VULTR

Node.js > Article

Implement InfluxDB Server with Node.js on Ubuntu 20.04

Learn how to implement an InfluxDB server with Node.js on Ubuntu 20.04. Step-by-step guide for installation, configuration, and integration for time-series data.

Contents

01 Introduction 3

02 Prerequisites 3

03 1. Set Up a Sample Database 4

04 2. Create a Database Gateway Class 6

05 3. Create the Database Metric class 8

06 4. Create the Application's Entry Point 10

07 5. Test the Application 15

08 Conclusion 18

Introduction

InfluxDB is a high-performance time-series database written in the Go programming language. The open-source database application supports several hundred thousand writes per second. InfluxDB optimization for speed makes it the most suitable database server for storing and monitoring time-stamped metrics. The metrics are useful in IoT, financial, transportation, and telecommunication industries.

Node.js is a cross-platform run-time engine that executes Javascript code. Node.js competes with other modern backend programming languages like PHP, Go, and Python. The Node.js non-blocking model makes it a suitable choice for highly-available applications such as chat applications, real-time collaborating tools, streaming applications, and more.

This guide shows you how to code a Node.js application that interacts with an InfluxDB database server On Ubuntu 20.04. The application writes data points to InfluxDB and queries the database to present the information in an HTML table.

Prerequisites

To follow along with this guide:

1. Deploy an Ubuntu 20.04 server.
2. Create a non-root `sudo` user and install the following packages:
    - InfluxDB.
    - Node.js PPA Version (**Option 2**).

1. Set Up a Sample Database

In this guide, your sample application stores data permanently on an InfluxDB server. The data may come from IoT devices such as smartwatches, energy meters, bank applications, and more in a production environment. This guide uses sample data from solar customers. The energy consumed by the customers in kilowatt-hours, otherwise known as a KWh, is live-streamed to the application via the HTTP protocol. Then, the application stores the data in the InfluxDB server for further processing.

SSH to your server and execute the following steps to create and initialize a sample InfluxDB database:

1. Log in to the InfluxDB server. Replace `admin` and `EXAMPLE_PASSWORD` with the correct InfluxDB administrator credentials.

   ```
   $ influx -username 'admin' -password 'EXAMPLE_PASSWORD'
   ```

   Output.

   ```
   >
   ```

2. Create a sample `solar_customers` database.

   ```
   > CREATE DATABASE solar_customers
   ```

   Output.

   ```
   >
   ```

3. Switch to the new `solar_customers` database.

   ```
   > USE solar_customers
   ```

   Output.

© Vultr 2025 | Page 4

```
Using database solar_customers
```

4. Insert sample data points into the `solar_customers` database. Specify `power_usage` as the metric name. In the InfluxDB model, a data point (For instance, `power_usage,customer_id=1 KWh=800`) is like a record, and a metric (For example, `power_usage`) is like a table in SQL-based databases.

```
> INSERT power_usage,customer_id=1 KWh=800
> INSERT power_usage,customer_id=2 KWh=1750
> INSERT power_usage,customer_id=3 KWh=2560
> INSERT power_usage,customer_id=4 KWh=652
> INSERT power_usage,customer_id=5 KWh=920
```

Output.

```
...
>
```

5. Query the `power_usage` metric to ensure the data is in place.

```
> SELECT "customer_id", "KWh" FROM "power_usage"
```

Output.

```
time                customer_id KWh
----                ----------- ---
1662015648115956829 1           800
1662015655675425278 2           1750
1662015662345951963 3           2560
1662015669315582055 4           652
1662015675915505403 5           920
```

6. Log out from the InfluxDB server.

```
> quit
```

The sample database is now ready to accept new data points. Proceed to the next step, creating a Node.js module for accessing the InfluxDB database.

