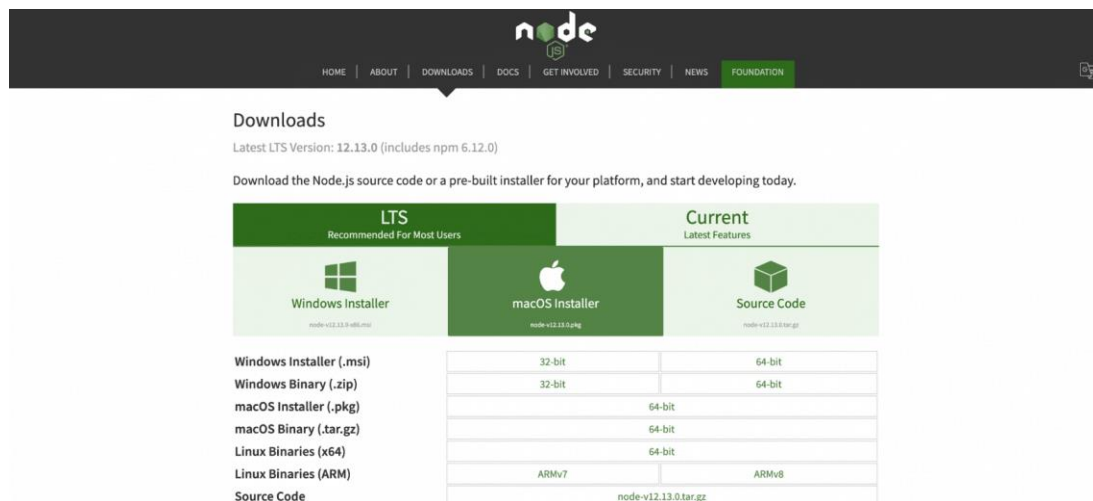
Finally, you have successfully installed Node JS and NPM on your Ubuntu machine.

How to Install Node.js and NPM on Mac?

Now, if you are using a macOS, let's understand the process of Node.js and NPM installation.

Install Node Using .pkg Installer

Well, it's a similar process as Windows. Here, Node provides a **.pkg** installer for Mac. Besides, we can also [download from its official website](#)



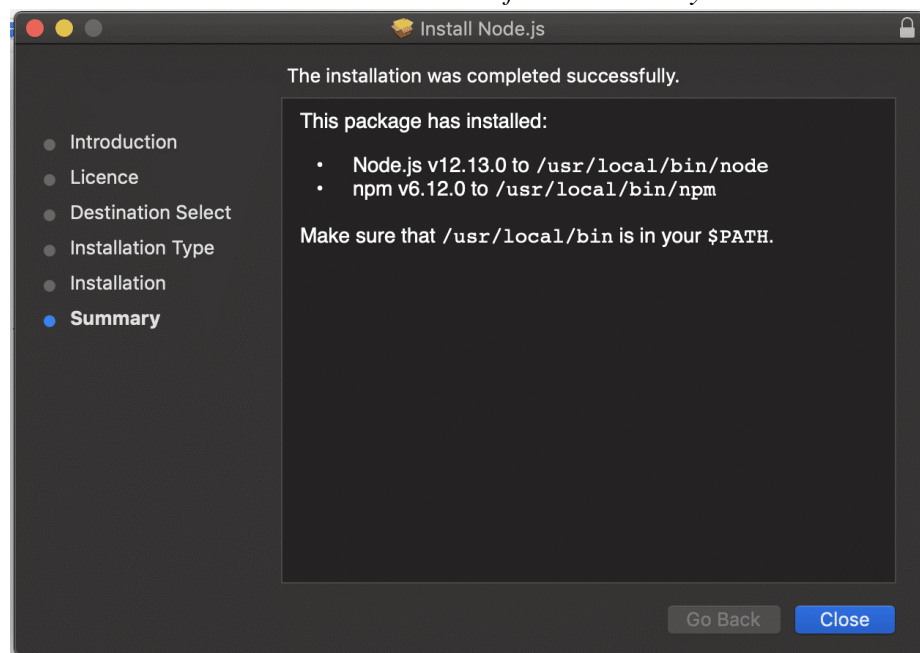
Step 1: Download the .pkg Installer

Click on the “**macOS Installer**” option to download the .pkg installer. Make sure you download it to your desired location.

Step 2: Run Node.js Installer

Now, your installer is ready to run. However, it will not take your much time. So, let’s explain in detail here.

Introduction > Continue License > Select Continue > Agree Installation Type > Install > Authenticate with your Mac to allow the Installation > Install Software Summary > Close



Step 3: Verify Node.js Installation

To verify whether you have properly installed Node.js in your macOS, run the following command in your terminal:

node -v node -v // The command we ran - tests the version of Node.js that's currently installed v12.13.0 // The version of Node.js that's installed. It can be some other version.

Step 4: Update Your NPM Version

Node.js doesn’t automatically update the version of npm.

Write a given command and your npm version will be updated.

\$ sudo npm install npm --global // Update thenpmCLI client

Tadda! Node.js is there on your Mac system. Interesting, right? Let’s explore it further.

Set NODE HOME in Environment Variable

Now, run the given command in your terminal for Mac or Linux systems. This ensures that the NODE path is set in the PATH variable.

export PATH=/usr/local/git/bin:/usr/local/bin:\$PATH

Here, `*/usr/local/bin*` is the default path where Node will be installed.

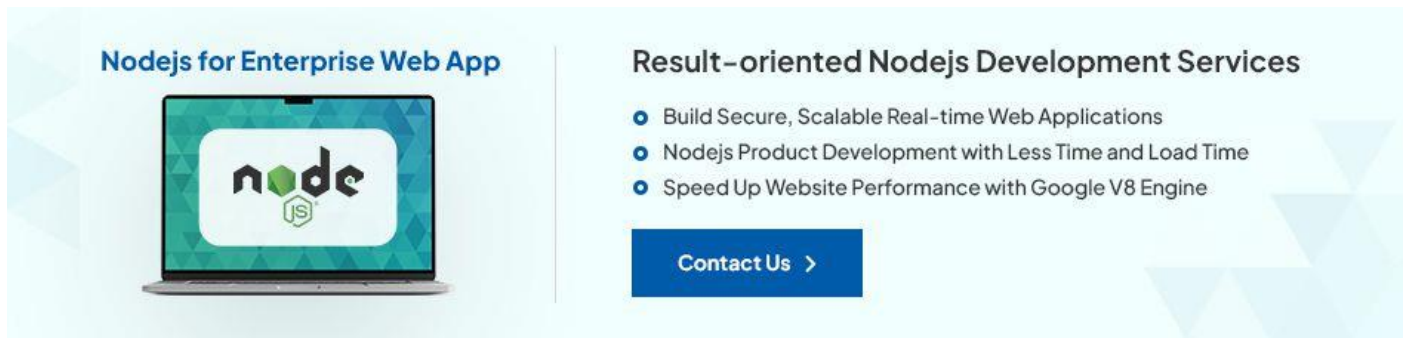
Actually, it’s advisable to write this command in **.bash_profile** or **.zshrc**. However, it depends on which shell you are using so that these path export stay on your system and nobody can recognize them from all the terminal instances.

Let's run the given command to add the details in **.bash_profile** file permanently.

```
echo 'export PATH=/usr/local/bin:$PATH' >> ~/.bash_profile
```

Run the given command to make it available to all users on the system.

```
source ~/.bashrc
```



How to Install Node Using “Homebrew”?

Homebrew is an open-source and free software package management system, simplifying the installation process on macOS and Linux. It installs packages to their directory and their files into **/usr/local**.

Here, we can use **Homebrew** to install Node.js. After this installation, let's move to the next step.

Step 1: Install Node.js and NPM

It's very easy to install Node.js and NPM using Homebrew. It lets you download, unpack and install Node and NPM on your device.

Run the given command in your terminal.

```
brew install node
```

Some files will be downloaded. So, let's install them.

Step 2: Verify Node.js and NPM Installation

Same as the windows system, you just have to type the below-given command and run it in your terminal.

For Node: `node -v`

For NPM: `npm -v`

Homebrew will update the version of Node and NPM you have installed. Make sure that your Homebrew has the latest version of the Node package.

If not, then run the given command to update the Homebrew.

```
brew update
```

Run given the command to update version of node:

```
brew upgrade node
```

It will update Node and NPM to the latest version.

How to Install Node Using NVM – Node Version Manager?

Being a [professional NodeJs developer](#), I have got many requests asking to install multiple versions of Node.js on their devices. Yes, it is possible with NVM – Node Version Manager.

Let's follow the given steps and install Node.

Step 1: Download NVM with Install Script

Now, you have two options to [install NVM](#).

With cURL:

```
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.0/install.sh | bash // Installation using cURL
```

With wget:

```
wget -qO- https://raw.githubusercontent.com/creationix/nvm/v0.33.0/install.sh | bash // Installation using Wget
```

Now NVM will be downloaded and installed.

Step 2: Verify NVM Installation

You can verify NVM with the given command:

```
nvm --version
```

The command will show something like (not necessarily exact):

```
*0.33.0 # The current version of nvm - yours may differ!*
```

Actually, it's advisable to write this command in **.bash_profile** or **.zshrc**. However, it depends on which shell you are using so that these path export stay on your system and nobody can recognize them from all the terminal instances. Let's run the given command to add the details in the **.bash_profile** file permanently.

```
echo 'export PATH=/usr/local/bin:$PATH' >> ~/.bash\_profile
```

Run the given command to make it available to all users on the system.

```
source ~/.bashrc
```

Step 3: Install the Latest Node Version Using NVM

Run the given command to install the latest version of Node.js.

```
nvm install node
```

Moreover, you can select all the available Node versions if you want to install a specific version.

Let's run the given command:

```
nvm ls-remote // This will list down all the available Node version
```

```
nvm install v12.15.0 // This will install Node 12.15.0
```

Now, let's check the installation of different [Node framework](#) versions and which specific versions are used with the given command.

```
nvm ls // This will list all the Node versions installed on your machine.
```

```
nvm use v12.15.0 // This will set the Node version to 12.15.0
```

Also, verify the existing version with the given command:

```
node --version
```

It will show "12.15.0", as it is used in the above commands.

Done!

Now you are all set with your Node installation and can further develop the Node application.

Running your first Hello World application in Node.js

Once you have Node.js download and installed on your computer, let's try to display "Hello World" in a web browser.

Create file Node.js with file name firstprogram.js

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('Hello World!');
}).listen(8080);
```

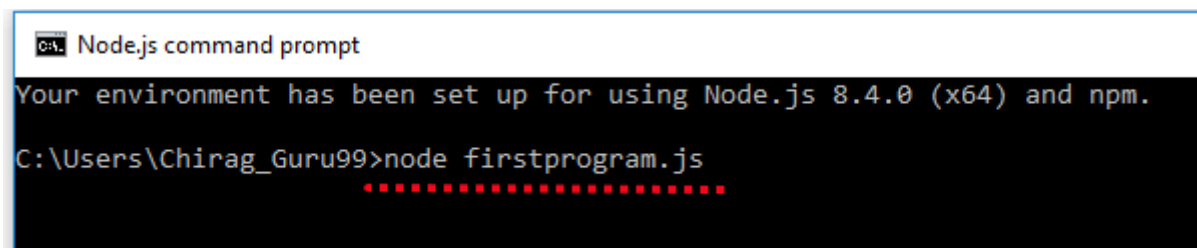
Code Explanation:

1. The basic functionality of the "require" function is that it reads a [JavaScript](#) file, executes the file, and then proceeds to return an object. Using this object, one can then use the various functionalities available in the module called by the require function. So in our case, since we want to use the functionality of HTTP and we are using the require(http) command.
2. In this 2nd line of code, we are creating a server application which is based on a simple function. This function is called, whenever a request is made to our server application.

3. When a request is received, we are asking our function to return a “Hello World” response to the client. The `writeHead` function is used to send header data to the client, and while the `end` function will close the connection to the client.
4. We are then using the `server.listen` function to make our server application listen to client requests on port no 8080. You can specify any available port over here.

Executing the code

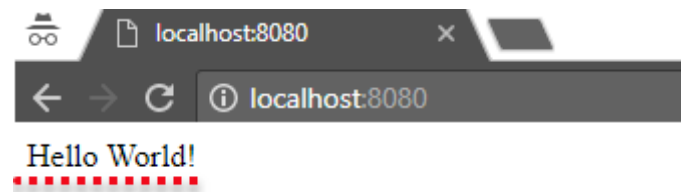
1. Save the file on your computer: `C:\Users\Your Name\ firstprogram.js`
2. In the command prompt, navigate to the folder where the file is stored. Enter the command `Node firstprogram.js`



```
C:\> Node.js command prompt
Your environment has been set up for using Node.js 8.4.0 (x64) and npm.
C:\Users\Chirag_Guru99>node firstprogram.js
.....
```

1. Now, your computer works as a server! If anyone tries to access your computer on port 8080, they will get a “Hello World!” message in return!
2. Start your internet browser, and type in the address: `http://localhost:8080`

OutPut



Verify installation: Executing a File

Create a js file named **main.js** on your machine (Windows or Linux) having the following code.

```
/* Hello, World! program in node.js */
console.log("Hello, World!")
```

Now execute `main.js` file using Node.js interpreter to see the result –

```
$ node main.js
```

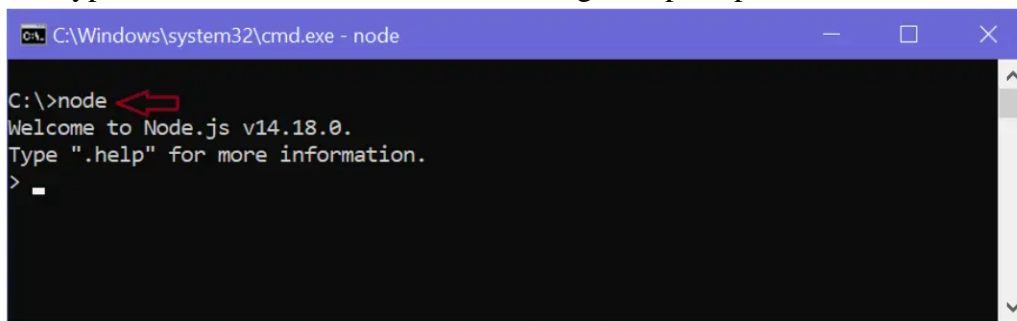
If everything is fine with your installation, this should produce the following result –

```
Hello, World!
```

Node.js Console/REPL

Node.js comes with virtual environment called REPL (aka Node shell). REPL stands for Read-Eval-Print-Loop. It is a quick and easy way to test simple Node.js/JavaScript code.

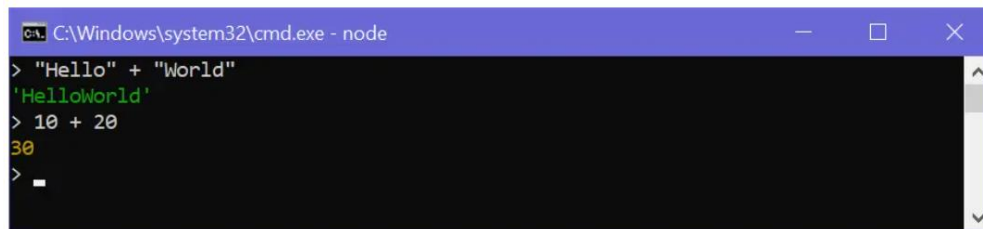
To launch the REPL (Node shell), open command prompt (in Windows) or terminal (in Mac or UNIX/Linux) and type *node* as shown below. It will change the prompt to > in Windows and MAC.



Launch Node.js REPL

You can now test pretty much any Node.js/JavaScript expression in REPL. $10 + 20$ will display 30 immediately in new line.

The + operator also concatenates strings as in browser's JavaScript.



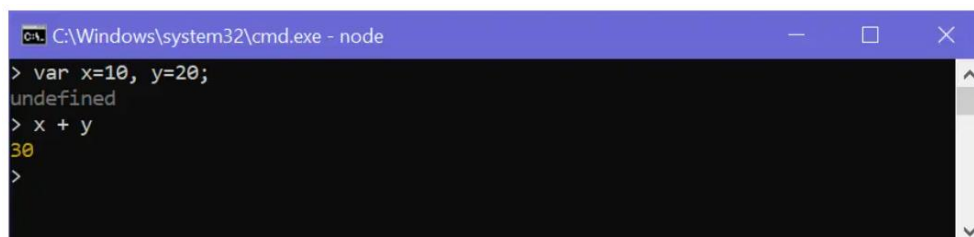
Node.js Examples

You can also define variables and perform some operation on them.

Define Variables on REPL

If you need to write multi line JavaScript expression or function then just press **Enter** whenever you want to write something in the next line as a continuation of your code. The REPL terminal will display three dots (...), it means you can continue on next line. Write .break to get out of continuity mode.

For example, you can define a function and execute it as shown below.



Node.js Example in REPL

You can execute an external JavaScript file by executing the `node fileName` command. For example, the following runs `mynodejs-app.js` on the command prompt/terminal and displays the result.

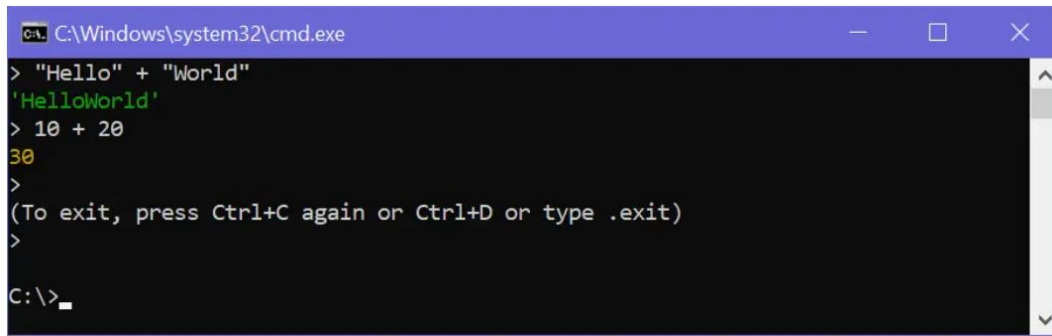
```
console.log("Hello World");
```

Now, you can execute `mynodejs-app` from command prompt as shown below.



Run External JavaScript file

To exit from the REPL terminal, press Ctrl + C twice or write .exit and press Enter.

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background with white text. The text shows a Node.js REPL session where the user has entered two commands: "> 'Hello' + 'World'" which resulted in the output "'HelloWorld'", and "> 10 + 20" which resulted in the output "30". Below these, there is a prompt ">" followed by the text "(To exit, press Ctrl+C again or Ctrl+D or type .exit)" and another prompt ">". At the bottom, the command prompt shows "C:\>".

Quit from REPL

Thus, you can execute any Node.js/JavaScript code in the node shell (REPL). This will give you a result which is similar to the one you will get in the console of Google Chrome browser.

The following table lists important REPL commands.

REPL Command	Description
.help	Display help on all the commands
tab Keys	Display the list of all commands.
Up/Down Keys	See previous commands applied in REPL.
.save filename	Save current Node REPL session to a file.
.load filename	Load the specified file in the current Node REPL session.
ctrl + c	Terminate the current command.
ctrl + c (twice)	Exit from the REPL.
ctrl + d	Exit from the REPL.
.break	Exit from multiline expression.
.clear	Exit from multiline expression.

Install MongoDB Community Edition on Ubuntu

MongoDB Atlas

[MongoDB Atlas](#) is a hosted MongoDB service option in the cloud which requires no installation overhead and offers a free tier to get started.

MongoDB Version

This tutorial installs MongoDB 7.0 Community Edition. To install a different version of MongoDB Community, use the version drop-down menu in the upper-left corner of this page to select the documentation for that version.

Considerations

Platform Support

MongoDB 7.0 Community Edition supports the following **64-bit** Ubuntu LTS (long-term support) releases on [x86_64](#) architecture:

- 22.04 LTS ("Jammy")
- 20.04 LTS ("Focal")

MongoDB only supports the 64-bit versions of these platforms. To determine which Ubuntu release your host is running, run the following command on the host's terminal:

