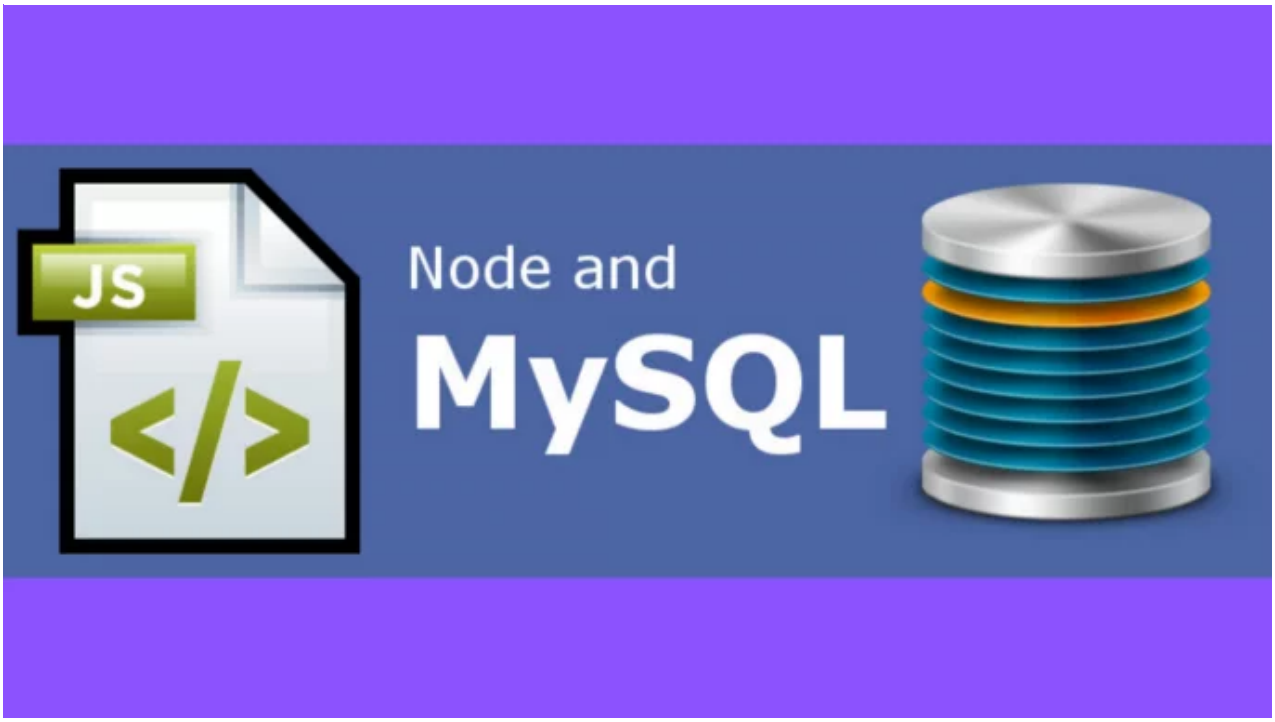
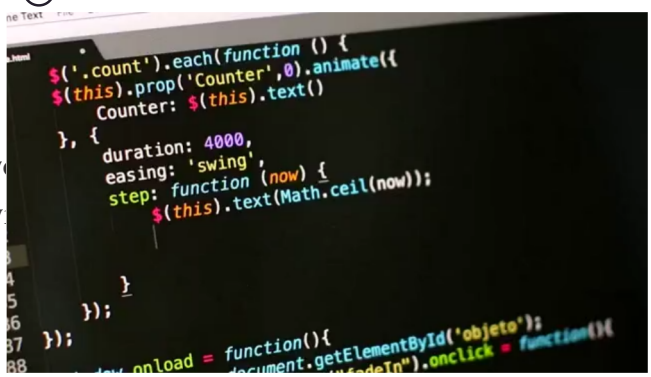
Node.js and MySQL Complete Tutorial

SHAHID / APR 8, 2021 / NODE TUTORIALS



Node.js and MySQL are some of the necessary binding
one of the most popular open-source databases in the w
popular programming language like Java and PHP prov
operations with MySQL.



