

# Node-tify TA project

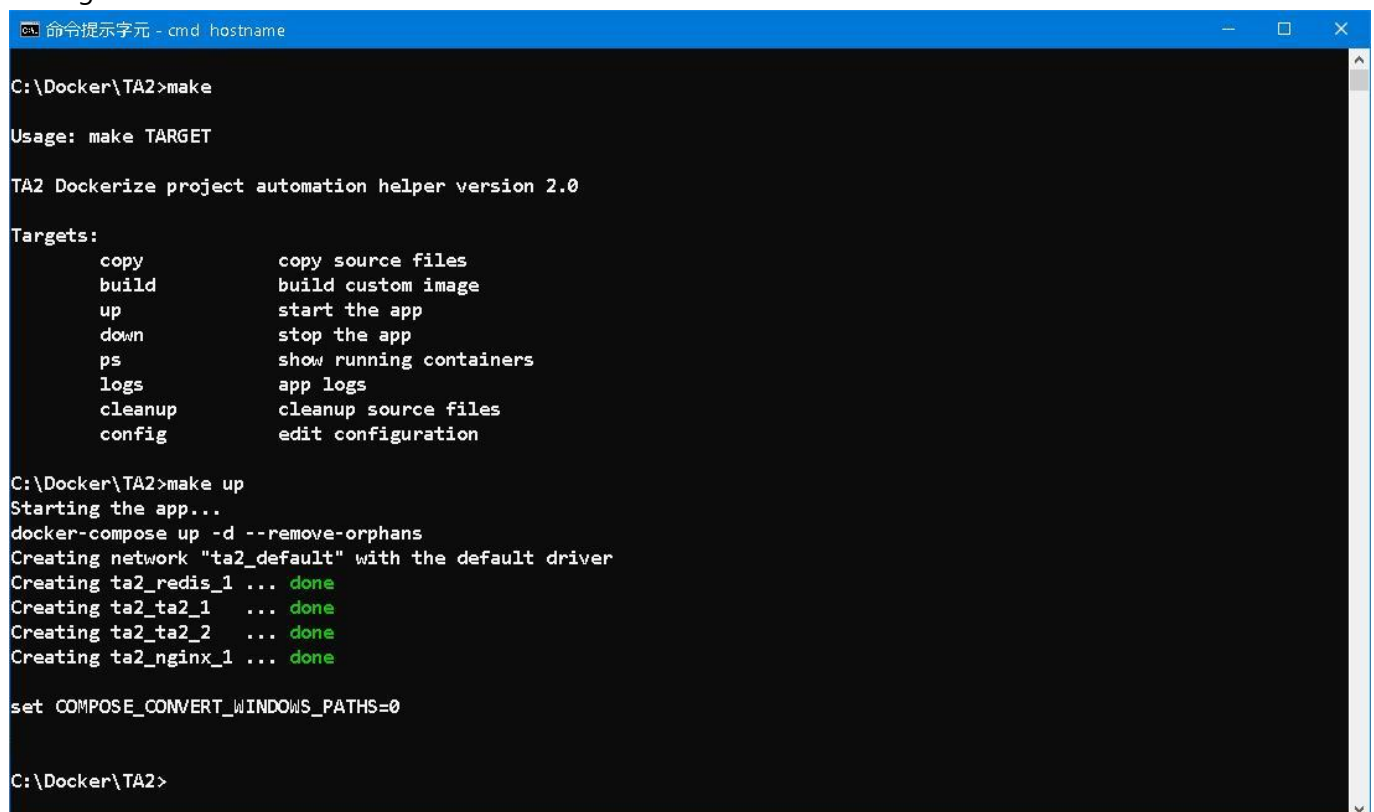
## — “Old soldiers never die--they just fade away.”

---

### Forward

I have a story, i.e. **TA**, but I won't tell you, not because it is insignificant, just because I don't want to... Being stagnated in a pond for over two decades, I became so accustomed to the placid posture of it. The ways of gearing things up, the steps to do things right which I was so acquaintance. They just deeply implanted in my brain unconsciously.

Two years ago, I came across the other way when I was finding means to *modernize* my artefacts. All my works proved to be in vain, as you may know, changing the clothes of a dying man doesn't prevent him from cessation, the only effect was to put a new shroud on an old carcase... At the first sight of the way, I was startled and dropped to faint immediately at it's velocity and delicacy... With tremendous mental exertion, I managed to finish **TA2**.



```
命令提示字元 - cmd hostname

C:\Docker\TA2>make

Usage: make TARGET

TA2 Dockerize project automation helper version 2.0

Targets:
    copy          copy source files
    build          build custom image
    up            start the app
    down          stop the app
    ps            show running containers
    logs          app logs
    cleanup        cleanup source files
    config        edit configuration

C:\Docker\TA2>make up
Starting the app...
docker-compose up -d --remove-orphans
Creating network "ta2_default" with the default driver
Creating ta2_redis_1 ... done
Creating ta2_ta2_1 ... done
Creating ta2_ta2_2 ... done
Creating ta2_nginx_1 ... done

set COMPOSE_CONVERT_WINDOWS_PATHS=0

C:\Docker\TA2>
```

While sitting miserably doesn't change anything. I decided to renounce my old ways, my old days and search for new dream, new deed, of course, with new way.