```
cat /etc/lsb-release
```

MongoDB 7.0 Community Edition on Ubuntu also supports the [ARM64](#) architecture on select platforms.

See [Platform Support](#) for more information.

Production Notes

Before deploying MongoDB in a production environment, consider the [Production Notes for Self-Managed Deployments](#) document which offers performance considerations and configuration recommendations for production MongoDB deployments.

Official MongoDB Packages

To install MongoDB Community on your Ubuntu system, these instructions will use the official `mongodb-org` package, which is maintained and supported by MongoDB Inc. The official `mongodb-org` package always contains the latest version of MongoDB, and is available from its own dedicated repo.

IMPORTANT

The `mongodb` package provided by Ubuntu is **not** maintained by MongoDB Inc. and conflicts with the official `mongodb-org` package. If you have already installed the `mongodb` package on your Ubuntu system, you **must** first uninstall the `mongodb` package before proceeding with these instructions.

See [MongoDB Community Edition Packages](#) for the complete list of official packages.

Install MongoDB Community Edition

Follow these steps to install MongoDB Community Edition using the `apt` package manager.

Import the Public Key

From a terminal, install `gnupg` and `curl` if they are not already available:

```
sudo apt-get install gnupg curl
```

To import the MongoDB public GPG key, run the following command:

```
curl -fsSL https://www.mongodb.org/static/pgp/server-7.0.asc | \
sudo gpg -o /usr/share/keyrings/mongodb-server-7.0.gpg \
--dearmor
```

Create the List File

Create the list file `/etc/apt/sources.list.d/mongodb-org-7.0.list` for your version of Ubuntu.

Ubuntu 22.04 (Jammy)

Ubuntu 20.04 (Focal)

Create the list file for Ubuntu 22.04 (Jammy):

```
echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-server-7.0.gpg ]
https://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/7.0 multiverse" | sudo tee
/etc/apt/sources.list.d/mongodb-org-7.0.list
```

Reload the Package Database

Issue the following command to reload the local package database:

```
sudo apt-get update
```

Install MongoDB Community Server

You can install either the latest stable version of MongoDB or a specific version of MongoDB.

Latest Release

Specific Release

To install the latest stable version, issue the following

```
sudo apt-get install -y mongodb-org
```

For help with troubleshooting errors encountered while installing MongoDB on Ubuntu, see our [troubleshooting](#) guide.

Run MongoDB Community Edition

ulimit Considerations

Most Unix-like operating systems limit the system resources that a process may use. These limits may negatively impact MongoDB operation, and should be adjusted. See [UNIX ulimit Settings for Self-Managed Deployments](#) for the recommended settings for your platform.

NOTE

If the `ulimit` value for number of open files is under `64000`, MongoDB generates a startup warning.

Directories

If you installed through the package manager, the data directory `/var/lib/mongodb` and the log directory `/var/log/mongodb` are created during the installation.

By default, MongoDB runs using the `mongodb` user account. If you change the user that runs the MongoDB process, you **must** also modify the permission to the data and log directories to give this user access to these directories.

Configuration File

The official MongoDB package includes a [configuration file](#) (`/etc/mongod.conf`). These settings (such as the data directory and log directory specifications) take effect upon startup. That is, if you change the configuration file while the MongoDB instance is running, you must restart the instance for the changes to take effect.

Procedure

Follow these steps to run MongoDB Community Edition on your system. These instructions assume that you are using the official `mongodb-org` package -- not the unofficial `mongodb` package provided by Ubuntu -- and are using the default settings.

Init System

To run and manage your `mongod` process, you will be using your operating system's built-in [init system](#). Recent versions of Linux tend to use **systemd** (which uses the `systemctl` command), while older versions of Linux tend to use **System V init** (which uses the `service` command).

If you are unsure which init system your platform uses, run the following command:

```
ps --no-headers -o comm 1
```

Then select the appropriate tab below based on the result:

- `systemd` - select the **systemd (systemctl)** tab below.
- `init` - select the **System V Init (service)** tab below.

systemd (systemctl)

System V Init (service)

Start MongoDB.

Issue the following command to start `mongod`:

```
sudo service mongod start
```

Verify that MongoDB has started successfully

Verify that the `mongod` process has started successfully:

```
sudo service mongod status
```

You can also check the log file for the current status of the `mongod` process, located at: `/var/log/mongodb/mongod.log` by default. A running `mongod` instance will indicate that it is ready for connections with the following line:

```
[initandlisten] waiting for connections on port 27017
```

Stop MongoDB.

As needed, you can stop the `mongod` process by issuing the following command:

```
sudo service mongod stop
```

Restart MongoDB.

Issue the following command to restart [mongod](#):

```
sudo service mongod restart
```

Begin using MongoDB.

Start a [mongosh](#) session on the same host machine as the [mongod](#). You can run [mongosh](#) without any command-line options to connect to a [mongod](#) that is running on your localhost with default port 27017.

```
mongosh
```

For more information on connecting using [mongosh](#), such as to connect to a [mongod](#) instance running on a different host and/or port, see the [mongosh documentation](#).

To help you start using MongoDB, MongoDB provides [Getting Started Guides](#) in various driver editions. For the driver documentation, see [Start Developing with MongoDB](#).

Uninstall MongoDB Community Edition

To completely remove MongoDB from a system, you must remove the MongoDB applications themselves, the configuration files, and any directories containing data and logs. The following section guides you through the necessary steps.

WARNING

This process will *completely* remove MongoDB, its configuration, and *all* databases. This process is not reversible, so ensure that all of your configuration and data is backed up before proceeding.

Stop MongoDB.

Stop the [mongod](#) process by issuing the following command:

```
sudo service mongod stop
```

Remove Packages.

Remove any MongoDB packages that you had previously installed.

```
sudo apt-get purge "mongodb-org*"
```

Remove Data Directories.

Remove MongoDB databases and log files.


```
sudo rm -r /var/log/mongodb
sudo rm -r /var/lib/mongodb
```

Build a CRUD Command Line Application With Node.js

- First off, let's break down what a CRUD API is. CRUD stands for Create, Read, Update, and Delete.
- These are the four basic operations you can perform on data in any database.
- An API, or Application Programming Interface, is a set of rules and protocols for building and interacting with software applications.
- So, a CRUD API allows you to create, read, update, and delete data through defined interfaces.
- You can use JavaScript to build almost any software (web, mobile, bots, and so on).
- The reason is that computers no longer depend on browsers only to understand JavaScript code.
- Node.js is used to run backend applications that are written in JavaScript.
- I will teach you how to build a command line application that can read, write, edit, and delete data using Node.js.
- It will not require connecting to external databases like MySQL, MongoDB, Postgresql, and so on.
- By the end of the topic, you should be able to set up a basic Node project, manipulate files, use modules, navigate promises, collect input, and so on.

Prerequisites

- You do not need any prior programming knowledge to understand this tutorial.

How the Finished Project Will Function

The application that we are going to build will be able to do the following:

- Check if a database exists
- Retrieve data from the database
- Add new data to the database
- Update database with new data
- Remove data from the database

How to Install Node and NPM

Please go to the Nodejs website and download the one recommended for all users. Install it after the download is complete.

Use the command below to check if the installation was successful:

- For Node

node -v

- For npm

npm -v

The installation is successful if each command returns a version number.

How to Setup a Node.js Project

The steps below will help you set up your project:

Open your terminal or CMD and create the project directory:

mkdir node_CLI_app

Navigate into the directory:

cd node_CLI_app

Initialize the project:

npm init

It will bring up some questions. Press the Enter button for every prompt. A new file (package.json) should have been added to your project directory. We will use the file to add external code (module) to the project.

Add the following line before the closing curly brace (}) to enable ES6 import:

"type": "module"

That completes the project setup!

How to Install Dependencies

You will recall that the package.json file is for adding external code to the project. This external code is also called Dependencies, Modules, or Packages. It's written by other programmers (usually free of charge) to help others build applications faster. You will find a lot of these on the [npm website](#).

We need two (2) packages for this project: inquirer and uuid. I will show you how to install them in this section.

Installing of packages follows this pattern:

npm install <package_name>

We can install more than one package at a time by separating the package names with a space:

npm install <package_name_1> <package_name_2> <package_name_3>

The install command can be replaced with i for convenience.

So run the following command to install the packages:

```
npm i inquirer uuid
```

Your package.json file should have the following lines of code added to it when the installation is complete:

```
"dependencies": {  
  "inquirer": "^9.1.4",  
  "uuid": "^9.0.0"  
}
```

The version numbers may differ, but that's fine.

You will also notice that a file (package.lock.json) and a folder (node_modules) have been added. You do not have to worry about them. Just know that they help to manage the external code we just added.

That completes the installation of dependencies!

How to Create a New File from the Terminal

You can create a file from the terminal using the following command:

- On Mac:

```
touch <file_name>
```

- On Windows:

```
echo.><file_name>
```

You can also create more than one file at a time by separating the files with a space.

That is how we will generate the files for this project:

```
touch addData.js removeData.js retrieveData.js updateData.js queryDB.js  
dbFileCheck.js
```

Each of those files will play a unique role in the application.

- **addData.js** adds data to the database. It also fabricates the database file if it doesn't exist.
- **removeData.js** removes selected data.
- **retrieveData.js** fetches all data.
- **updateData.js** edits data.
- **queryDB.js** checks if a database exists and executes a function passed to it.
- **dbFileCheck.js** confirms if the database file has been created.

There is one more file that we did not create: the database file (db.json). We will auto-generate it using our code.

How to Check if a File Exists

Node.js provides a built-in module for manipulating files – file system (fs). One of the methods that the module contains is the **existsSync** method. It

returns true or false depending on if the file has been created. I will use that to check if the **db.json** file is in the project. See the code below:

```
import fs from "fs";
import { exit } from "process";

if (fs.existsSync("db.json")) {
  console.log("File exists!");
  exit(1);
}
```

The exit method terminates a process.

That is what we want to do in the **dbFileCheck.js** file. However, we want to invert the result by adding a ! before the **fs.existsSync** method like this:

```
import fs from "fs";
import { exit } from "process";

if (!fs.existsSync("db.json")) {
  console.log("File does not exist!");
  exit(1);
}
```

This will be necessary when building other functionalities like updating and deleting data.

So how will those files gain access to this code? It is through a modular structure. That entails bundling this code and making it accessible through other files in the project. The export command makes this possible:

```
import fs from "fs";
import { exit } from "process";
export default function dbFileCheck() {
  if (!fs.existsSync("db.json")) {
    console.log("Database is Empty. Create some data!");
    exit(1);
  }
}
```

The code above puts the code in a function (dbFileCheck) and exports it using the export command. This function can now be imported and used in other files within this project.

Note that the default keyword is necessary for the first export from a file.

How to Query the Database

Another method the file system has is the `readFile` method. It returns the content of any file passed to it.

The `readFile` method takes in two parameters: the file to be read and a `callback` function that returns the result of the operation.

We will use the following code to retrieve data from our database:

```
import fs from "fs";
```

```
fs.readFile("db.json", function (err, data) {  
  if (err) {  
    console.log(err);  
  }  
  console.log(data.toString());  
});
```

The code above imports the file system module and tries to read the database file. If there is an error, it will return the error. If there are no errors, it will return the data it got.

The `.toString()` method attached to the data returned (`data.toString()`) is because the data retrieved is of the type buffer by default, which is not readable.

```
<Buffer 54 68 65 20 60 72 65 61 64 46 69 6c 65 60 20 6d 65 74 68 6f 64 20 74 61 6b 65 73 20 69 6e 20 74 77  
6f 20 70 61 72 61 6d 65 74 65 72 73 3a 20 74 68 65 ... 150 more bytes>
```

buffer output

Open your project in a code editor and paste that code into the `queryDB.js` file. Run the command below to see if it is working:

```
node queryDB
```

The command above will execute every code in the `queryDB.js`. The `.js` extension is optional. It will run either way.

The result of that operation will be an error because we do not have the file.

```
[Error: ENOENT: no such file or directory, open 'db.json'] {  
  errno: -2,  
  code: 'ENOENT',  
  syscall: 'open',  
  path: 'db.json'  
}  
undefined
```

You can create the `db.json` file, add some content, and check the output.

But we do not want our app to try to read the database file if it has not been generated. So use the `existsSync` method to check if the file has been created. See how I use it in the code below:

```
import fs from "fs";
```

```

if (fs.existsSync("db.json")) {
  fs.readFile("db.json", function (err, data) {
    if (err) {
      console.log(err);
    }
    console.log(data.toString());
  });
} else {
  console.log("No data available!");
}

```

The code above checks if the file exists by passing the file name (**db.json**) to the **fs.existsSync** method. If it is true, then it reads the file. If it is false, it returns a string.

Now that we have that clean code, we want to make it even more robust.

Since we are building a CRUD application, we must have access to and keep track of the data returned. I have introduced a new variable in the code below to make that happen:

```

import fs from "fs";

let info = [];
if (fs.existsSync("db.json")) {
  fs.readFile("db.json", function (err, data) {
    if (err) {
      console.log(err);
    }
    info = JSON.parse(data.toString());
    console.log(info);
  });
} else {
  console.log("No data available!");
}

```

The code above now has a variable, **info**. It keeps track of the data returned. I passed the data through the **JSON.parse** method to convert the data from string to array.

We can now manipulate the info as we deem fit. We have to export the code and accept a function to make this possible. That function will take the info variable as input and then use it as required.

```
import fs from "fs";
```

```
export default async function queryDB(externalFunction) {  
  try {  
    let info = [];  
  
    if (fs.existsSync("db.json")) {  
      await fs.readFile("db.json", function (err, data) {  
        info = JSON.parse(data.toString());  
        console.log(info);  
  
        if (err) {  
          console.log(err);  
          return;  
        }  
  
        if (externalFunction && !err) {  
          externalFunction(info);  
          return;  
        }  
      });  
    } else {  
      if (externalFunction) {  
        externalFunction(info);  
        return;  
      }  
    }  
  } catch (error) {  
    console.error(`Something Happened: ${error.message}`);  
  }  
}
```

This code takes in a function and executes it only if there are no errors. However, there will be one exception – when adding data. In that case, the database will be created.

The `try...catch...` block helps pick out errors, and the `async` `await` keywords are used when executing code that will take a long time to run.

The `queryDB.js` will no longer return any output when we execute the node `queryDB` command. But that is fine. We will trigger it from other files.

How to Add Data to a File

In this section, I will teach you how to add data to the store. The file for this section is the `addData.js` file.

We will start by importing all necessary modules:

```
import inquirer from "inquirer";
```

```
import fs from "fs";
```

```
import { v4 as uuidv4 } from "uuid";
```

```
import queryDB from "../queryDB.js";
```

Next, create the boilerplate for the function:

```
export default async function addData(info) {
```

```
  try {  
  
  } catch (error) {  
    console.log("Something went wrong!", error);  
  }  
}
```

This is similar to what we have done before now. All the code we would write next will go into the `try...` block. The `info` array passed into the function is coming from the `queryDB` file.

The first thing to do in the `try...` block is to collect the data to be stored. We will use `inquirer` to do that with the code below:

```
const answers = await inquirer.prompt([  
  {  
    type: "input",  
    name: "name",  
    message: "What's your name?",  
  },  
  {
```



```

    type: "number",
    name: "phone",
    message: "What's your phone?",
  },
  {
    type: "list",
    name: "age",
    message: "Are you an adult?",
    choices: [
      { name: "Y", value: "Adult" },
      { name: "N", value: "Minor" },
    ],
  },
]);

```

The inquirer package helps in building interactive command line interfaces. It contains a method called `prompt` used to ask for inputs. It takes an array of questions in object format. Each object must have the `name`, `type`, and `message` keys.

The `choices` key is optional. It is used when there is a list of options.

So the code above collects three (3) inputs (`name`, `phone`, and `age`) and stores them in a variable called `answers`.

Next, we assign this set of input a unique ID by calling the `uuidv4()` function and push everything into the `info` array:

```

const data = {
  id: uuidv4(),
  name: answers.name,
  phone: answers.phone,
  age: answers.age,
};
info.push(data);

```

Finally, we check if the database file exists. We will update the database with the new data if the file has been created or create it and add the new data if it is false.

```

if (fs.existsSync("db.json")) {
  createDetails(info);
} else {
  fs.appendFile("db.json", "[]", (err) => {

```

```

    if (err) {
        console.log("Could not create db.json", err);
        return;
    }
    createDetails(info);
  });
}

```

The **createDetails** function will be used to overwrite the database data. I will create that in a bit.

In the else (if the file doesn't exist) block, I created the file using the file system's *appendFile* method. This method is used to create a file if it is not there already and add or append data to the bottom of the file.

I am appending [] to the file. So the newly created file will have just [] in it. I called the createDetails function next to add the data collected from the CLI.

Please type the following code for the createDetails function below the addData function.

```

async function createDetails(info) {
  await fs.writeFile("db.json", JSON.stringify(info), function (err) {
    if (err) {
      console.log(err);
    }
    console.log("saved!");
  });
}

```

This function uses the file system's writeFile method to overwrite the database with current data. I am using JSON.stringify to convert the info variable to a string because that is the acceptable format when writing to a file. That also explains why I used JSON.parse to convert it to the original type when I retrieved it in the previous section.

writeFile is different from appendFile because writeFile overwrites a file while appendFile adds to the existing content. They are similar in that both methods will create the file if it has not been created.

The last thing to do in this file is invoke or call the function. The way to do that is by typing its name followed by opening and closing braces like this:

```
addData();
```

This might work fine, but it wouldn't do what we want. We need to pass the function as an argument into the queryDB like this:

```
queryDB(addData);
```

Now, the addData will be able to communicate with the queryDB function.

How to Test the addData File

Run the command below to test if the addData file works as expected:

```
node addData
```

It will prompt you for answers. Fill in the answers. After that, your screen should look like mine:

```
? What's your name? Sam
? What's your phone? 90
? Are you an adult? Y
saved!
```

addData output

It should also have auto-generated a db.json file with the data you just added.

And that is it for the addData file!

How to Retrieve Data

This section shows how to get the content of the database. All you need to achieve this functionality is the code below:

```
import queryDB from "./queryDB.js";
```

```
queryDB();
```

What the code does is import the queryDB function and invoke it. Paste the code in the retrieveData.js file and execute the file with:

```
node retrieveData
```

It will return an output similar to this:

```
[
  {
    id: '634da119-7280-4f32-8e63-88b1e73bf845',
    name: 'Sam',
    phone: 90,
    age: 'Adult'
  }
]
```

retrieveData.js output

You might wonder: why did I not call this function in the queryDB file? The reason is that the file performs more than just retrieving data. Calling the queryDB function in its file will alter the result of other files.

How to Edit Data

This section will now focus on how to update data. The pattern to follow here is:

- Collect the user's ID.
- Search for the user.
- Return the user's data if the user exists, and set it as the default option.
- Ask for updated info. The initial value will be kept if the user presses the Enter key.
- Finally, overwrite the database with the updated information.

Let's get to it...

Navigate into the **updateData.js** file and paste the following code:

```
import inquirer from "inquirer";
import fs from "fs";
import queryDB from "../queryDB.js";
import dbFileCheck from "../dbFileCheck.js";

export default async function updateData(info) {
  dbFileCheck();

  try {

  } catch (error) {
    console.log("Something went wrong!", error);
  }
}
```

This is the boilerplate for the updateData function. The new thing here is the dbFileCheck function (made it a while ago) to terminate an operation if the database has not been created.

The rest of the code will be inside the try... block.

The first thing is to collect the user ID with this code:

```
const answers = await inquirer.prompt([
  {
    type: "input",
```

```

    name: "recordID",
    message: "Enter Record ID",
  },
]);

```

The second is to search for the user:

```

let current;

info.forEach((element) => {
  if (element.id === answers.recordID) {
    current = element;

    updateDetails(current, info);
  }
});

```

The code above searches through the users (info) and sets the user found to the current variable. Finally, the updateDetails is called to collect the updated information and overwrite the database with the new data.

How to build the updateDetails function

This part will show how to keep track of a user's initial data while collecting new data. It will update the database with the new data afterward.

The following code is the boilerplate for the function:

```

async function updateDetails(current, info) {
  try {

  } catch (error) {
    console.log("Something went wrong!", error);
  }
}

```

This code goes under the updateData operation.

The code below goes into the try... block. It is for collecting updated information from the user.