In this **Node js and MySQL tutorial**, we are going to learn how to connect the Node js server with a MySQL database. We will also learn how to pool connections to improve performance, query the tables, and call stored procedures.

To be able to follow up with the code examples in this Node.js and MySQL tutorial, you should have MySQL installed on your computer, ~~click here~~ to download MySQL.

Also Read: ***NodeJS MySQL Create Table***

Quick Start: How to Use MySQL in Node

Assuming you have Node and MySQL installed on your computer. Let's quickly use MySQL in Node in three easy steps:

Step 1: Create a new Node.js project

Create a new directory and initialize a Node project usi

```
$ mkdir mysqlexperiment && cd mysql
$ npm init --y
```

Step 2: Install mysql node module

Install the **mysql** node module using the NPM.

```
npm install --save mysql
```

Step 3: Connect with MySQL

Create an **app.js** file and copy/paste the code shown below. Change the MySQL credentials accordingly with your system.

```
const mysql = require('mysql');
const connection = mysql.createConnection({
    host: 'localhost',
    user: 'user',
    password: 'password',
    database: 'databasename'
});

connection.connect((err) => {
    if (err) throw err;
    console.log('Connected to MySQL Server!');
});
```

Run the code using the following command.

```
node app.js
```

Observe the *'Connected to MySQL Server!'* message in

If you have the latest MySQL server installed, you migł following.

```
{
 code: 'ER_NOT_SUPPORTED_AUTH_MODE',
 errno: 1251,
 sqlMessage: 'Client does not support authentication protocol re
 sqlState: '08004',
 fatal: true
}
```

To tackle this issue, create a new user in your MySQL server with **'mysql_native_password'** authentication mechanisum.

Here is how you can do it quickly. First, log in to the MySQL server using root access.

```
mysql -u root -p
```

Then run these commands one by one.

```
CREATE USER 'newuser'@'localhost' IDENTIFIED WITH 'mysql_native_
GRANT ALL PRIVILEGES ON * . * TO 'newuser'@'localhost';
FLUSH PRIVILEGES;
```

In the code, pass the new credentials to connect to the MySQL server. Let's proceed further.

Pooling MySQL Connections

The code shown earlier is not meant for production use.
and MySQL. In a production scenario, we must use con
performance of MySQL and not overload the MySQL s

Let's explain it with a simple example.

Consider the code shown below.

```
const express = require("express");
const app = express();
const mysql = require('mysql');

const connection = mysql.createConnection({
    host: 'localhost',
    user: 'username',
    password: 'password',
    database: 'databasename'
});

connection.connect((err) => {
    if (err) throw err;
    console.log('Connected to MySQL Server!');
});

app.get("/", (req, res) => {
    connection.query('SELECT * from users LIMIT 1', (err, rows) =
        if (err) throw err;
        console.log('The data from users table are: \n', rows);
        connection.end();
    });
});

app.listen(3000, () => {
    console.log('Server is running at port 3000');
});
```

We are integrating *express* module to create a web server. Install the module using the following command.

```
npm install --save express
```

We are creating a MySQL connection on every request
multiple concurrent requests, the MySQL server will ge

To simulate the concurrent connection scenario, we are

Use this command to install it in Ubuntu system.

```
sudo apt-get install siege
```

Run our Node server.

