PM2 — Cluster Mode and Zero-Downtime Restarts

by Marcus Pöhls on December 21 2015, tagged inServer, PM2, 17 min read

You've already learned how to start, stop, or delete application processes with PM2. There is a major downside having only one app process running: app updates. Deploying updates to your application requires a restart to make the changes available to every visitor. These app restarts will cause a downtime, even though they are quite short, like 5 seconds maximum. Once you have enough traffic that your analytics shows almost always that somebody is currently viewing your site. And you'll start holding yourself back from just restarting your app instances.

This article will show you how to use PM2's cluster mode to spawn multiple worker processes of your app which will result in 0s downtimes during app restarts!

PM2 Series Overview

Cluster Mode Requirements

Before we start out and start your app in cluster mode, make sure you have **Node.js `0.12` or newer installed**. Node.js `0.10` is incompatible and not supported with PM2’s cluster module!

That’s all you need to prepare before using the cluster mode of PM2.

Node.js Cluster Module

Knowing how things work out under the hood is important in production environments. You don’t want any magic running on your server which was configured once by a colleague of yours (or even earlier) and nobody else can help in emergency situations.

Node.js is single threaded and therefore doesn’t leverage all available CPU cores of your machine. However, it has a built-in cluster module which spawns worker processes to share the same TCP connection. That means, your application can run on the same IP and port, but have multiple workers responding to incoming requests. Node’s cluster module works in round robin manner to distribute work load among available workers.

Let’s have a look at the following code to show how Node’s integrated cluster module would look like for an application:

```
var http = require('http');
var cpuCount = require('os').cpus().length;

if (cluster.isMaster) {
  // Fork workers.
  for (var i = 0; i < cpuCount; i++) {
    cluster.fork();
  }

  cluster.on('exit', function(worker, code, signal) {
    console.log('worker ' + worker.process.pid + ' died');
  });
} else {
  // Workers can share any TCP connection
  // In this case it is an HTTP server
  http.createServer(function(req, res) {
    res.writeHead(200);
    res.end("hello world\n");
  }).listen(8000);
}
```

Internally, the cluster module creates a master and forks your app instances to worker processes. You can have as many workers as you want, they're not limited by the number of available CPU cores. The master and worker processes use inter process communication, IPC, to share internal data.

PM2 Cluster Module

When using PM2, it will automatically handle all Node.js's cluster logic for you. There is no need to change your actual application to run multiple processes of it. You can create your application and hand off the clustering to PM2.

That means, the code above would shrink to its core:

app.js

```
http.createServer(function(req, res) {
  res.writeHead(200);
  res.end("hello world\n");
}).listen(8000);
```

Then you benefit from PM2’s built-in cluster module which handles multiple processes for your:

This command will start two processes of `app.js`. Using the `-i <number-of-instances>` option tells PM2 to start the process in `cluster` mode instead of `fork` mode.

```
[PM2] Starting server in cluster_mode (4 instances)
[PM2] Done.
----------------------------------------------------------------------------------------------------
│ App name    │ id │ mode    │ pid   │ status │ restart │ uptime │ cpu │ memory      │ watching │
----------------------------------------------------------------------------------------------------
│ maintenance │ 0  │ fork    │ 4114  │ online │ 0       │ 2h     │ 12% │ 53.945 MB   │ disabled │
│ homepage    │ 3  │ cluster │ 23568 │ online │ 0       │ 2s     │  9% │ 76.566 MB   │ disabled │
│ homepage    │ 4  │ cluster │ 23569 │ online │ 0       │ 2s     │ 16% │ 66.047 MB   │ disabled │
│ homepage    │ 5  │ cluster │ 23586 │ online │ 0       │ 2s     │  2% │ 55.406 MB   │ disabled │
│ homepage    │ 6  │ cluster │ 23603 │ online │ 0       │ 2s     │ 10% │ 53.160 MB   │ disabled │
----------------------------------------------------------------------------------------------------
```

PM2 allows various values for the `-i` option:

```
pm2 start app.js -i 5

# start as many processes as CPU cores available
pm2 start app.js -i 0

# start as many processes depending on CPU cores minus 1
# e.g. 4 cores - 1 = 3
pm2 start app.js -i -1
```

PM2 also handles possible failures and makes sure your workers processes get back up online in case of an error. You manage the processes in cluster_mode like any other process (start, restart, stop, delete, etc).

Scale Your Cluster

Using PM2 and its cluster mode allows you to scale your applications in real-time. If you need more or less workers than currently available, use PM2's `scale` command and adjust the cluster size respectively:

The `scale` command is only available for processes that already run in `cluster` mode. PM2 won't restart your app which currently runs in `fork` mode. You need to manually delete and restart it in `cluster` mode.

The following code block shows you how to scale up the number of processes for the examplary `homepage` app.

```
[PM2] Scaling up application
[PM2] Scaling up application
[PM2] Scaling up application
[PM2] Scaling up application
------------------------------------------------------------------------------------------
│ App name    │ id │ mode    │ pid   │ status │ restart │ uptime │ memory       │ watching │
------------------------------------------------------------------------------------------
│ maintenance │ 0  │ fork    │ 26400 │ online │ 2       │ 16h    │ 40.844 MB    │ disabled │
│ homepage    │ 3  │ cluster │ 23568 │ online │ 0       │ 2s     │ 76.566 MB    │ disabled │
│ homepage    │ 4  │ cluster │ 23569 │ online │ 0       │ 2s     │ 66.047 MB    │ disabled │
│ homepage    │ 5  │ cluster │ 23586 │ online │ 0       │ 2s     │ 55.406 MB    │ disabled │
│ homepage    │ 6  │ cluster │ 23603 │ online │ 0       │ 2s     │ 53.160 MB    │ disabled │
│ homepage    │ 7  │ cluster │ 36299 │ online │ 0       │ 0s     │ 49.340 MB    │ disabled │
│ homepage    │ 8  │ cluster │ 36316 │ online │ 0       │ 0s     │ 37.949 MB    │ disabled │
│ homepage    │ 9  │ cluster │ 36332 │ online │ 0       │ 0s     │ 28.406 MB    │ disabled │
│ homepage    │ 10 │ cluster │ 36350 │ online │ 0       │ 0s     │ 17.609 MB    │ disabled │
------------------------------------------------------------------------------------------
```