```

const feedbacks = await inquirer.prompt([
  {
    type: "input",

```

```

    default: current.name,
    name: "name",
    message: "What's your name?",
  },
  {
    type: "number",
    default: current.phone,
    name: "phone",
    message: "What's your phone?",
  },
  {
    type: "list",
    default: current.age,
    name: "age",
    message: "Are an adult?",
    choices: [
      { name: "Y", value: "Adult" },
      { name: "N", value: "Minor" },
    ],
  },
]);

```

The default key is new. It holds input that will be used if the user doesn't provide one. It keeps track of the user's current data for this code. So the user can hit the Enter button instead of entering the former value again.

The user's details should be updated accordingly using this code:

```

current.name = feedbacks.name;
current.phone = feedbacks.phone;
current.age = feedbacks.age;

```

Finally, overwrite the database with the new value:

```

await fs.writeFile("db.json", JSON.stringify(info), function (err) {
  if (err) {
    console.log(err);
  }
  console.log("updated");
});

```

Call the `updateData` function to bring everything together:

```
queryDB(updateData)
```

How to test the `updateData` file

Run the command below to see how the `updateData` function performs:

```
node updateData
```

It will prompt you for an ID. Enter the ID of any record in the database.

It will then prompt for updated information about the user. Fill in the information and press Enter for any detail that does not need an update. The terminal should look like this at the end:

```
? Enter Record ID 634da119-7280-4f32-8e63-88b1e73bf845
? What's your name? Samson Ebere
? What's your phone? 90
? Are an adult? Y
updated
```

`updateData` Output

That wraps it up for the `updateData` file. Records can now be updated.

How to Delete Data

A CRUD application cannot be completed without the DELETE part. That will be the focus here. It will borrow a bit from the previous sections. So it will not be too much to grasp.

You'll begin by typing the following code in the `removeData.js` file:

```
import inquirer from "inquirer";
```

```
import fs from "fs";
```

```
import queryDB from "../queryDB.js";
```

```
import dbFileCheck from "../dbFileCheck.js";
```

```
export default async function removeData(info) {
  dbFileCheck();
```

```
  try {
    const answers = await inquirer.prompt([
      {
        type: "input",
        name: "recordID",
        message: "Enter Record ID",
      },
    ],
```

```
]);
```

```
let remnantData = [];  
info.forEach((element) => {  
  if (element.id !== answers.recordID) {  
    remnantData.push(element);  
  }  
});
```

```
fs.writeFile("db.json", JSON.stringify(remnantData), function (err) {  
  if (err) {  
    console.log(err);  
  }  
  console.log("Deleted!");  
});  
} catch (error) {  
  console.log("Something went wrong!", error);  
}  
}
```

First, the code above collects an ID.

Next, it uses the ID to search the database data and keeps only data with a different ID in the remnantData array.

Finally, it overwrites the database with the updated data.

Call the removeData function at the bottom to put everything together like this:

```
queryDB(removeData)
```

Now try to test the file using this command:

```
node removeData
```





The is my output:

```
? Enter Record ID 634da119-7280-4f32-8e63-88b1e73bf845  
Deleted!
```

removeData Ouput

NODECLIPSE PLUGIN


In short words, install Node.js with needed packages, Java and Eclipse package like Enide 2015:

1. If you don't have, get latest Node.js or io.js
2. Then install Nodeclipse CLI and Express with Express' application generator
\$ npm install -g nodeclipse
\$ npm install -g express
\$ npm install -g express-generator
If you are going to use CoffeeScript, do
\$ npm install -g coffee-script
3. If you don't have, download & install JDK 8 from Oracle Java SE Downloads.
 - For example "Java Platform (JDK) 8u162".
 - There are issues with Java 7 and 6 #71 #72 and newer Java 9 or 10.
 - **Note 0:** If you don't know what version is used by Eclipse, check in Preferences -> Nodeclipse.
 - The Eclipse most recommended way to configure what JDK to use is to configure in eclipse.ini.
- 4.a Get Enide 2015 with all Nodeclipse plugins included.
 - see also Enide 2015-7
 - for  win32  win64  MacOS X  Linux
 - See all SourceForge downloads. served.
 - See java-start and android-start pages.
 - That is enough for Node.js, JavaScript and Java development.
 - For existing Node.js project you can run nodeclipse -p to add project files and then import as existing project.
 - For any project or folder you can run nodeclipse -g and then import as existing project (this will add only 1 .project file).
 - You can update to the latest version later and install even more from Nodeclipse Updates repository, see "b) update site".
- 4.b or Download and install Eclipse for your OS.
 - Recommended is "Eclipse IDE for Committers" (actually it is Standard/SDK), "Eclipse IDE for Java Developers" or "Eclipse for Java/DSL Developers".


- You can also install Nodeclipse/Enide plugins into Aptana Studio (see issue #181), Red Hat JBoss Developer Studio, STS Spring Tool Suite or other Eclipse-based software like Adobe Flash Builder, IBM Rational products and SAP NetWeaver Developer Studio.

INSTALLATION\UPDATE INSTRUCTIONS

(for Eclipse or Enide [Studio])

In short words, drag-and-drop  on Eclipse main toolbar and get Enide 2015 plugins.

A) MARKETPLACE (RECOMMENDED WAY: A BIT QUICKER)

1. Start Eclipse.
2. Drag and drop  into a running Eclipse (menu area) to install Nodeclipse.

See tip how

2. Review the features that you are about to install. Click Next.
3. Read the license agreements, and then select I accept the terms of the license agreements. Click Finish.
4. You will then be asked if you would like to restart Eclipse. Click Restart Now.
5. Switch to Node perspective (e.g. Window -> Open Perspective -> Other ... -> Node).
6. Now you are ready to develop Node.js applications with Nodeclipse!

B) UPDATE SITE (BIGGER CHOICE)

1. Start Eclipse, then select Help > Install New Software...
2. Enter the update site URL into the Work with text box:

<https://nodeclipse.github.io/updates/>

or the one that is quicker but changes every release

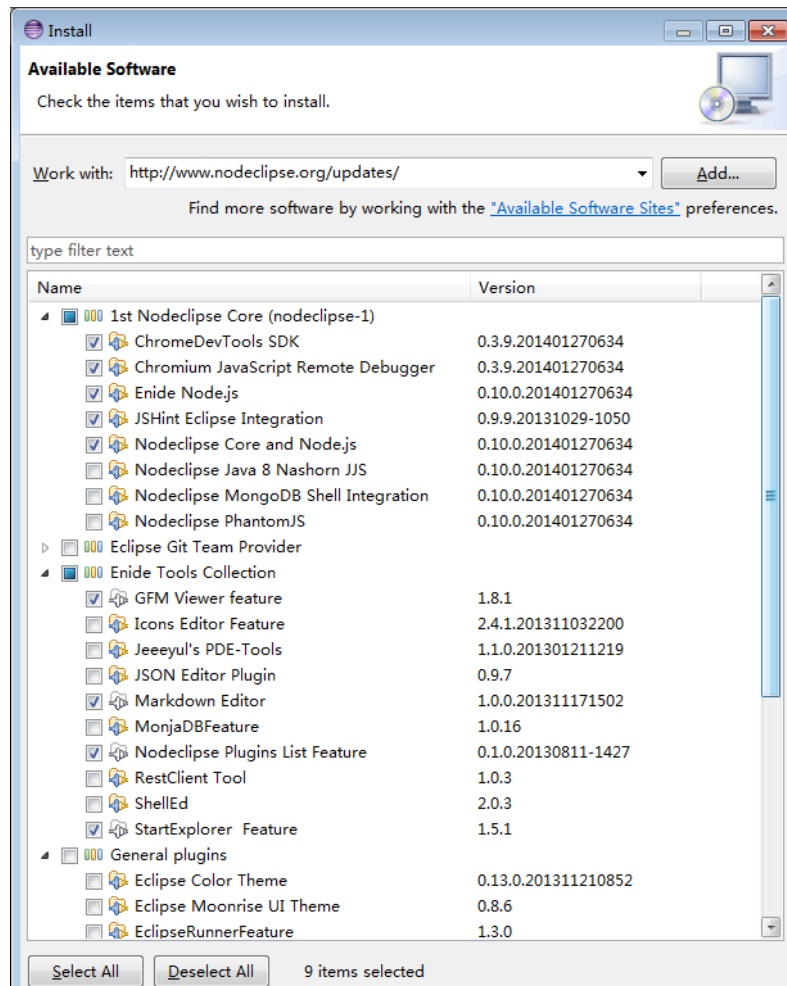
<http://dl.bintray.com/nodeclipse/nodeclipse/1.0.2f/>

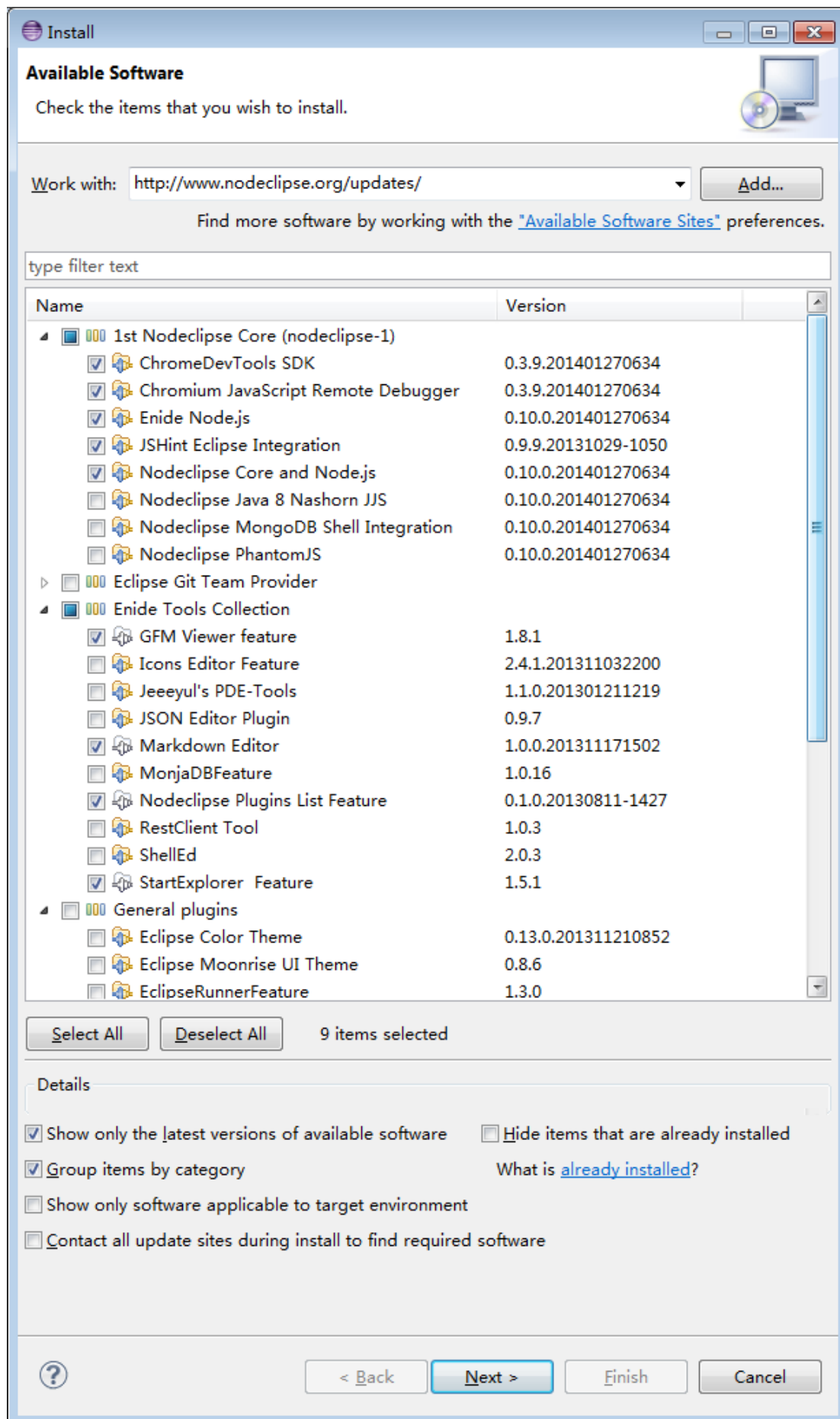
Select with checkbox what you want to install.

("The more, the better" does not apply here.

Take note of what features names you are installing.)

Below is Enide-Recommended-Set-for-Nodejs for example.





3. Uncheck "Contact all updates site during install to find required software" to make installation quicker. Note that different Eclipse version have different behaviour for that checkbox. Press the Enter key or Next button.

4. After a while depending of what you had and what you selected, you should see the center box filled with Eclipse plugins to install and Click Next.
5. Review the features that you are about to install. Click Next.
6. Read the license agreements, and then select I accept the terms of the license agreements. Click Finish.
7. You will then be asked if you would like to restart Eclipse. Click Restart Now.
8. Switch to Node perspective (e.g. Window -> Open Perspective -> Other ... -> Node).
9. Now you are ready to develop Node.js applications with Nodeclipse and great tools!

C) ARCHIVE (WHEN INTERNET IS SLOW, NOT STABLE OR INSTALLING MANY TIMES)

zip archive has content identical to update p2 site

0. Download zip from SourceForge.

1. Start Eclipse.
2. Help -> Install New Software -> Add -> Archive.

Hint for b) & c): keep the file name the same when updating, or remove old path and URLs as Eclipse keeps checking them.

D) ENIDE P2F FILE

0. Read more at Marketplace page.

E) NODECLIPSE CLI INSTALLER

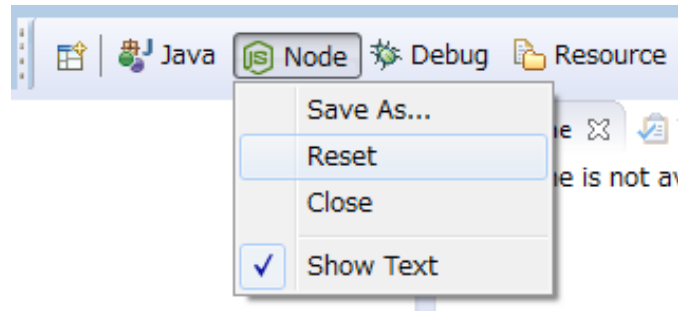
0. Read more at GitHub page.

1. cd path/to/eclipse
2. nodeclipse install nodejs markdown gfm startexplorer
3. Start Eclipse.

UPDATING

- If you used <http://dl.bintray.com/nodeclipse/nodeclipse/1.0.2f/> like URL or update site zip archive, you will just need to install with any above option. (Also you will need to clean up old update sites in Preferences -> Update/Install -> Available Software Sites.) Otherwise Eclipse "Help -> Check for updates" should work.
- You will be notified about updates if you have "Automatic Updates" enabled in Preferences.
- Check home page for announces and news about new plugins and workflow.

- Reset perspective after an update for new UI elements to appear:



- If you have run into the problem while updating, e.g. "#72",
 - 1) **let us know by raising new issue (tell exactly what OS, Java version, Eclipse version and Nodeclipse version is used.)**
 - 2) **Update to latest JDK.**
 - 3) **Uninstall then install:**
 - 3.1) **Uninstall Nodeclipse Core (only 1 feature entry) via Help -> About Eclipse -> Installation details, select Nodeclipse, then Uninstall.. button**
 - 3.2) **install again.**

TROUBLESHOOTING

- If you can't install, try different way to get Nodeclipse, e.g. other update site, Enide or Enide Studio.
- Check home page for announcements.
- Try support options.

Source:

1. <https://github.com/kamaljitsingh76/nodeclipse>
2. <https://nodeclipse.github.io/>
3. <https://www.youtube.com/watch?v=gk3yTklK8Rw>
4. <https://github.com/nodejs>

How to use Node.js with Eclipse IDE on Ubuntu

- Here you will learn, how to use Node.js with Eclipse IDE using the `nodeclipse` package.
- We install Node.js and the Eclipse IDE on Ubuntu before using Node.js with the IDE.
- You can jump to section 3 if you have already installed Node.js (section 1) and Eclipse IDE (section 2) on your system.

Section-1: Installing Node.js

Check if Node.js is installed

Knowing how to use Node.js with Eclipse IDE starts by installing Node.js and Eclipse IDE. First, check for a Node.js installation.

```
bash
# Check the binaries
$ which node
$ which npm

# or their versions
$ node -v
```

```
$ npm -v
```

Output

```
bash
```

```
user@hostname:~$ which node
```

```
user@hostname:~$ which npm
```

```
user@hostname:~$ node -v
```

```
Command 'node' not found, but can be installed with:
```

```
user@hostname:~$ npm -v
```

```
Command 'npm' not found, but can be installed with:
```

Node.js is not installed on the system. We can install the latest version as follows.

Install Node.js from the source

We have already covered this topic in detail in [Install Node.js on Ubuntu 20.04 \[3 Different Methods\]](#) so we will keep this section short and to the topic:

Update, then upgrade the system.

```
bash
```

```
$ sudo apt update
```

```
$ sudo apt upgrade
```

Install the curl tool before downloading Node.js version 18+ from the sources.

```
bash
```

```
# install curl
```

```
sudo apt install curl
```

```
# download the files from Node.js sources to Ubuntu repositories
```

```
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
```

Then, install the downloaded files.

```
bash
```

```
sudo apt-get install -y nodejs
```

Now let's confirm the installations.

Input

```
bash
```

```
$ node -v
```

```
$ npm -v
```

Output

```
bash
```

```
user@hostname:~$ node -v
```

```
v18.10.0
```

```
user@hostname:~$ npm -v
```

```
8.19.2
```

Node.js version 18.10.0 is now installed on the system.