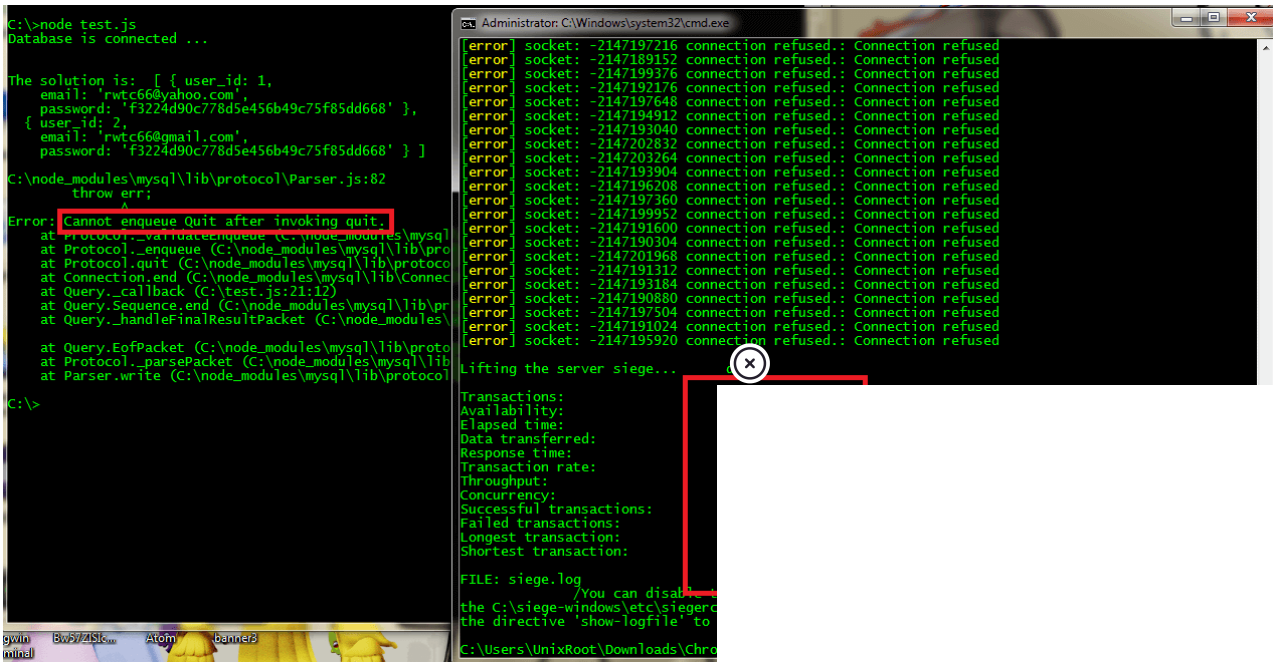
```
node app.js
```

Let's simulate the concurrent requests.

```
siege -c10 -t1M http://localhost:3000
```

Assuming you are running the Node server on Port 3000.

Here is the output.



As you can see from the output above, our server crashed while handling concurrent requests. To tackle this scenario, we use the Pooling mechanism.

> Connection Pooling is a mechanism to maintain a cache of database connection so that the connection can be reused after releasing it.

Let's rewrite our code to support connection pooling.

```
const express = require("express");
const app = express();
const mysql = require('mysql');

const pool = mysql.createPool({
    host: 'localhost',
    user: 'username',
    password: 'password',
    database: 'databasename'
});

app.get("/", (req, res) => {
    pool.getConnection((err, connection) => {
        if (err) throw err;
        console.log('connected as id ' + connection.threadId);
        connection.query('SELECT * from users LIMIT 1', (err, rov
            connection.release(); // return the connection to poo
            if (err) throw err;
            console.log('The data from users table are: \n', rows
        });
    });
});

app.listen(3000, () => {
    console.log('Server is running
});
```