2. Create a Database Gateway Class

When creating a data-driven application with Node.js, it is conventional to have a separate class that provides database functions to the rest of the Node.js modules. A separate database gateway class allows you to define your database credentials in a single file making troubleshooting straightforward in case the application fails to connect to the database. Follow the steps below to create the class:

1. Create a `project` directory for the Node.js application.

```
$ mkdir project
```

2. Switch to the new `project` directory.

```
$ cd project
```

3. Open a new `database_gateway.js` file in a text editor.

```
$ nano influxdb_gateway.js
```

4. Enter the following information into the `influxdb_gateway.js` file. Replace `EXAMPLE_PASSWORD` with the correct password for the InfluxDB `admin` account.

```
class influxdb_gateway {

    dbConn() {

        const influxDb = require('influxdb-nodejs');

        const dbHost = '127.0.0.1';
        const dbPort = '8086';
        const dbUser = 'admin';
        const dbPass = 'EXAMPLE_PASSWORD';
        const dbName = 'solar_customers';

        const conString = 'http://' dbUser ':' dbPass '@' dbHost ':'
dbPort '/' dbName;
```

```
        const influxDbClient = new influxDb(conString);

        return influxDbClient;
    }
}

module.exports = influxdb_gateway;
```

5. Save and close the `database_gateway.js` file.

The `influxdb_gateway.js` file explained:

1. The `influxdb_gateway.js` contains a single `influxdb_gateway` class. The class then houses a `dbConn()` method.

```
class influxdb_gateway {
    dbConn() {
        ...
    }
}
...
```

2. The `dbConn()` method loads the `influxdb-nodejs` module. This module provides InfluxDB functionalities to the Node.js application. Then, the `dbConn()` method connects to the InfluxDB server and returns the connection using the `const influxDbClient = new influxDb(conString);` and `return influxDbClient;` statements.

3. The `module.exports = influxdb_gateway;` at the end of the file allows you to import and use the `influxdb_gateway` module in other Node.js source code files using the following declarations.

```
const influxdb_gateway = require('./influxdb_gateway');
const dg = new influxdb_gateway();
const dbConn =  dg.dbConn();
```

After coding the database gateway class, the next step involves creating a module that manages the InfluxDB `power_usage` metric.

© Vultr 2025 | Page 7

3. Create the Database Metric class

This application measures a single `power_usage` metric for demonstration purposes. Complex applications may contain dozens or hundreds of metrics. Because each metric is different, you should code its functionalities in a separate file. Follow the steps below to create a `power_usage` metric class:

1. Open a new `power_usage.js` file in a text editor.

```
$ nano power_usage.js
```

2. Enter the following information into the `power_usage.js` file.

```
class power_usage {

    constructor(dbConn) {
        this.dbConn = dbConn;
        this.measurement = 'power_usage';
    }

    readPoints(callBack) {

        const reader = this.dbConn.query(this.measurement);

        reader.then(function (data) {

            const dataPoints  = data.results[0].series[0].values;
            const dataColumns = data.results[0].series[0].columns;

            callBack(dataPoints, dataColumns)

        }).catch(
            console.error
        );
    }

    writePoint(pointData, callBack) {

        this.dbConn.write(this.measurement).tag({
            customer_id: pointData["customer_id"]
```

```
        }).field({
            KWh: pointData["KWh"]
        }).then(
            callBack("Success.")
        ).catch(
            console.error
        );

    }
}

module.exports = power_usage;
```

3. Save and close the `power_usage.js` file.

The `power_usage.js` file explained:

1. The `power_usage.js` file contains one `power_usage` class and three methods.

```
class power_usage {

    constructor(dbConn) {
        ...
    }

    readPoints(callBack) {
        ...
    }

    writePoint(pointData, callBack) {
        ...
    }
}
```

2. The methods in the `power_usage` class perform the following functions:

    - `constructor(dbConn)`: This method initializes the `power_usage` class. The constructor method accepts a `dbConn` argument containing a connection to the InfluxDB server. The `this.measurement = 'power_usage';` statement sets the name of the measurement. When initializing the `power_usage` class, you must pass an InfluxDB server connection through the constructor function per the following illustration.

```
const power_usage = require('./power_usage');
const pu = new power_usage(dbConn);
```

- `readPoints(callBack)`: This method runs the `this.dbConn.query(this.measurement);` function that queries and retrieves data points from the InfluxDB server. The `callBack` argument accepts a function that runs when the method completes executing. After the `readPoints(callBack)` method runs, it returns the data points and data columns to the calling function using the `callBack(dataPoints, dataColumns)` statement.
- `writePoint(pointData, callBack)`: This method writes a new measurement data point to the InfluxDB server by running the `this.dbConn.write(this.measurement).tag({...}).field({...})` function. The `writePoint(...)` function accepts the `pointData` in JSON format and the `callBack` function that runs when the method completes.