```
user@hostname:~$ node -v
v18.10.0
user@hostname:~$ npm -v
8.19.2
user@hostname:~$
```


Section-2: Installing Eclipse

Eclipse IDE requires Java installation before being installed on the local machine. Let's update the system and then install Java Development Kit.

Install Java

We have covered detailed guide on [installing Java on Liux, Windows and MAC Platform](#). So, we will keep this section short by just giving brief command outputs:

```
bash
$ sudo apt update
$ sudo apt install default-jre
```

Check the installed JDK version.

Input

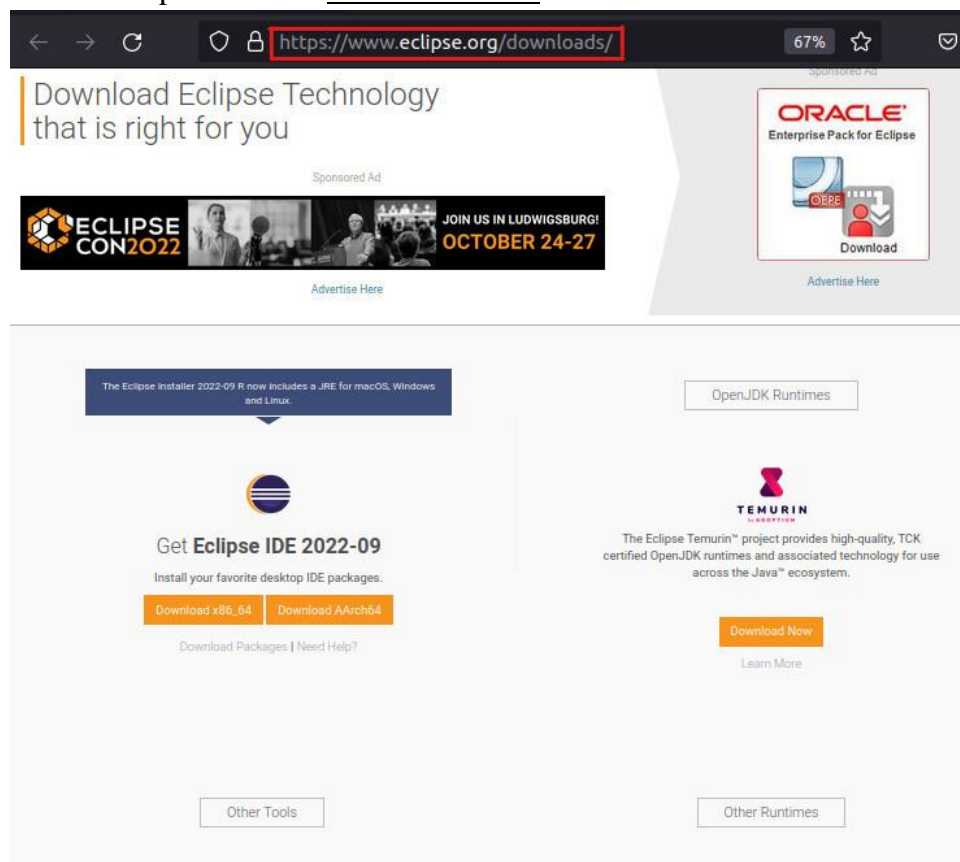
```
bash
$ java --version
```

Output

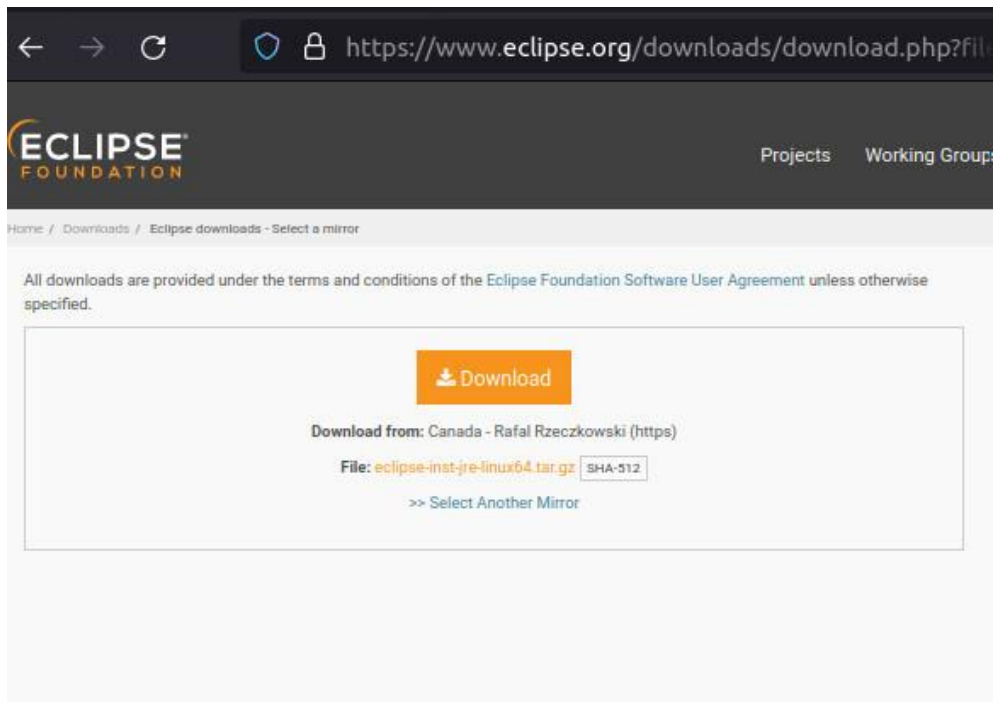
```
bash
user@hostname:~$ java --version
openjdk 11.0.16 2022-07-19
OpenJDK Runtime Environment (build 11.0.16+8-post-Ubuntu-0ubuntu122.04)
OpenJDK 64-Bit Server VM (build 11.0.16+8-post-Ubuntu-0ubuntu122.04, mixed mode, sharing)
```

Download Eclipse

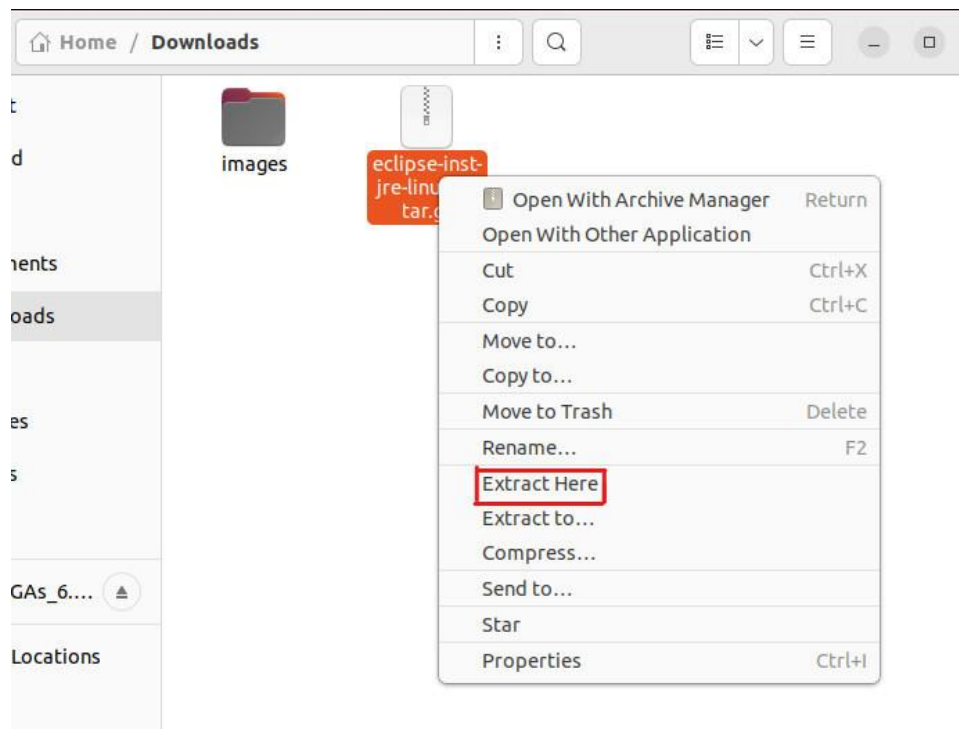
Get the latest version of Eclipse from the [Official Website](#).



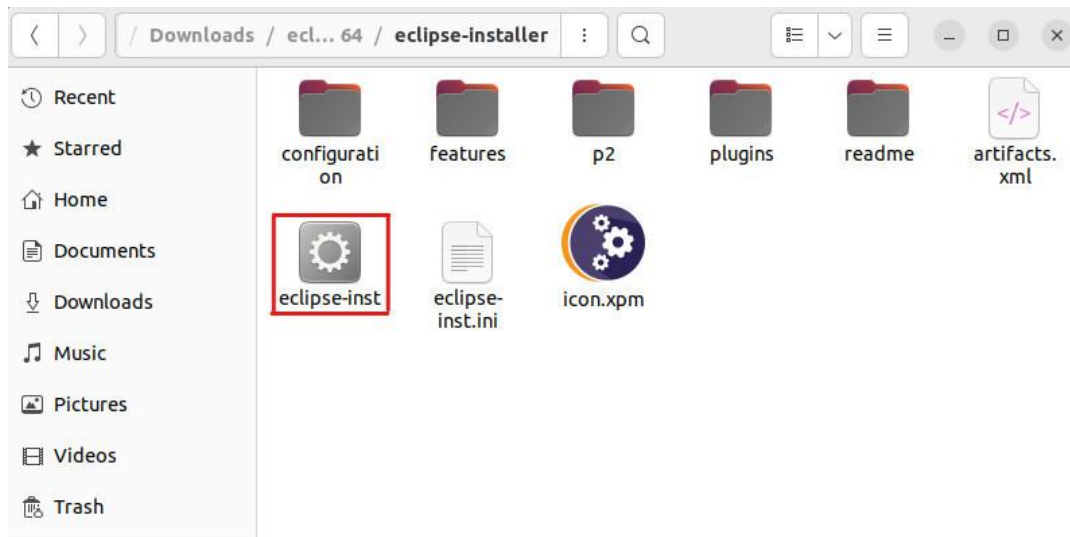
Click on *Download x86_64* followed by *Download*.



An archived file gets saved in the Downloads directory. Extract the file by right-clicking it and choosing *Extract Here*.



Open the new directory followed *eclipse-installer* → *eclipse-inst*.



Install Eclipse

The installer launches.

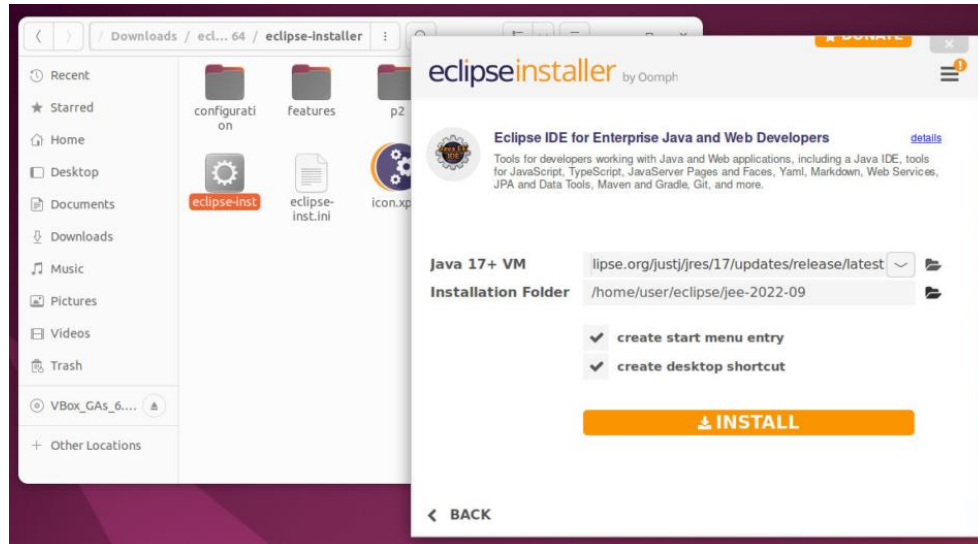


Select *Eclipse IDE for Enterprise Java and Web Developers*.

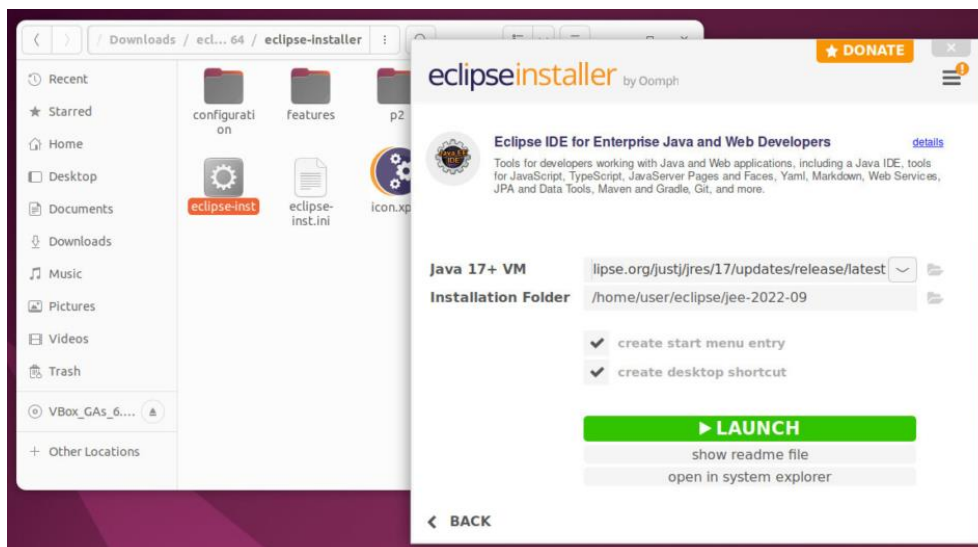


The installer selects the installed Java version. Click

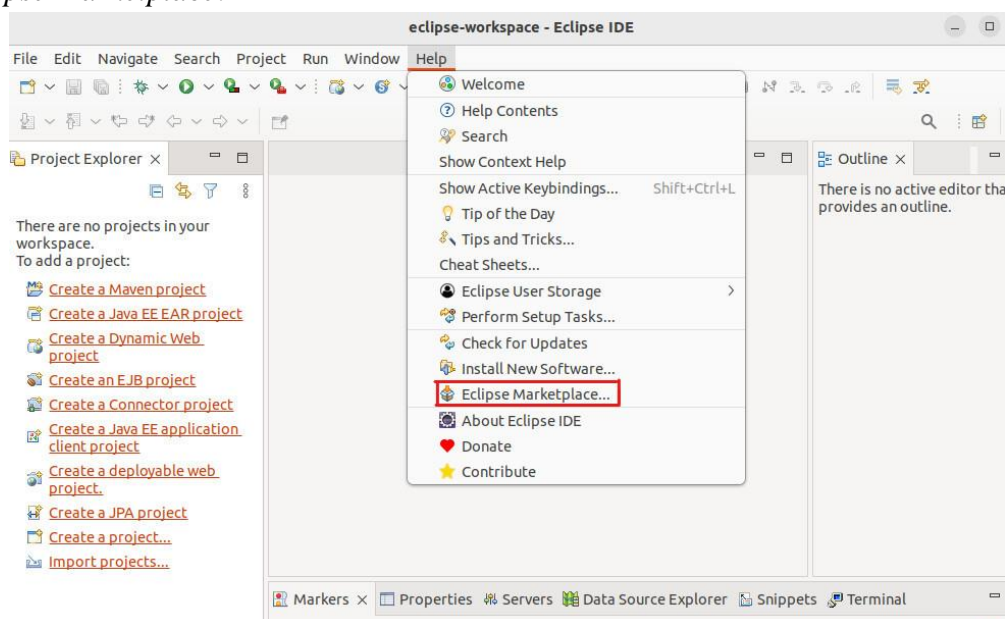
on the *INSTALL* button, then accept the terms and conditions to start downloading the software.



Lastly, we can launch the IDE.



Let's install a Node.js package to enable us to create a Node.js project on the Eclipse IDE. Go to *Help* → *Eclipse Marketplace*.




Search for Node.js. The Enide (Studio) 2015 - Node.js, JavaScript, Java and Web Tools 1.0.2 appears. Install it.

Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installation.
Press the "more info" link to learn more about a solution.

Search Recent Popular Favorites Installed Giving IoT an Edge

Find: All Markets All Categories Go




Enide (Studio) 2015 - Node.js, JavaScript, Java and Web Tools 1.0.2

Nodeclipse "Enide Studio 2015" is Tool Suite For Node.js, JavaScript, Java Development. This is the most feature-rich pack. Unless there is the latest version... [more info](#)

by Nodeclipse/Enide, EPL

[Node.js](#) [javascript](#) [es5](#) [es6](#) [java](#)

★ 67 Installs: **84.8K** (431 last month) Install



Enide Studio 2014 - Node.js, JavaScript, Java and Web Tools 1.0.1

Nodeclipse "Enide Studio 2014" is Tool Suite For Node.js, JavaScript, Java Development. This is the most feature-rich pack. Unless there is the latest version... [more info](#)

by Nodeclipse/Enide, EPL

[Node.js](#) [javascript](#) [java](#) [maven](#) [gradle](#)


★ 41 Installs: **50.0K** (0 last month) Install



Wild Web Developer: HTML, CSS, JavaScript, TypeScript, Node.js, Angular, JSON, YAML (+Kubernetes), XML 0.13.1

Confirm the selected features.

Confirm Selected Features

Press Confirm to continue with the installation. Or go back to choose more solutions to install.



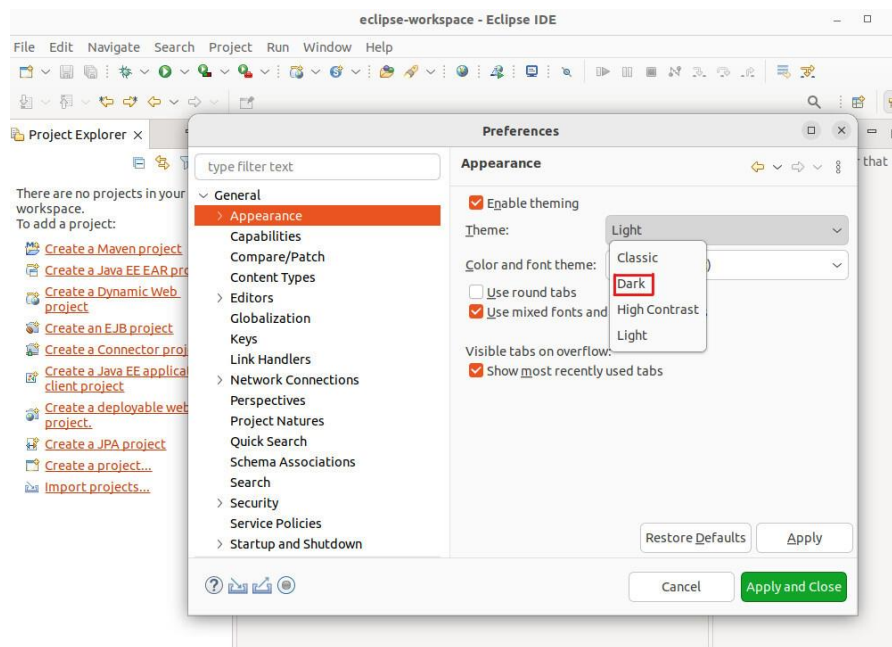
Enide (Studio) 2015 - Node.js, JavaScript, Java and Web Tools 1.0.2 <https://nodeclipse.github.io/updates/>

- ☒ org.nodeclipse.enide.editors.gradle.feature (required)
- ☒ org.nodeclipse.pluginslist.feature (required)
- ☒ AngularJS Eclipse
- ☒ code.google.restclient.tool.feature
- ☒ code.satyagraha.gfm.viewer.feature
- ☒ com.github.eclipsecolortheme.feature
- ☒ de.bastiankrol.startexplorer.feature
- ☒ Eclipse WTP HTML-Web resources
- ☒ gitaddon.feature
- ☐ io.emmet.eclipse
- ☒ jdt.spelling.feature
- ☒ JSHint Eclipse Integration
- ☒ markdown.editor.feature
- ☒ Mirur Visual Debugger
- ☒ net.mihai-nita.ansicon
- ☒ org.dadacoalition.yedit
- ☒ org.nodeclipse.enide.gradle.feature
- ☐ org.nodeclipse.enide.maven.feature
- ☒ org.nodeclipse.enide.nodejs.feature
- ☐ org.nodeclipse.jjs.feature
- ☐ org.nodeclipse.mongodb.feature
- ☒ org.nodeclipse.phantomjs.feature
- ☐ org.nodeclipse.vertx.feature
- ☒ org.sweetlemonade.eclipse.json.feature
- ☒ pm.eclipse.editbox.feature
- ☒ Terminals View
- ☒ Tern feature

?
< Install More
Confirm >
Cancel
Finish

Accept the License Agreement and let the Software be installed. Besides, we can install *Emmet* (*ex-Zen Coding*) for HTML auto-completion.

Now we can develop our Node.js web server. Before that, we can switch to the dark theme by visiting *Window* → *Preferences* → *General* → *Appearance*, then selecting *Dark* from the dropdown menu. Lastly, click on the *Apply and Close* button, then *Restart* the IDE for the changes to be effective.



Section-3: How to use Node.js with Eclipse IDE by developing an Express Web Server

This section teaches you how to use Node.js with Eclipse IDE by loading a styled HTML page using a web server built with the express framework.

Set up a lab environment

Here is the project structure. Create the structure before initializing an NPM package and installing express.

