How To Write a Linux Daemon with Node.js

Authored by: ***ASPHostServer Administrator*** *[asphostserver@gmail.com]*
Saved From: *http://faq.asphosthelpdesk.com/article.php?id=225*

Introduction

A daemon is a program that runs in background and has no controlling terminal. They are often used to provide background services. For example, a web-server or a database server can run as a daemon.

This tutorial will show you how to write a daemon with Node.js and deploy it on your server with Upstart.

I will focus on implementing a standard daemon. I'm using upstart just for simplicity, however you can write a System-V init script or use anything you like to start your daemon.

Requirements

For this tutorial, you will need a Linux server(Preferably Ubuntu or CentOS), Node.js, and Upstart.

Node.js

There are several ways to install Node.js. The easiest way in my opinion is to use nvm. It also allows you to manage different versions of Node.

Alternatively, you can follow one of these guides:

1. How To Install an Upstream Version of Node.js on Ubuntu 12.04
2. Installing Node.js via package manager

Upstart

Upstart comes pre-installed on many Linux distros. If it's not installed on the distro of your choice, you should be able to install it from official repositories. You can also compile it from the source-code. Refer to Upstart Getting Started page for more info.

How Daemons Work

Basically a daemon starts like a normal process. the following will occur afterwards:

1. It creates a copy of itself as it's child process.
2. The child detaches itself from the parent process.
3. The child process closes its standard file descriptors (See below.)
4. The parent process exits.
5. The daemon continues its work in background.

Instead of closing the standard file descriptors, the parent process can open the null device and attach it to the child's standard file descriptors.

The Example Daemon

For the sake of this tutorial, we're going to create a simple **HTTP** daemon.

Our daemon will be able to:

Start in the background (we're going to use a module called "daemon" for this.) **Spawn multiple workers for HTTP** server. **Restart the workers on SIGHUP**. **Terminate the workers on SIGTERM**. **Drop workers' privileges after the HTTP** server gets started.

Initial project structure

The following is the initial project structure:

```
node-simple-http-daemon/
|
|-- README.md
|-- bin/
|   `-- node-simple-http-daemon
`-- lib/
    `-- app.js
```

package.json

Let's start by creating a package.json file:

```
$ npm init
```

Install daemon module:

```
$ npm --save install daemon
```

And here is how the package.json should look like:

```
{
  "name": "node-simple-http-daemon",
  "version": "0.0.0",
  "description": "Simple HTTP Daemon written in Node.js",
  "main": "lib/app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node lib/app.js"
  },
  "author": "Fardjad Davari",
  "license": "MIT",
  "dependencies": {
    "daemon": "~1.1.0"
```

```
  }
}
```

Note that we added a start script, So we can start the app with `npm start` command later.

HTTP Server

For now, we're going to create a simple HTTP server that starts listening on port 80 and responds with "Hello world" for every request.

Put the following in lib/app.js file:

```
/**
 * lib/app.js
 */
const PORT = 80;
const ADDRESS = '0.0.0.0';
var http = require('http');
var server = http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
});
server.listen(PORT, ADDRESS, function () {
    console.log('Server running at http://%s:%d/', ADDRESS, PORT);
    console.log('Press CTRL+C to exit');
});
```

You can start the server by running `sudo npm start`.

Daemon executable

The following is the daemon executable code.