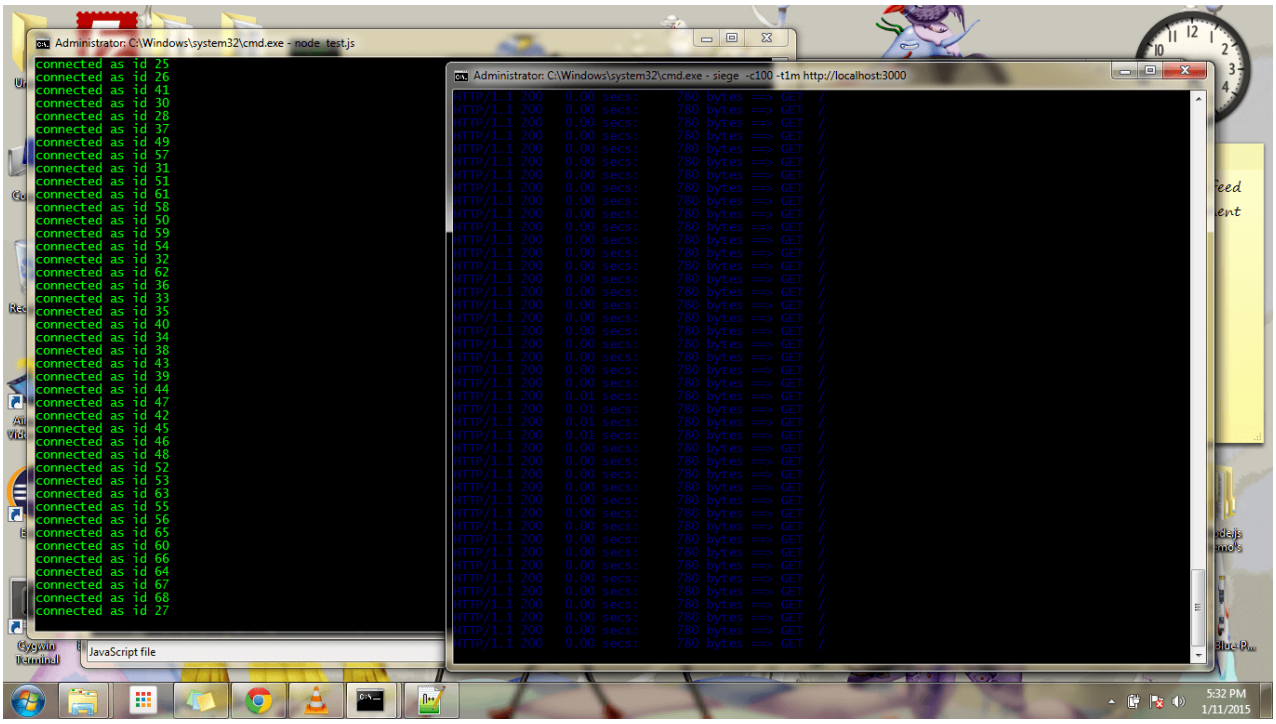
Run the app using the following command.

```
node app.js
```

Let's fire up 10 concurrent users for 1 minute using siege by using this command.

```
siege -c10 -t1M http://localhost:3000
```

Here is the output.



Our server is effectively handling multiple requests with ease. I have used this approach in multiple production software solutions with heavy payload and it works like charm.

Let's learn how to execute various MySQL queries usin

Executing Queries

Let's learn how to execute queries using Node.js.

Inserting data into Table

Here is the code to add new rows to the table.

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

const pool = mysql.createPool({
    connectionLimit: 100, //important
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'todolist',
    debug: false
});

// add rows in the table

function addRow(data) {
    let insertQuery = 'INSERT INTO ?? (??,??) VALUES (?,?)';
    let query = mysql.format(insertQuery, ["todo", "user", "notes
    pool.query(query, (err, response) => {
        if (err) {
            console.error(err);
            return;
        }
        // rows added
        console.log(response.insertId);
    });
}

// timeout just to avoid firing query before connection happens

setTimeout(() => {
    // call the function
    addRow({
        "user": "Shahid",
        "value": "Just adding a note"
    });
}, 5000);
```

The **mysql.format** function will perform the query escape.

Selecting data in Table

Here is the code to query rows in the table.

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

const pool = mysql.createPool({
    connectionLimit: 100, //important
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'todolist',
    debug: false
});

// query rows in the table

function queryRow(userName) {
    let selectQuery = 'SELECT * FROM ?? WHERE ?? = ?';
    let query = mysql.format(selectQuery, ["todo", "user", userN
    // query = SELECT * FROM `todo` where `user` = 'shahid'
    pool.query(query, (err, data) => {
        if (err) {
            console.error(err);
            return;
        }
        // rows fetch
        console.log(data);
    });
}

// timeout just to avoid firing query before connection happens

setTimeout(() => {
    // call the function
    // select rows
    queryRow('shahid');
}, 5000);
```

If you would like to add multiple rows in a single query, you can pass an array in the values. Like this.

```
let insertQuery = 'INSERT INTO ?? (
let values = [["shahid","hello"],["
let query = mysql.format(insertQuery
```

Updating data in Table

Here is the code to update the data in the table.