```
bash
eclipseNode
├── index.js
├── public
│   ├── index.html
│   └── style.css
```

1 directory, 3 files

I create the project structure as follows.

```
bash
$ mkdir eclipseNode && cd eclipseNode
$ mkdir public
$ touch public/index.html public/style.css
$ touch index.js
$ npm init -y
$ npm i express
```

```

user@hostname:~$ mkdir eclipseNode && cd eclipseNode
user@hostname:~/eclipseNode$ mkdir public
user@hostname:~/eclipseNode$ touch public/index.html public/style.css
user@hostname:~/eclipseNode$ touch index.js
user@hostname:~/eclipseNode$ npm init -y
Wrote to /home/user/eclipseNode/package.json:

{
  "name": "eclipsenode",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

user@hostname:~/eclipseNode$ npm i express

added 57 packages, and audited 58 packages in 4s

7 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
user@hostname:~/eclipseNode$

```

Import the project in Eclipse

Install nodeclipse package with NPM.

```
bash
```

```
$ sudo npm install -g nodeclipse
```

The package enables us to use a CLI with NPM, install third-party packages, and run multiple utility commands.

Input

```
bash
```

```
$ nodeclipse
```

Output

```
bash
```

```
user@hostname:~/projectDir$ nodeclipse
```

```
Usage: nodeclipse [arguments]
```

```
`nodeclipse --help install` for Nodeclipse CLI Installer Help
```

Arguments:

- c, --create <name> create project folder and prepare it
- u, --use <template> use/copy specified template when creating project
- p, --prepare prepare Nodeclipse [Node.js] project for import, i.e. add needed ``.project`` file and other ``.*/`` files (``.gitignore``, ``.jshinttrc``, ``.settings/``) if there is no ``.project`` yet
- g, --eclipse_project_general prepare General Eclipse project for import, i.e. add only needed ``.project`` file
- n, --name [<name>] project name (default is folder name)
- h, --help this help screen
- v, --version print nodeclipse CLI's version
- V, --verbose be verbose

Templates are just folders in this project sources:

hello-world	The famous hello world HTTP server in 6 lines
hello-coffee	The same server written in CoffeeScript
hello-typescript	The same server written in TypeScript
hello-html	Template with HTML file
template-gradle-java	Gradle Java project
template-maven-java	Maven Java project

Check `README.md` and sources at `/usr/lib/node_modules/nodeclipse` or <https://github.com/Nodeclipse/nodeclipse-1/tree/master/org.nodeclipse.ui/templates/>

For example, we can use the `-p` option to add a `.project` directory to the current working directory.

Input

```
bash
$ nodeclipse -p
```

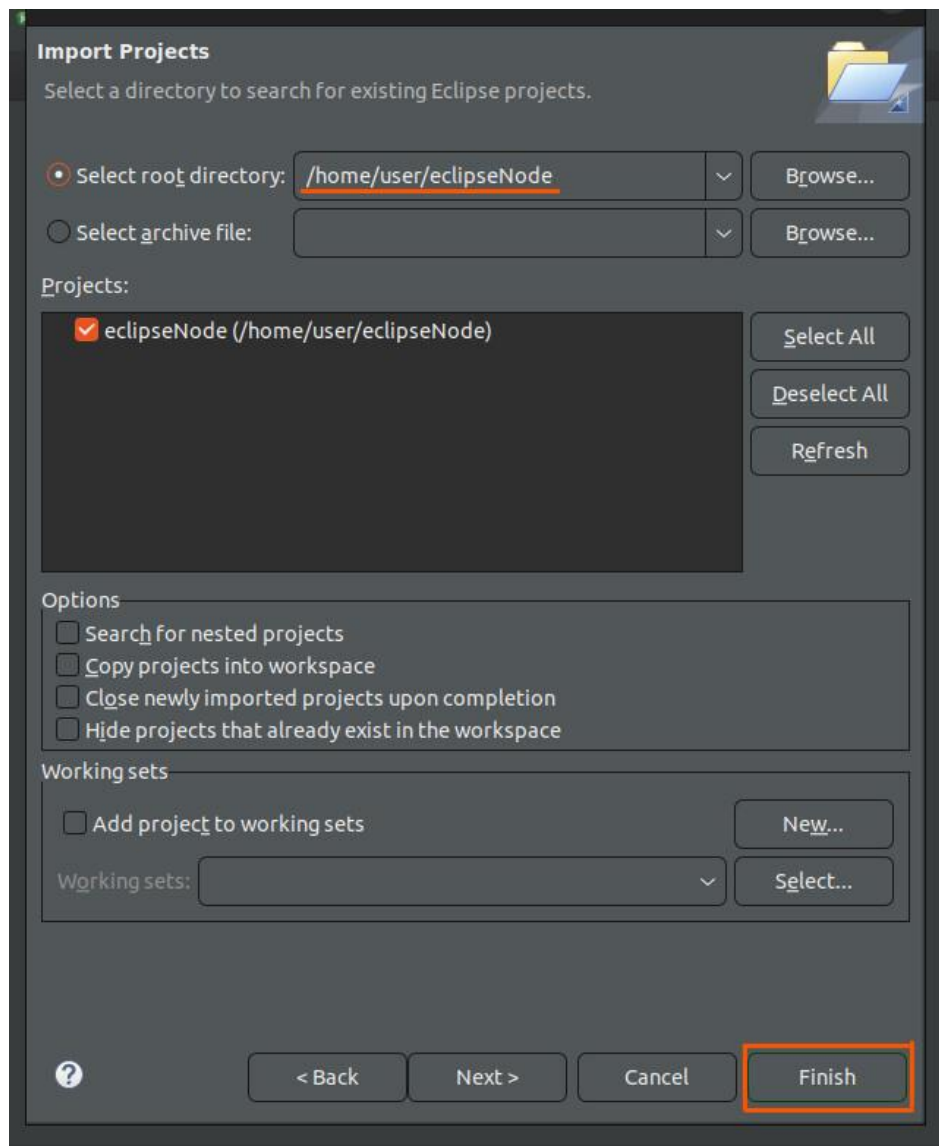
Output

```
bash
user@hostname:~/eclipseNode$ nodeclipse -p
In Eclipse/Enide select File -> Import... -> General / Existing Projects into Workspace
and enter project directory: /home/user/eclipseNode
```

Visit <http://www.nodeclipse.org/> for News, post Shares, Installing details, Features list, Usage (incl Video, Demo) with all shortcuts, Help and Hints, Support options, Where Helping needed, How to thank and Contact us, also History page.

Now launch the Eclipse IDE and import the project directory by following this path:

1. File → *Import* → *General* → *Existing Projects into Workspace*.
2. Click the *Next >* button and paste the project path into the search box.
3. Click on the *Finish* button.



Update the files with the following content.

package.json

```
json
{
  "name": "eclipsenode",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

We configure the application to use ES modules

```
json
```

```
"type": "module"
```

For example, we can import the express module in the index.js file.

index.js

```
javascript
```

```
import express from "express";
```

```
const app = express();
```

```
app.use(express.static("public"));
```

```
const PORT = process.env.PORT || 3000;
```

```
app.listen(PORT, console.log(`Make requests at http://localhost:${PORT}`));
```

We import express and create a web server that listens for requests on 3000. On making a request on localhost, we load static assets stored in the public folder.

public → *index.html*

```
xml
```

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
  <meta charset="UTF-8">
```

```
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
```

```
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
  <link rel="stylesheet" href="style.css">
```

```
  <title>How to use node.js with Eclipse IDE</title>
```

```
</head>
```

```
<body>
```

```
  <h1>How to use node.js with Eclipse IDE</h1>
```

```
  <h3>Learn how to use node.js with Eclipse IDE step-by-step</h3>
```

```
  <h4>by loading this page</h4>
```

```
  <h5>with an Express web server.</h5>
```

```
</body>
```

```
</html>
```

We create a page with four headings. The page is styled with style.css.

public → **style.css**

```
css
```

```
body {
```

```
  width: 70%;
```

```
  margin: 7% auto;
```

```
  text-align: center;
```

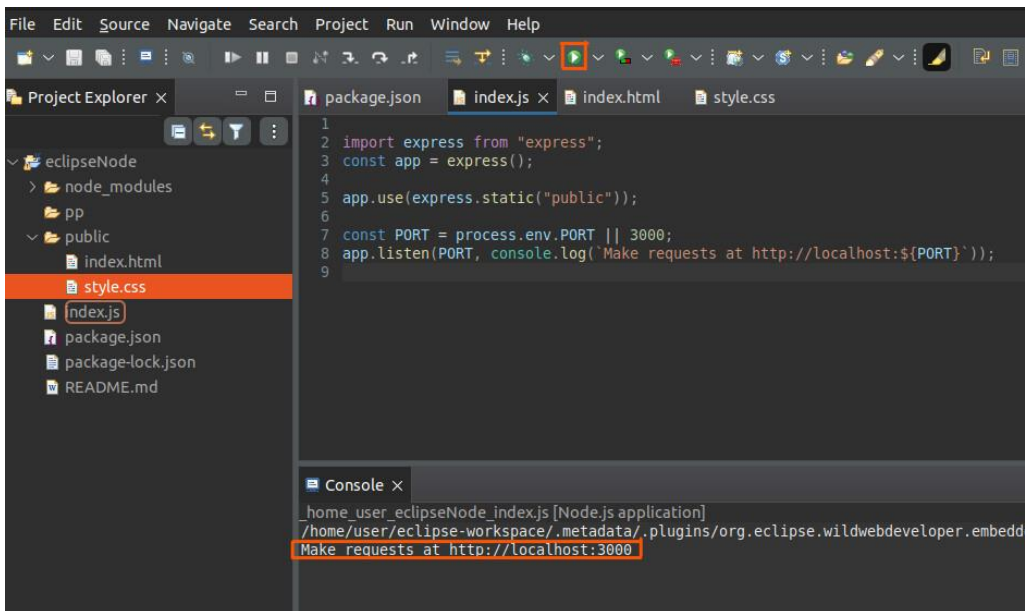
```
  background-color: black;
```