3. The `module.exports = power_usage;` declaration at the end of the `power_usage` class avails the module to other source code files. Using the following code block, you may initialize and use the `power_usage` module.

```
const power_usage = require('./power_usage');
const pu = new power_usage(dbConn);
pu.readPoints(callBack);
pu.writePoint(dataPoints, callBack);
```

After coding the `power_usage` metric class, create the application's entry point in the next step.

4. Create the Application's Entry Point

Every Node.js application must have an entry point. This is the main file that runs when the application starts. The sample application in this guide runs through an `index.js` file. Create the file by following the steps below.

1. Open a new `index.js` file for editing purposes.

```
$ nano index.js
```

2. Enter the following information into the `index.js` file. Replace `172.16.0.1` with your server's private IP address.

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

const influxdb_gateway = require('./influxdb_gateway');
const power_usage = require('./power_usage');

const httpHost = '172.16.0.1';
const httpPort = 8080;

const server = http.createServer(httpHandler);

server.listen(httpPort, httpHost, () => {
    console.log(`Server running at http://${httpHost}:${httpPort}/`);
});

function httpHandler(req, res) {

    const dg = new influxdb_gateway();
    const dbConn =  dg.dbConn();

    const pu = new power_usage(dbConn);

    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/html');

    if (req.method == 'POST') {

        var httpPayload = "";

        req.on('data', function (data) {
            httpPayload = data;
         });

        req.on('end', function () {
            pu.writePoint(JSON.parse(httpPayload), writeCallBack);
        });
    }
```

```
    function writeCallBack(response) {
        res.write(response  'rn');
        res.end();
    }

    if (req.method == 'GET') {
        pu.readPoints(readCallBack);
    }

    function readCallBack(dataPoints, dataColumns) {

        var tableHeaders = '';

        for (var columnIndex in dataColumns) {
            tableHeaders = '<th>'  dataColumns[columnIndex]  '</th>';
        }

        var rows = '';

        for (var rowIndex in dataPoints) {

            var rowData = '<tr>';

            for (var columnIndex in dataPoints[rowIndex]) {
                rowData = '<td>'  dataPoints[rowIndex][columnIndex]  '</td>';
            }

            rowData = '</tr>';

            rows = rowData;
        }

        const tableData = "<table border = '1' align = 'center'><tr>"
tableHeaders  "</tr>"  rows  "</table>";

        const html = `<!DOCTYPE html>
                    <html>
                      <body align = 'center'>
                        <h1>Power Usage For Customers</h1><hr>`
                          tableData
                       `</body>
                    </html>`;

        res.write(html);
        res.end();
```

```
    }
}
```

3. Save and close the `index.js` file.

The `index.js` file explained:

1. The `...require(...)` section loads all the modules the application requires to run. The `http` module offers HTTP functionalities to your application and allows you to run a web server. The `influxdb_gateway` and `power_usage` statements load the custom classes you coded earlier.

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

const influxdb_gateway = require('./influxdb_gateway');
const power_usage = require('./power_usage');
...
```

2. The next four lines initialize a web server that listens for incoming connections under port `8080`. Then, the `http.createServer(httpHandler);` statement passes `httpHandler` as the handler function for incoming HTTP requests.

```
 ...
const httpHost = '172.16.0.1';
const httpPort = 8080;

const server = http.createServer(httpHandler);

server.listen(httpPort, httpHost, () => {
    console.log(`Server running at http://${httpHost}:${httpPort}/`);
});
...
```

3. The `httpHandler(req, res)` function initializes the `influxdb_gateway` and the `power_usage` modules using the following statements.

```
     ...
    const dg = new influxdb_gateway();
    const dbConn =  dg.dbConn();
```


© Vultr 2025 | Page 13

```
const pu = new power_usage(dbConn);
...
```

4. The next two lines set the correct HTTP headers for the application to allow HTTP clients to treat the data as HTML.

```
 ...
res.statusCode = 200;
res.setHeader('Content-Type', 'text/html');
...
```

5. The logical `if (...){...}` statements examine the HTTP method to determine the appropriate method to run. For `POST` requests (`if (req.method == 'POST') {...}`), the application runs the `pu.writePoint(JSON.parse(httpPayload), writeCallBack);` function to write the data points to the InfluxDB database. For `GET` requests (`if (req.method == 'GET') {...}`), the application runs the `pu.readPoints(readCallBack);` function to read the data points from the InfluxDB database.