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

const pool = mysql.createPool({
    connectionLimit: 100, //important
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'todolist',
    debug: false
});

// update rows

function updateRow(data) {
    let updateQuery = "UPDATE ?? SET ?? = ? WHERE ?? = ?";
    let query = mysql.format(updateQuery, ["todo", "notes", data
    // query = UPDATE `todo` SET `notes`='Hello' WHERE `name`='sh
    pool.query(query, (err, response) => {
        if (err) {
            console.error(err);
            return;
        }
        // rows updated
        console.log(response.affectedRows);
    });
}

// timeout just to avoid firing query before connection happens

setTimeout(() => {
    // call the function
    // update row
    updateRow({
        "user": "Shahid",
        "value": "Just updating a note"
    });
}, 5000);
```

Deleting Rows in the table

Here is the code to delete a row from the table.

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

const pool = mysql.createPool({
    connectionLimit: 100, //importar
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'todolist',
    debug: false
```

```
});

function deleteRow(userName) {
    let deleteQuery = "DELETE from ?? where ?? = ?";
    let query = mysql.format(deleteQuery, ["todo", "user", userN
    // query = DELETE from `todo` where `user`='shahid';
    pool.query(query, (err, response) => {
        if (err) {
            console.error(err);
            return;
        }
        // rows deleted
        console.log(response.affectedRows);
    });
}

// timeout just to avoid firing query before connection happens

setTimeout(() => {
    // call the function
    // delete row
    deleteRow('shahid');
}, 5000);
```

Calling MySQL Stored Procedure Using Node

When a SQL query run in order to retrieve some data from the MySQL database, MySQL executes that query and returns the requested data, and if our system requires querying the same data regularly we have to write over and over again multiple times, so to solve that problem stored procedure comes into existence. A store procedure can store SQL statements in the MySQL server which can be directly run by calling that stored procedure.

You can also call a stored procedure directly using Node.js. If you don't have stored procedures created in MySQL, you can refer to the code below to do the same.

```
CREATE PROCEDURE `getAllTodo`()
BEGIN
SELECT * FROM todo;
END$$
```

Here is the code to call this stored procedure from the Node.js code:

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

const pool = mysql.createPool({
    connectionLimit: 100, //important
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'todolist',
    debug: false
});

function callSP(spName) {
    let spQuery = 'CALL ??';
    let query = mysql.format(spQuery, [spName]);
    // CALL `getAllTodo`
    pool.query(query, (err, result) => {
        if (err) {
            console.error(err);
            return;
        }
        // rows from SP
        console.log(result);
    });
}