```
  color: gold;
```

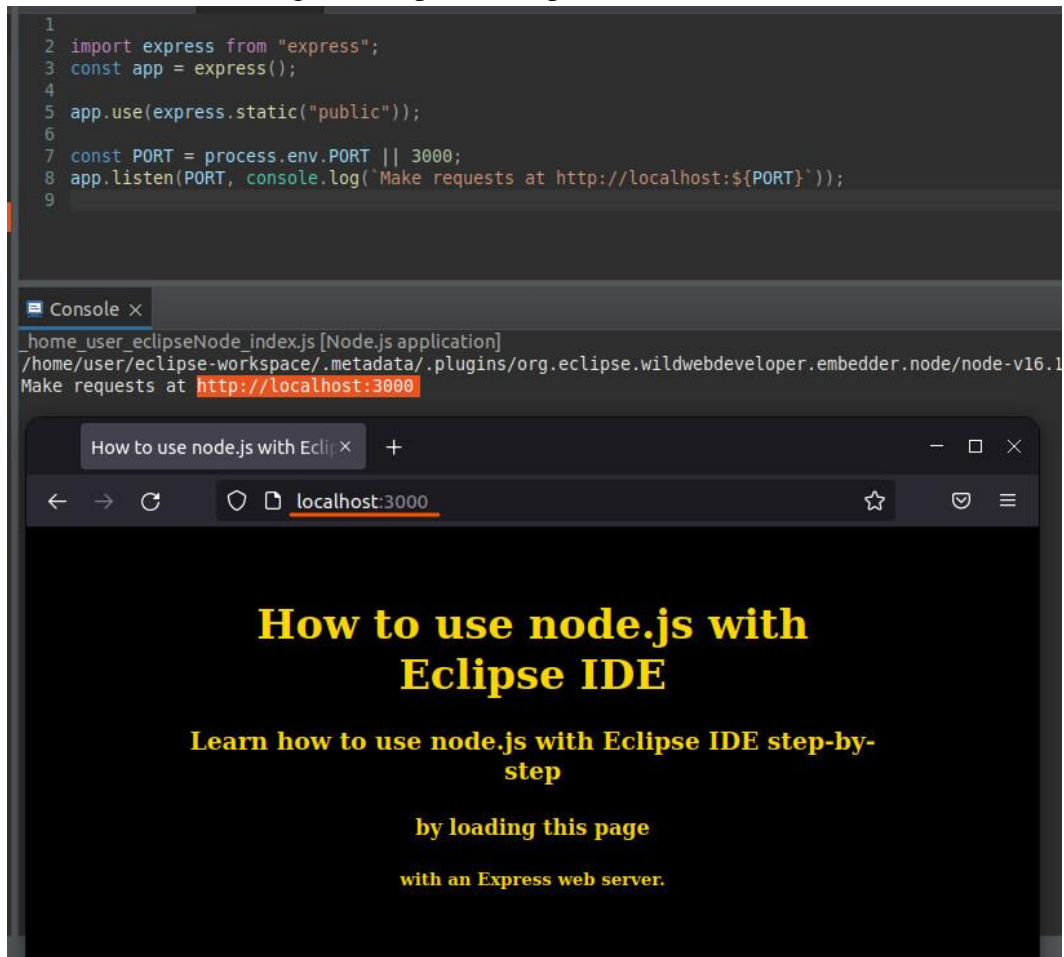
```
}
```

We center the golden text on the page.

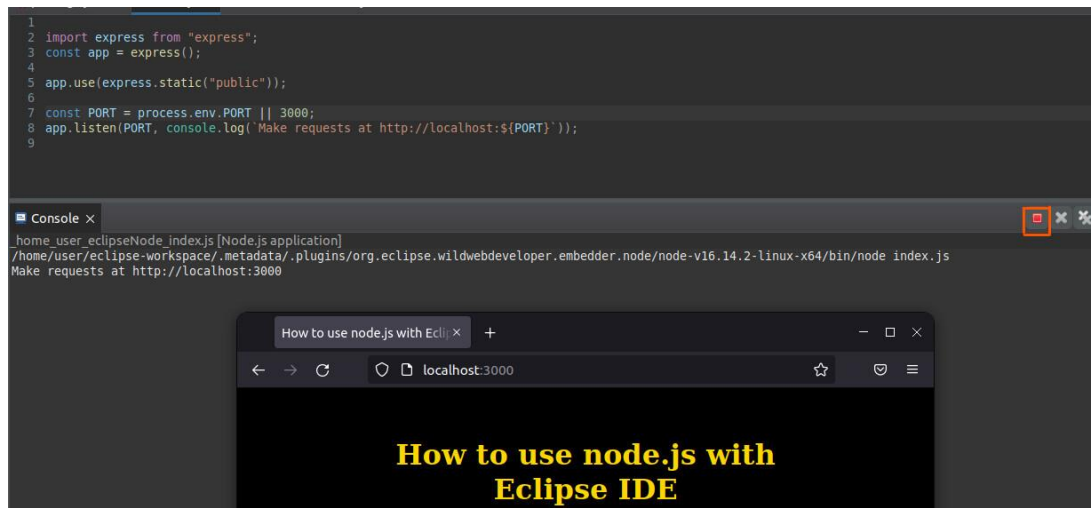
Now we can start the server by clicking on the green run button. The console output appears with the message *Make requests at <http://localhost:3000>*.



Open the URL with a browser. We get the expected output.



We can stop the server by clicking on the red button on the right-hand side of the console menu.



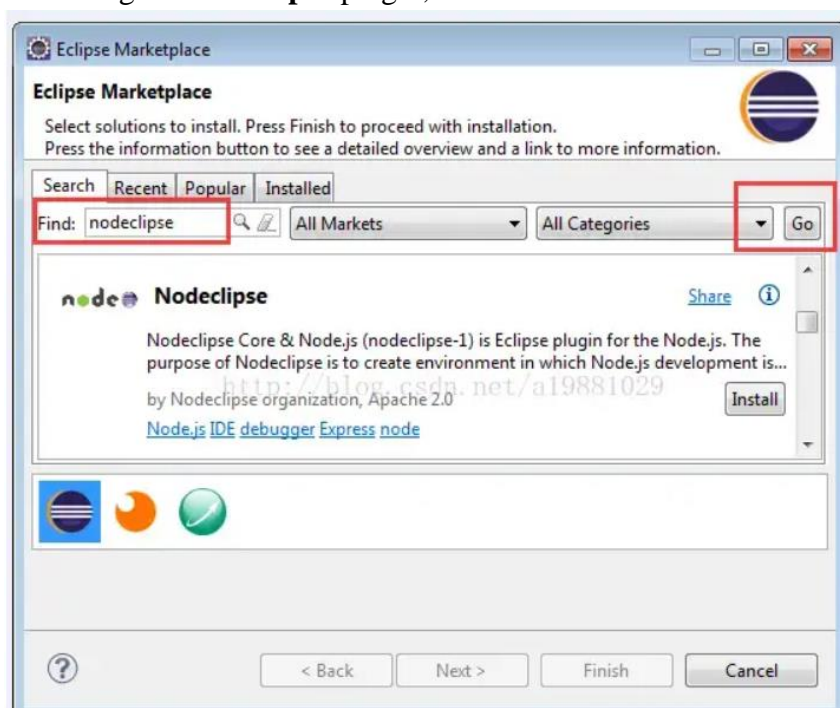
Node JS Hello World Example With Eclipse

This article will show you examples of how to use the **Nodeclipse** plugin to create and debug node js applications in eclipse.

1. Install Eclipse Node JS Plugin Nodeclipse.

Before you can create a Node.js application in eclipse, you should download and install the **Nodeclipse** plugin in your Eclipse. Eclipse version is eclipse-jee-neon-3-win32-x86_64. Nodeclipse plugin version is 1.0.2.

1. Open Eclipse, click **Help** —> **Eclipse Marketplace** menu in the top toolbar.
2. Input the keyword **nodeclipse** in the popup window search text box, click the **Go** button to search. When finding the **nodeclipse** plugin, click the **Install** button at its end to install it.

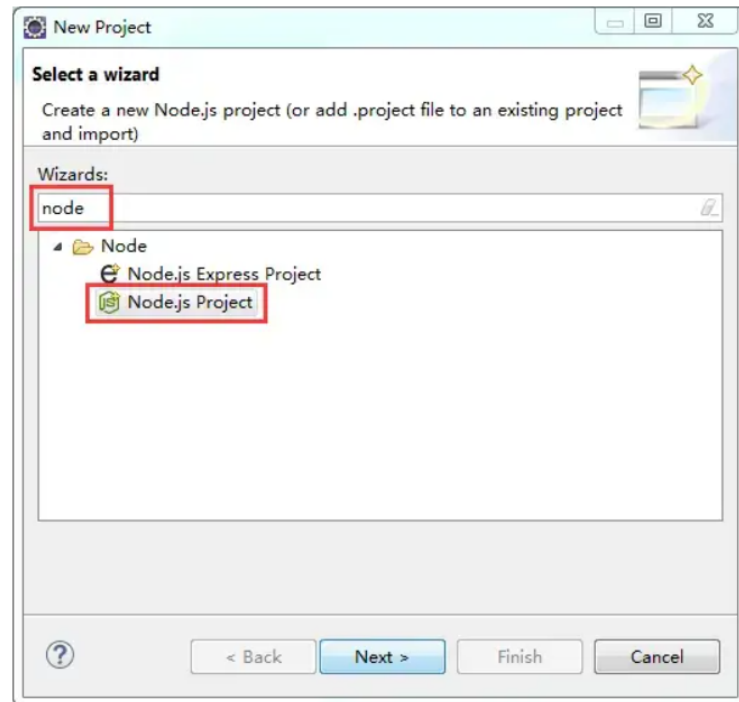


3. Check all **nodeclipse** features in the next dialog and click **Confirm** button.
4. Select the accept license agreement radio button in the next dialog.
5. Click the **Finish** button to complete the **nodeclipse** plugin installation.

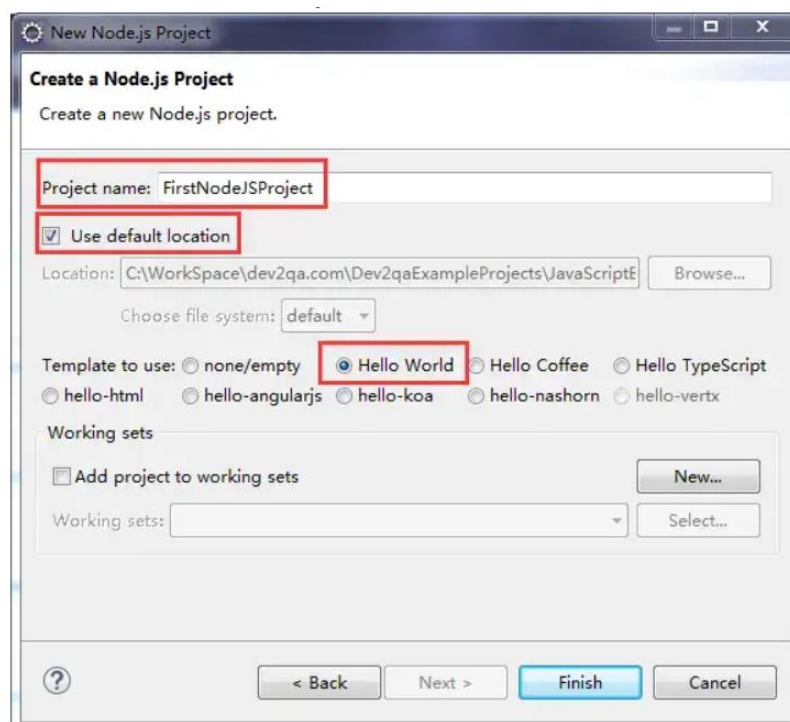
2. Create Eclipse Node JS Project.

Now you can use eclipse to create a Node.js project follow the below steps.

1. Click the eclipse **File** —> **New** —> **Project** menu item. Input the keyword **node** in Wizards search box, then selects **Node.js Project**.



2. In the next dialog, input the project name and select the default workspace. Select the **Hello World** radio button in **Template to use** section.

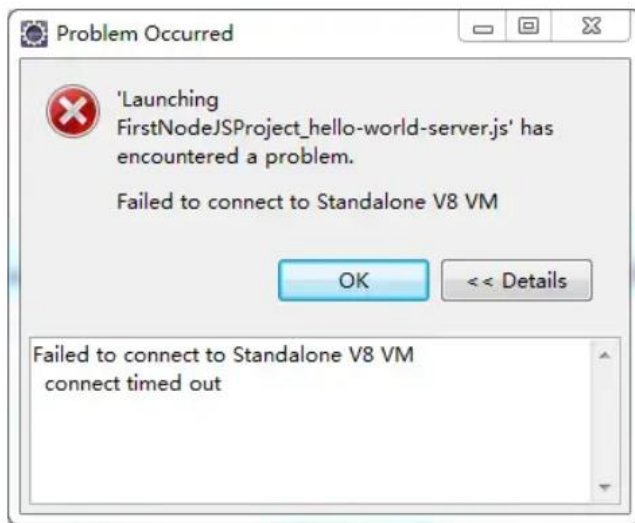


3. Click the **Finish** button to complete the eclipse node js project creation. Then you can see the wizard added files in the left project panel. Click the **hello-world-server.js** file in the file list, it just implements a simple HTTP web server.

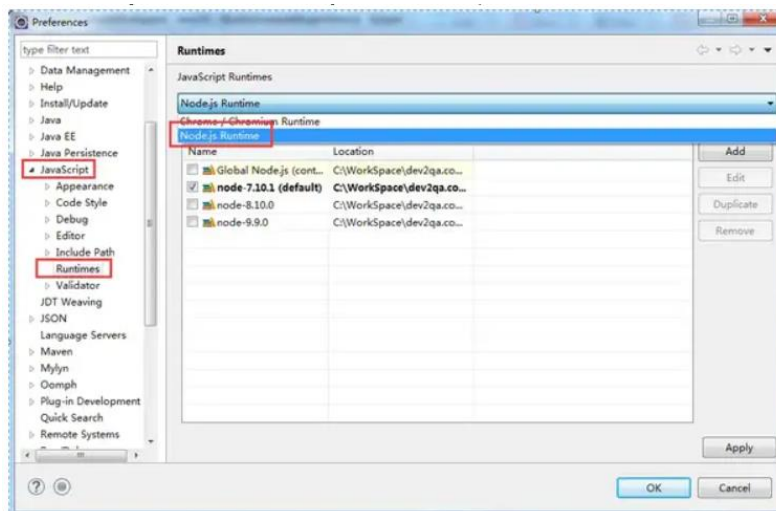
3. Debug Eclipse Node JS Project.

1. Double click the first column in the **hello-world-server.js** file in line 3, then it will set a breakpoint there.
2. Right-click the js file, click **Debug As** —> **Node.js Application** menu in the popup menu list. Then the debug process will start. If you meet the below error.

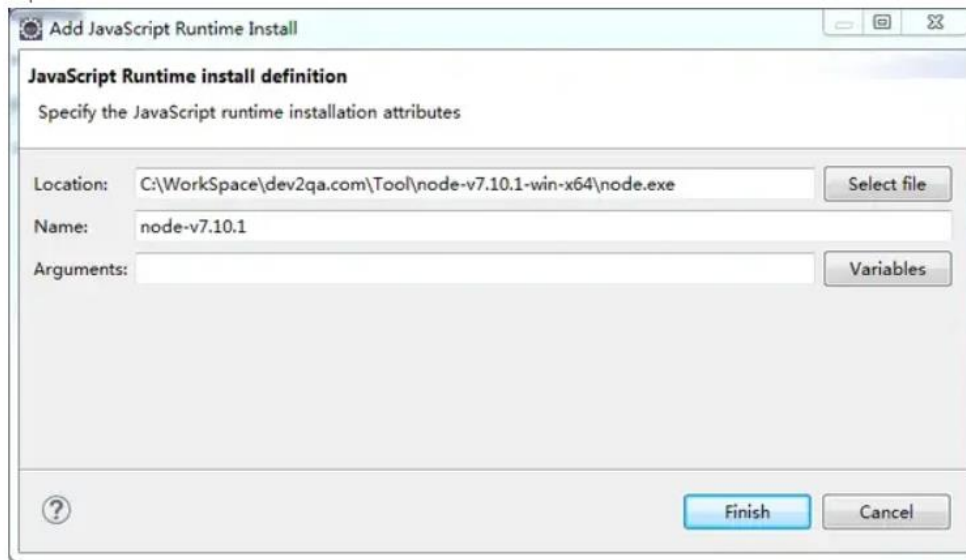
Failed to connect to Standalone V8 VM
connect timed out



3. That is because you use Node.js version 8 or 9 as node runtime. Those two versions are not stable, so you need to follow the below steps to fix the above error.
4. Download and install Node.js version 7.10.1, please read the article [How To Install Node JS In Windows](#) for detail.
5. Change node app runtime by click eclipse **Window** —> **Preferences** menu in the top toolbar.
6. In the popup window left panel, click **JavaScript** —> **Runtimes** menu. In the right panel choose **Node.js Runtime** in **JavaScript Runtimes** drop-down list box.



7. Click **Add** button to open **Add JavaScript Runtime Install** popup window.
8. Select the **node.exe** file in the **Location** input box, give the runtime a name in the **Name** input box.



9. Click the **Finish** button to complete the settings and return to the node runtime list dialog. Check the checkbox before the created runtime and click **OK**.
10. Now you can right-click the js file, click **Debug As** —> **Node.js Application** menu in the popup menu list. You will find the execution stopped at the first line of the js file. Then you can click debug button at the top toolbar to debug the js files.

Node.js CRUD Operations Using Mongoose and MongoDB Atlas

- Before we dive into the major operations and functionality of Mongoose, let us get a brief idea about what it actually is and how it makes our Node.js project more flexible and user-friendly.
- **MongooseJs:** Mongoose is basically a package that serves as a mediator between the NodeJS application and the MongoDB server. It is an Object Document Mapper(ODM) that allows us to define objects with a strongly-typed schema that is mapped to a MongoDB document. Mongoose supports all the CRUD operations – Creating, Retrieving, Updating, and Deleting.
- **Prerequisites:** Since we will be using Express to set up our basic server, We would recommend going through some articles on express and official express documents. Other requirements include MongoDB Atlas and Postman.
- **Installation:** Install the Mongoose and the express module through npm using the below command:
npm install express mongoose --save
- **MongoDB Atlas Setup:**
 - Setup an account.
 - Build a new cluster.
 - Go to Database Access and hit “Add New User”. Add a username and password, if you autogenerate a password make sure you copy it, we’ll need it later.
 - Whitelist your IP Address.Hit “Add Current IP address” and Confirm.
 - Go to Clusters, if your cluster building is done then hit Connect, “Connect Your Application”, and copy the URL it gives you.
- **Postman Setup:** We will be using Postman to manage our requests. Once it is downloaded, hit the “Create a request” option. Every time we make a new API endpoint we’ll be setting up another request for it. This will help you manage everything so you don’t have to copy/paste HTTP requests everywhere.
- **Server Setup:** Here, we’ll set up our server on port 3000 and call the express function that returns a server object in a variable named app. Then we start the listener saying app.listen with the port address. Finally, we create the /api route which will be triggered once request localhost:3000/api is received from the browser.

File Name : Server.js

```
const express = require('express');
const bodyParser = require('body-parser');
const api = require('./api');

const port = 3000;
const app = express();

app.listen(port, function () {
  console.log("Server is listening at port:" + port);
});

// Parses the text as url encoded data
app.use(bodyParser.urlencoded({ extended: true }));

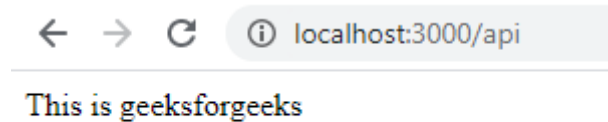
// Parses the text as json
app.use(bodyParser.json());

app.use('/api', api);
```



```
C:\Project>node server.js
Server is listening at port:3000
```

Server running on desired port



Sample output to check working of api route

Schema: Schema is a representation of the structure of the data. It allows us to decide exactly what data we want, and what options we want the data to have as an object.

Filename : studentschema.js

```
const mongoose = require('mongoose');

const StudentSchema = new mongoose.Schema({
  StudentId: Number,
  Name: String,
  Roll: Number,
  Birthday: Date,
  Address: String
});

module.exports = mongoose.model(
  'student', StudentSchema, 'Students');
```

A schema named “StudentSchema” is created that accepts the fields Id, Name, Roll, Birthday, and Address. Models basically provide a list of predefined methods that are used to manipulate the data for inserting, updating, deleting, and retrieving from the database collection.

With that basic pattern, we’ll use the mongoose.model method to make it usable with actual data and export it so that we can use it in api.js.

Advanced Routing and MongoDB Connections:

Filename : api.js When you make a request to localhost:3000/api, express will search for api route and execute the api.js file.

```
const mongoose = require('mongoose');
const express = require('express');
const router = express.Router();
const StudentModel = require('./studentschema');

// Connecting to database
const query = 'mongodb+srv://Username:<password>'
  + '@student.tuufn.mongodb.net/College?'
  + 'retryWrites=true&w=majority'
```

```

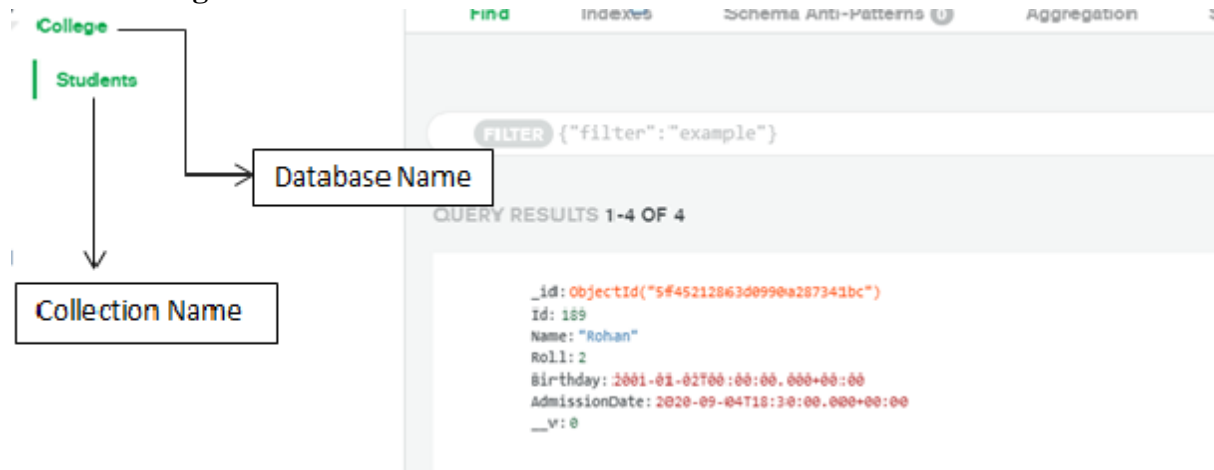
const db = (query);
mongoose.Promise = global.Promise;

mongoose.connect(db, {
  useNewUrlParser: true,
  useUnifiedTopology: true
}, function (error) {
  if (error) {
    console.log("Error!" + error);
  }
});

module.exports = router;

```

The database is **College** and the collection inside the database is **Students**.



A Glimpse of the Mongo Database

CRUD OPERATIONS

- **Create:** We'll be setting up a post request to '/save' and we'll create a new student object with our model and pass with it the request data from Postman.

Once this is done, we will use `.save()` to save it to the database.

```

router.get('/save', function (req, res) {
  const newStudent = new StudentModel({
    StudentId: 101,
    Name: "Sam", Roll: 1, Birthday: 2001 - 09 - 08
  });

  newStudent.save(function (err, data) {
    if (err) {
      console.log(error);
    }
    else {
      res.send("Data inserted");
    }
  });
});

```

A new instance of the student is created using StudentModel and the reference is stored in the variable newStudent. Using the newStudent variable we save the document of the new student to the database collection.

For achieving this, in Postman we will make a GET request localhost:3000/api/save

Note: We can even insert new documents without hardcoding the fields as done above. For that, we need to change the request from GET to POST and use the body-parser middleware to accept the new student's data. This ensures that we can insert details of as many students as we need.

```
router.post('/save', function (req, res) {
  const newStudent = new StudentModel();
  newStudent.StudentId = req.body.StudentId;
  newStudent.Name = req.body.Name;
  newStudent.Roll = req.body.Roll;
  newStudent.Birthday = req.body.Birthday;

  newStudent.save(function (err, data) {
    if (err) {
      console.log(error);
    }
    else {
      res.send("Data inserted");
    }
  });
});
```

- **Retrieve:** To retrieve records from a database collection we make use of the .find() function.

```
router.get('/findall', function (req, res) {
  StudentModel.find(function (err, data) {
    if (err) {
      console.log(err);
    }
    else {
      res.send(data);
    }
  });
});
```

In Postman, we make a new GET request with the URL localhost:3000/api/findall and hit send. It makes our HTTP GET request and returns documents of all the students from our database collection.

- To retrieve a single record or the first matched document we make use of the function findOne().

```
router.get('/findfirst', function (req, res) {
  StudentModel.findOne({ StudentId: { $gt: 185 } },
    function (err, data) {
      if (err) {
        console.log(err);
      }
      else {
        res.send(data);
      }
    });
});
```

In Postman, we make a new GET request with the URL localhost:3000/api/findfirst and hit send. It makes our HTTP GET request and returns the first document that matches the condition **StudentId:\$gt:185** (\$gt means greater than).

- **Delete:** To delete a record from the database, we make use of the function `.remove()`. It accepts a condition that is the parameter according to which it performs deletion. Here the condition is `Id:188`.

```
router.get('/delete', function (req, res) {
  StudentModel.remove({ StudentId: 188 },
    function (err, data) {
      if (err) {
        console.log(err);
      }
      else {
        res.send(data);
      }
    });
});
```

- We can also use the `.findByIdAndDelete()` method to easily remove a record from the database. Every object created with Mongoose is given its own `_id`, and we can use this to target specific items with a DELETE request.

```
router.post('/delete', function (req, res) {
  StudentModel.findByIdAndDelete((req.body.id),
    function (err, data) {
      if (err) {
        console.log(err);
      }
      else {
        res.send(data);
        console.log("Data Deleted!");
      }
    });
});
```

- **Update:** Just like with the delete request, we'll be using the `_id` to target the correct item. `.findByIdAndUpdate()` takes the target's id, and the request data you want to replace it with.

```
router.post('/update', function (req, res) {
  StudentModel.findByIdAndUpdate(req.body.id,
    { Name: req.body.Name }, function (err, data) {
      if (err) {
        console.log(err);
      }
      else {
        res.send(data);
        console.log("Data updated!");
      }
    });
});
```

How to retrieve the latest record from database collection: To retrieve the latest record we need two basic functions:

- `.sort()` – It accepts a parameter according to which it sorts the data in descending (-1) or ascending(1) order.
- `.limit()` – It decides the number of documents needed to be retrieved.

Example: Suppose I want to fetch the record of the student who has most recently taken admitted to the College. The following code snippet does this job for us.

