



TUTORIAL

How To Create and Run Scheduled Jobs with Node.js

Node.js Development JavaScript

By Chris Nwamba

Posted December 12, 2019 27.8k



While this tutorial has content that we believe is of great benefit to our community, we have not yet tested or edited it to ensure you have an error-free learning experience. It's on our list, and we're working on it! You can help us out by using the "report an issue" button at the bottom of the tutorial.

Introduction

Ever wanted to do specific things on your application server at certain times without having to manually run them yourself? This is where scheduled tasks come in handy.

In this article, you'll create scheduled tasks in Node applications using the `node-cron` module. You'll schedule the execution of different scripts at different intervals from your application.

```
cron-jobs-node on  master [!?] is  v1.0.0 via  v7.2.1
→ npm start

> cron-jobs-node@1.0.0 start /home/captog/work/cron-jobs-node
> node index.js

running a task every minute
running a task every minute
running a task every minute
```

In this tutorial you'll build a small application that automatically deletes auto-generated log files from the server. Then you'll look at a few additional use cases.

Prerequisites

[SCROLL TO TOP](#)

To follow through this tutorial, you'll need:

- A local development environment for Node.js. Follow [How to Install Node.js and Create a Local Development Environment](#)

Step 1 – Creating A Node Application and Installing Dependencies

To get started, create a new Node application by opening your terminal and creating a new folder for your project.

```
$ mkdir cron-jobs-node && cd cron-jobs-node
```

Then initialize it, which creates a `package.json` file which you'll use to track dependencies:

```
$ npm init -y
```

Add the `express` web framework as a dependency, as well as the `node-cron` and the `fs` modules by running the following command:

```
$ npm install express node-cron fs
```

The `express` module powers the web server you'll build. `node-cron` is the task scheduler, and `fs` is the Node file system module.

The dependencies are installed; let's build the server.

Step 2 – Building the Backend Server

Create an `index.js` file and then import the necessary node modules:

```
$ nano index.js
```

Add this code to `index.js` to include `node-cron` and `express` and create an instance of Express:

SCROLL TO TOP

index.js

```
const cron = require("node-cron");
const express = require("express");
const fs = require("fs");

app = express();
```

This application will generate logs, and after a while, you'll want to delete the error log files at intervals automatically. You'll use `node-cron` to do this.

To see how this module works, add the following to your `index.js` file which displays a message every minute:

index.js

```
// schedule tasks to be run on the server
cron.schedule("* * * * *", function() {
  console.log("running a task every minute");
});

app.listen(3128);
```

Now, when you run the server, we get the following result:

```
$ node index.js
```

Output

```
running a task every minute
running a task every minute
```

You have a task running every minute. Stop the server with `CTRL+C`.

Now let's look at how to run tasks in more detail.

Step 2 – Scheduling Tasks at Different intervals

With `node-cron`, you can schedule tasks for different intervals.

In the previous example, you created a job that ran every minute by passing `* * * * *` to the `schedule` function. Each of those `SCROLL TO TOP` special meaning:

```

* * * * *
| | | | |
| | | | day of week
| | | | month
| | | day of month
| | hour
| minute
second ( optional )

```

Learn more about how this notation works in [How To Use Cron to Automate Tasks on a VPS](#).

To delete the log file from the server on the 21st of every month, update the `index.js` to look like this:

`index.js`


```

// schedule tasks to be run on the server
cron.schedule("* * 21 * *", function() {
  console.log("-----");
  console.log("Running Cron Job");
  fs.unlink("./error.log", err => {
    if (err) throw err;
    console.log("Error file successfully deleted");
  });
});

app.listen("3128");

```

Now, when the server is run, you'll get the following output on the 21st of the month:



A terminal window showing the output of a cron job. The text "Cron Job automatically deleting error file" is displayed in a monospaced font. The terminal has a light gray background and a dark gray border.

To simulate this yourself, change the scheduler to run in a shorter interval.

You can run any actions inside the scheduler. Actions ranging from creating a file, to sending emails and running scripts. Let's take a look at more use cases

Use Case: Backing Up Databases

Ensuring the accessibility of user data is very key to any business. If an unforeseen event happens and your database becomes damaged, you'll need to restore your

SCROLL TO TOP

database from a backup. You'll be in serious trouble if you don't have any form of existing backup for your business. You can use what you've learned so far to periodically backup the existing data in your database. Let's take a look at how to do this. In this example, you'll use the SQLite database, as it's less complex than other options and works in the context of this example.

Visit the [SQLite3 download page](#) to download SQLite3 for your platform. You can also install SQLite3 on Ubuntu with the following commands:

```
$ sudo apt-get update
$ sudo apt-get install sqlite3 libsqlite3-dev
```

Next, install a Node module that lets your app run shell scripts:

You'll use this library to execute a command that exports your data.

Now create a sample database by running the command:

```
$ sqlite3 database.sqlite
```

To backup your database at 11:59pm every day, update your `index.js` file to look like this:

index.js

```
const fs = require("fs");
let shell = require("shelljs");
const express = require("express");

app = express();

// To backup a database
cron.schedule("59 23 * * *", function() {
  console.log("-----");
  console.log("Running Cron Job");
  if (shell.exec("sqlite3 database.sqlite .dump > data_dump.sql").code !== 0) {
    shell.exit(1);
  }
  else{
    shell.echo("Database backup complete");
  }
});
app.listen("3128");
```

SCROLL TO TOP

Save the file and run your server:

```
$ node index.js
```

You'll see the following results:



Next, let's look at sending periodic emails.