// timeout just to avoid firing query before connection happens

setTimeout(() => {
    // call the function
    // call sp
    callSP('getAllTodo')
}, 5000);
```

Conclusion

In this tutorial we have learned how to use MySQL in N
mysql module, we have also learned connection pooling
Node.js to insert data into tables, query data in tables, u
finally, we have learned to call MySQL stored procedur
of databases but it works very well with MySQL, so do

Reference

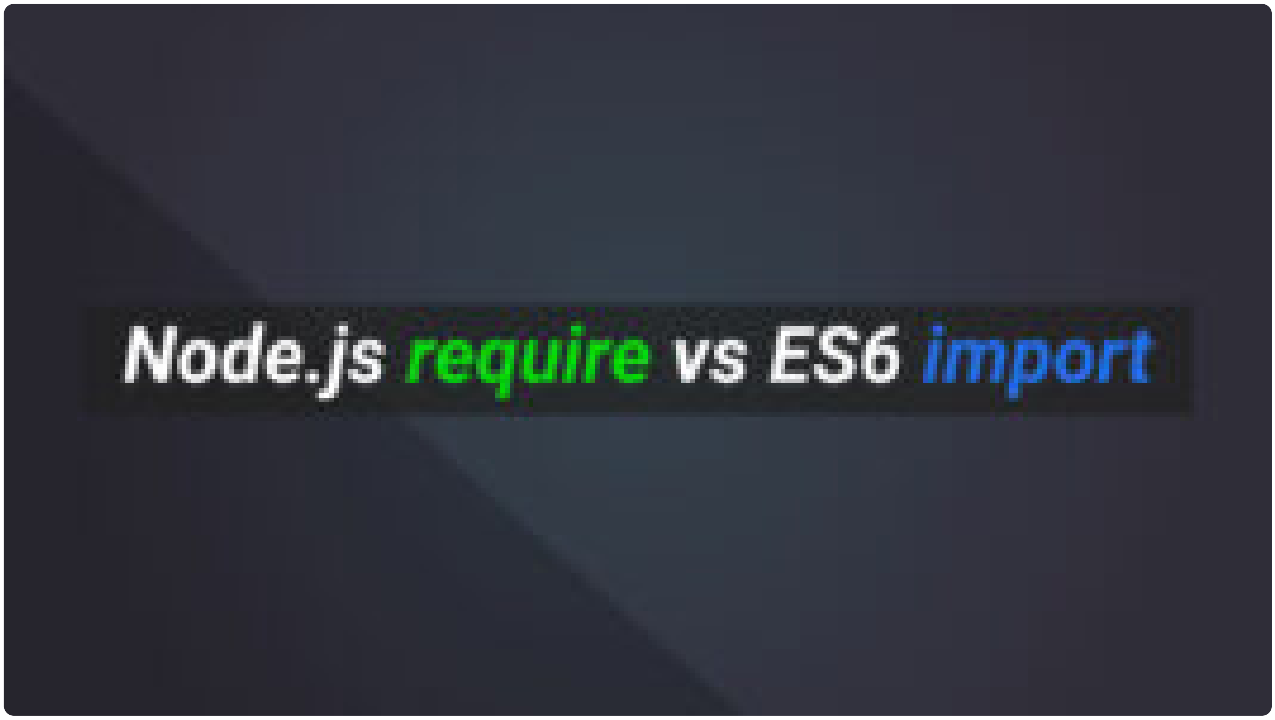
https://www.npmjs.com/package/mysql