```
99  //retrieving the latest record
100 router.get('/latest',function(req,res){
101     StudentModel.find({}).sort({AdmissionDate : -1}).limit(1).exec(function(err,data){
102         if(err)
103         {
104             console.log(err);
105         }
106         else
107         {
108             res.send(data);
109         }
110     });
111 });
```

Code Snippet that retrieves the latest data

NOTE: `limit()` should not be used without `.sort()` as it may cause bugs later that are difficult to track down. This is because we can't otherwise guarantee the order of the result. We would get different records at the top of the results which isn't desirable. To make a query deterministic, they must give the same results every time they are executed.

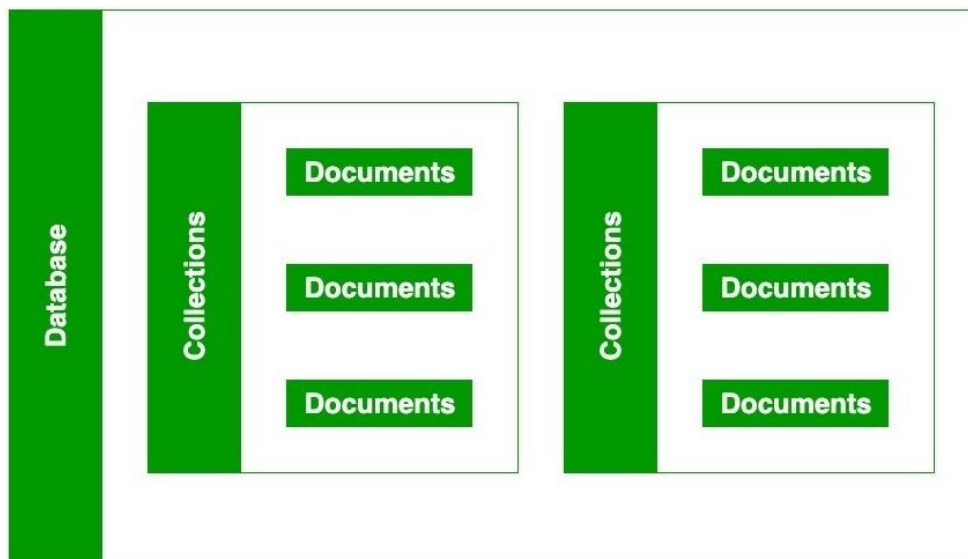
What is MongoDB – Working and Features

Introduction

- MongoDB is an open-source document-oriented database that is designed to store a large scale of data and also allows you to work with that data very efficiently.
- It is categorized under the NoSQL (Not only SQL) database because the storage and retrieval of data in the MongoDB are not in the form of tables.
- The MongoDB database is developed and managed by MongoDB.Inc under SSPL(Server Side Public License) and initially released in February 2009.
- It also provides official driver support for all the popular languages like C, C++, C#, and .Net, Go, Java, Node.js, Perl, PHP, Python, Motor, Ruby, Scala, Swift, Mongoid.
- So, that you can create an application using any of these languages.
- Nowadays there are so many companies that used MongoDB like Facebook, Nokia, eBay, Adobe, Google, etc. to store their large amount of data.

Working of MongoDB

- Now, we will see how actually thing happens behind the scene.
- As we know that MongoDB is a database server and the data is stored in these databases.
- Or in other words, MongoDB environment gives you a server that you can start and then create multiple databases on it using MongoDB.
- Because of its NoSQL database, the data is stored in the collections and documents.
- Hence the database, collection, and documents are related to each other as shown below:

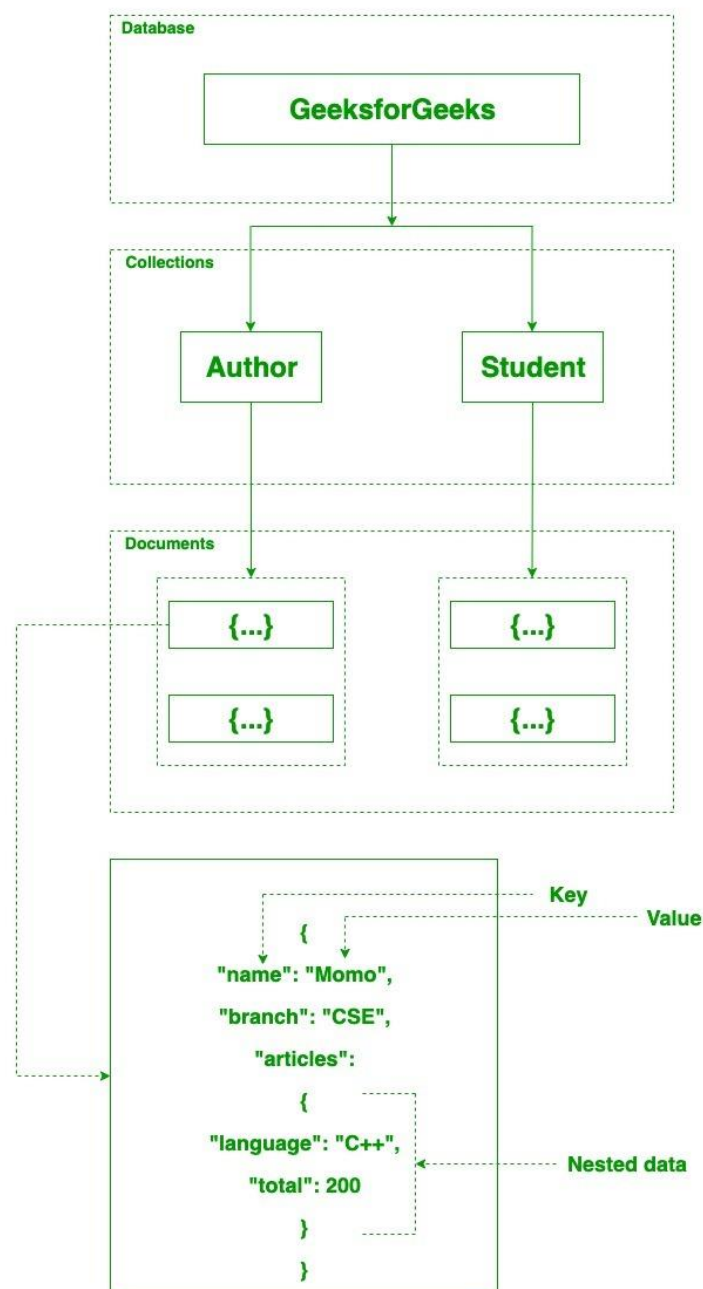


- The MongoDB database contains collections just like the MYSQL database contains tables. You are allowed to create multiple databases and multiple collections.
- Now inside of the collection we have documents. These documents contain the data we want to store in the MongoDB database and a single collection can contain multiple documents and you are schema-less means it is not necessary that one document is similar to another.
- The documents are created using the fields. Fields are key-value pairs in the documents, it is just like columns in the relation database. The value of the fields can be of any BSON data types like double, string, boolean, etc.

- The data stored in the MongoDB is in the format of BSON documents. Here, BSON stands for Binary representation of JSON documents. Or in other words, in the backend, the MongoDB server converts the JSON data into a binary form that is known as BSON and this BSON is stored and queried more efficiently.
- In MongoDB documents, you are allowed to store nested data. This nesting of data allows you to create complex relations between data and store them in the same document which makes the working and fetching of data extremely efficient as compared to SQL. In SQL, you need to write complex joins to get the data from table 1 and table 2. The maximum size of the BSON document is 16MB.

NOTE: In MongoDB server, you are allowed to run multiple databases.

For example, we have a database named GeeksforGeeks. Inside this database, we have two collections and in these collections we have two documents. And in these documents we store our data in the form of fields. As shown in the below image:



Difference Between mongoDB and RDBMS

Some major differences in between MongoDB and the RDBMS are as follows:

MongoDB	RDBMS
It is a non-relational and document-oriented database.	It is a relational database.
It is suitable for hierarchical data storage.	It is not suitable for hierarchical data storage.
It has a dynamic schema.	It has a predefined schema.
It centers around the CAP theorem (Consistency, Availability, and Partition tolerance).	It centers around ACID properties (Atomicity, Consistency, Isolation, and Durability).
In terms of performance, it is much faster than RDBMS.	In terms of performance, it is slower than MongoDB.

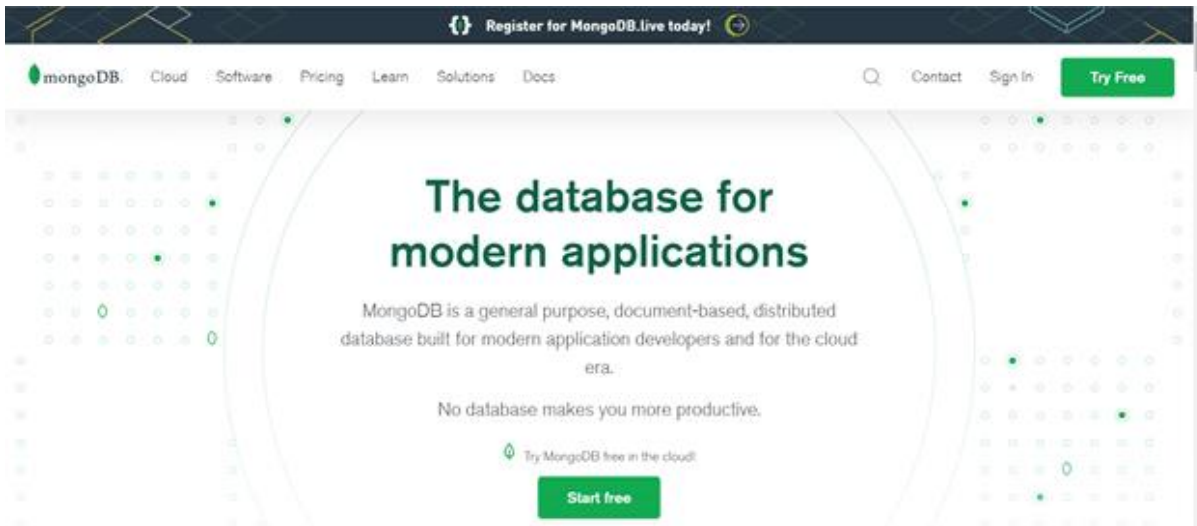
Features of MongoDB –

- **Schema-less Database:** It is the great feature provided by the MongoDB. A Schema-less database means one collection can hold different types of documents in it. Or in other words, in the MongoDB database, a single collection can hold multiple documents and these documents may consist of the different numbers of fields, content, and size. It is not necessary that the one document is similar to another document like in the relational databases. Due to this cool feature, MongoDB provides great flexibility to databases.
- **Document Oriented:** In MongoDB, all the data stored in the documents instead of tables like in RDBMS. In these documents, the data is stored in fields(key-value pair) instead of rows and columns which make the data much more flexible in comparison to RDBMS. And each document contains its unique object id.
- **Indexing:** In MongoDB database, every field in the documents is indexed with primary and secondary indices this makes easier and takes less time to get or search data from the pool of the data. If the data is not indexed, then database search each document with the specified query which takes lots of time and not so efficient.
- **Scalability:** MongoDB provides horizontal scalability with the help of sharding. Sharding means to distribute data on multiple servers, here a large amount of data is partitioned into data chunks using the shard key, and these data chunks are evenly distributed across shards that reside across many physical servers. It will also add new machines to a running database.
- **Replication:** MongoDB provides high availability and redundancy with the help of replication, it creates multiple copies of the data and sends these copies to a different server so that if one server fails, then the data is retrieved from another server.
- **Aggregation:** It allows to perform operations on the grouped data and get a single result or computed result. It is similar to the SQL GROUPBY clause. It provides three different aggregations i.e, aggregation pipeline, map-reduce function, and single-purpose aggregation methods
- **High Performance:** The performance of MongoDB is very high and data persistence as compared to another database due to its features like scalability, indexing, replication, etc.

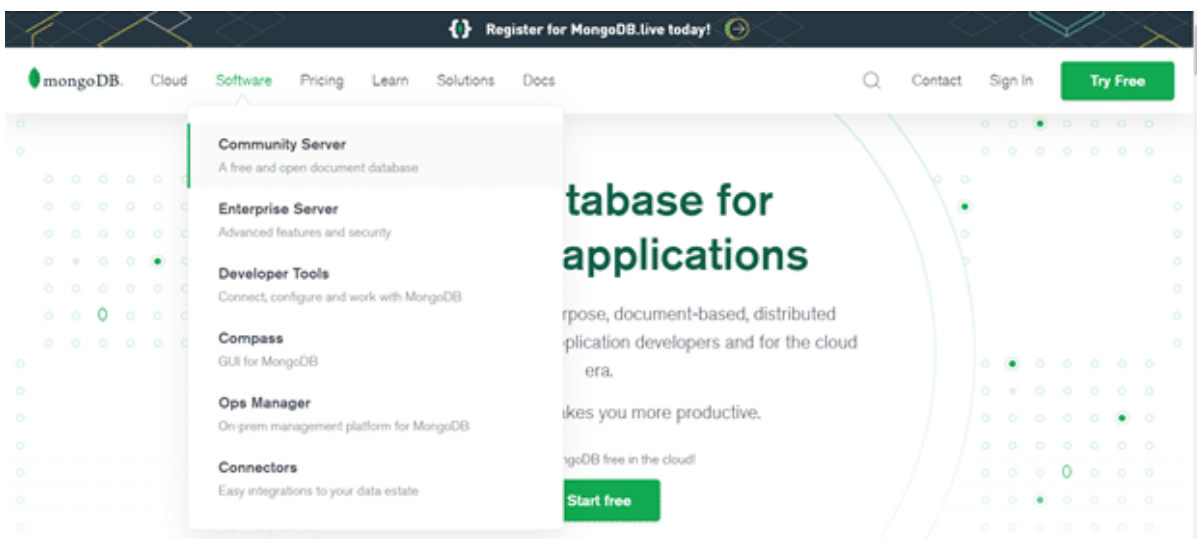
Installing MongoDB on Windows 10

Now, let's see the installation process of MongoDB on Windows 10. Follow the steps mentioned below:

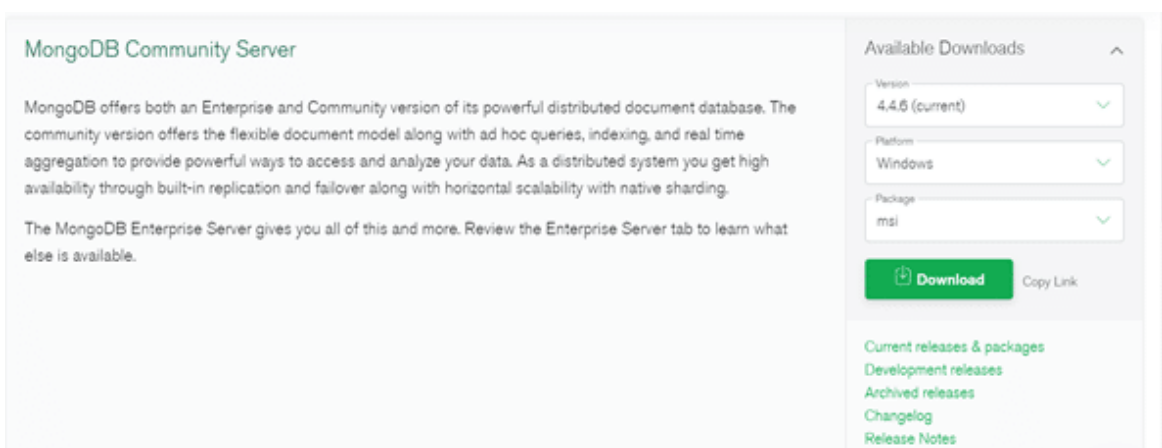
Step 1: Go to the MongoDB official page. Then click on the **software** on your top left.



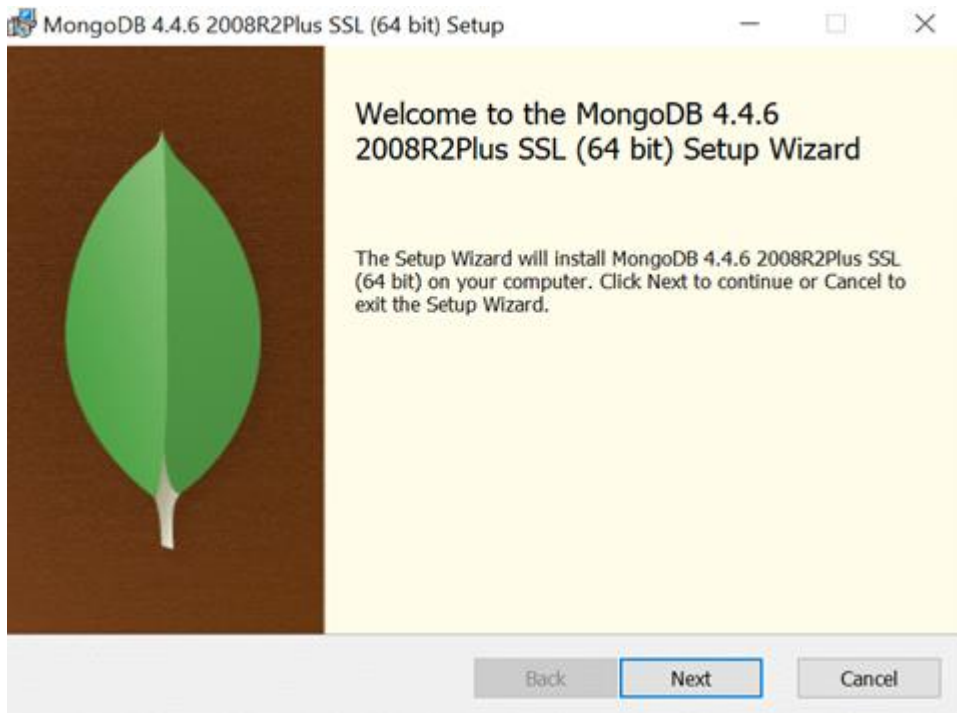
Step 2: From the software, click on the **community server** option.



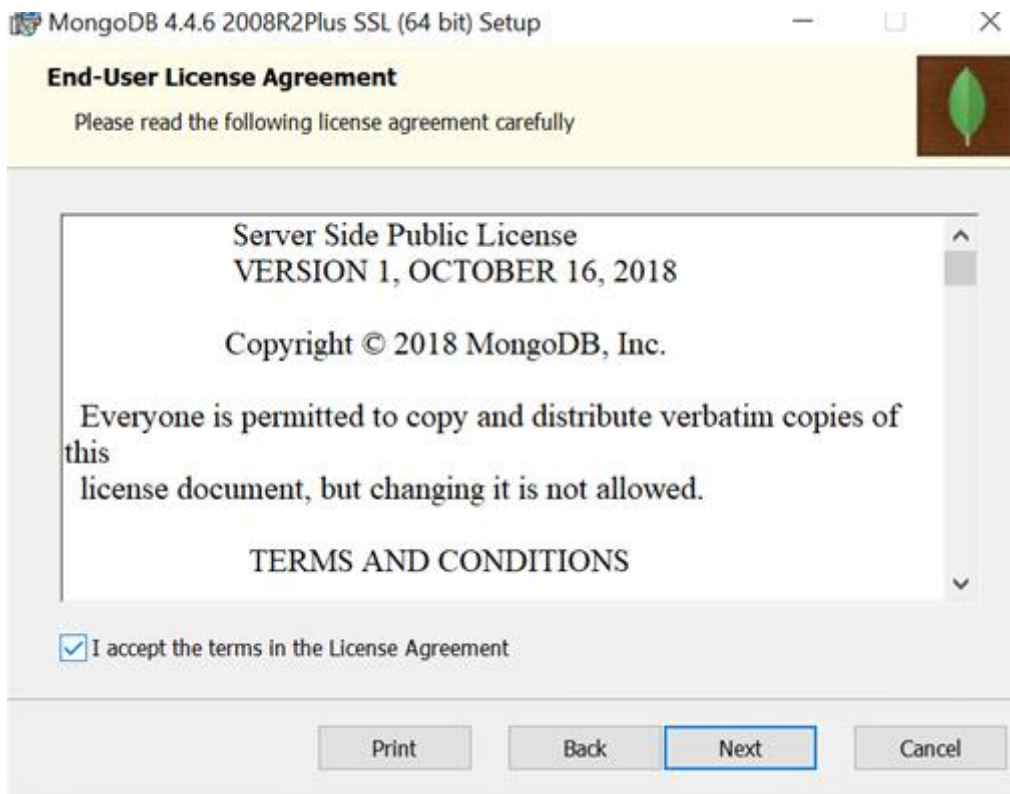
Step 3: Then you will see this MongoDB Community Server which is 4.4.6 version and msi package. Click on the download button to **Download** the software.



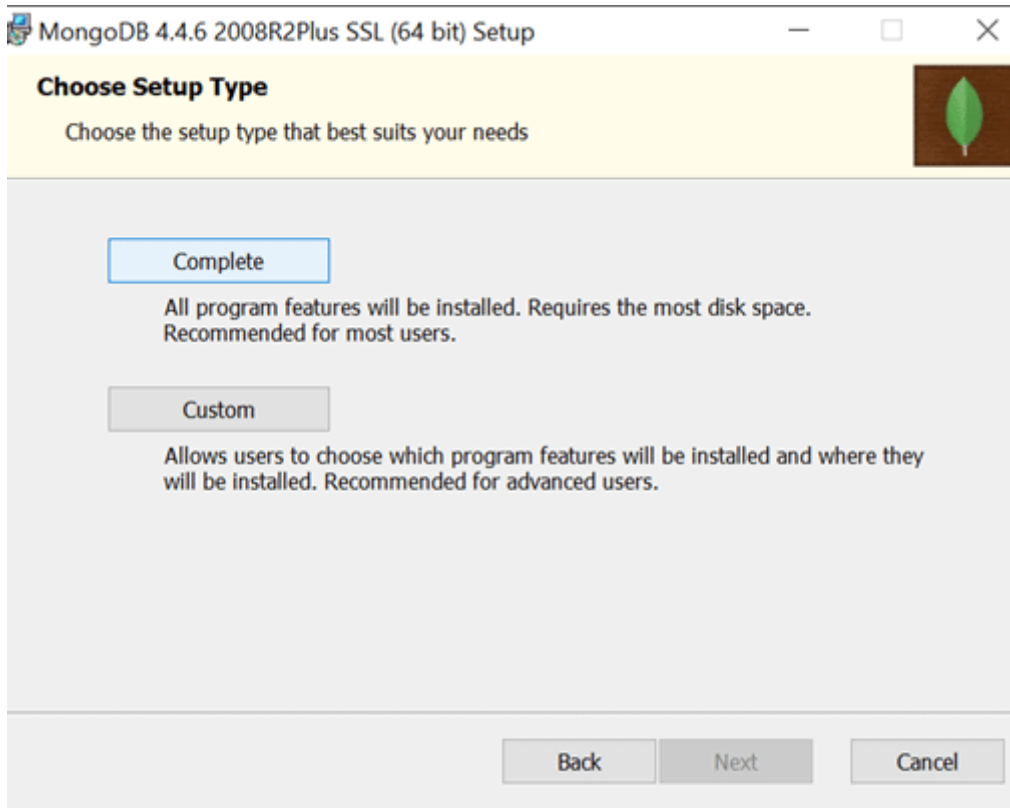
Step 4: After completing of downloading, open the downloaded .msi file. You will see something like the below image. Click on **Next** to continue.



Step 5: Then **Accept** the license agreement and click on **Next**.



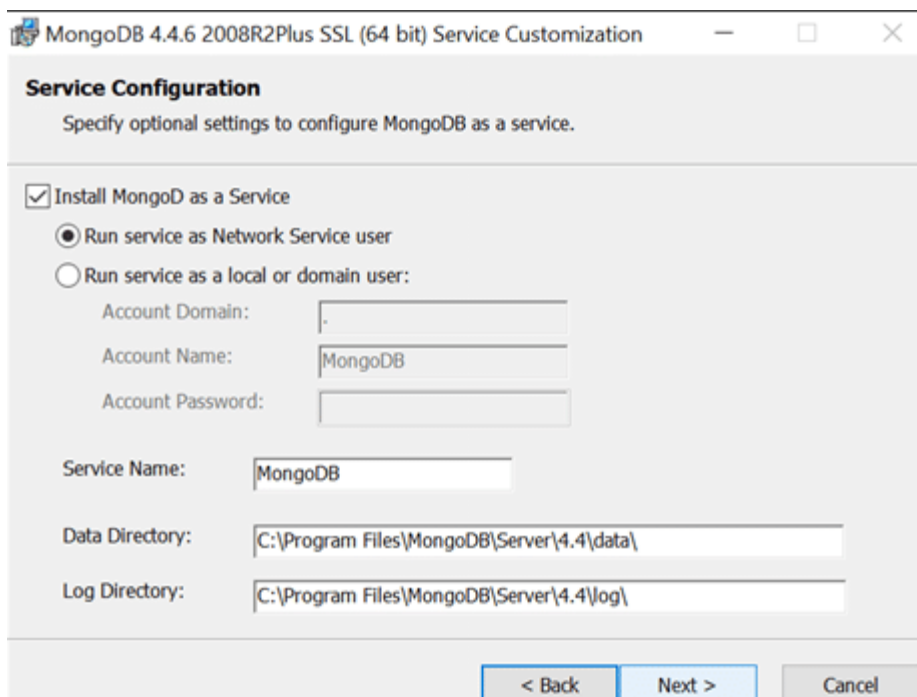
Step 6: Then you have to choose the setup type. Click on the **Complete** option



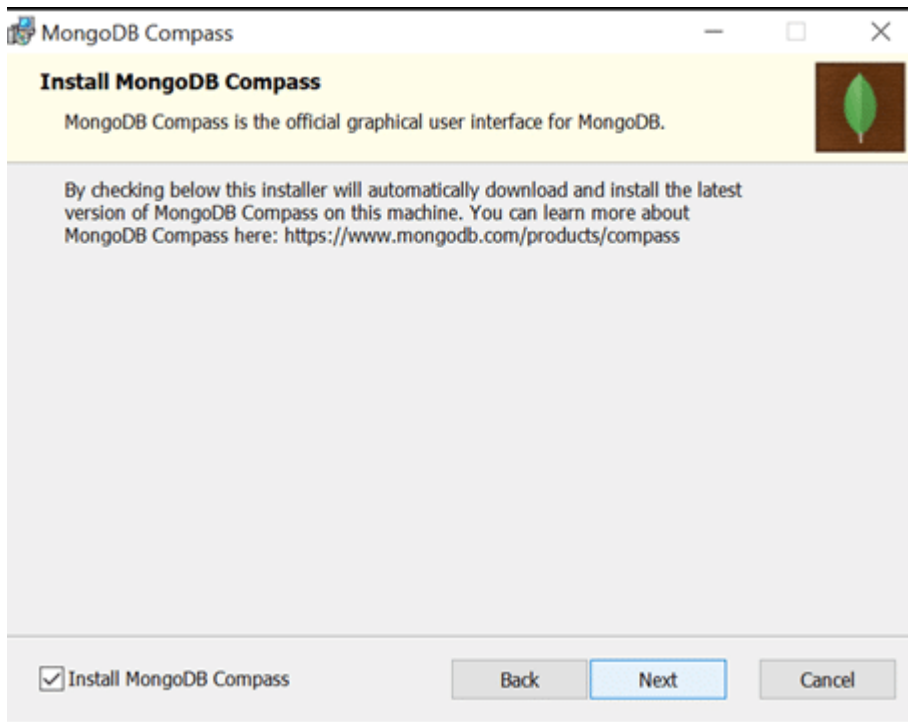
Step 7: Then you will be asked to specify optional settings to configure MongoDB as a service. To specify options follow the steps below:

- Install MongoDB as a service
- Run service as a Network service user.

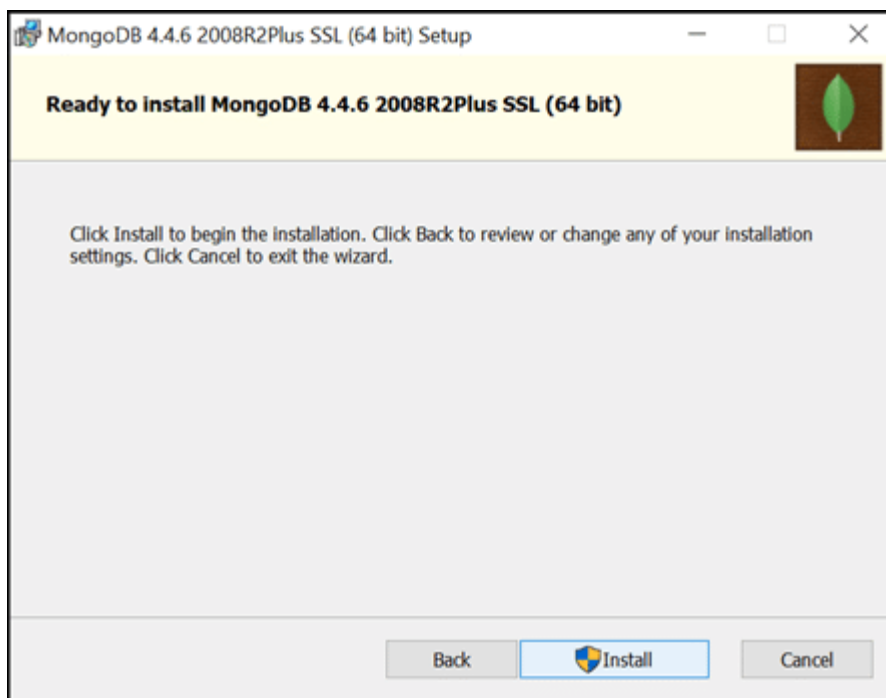
Then click on **Next** to continue.



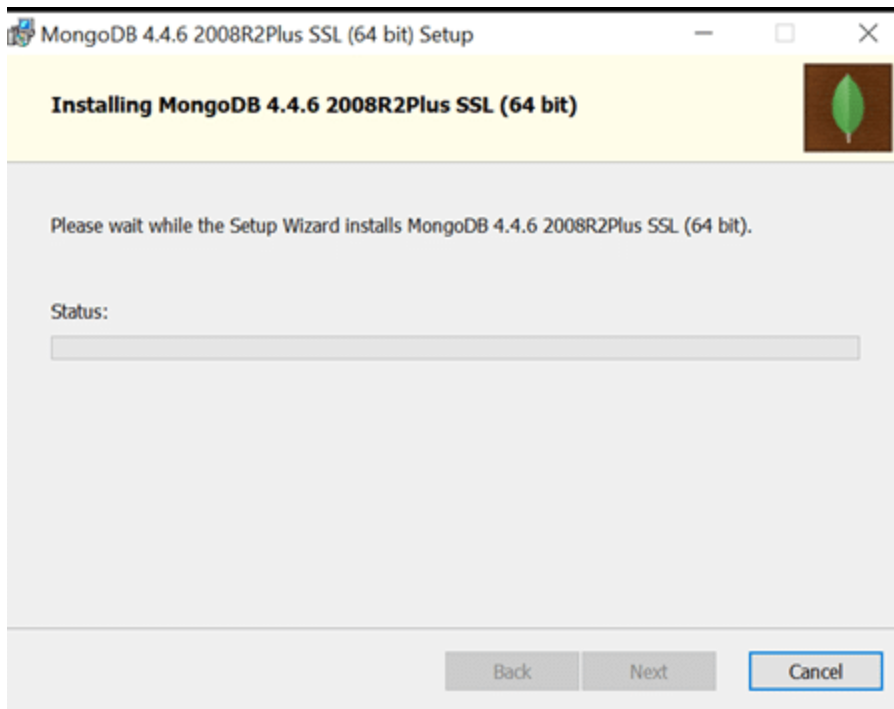
Step 8: Next, you will see an option to install MongoDB compass which is optional. You can keep it selected if you want or deselect it if you don't. Click on **Next**.



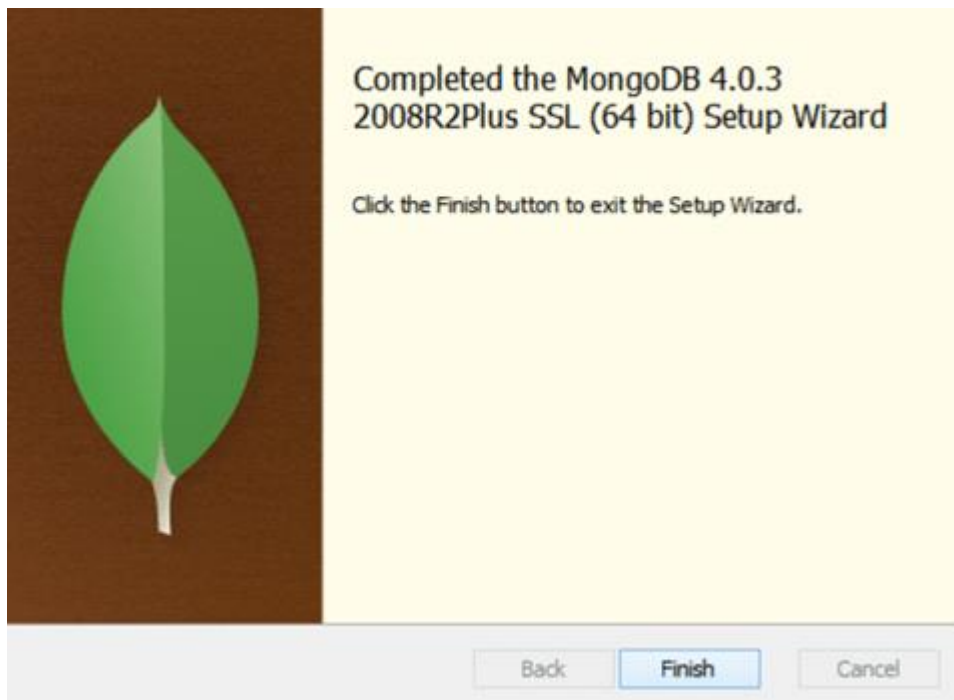
Step 9: From here you can install the software. Click on **Install** to start the installation process.



The installation process starts.



Step 10: Finally, the installation process is completed. Click on **Finish**.



This is the installation process of MongoDB. Now let's talk about the scope of MongoDB NoSQL.

Installation of MongoDB on Ubuntu

MongoDB can be installed on Ubuntu with the use of the following commands. These commands are easy to run on the terminal and make the installation process handy. Follow the steps given below to install MongoDB:

Step 1: First you need to update and upgrade your system repository in order to install MongoDB. Type the following command in your terminal and then press Enter.