Of course, PM2 allows you to scale down any app in real-time:

```
[PM2] deleteProcessId process id 3
[PM2] deleteProcessId process id 4
[PM2] deleteProcessId process id 5
[PM2] deleteProcessId process id 6
[PM2] deleteProcessId process id 7
[PM2] deleteProcessId process id 8
-----------------------------------------------------------------------------------------
│ App name    │ id │ mode    │ pid   │ status │ restart │ uptime │ memory      │ watching │
-----------------------------------------------------------------------------------------
│ maintenance │ 0  │ fork    │ 26400 │ online │ 2       │ 16h    │ 41.688 MB   │ disabled │
│ homepage    │ 9  │ cluster │ 36332 │ online │ 0       │ 115s   │ 71.020 MB   │ disabled │
│ homepage    │ 10 │ cluster │ 36350 │ online │ 0       │ 115s   │ 71.242 MB   │ disabled │
-----------------------------------------------------------------------------------------
```

The previously running processes are deleted immediately and the number of workers reduced to your defined number.

You can also use a shortcut within PM2 which allows you to scale up or down your number of workers: `pm2 scale homepage +1` or `pm2 scale homepage -1` . The first command will add another worker to `homepage` and the second one removes a worker process.

Zero-Downtime Deployments

Everybody wants to minimize the downtime for apps running in production. This applies to any failure situation as well as to update deployments. When running an app in `fork` mode and you want to publish app changes to production, you need to restart the app process which causes a short downtime.

Reload Your App

To omit this downtime, you can leverage PM2's `cluster` mode and cluster module to restart your app processes in 0s downtime manner. In opposite to the `restart` command, you'll use `reload` .

The `reload` command will restart one process after another and wait until the currently restarted one is back online. This way, you're taking a ton of mental stress from your deployments, because you don't need to wait until less visitors are actively reading your content.

```
[PM2] Reloading process by name homepage
[PM2] Process homepage successfully reloaded
[PM2] Process homepage successfully reloaded
[PM2] All processes reloaded
-----------------------------------------------------------------------------------------------------
| App name    | id | mode    | pid   | status | restart | uptime | cpu | memory     | watching |
-----------------------------------------------------------------------------------------------------
| maintenance | 0  | fork    | 26400 | online | 2       | 17h    | 17% | 45.078 MB  | disabled |
| homepage    | 19 | cluster | 37983 | online | 1       | 2s     | 11% | 70.109 MB  | disabled |
| homepage    | 20 | cluster | 38003 | online | 1       | 1s     | 32% | 70.277 MB  | disabled |
-----------------------------------------------------------------------------------------------------
```

You'll notice that the reload takes much longer than the hard restart. However, that's the trade-off you're happily paying to deploy changes whenever you want without causing downtimes for your visitors.

In case PM2 couldn't reload your app, it will fallback to the classic `restart` command.

Gracefully Reload Your App

There are situations when reloading your app might take a long time or the reload even fails. Those cases indicate that your app has a lot open connections or you need to close a database connection manually. Experiencing this situation, you can leverage the `SIGINT` signal to intercept the shutdown process of your application. Within your app, listen for this `SIGINT` signal and close the connections.

```
    // process reload ongoing
    // close connections, clear cache, etc
    // by default, you have 1600ms
    process.exit(0);
});
```

You can control the timeout for your app has to close all connections once the shutdown signal was sent. Use the `PM2_GRACEFUL_LISTEN_TIMEOUT` environment variable and define your desired value in milliseconds. The default value is 1600ms.

By listening on the `SIGINT` signal, the `pm2 reload` command becomes graceful.

The output of the graceful `reload` command is the same as for the normal `reload` :

```
[PM2] Reloading process by name homepage
[PM2] Process homepage successfully reloaded
[PM2] Process homepage successfully reloaded
[PM2] All processes reloaded
----------------------------------------------------------------------------------------------------
│ App name    │ id │ mode    │ pid   │ status │ restart │ uptime │ cpu │ memory      │ watching │
----------------------------------------------------------------------------------------------------
│ maintenance │ 0  │ fork    │ 26400 │ online │ 2       │ 17h    │ 15% │ 48.379 MB   │ disabled │
│ homepage    │ 19 │ cluster │ 38922 │ online │ 2       │ 18s    │ 22% │ 79.867 MB   │ disabled │
│ homepage    │ 20 │ cluster │ 38943 │ online │ 2       │ 9s     │ 11% │ 80.273 MB   │ disabled │
----------------------------------------------------------------------------------------------------
```

Downsides of Cluster Mode

If you rely on sessions, the cluster module may cause you some headache. You can imagine that sticky sessions are bound to a given process and since PM2 passes requests in round robin manner to the workers, you might not be sent to the same worker as before. That means, you need to store session and websocket information outside of your app, like in a database.