6. The `writeCallBack(...)` function executes when the application writes data successfully into the database. The `writeCallBack(...)` function returns a `Success.` message.

```
 ...
function writeCallBack(response) {
    res.write(response  'rn');
    res.end();
}
...
```

7. The `readCallBack(dataPoints, dataColumns)` function runs when the application retrieves data from the InfluxDB database and displays the data in HTML format. The `for (var columnIndex in dataColumns){...}` loop sets the HTML table headers. Then, the `for (var rowIndex in dataPoints) {...}` loop populates the HTML table rows.

```
 ...
function readCallBack(dataPoints, dataColumns) {
    ...
    for (var columnIndex in dataColumns) {
```

```
            tableHeaders = '<th>'  dataColumns[columnIndex]  '</th>';
        }

        var rows = '';

        for (var rowIndex in dataPoints) {

            var rowData = '<tr>';

            for (var columnIndex in dataPoints[rowIndex]) {
                rowData = '<td>'  dataPoints[rowIndex][columnIndex]  '</td>';
            }

            rowData = '</tr>';

            rows = rowData;
        }

        ...

        res.write(html);
        res.end();
    }
    ...
```

You now have all the required application's source code files. Test the application in the next step.

5. Test the Application

The final step in this guide involves the following:

- Initializing the `project` directory.
- Installing the `influxdb-nodejs` module using the `npm` package.
- Inserting new data points using the Linux `curl` command.
- Displaying the InfluxDB data points on an HTML web page.

Follow the steps below to complete the above steps:

1. Ensure your Node.js `npm` package is up to date.

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

Output.

```
changed 54 packages, and audited 212 packages in 2s
```

2. Initialize the `project` directory.

```
$ npm init
```

3. Respond to the following prompts. For most prompts, you only need to press the `Enter` to proceed.

```
package name: (project) :key_enter:
version: (1.0.0):key_enter:
description: InfluxDB and Node.js :key_enter:
entry point: (index.js) :key_enter:
test command: :key_enter:
git repository: :key_enter:
keywords: influxdb, node.js :key_enter:
author: Test author :key_enter:
license: (ISC) :key_enter:
About to write to ...

{
...
}

Is this OK? (yes) yes :key_enter:
```

4. Install the `influxdb-nodejs` module.

```
$ npm install influxdb-nodejs
```

Output.

© Vultr 2025

```
added 41 packages, and audited 42 packages in 2s
```

5. Run the project.

```
$ node index.js
```

The application starts a web server that listens for incoming HTTP connections on port `8080`. The previous command has a blocking function. Don't run more commands on the active terminal window.

```
Server running at http://172.16.0.1:8080/
```

6. Establish another `SSH` connection in a new terminal window and run the following commands.

```
 $ curl -X POST http://172.16.0.1:8080/ -H 'Content-Type: application/json' -d
'{"customer_id": 6, "KWh": 2687}'
 $ curl -X POST http://172.16.0.1:8080/ -H 'Content-Type: application/json' -d
'{"customer_id": 7, "KWh": 1265}'
 $ curl -X POST http://172.16.0.1:8080/ -H 'Content-Type: application/json' -d
'{"customer_id": 8, "KWh": 4268}'
 $ curl -X POST http://172.16.0.1:8080/ -H 'Content-Type: application/json' -d
'{"customer_id": 9, "KWh": 1398}'
 $ curl -X POST http://172.16.0.1:8080/ -H 'Content-Type: application/json' -d
'{"customer_id": 10, "KWh": 7625}'
```

Output.

```
 ...
Success.
```

7. Open a web browser and visit the following URL. Replace `172.16.0.1` with your server's public IP address

   - http://172.16.0.1:8080/

The following HTML table lists the InfluxDB data points.

Power usage HTML table

Conclusion

This guide shows you how to implement the InfluxDB server with Node.js on Ubuntu 20.04 server. The sample application only uses a few data points for demonstration purposes. However, InfluxDB has rich grouping and mathematical functions that you can use to summarize the data when designing more detailed applications.

Follow the link below to read how to use the InfluxDB server with Golang:

- How to Use the InfluxDB with Golang on Ubuntu 20.04.

© Vultr 2025

VULTR