```
$ sudo apt update && sudo apt upgrade
```

```
rishabh@rishabh-Lenovo-V130-15IKB: ~  
File Edit View Search Terminal Help  
rishabh@rishabh-Lenovo-V130-15IKB:~$ sudo apt update && sudo apt upgrade  
[sudo] password for rishabh:  
Hit:1 http://in.archive.ubuntu.com/ubuntu bionic InRelease  
Hit:2 http://in.archive.ubuntu.com/ubuntu bionic-updates InRelease  
Hit:3 http://in.archive.ubuntu.com/ubuntu bionic-backports InRelease  
Ign:4 http://dl.google.com/linux/chrome/deb stable InRelease  
Hit:5 http://ppa.launchpad.net/obsproject/obs-studio/ubuntu bionic InRelease  
Hit:6 http://dl.google.com/linux/chrome/deb stable Release  
Get:7 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]  
Fetched 88.7 kB in 2s (57.5 kB/s)  
Reading package lists... 32%
```

Step 2: Now, install the MongoDB package using ‘**apt**’. Type the following command and press Enter.

\$ sudo apt install -y mongodb

```
rishabh@rishabh-Lenovo-V130-15IKB: ~  
rishabh@rishabh-Lenovo-V130-15IKB:~$ sudo apt install -y mongodb  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following packages were automatically installed and are no longer required:  
  efibootmgr libfwupd  
Use 'sudo apt autoremove' to remove them.  
The following additional packages will be installed:  
  libboost-program-options1.65.1 libgoogle-perftools4 libpcrecpp0v5  
  libtcmalloc-minimal4 libyaml-cpp0.5v5 mongo-tools mongodb-clients  
  mongodb-server mongodb-server-core  
The following NEW packages will be installed:  
  libboost-program-options1.65.1 libgoogle-perftools4 libpcrecpp0v5  
  libtcmalloc-minimal4 libyaml-cpp0.5v5 mongo-tools mongodb mongodb-clients  
  mongodb-server mongodb-server-core  
0 upgraded, 10 newly installed, 0 to remove and 0 not upgraded.  
Need to get 53.4 MB of archives.  
After this operation, 217 MB of additional disk space will be used.  
Get:1 http://in.archive.ubuntu.com/ubuntu bionic/main amd64 libboost-program-opti  
ons1.65.1 amd64 1.65.1+dfsg-0ubuntu5 [137 kB]  
Get:2 http://in.archive.ubuntu.com/ubuntu bionic/main amd64 libtcmalloc-minimal4  
amd64 2.5-2.2ubuntu3 [91.6 kB]
```

Step 3: Check the service status for MongoDB with the help of following command:

\$ sudo systemctl status mongodb


```
rishabh@rishabh-Lenovo-V130-15IKB: ~
File Edit View Search Terminal Help
rishabh@rishabh-Lenovo-V130-15IKB:~$ sudo systemctl status mongod
● mongod.service - An object/document-oriented database
   Loaded: loaded (/lib/systemd/system/mongod.service; enabled; vendor preset: e
   Active: active (running) since Tue 2020-03-03 15:49:56 IST; 29s ago
     Docs: man:mongod(1)
  Main PID: 14151 (mongod)
    Tasks: 23 (limit: 4915)
   CGroup: /system.slice/mongod.service
           └─14151 /usr/bin/mongod --unixSocketPrefix=/run/mongod --config /etc/

Mar 03 15:49:56 rishabh-Lenovo-V130-15IKB systemd[1]: Started An object/document-
lines 1-10/10 (END)
```

systemctl verifies that MongoDB server is up and running.

Step 4: Now check if the installation process is done correctly and everything is working fine. Go through the following command:

```
$ mongo --eval 'db.runCommand({ connectionStatus: 1 })'
```

```
rishabh@rishabh-Lenovo-V130-15IKB: ~
File Edit View Search Terminal Help
rishabh@rishabh-Lenovo-V130-15IKB:~$ mongo --eval 'db.runCommand({ connectionSta
tus: 1 })'
MongoDB shell version v3.6.3
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.3
{
  "authInfo" : {
    "authenticatedUsers" : [ ],
    "authenticatedUserRoles" : [ ]
  },
  "ok" : 1
}
rishabh@rishabh-Lenovo-V130-15IKB:~$
```

the value “1” in ok field indicates that the server is working properly with no errors.

Step 5: MongoDB services can be started and stopped with the use of following commands: To stop running the MongoDB service, use command :

```
$ sudo systemctl stop mongod
```

MongoDB service has been stopped and can be checked by using the status command:

```
$ sudo systemctl status mongod
```

```
rishabh@rishabh-Lenovo-V130-15IKB: ~
File Edit View Search Terminal Help
rishabh@rishabh-Lenovo-V130-15IKB:~$ sudo systemctl stop mongodb
rishabh@rishabh-Lenovo-V130-15IKB:~$ sudo systemctl status mongodb
● mongodb.service - An object/document-oriented database
   Loaded: loaded (/lib/systemd/system/mongodb.service; enabled; vendor preset: e
   Active: inactive (dead) since Tue 2020-03-03 15:51:06 IST; 10s ago
     Docs: man:mongod(1)
   Process: 14151 ExecStart=/usr/bin/mongod --unixSocketPrefix=${SOCKETPATH} --con
   Main PID: 14151 (code=exited, status=0/SUCCESS)

Mar 03 15:49:56 rishabh-Lenovo-V130-15IKB systemd[1]: Started An object/document-
Mar 03 15:51:06 rishabh-Lenovo-V130-15IKB systemd[1]: Stopping An object/document
Mar 03 15:51:06 rishabh-Lenovo-V130-15IKB systemd[1]: Stopped An object/document-
lines 1-10/10 (END)
```

As it can be seen that the service has stopped, to start the service we can use :

\$ sudo systemctl start mongodb

```
rishabh@rishabh-Lenovo-V130-15IKB: ~
File Edit View Search Terminal Help
rishabh@rishabh-Lenovo-V130-15IKB:~$ sudo systemctl start mongodb
rishabh@rishabh-Lenovo-V130-15IKB:~$ sudo systemctl status mongodb
● mongodb.service - An object/document-oriented database
   Loaded: loaded (/lib/systemd/system/mongodb.service; enabled; vendor preset: e
   Active: active (running) since Tue 2020-03-03 15:51:32 IST; 14s ago
     Docs: man:mongod(1)
   Main PID: 14974 (mongod)
     Tasks: 23 (limit: 4915)
    CGroup: /system.slice/mongodb.service
           └─14974 /usr/bin/mongod --unixSocketPrefix=/run/mongodb --config /etc/

Mar 03 15:51:32 rishabh-Lenovo-V130-15IKB systemd[1]: Started An object/document-
lines 1-10/10 (END)
```