Use Case: Sending Scheduled Emails

You can also use jobs to keep your users up to date by sending them emails at different intervals. For example, you can curate a list of interesting links and then send them to users every Sunday. To do this, use the `nodemailer` module and connect it to your SMTP server, like GMail.

Install `nodemailer` by running the command:

```
$ npm install --save nodemailer
```

Once that is done, update the `index.js` file to add a section that defines the mailer and sets the username and password for a GMail account:

index.js

```
const cron = require("node-cron");
const express = require("express");
let nodemailer = require("nodemailer");

app = express();

// create mail transporter
let transporter = nodemailer.createTransport({
  service: "gmail",
  auth: {
    user: "your_email_address@gmail.com",
    pass: "your_password"
  }
});
```

SCROLL TO TOP

Warning: You will need to temporarily allow non-secure sign-in for your Gmail account if you'd like to use it for testing purposes [here](#). For production applications, configure a specific secure Email account and use those credentials.

Then add the task to send the messages every Wednesday:

index.js

```
// sending emails at periodic intervals
cron.schedule("* * * * Wednesday", function(){
  console.log("-----");
  console.log("Running Cron Job");
  let mailOptions = {
    from: "COMPANYEMAIL@gmail.com",
    to: "sampleuser@gmail.com",
    subject: `Not a GDPR update ;)` ,
    text: `Hi there, this email was automatically sent by us`
  };
  transporter.sendMail(mailOptions, function(error, info) {
    if (error) {
      throw error;
    } else {
      console.log("Email successfully sent!");
    }
  });
});

app.listen("3128");
```

Now, when you run the server using the command `node index.js` , you get the following result:

 Server Running Cron Job

 Email Automatically sent by Cron Job

Conclusion

In this article, you used `node-cron` to schedule jobs in your Node.js applications. You can find the code examples for this tutorial in [this GitHub repository](#).

Was this helpful?

Yes

NO

SCROLL TO TOP



[Report an issue](#)

About the authors



Chris Nwamba

is a Community author on DigitalOcean.

Related

TUTORIAL

How To Use the Python Map Function

We can use the built-in function `map()` to apply a function to each item in an iterable (like a list or dictionary) and return a ...

TUTORIAL

How To Launch Child Processes in Node.js

Since Node.js instances create a single process with a single thread, JavaScript operations that take a long time to run ...

TUTORIAL

Understanding Arrow Functions in JavaScript

Arrow functions are a new way to write anonymous function expressions in JavaScript, and are similar to lambda functions in ...

TUTORIAL

How To Build a Discord Bot with Node.js

Discord is a chat application that allows millions of users across the globe to message and voice chat online in ...

[SCROLL TO TOP](#)

Still looking for an answer?

[Ask a question](#)[Search for more help](#)

Comments

1 Comment

Leave a comment...

[Sign In to Comment](#)



[nodejsdeveloperskh](#) May 3, 2020

0 hi. I have a question. why should I use node-cron package until I can use setInterval or some kind?

thanks for your answer

[Reply](#) [Report](#)

[SCROLL TO TOP](#)



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.



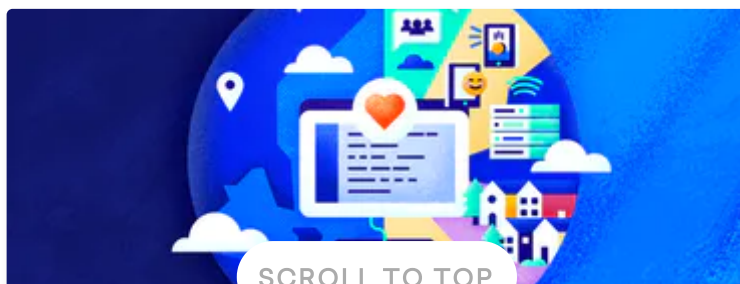
BECOME A CONTRIBUTOR

You get paid; we donate to tech nonprofits.



GET OUR BIWEEKLY NEWSLETTER

Sign up for Infrastructure as a Newsletter.



SCROLL TO TOP

HIRE FOR GOOD

Working on improving health
and education, reducing
inequality, and spurring
economic growth? We'd like to
help.

Featured on Community Kubernetes Course Learn Python 3 Machine Learning in Python
Getting started with Go Intro to Kubernetes

DigitalOcean Products Droplets Managed Databases Managed Kubernetes Spaces Object Storage
Marketplace

Welcome to the developer cloud

DigitalOcean makes it simple to launch in the
cloud and scale up as you grow – whether you're
running one virtual machine or ten thousand.

[Learn More](#)



Partners
Referral Program

Press
Legal
Security & Trust Center

Products

Pricing
Products Overview
Droplets
Kubernetes
Managed Databases
Spaces

Marketplace
Load Balancers
Block Storage
API Documentation
Documentation
Release Notes

Community

Tutorials
Q&A
Tools and Integrations
Tags
Product Ideas
Write for DigitalOcean

Presentation Grants
Hatch Startup Program
Shop Swag
Research Program
Open Source
Code of Conduct

Contact

Get Support
Trouble Signing In?
Sales
Report Abuse
System Status

SCROLL TO TOP