The following should be placed in bin/node-simple-http-daemon:

```
#!/usr/bin/env node
/**
 * bin/node-simple-http-daemon
 */
// Everything above this line will be executed twice
require('daemon')();
var cluster = require('cluster');
// Number of CPUs
var numCPUs = require('os').cpus().length;
/**
 * Creates a new worker when running as cluster master.
 * Runs the HTTP server otherwise.
 */
function createWorker() {
  if (cluster.isMaster) {
```

```
    // Fork a worker if running as cluster master
    var child = cluster.fork();
    // Respawn the child process after exit
    // (ex. in case of an uncaught exception)
    child.on('exit', function (code, signal) {
      createWorker();
    });
  } else {
    // Run the HTTP server if running as worker
    require('../lib/app');
  }
}
/**
 * Creates the specified number of workers.
 * @param  {Number} n Number of workers to create.
 */
function createWorkers(n) {
  while (n-- > 0) {
    createWorker();
  }
}
/**
 * Kills all workers with the given signal.
 * Also removes all event listeners from workers before sending the signal
 * to prevent respawning.
 * @param  {Number} signal
 */
function killAllWorkers(signal) {
  var uniqueID,
      worker;
  for (uniqueID in cluster.workers) {
    if (cluster.workers.hasOwnProperty(uniqueID)) {
      worker = cluster.workers[uniqueID];
      worker.removeAllListeners();
      worker.process.kill(signal);
    }
  }
}
/**
 * Restarts the workers.
 */
process.on('SIGHUP', function () {
  killAllWorkers('SIGTERM');
  createWorkers(numCPUs * 2);
});
/**
 * Gracefully Shuts down the workers.
 */
process.on('SIGTERM', function () {
  killAllWorkers('SIGTERM');
});
// Create two children for each CPU
createWorkers(numCPUs * 2);
```

Time to start our daemon! But before that, we should modify our app.js to handle SIGTERM.

Add the following to the end of the app.js file:

```
process.on('SIGTERM', function () {
  if (server === undefined) return;
  server.close(function () {
    // Disconnect from cluster master
    process.disconnect && process.disconnect();
  });
});
```

Make the daemon script executable:

```
$ chmod +x bin/node-simple-http-daemon
```

And run it (Make sure nothing else is running on port 80):

```
$ sudo bin/node-simple-http-daemon
```

Now, your daemon and it's workers should be running in background. You can confirm that by sending an HTTP GET request via cURL:

```
$ curl 127.0.0.1
```

Before moving to the next step, let's take a look at the process IDs:

```
$ ps -axf | grep [n]ode-simple-http-daemon | \
  awk '{ print "pid:"$ ps -axf | grep [n]ode-simple-http-daemon | \ awk '{ print
"pid:"$2", parent-pid:"$3 }'", parent-pid:"}'
```

Sample output:

```
pid:33811, parent-pid:1
pid:33853, parent-pid:33811
pid:33856, parent-pid:33811
pid:33857, parent-pid:33811
pid:33858, parent-pid:33811
pid:33859, parent-pid:33811
pid:33860, parent-pid:33811
pid:33861, parent-pid:33811
pid:33862, parent-pid:33811
```

Note that the daemon is the process with parent-pid of 1.

Try restarting the workers by sending SIGHUP to the daemon:

```
$ kill -HUP 33811 # (replace 33811 with the daemon PID)
```

Now if you list the PIDs again, you should be able to see new worker processes with new PIDs. Amazing, isn't it?

To stop the daemon, just run:

```
$ kill -TERM 33811 # (replace 33811 with the daemon PID)
```

Dropping privileges

We're almost done. We only need to make workers drop their privileges after the server gets started.

Modify `server.listen()` callback in **app.js so it reads:**

```
server.listen(PORT, ADDRESS, function () {
    console.log('Server running at http://%s:%d/', ADDRESS, PORT);
    console.log('Press CTRL+C to exit');
    // Check if we are running as root
    if (process.getgid() === 0) {
      process.setgid('nobody');
      process.setuid('nobody');
    }
});
```

And that's it for the daemon part.

Now you can install it system-wide:

```
$ sudo npm link
```

Upstart

Making an **Upstart** job is very easy. Create a file in `/etc/init/node-simple-http-daemon.conf` with the following contents:

```
# /etc/init/node-simple-http-daemon.conf
start on started network
stop on stopping network
respawn
expect daemon
exec https-proxy-daemon
```

Now you can:

```
$ sudo start node-simple-http-daemon # Start the job
$ initctl --system node-simple-http-daemon # Check the job status
$ sudo reload node-simple-http-daemon # Send SIGHUP (restart the workers)
$ sudo stop node-simple-http-daemon # Stop the job
```

Next Steps

You may want to modify the daemon to give up spawning workers when they're getting killed too frequently (spinning).

I can't emphasize enough how important it is to have adequate logging in your application. Make sure to check out node-bunyan.