Shahid

Founder of Codeforgeek. Technologist. Published Author. Engineer. Content Creator. Teaching Everything I learn!

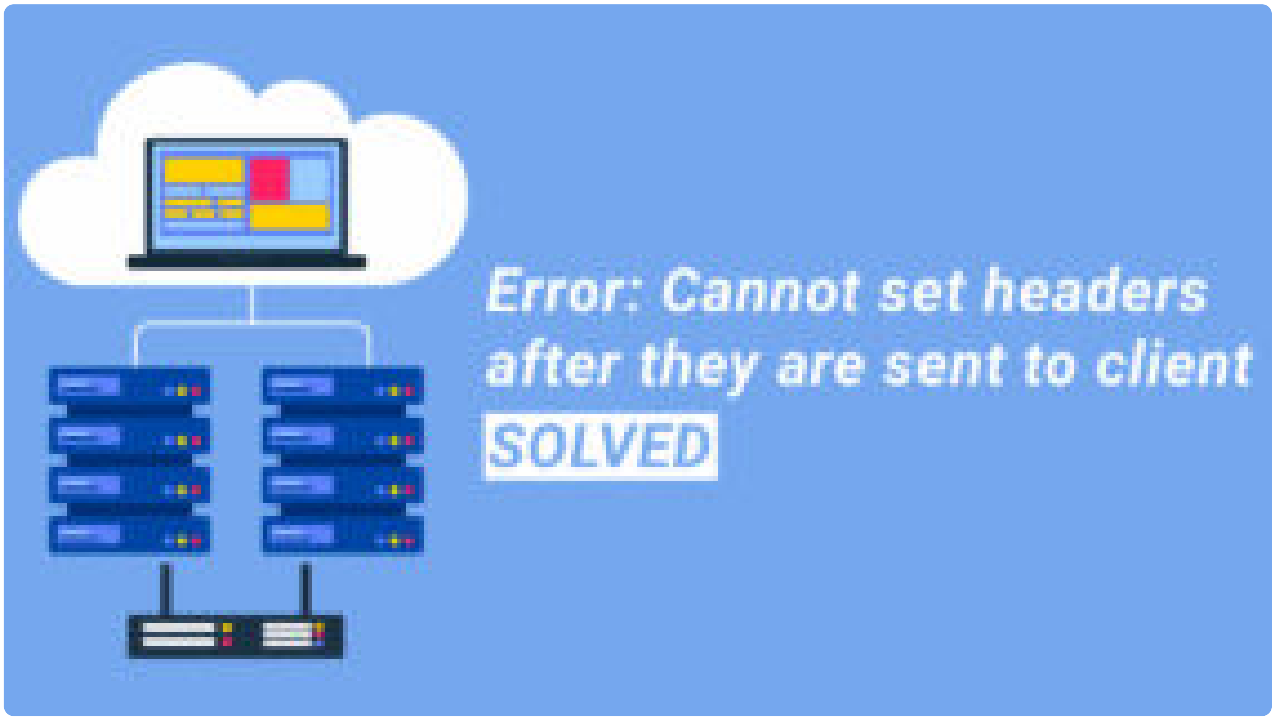
ARTICLES: 299

Related P



Explaining the Ultimate Difference Between Node.js require vs ES6 import

Mar 31, 2023

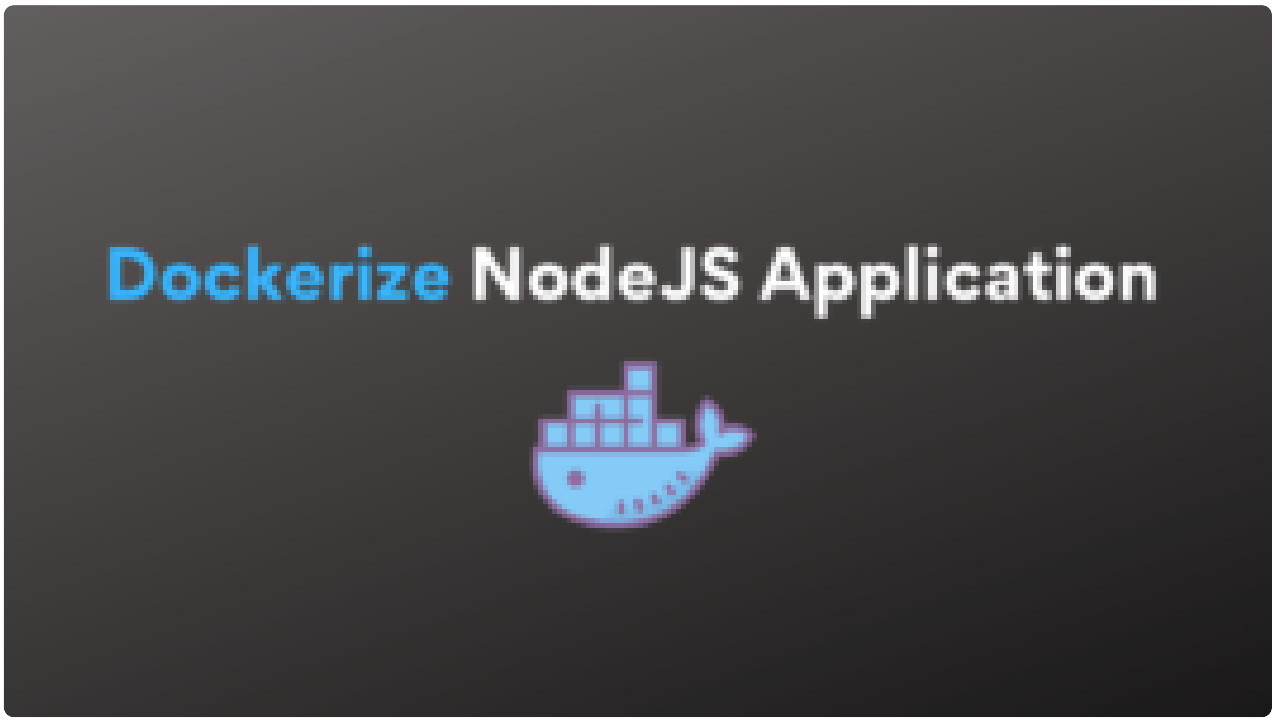


Cannot set headers after they are sent to client Solved

Mar 31, 2023

Difference between npm install and npm update (with examples)

Mar 30, 2023



Dockerize NodeJS Application: A Step-by-Step Guide

Mar 29, 2023