

Serve Static HTML Files with Node.js

Learn how to serve static HTML files with Node.js efficiently. This step-by-step guide covers setup, configuration, and best practices for web development.

Contents

01	Introduction	3
02	Prerequisites	3
03	1. Create a projectDirectory	3
04	2. Create Sample HTML Files	4
05	3. Create a main.js File	5
06	4. Test the Application	7
07	Conclusion	8

Introduction

By default, Node.js ships with a built-in `http` module that provides HTTP functionalities to your applications. The module can serve static or dynamic web content (For instance, HTML files or information fetched from a database like MySQL) depending on the configuration.

When you use the Node.js `http` module, you don't have to install third-party web servers like Apache when testing your web applications. This approach increases the developer's productivity. The `http` module has acceptable performance and contains a lot of functions for working with HTTP requests.

This guide implements the Node.js `http` module to serve static HTML files on an Ubuntu 20.04 server.

Prerequisites

To follow along with this guide:

1. [Deploy an Ubuntu 20.04 server.](#)
2. [Install Node.js.](#)

1. Create a `project` Directory

When working on any Node.js application, you must create a separate directory that hosts all the source code files.

1. Use the Linux `mkdir` command to set up the `project` directory. You can use any directory name.

```
$ mkdir project
```

2. Navigate to the new `project` directory.

```
$ cd project
```

3. Create a `public` sub-directory under the `project` directory. The `public` sub-directory stores the files you want to serve via HTTP.

```
$ mkdir public
```

4. Verify the following directories. The illustration below shows how your application directory structure looks.

```
home
----project
      ----public
```

Proceed to the next step to create and populate the HTML files.

2. Create Sample HTML Files

When serving static content, you may have many interlinked HTML files under the `public` directory, depending on the complexity of your website. This guide only uses two files for demonstration purposes.

1. Navigate to the `~/project/public` directory.

```
$ cd ~/project/public
```

2. Create an `index.html` file in a text editor.

```
$ nano index.html
```

3. Enter the following information into the file.

```
<html>
  <body>

  <h1>index.html</h1>
```

```
<p>A sample <b>index.html</b> web page.</p>

</body>
</html>
```

4. Save and close the file.
5. Create a `home.html` file in a text editor.

```
$ nano home.html
```

6. Enter the following information.

```
<html>
  <body>

    <h1>home.html</h1>
    <p>A sample <b>home.html</b> page.</p>

  </body>
</html>
```

7. Save and close the file.

3. Create a `main.js` File

In this step, you'll create a `main.js` file that contains the application's functions.

1. Navigate to the `project` directory.

```
$ cd ~/project
```

2. Open a new `main.js` file in a text editor.

```
$ nano main.js
```

3. Enter the following information into the `main.js` file. Replace **192.0.2.123** with your server's public IP address.

```
const http = require('http');
const fs = require('fs');

const host = '192.0.2.123';
const port = 8080;

const httpServer = http.createServer(httpHandler);

httpServer.listen(port, host, () => {
  console.log(`HTTP server running at http://${host}:${port}/`);
});

function httpHandler(req, res) {
  fs.readFile('./public/' + req.url, function (err, data) {

    if (err == null ) {

      res.writeHead(200, {'Content-Type': 'text/html'});
      res.write(data);
      res.end();
    }
  });
}
```

4. Save and close the `main.js` file.

The `main.js` file explained:

1. The `const http = require('http');` statement imports the `http` module in the application. This module allows your application to transfer data over the HTTP protocol.
2. The `const fs = require('fs');` declaration imports the `fs` module. This module provides file system functionalities.
3. The following two lines declare the web server's `host` and `port`. The application listens for incoming HTTP connections on port `8080` using the public IP address interface.

```
const host = '192.0.2.123';
const port = 8080;
```

4. The `const httpServer ...` statement creates an instance of the `http` server. The function `http.createServer(httpHandler);` instructs the `http` server to forward incoming HTTP requests to a `httpHandler(...)` function.
5. The `httpHandler(...)` function calls `fs.readFile()` function that takes two arguments:
 - The first argument is the filename. You're using the parameters `'./public/' + req.url` to extract the filename directly from the request URL.
 - The second function is the callback function (`function (err, data) {...}`) which returns either an error (`err`) or the `data` (content from the file).

You're then using the logical `if (err == null) {...}` statement to verify if there is an error. If there are no errors, you're writing the data in HTTP format using the `res.writeHead()`, `res.write(data)`, and `res.end()` functions. The `writeHead()` function sends the correct headers to browsers. That is a `200` response code for successful requests and a `text/html` header that tells the browser to treat the content as HTML.

```
function httpHandler(req, res) {
  fs.readFile('./public/' + req.url, function (err, data) {

    if (err == null) {

      res.writeHead(200, {'Content-Type': 'text/html'});
      res.write(data);
      res.end();
    }
  });
}
```

4. Test the Application

Your application is ready for testing.

1. Execute the `main.js` file.

```
$ node main.js
```

Output.

```
HTTP server running at http://192.0.2.123:8080/
```

2. Open a web browser and navigate to the following URLs. Replace **192.0.2.123** with your server's public IP address.

```
http://192.0.2.123:8080/index.html  
http://192.0.2.123:8080/home.html
```

3. The application should now serve the HTML pages using the `http` module, and your application is working as expected.

Conclusion

This guide highlights the basic steps of implementing the Node.js `http` module to serve static HTML pages. While the guide only uses two files, you can upload as many HTML files to the `public` folder depending on your needs.

Follow the links below to read more about Node.js:

- [A Quick Guide to Node.js.](#)
- [Create a CRUD Application with Node.js and MySQL.](#)



VULTR