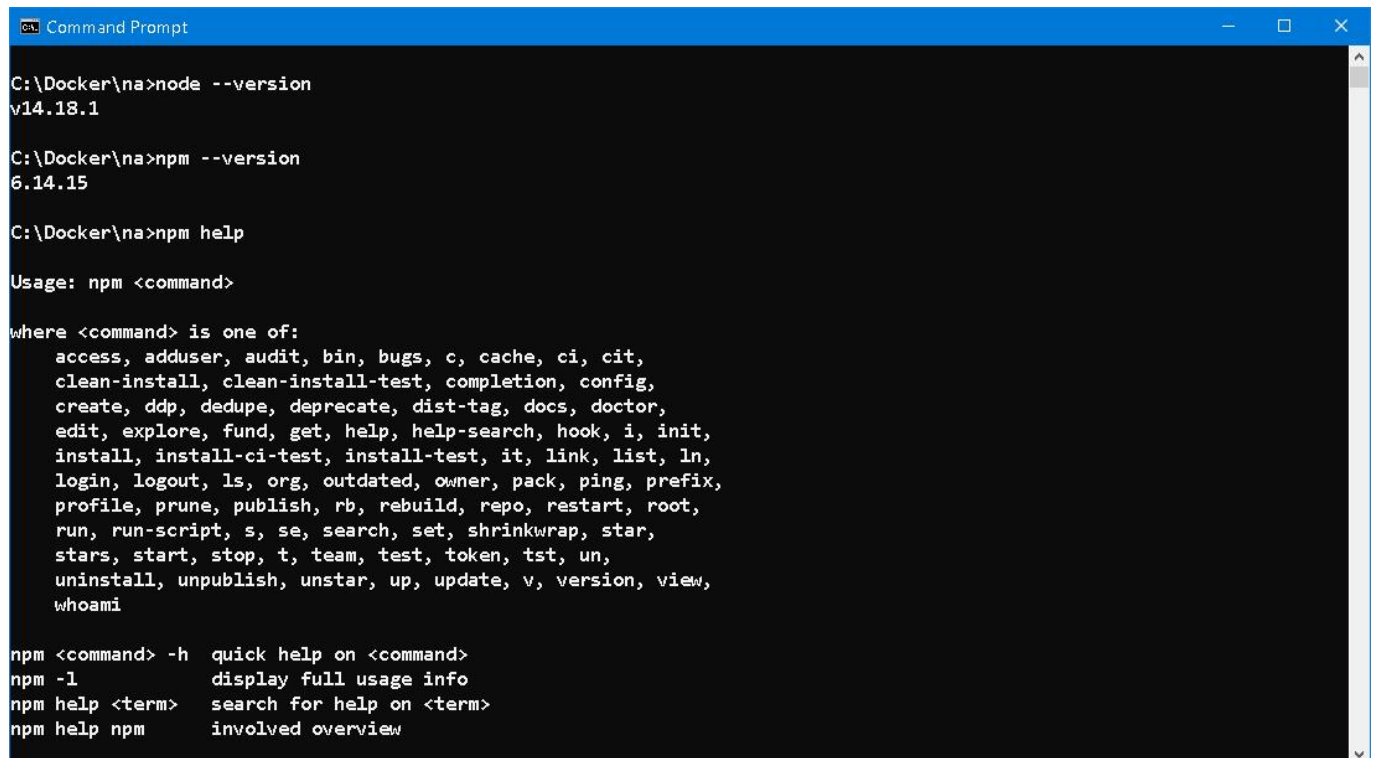
### I. HTTP - Back to Normal

[ASP.NET Web Forms](#) employs a simplified model to facilitate creation of web pages. The discussion of pros and cons is voluminous. The good things is it alleviates the burden of learning the underlay HTTP protocol, which gives novices a quick start. The bad things is, by default, you can not use HTTP verbs other than GET and POST<sup>[2]</sup> and no intrinsic [RESTful](#) URI support.

HTTP verbs such as PUT and DELETE are important when [AJAX](#) comes into play, RESTful URI are important when implementing API for front-end frameworks. While some strives to survive Web Forms in the Age of .NET 5/6+<sup>[1]</sup>, others seek ways to migrate or quit.

When comparing .NET Core with NodeJS, my scale is not on any language or environment heritage. While .NET Core is still under heavy development, NodeJS attains a more mature and fast-growing society, it is lightweight and more flexible, fits for both Bare-metal and Docker deployment. As a bonus, NodeJS integrates seamlessly with MongoDB and Redis, some state-of-the-art packages can not do without.

## II. [npm](#) - The Brothers [NodeJS](#)



```
Command Prompt

C:\Docker\na>node --version
v14.18.1

C:\Docker\na>npm --version
6.14.15

C:\Docker\na>npm help

Usage: npm <command>

where <command> is one of:
  access, adduser, audit, bin, bugs, c, cache, ci, cit,
  clean-install, clean-install-test, completion, config,
  create, ddp, dedupe, deprecate, dist-tag, docs, doctor,
  edit, explore, fund, get, help, help-search, hook, i, init,
  install, install-ci-test, install-test, it, link, list, ln,
  login, logout, ls, org, outdated, owner, pack, ping, prefix,
  profile, prune, publish, rb, rebuild, repo, restart, root,
  run, run-script, s, se, search, set, shrinkwrap, star,
  stars, start, stop, t, team, test, token, tst, un,
  uninstall, unpublish, unstar, up, update, v, version, view,
  whoami

npm <command> -h  quick help on <command>
npm -l           display full usage info
npm help <term>  search for help on <term>
npm help npm     involved overview
```

```
npm init -y
npm install express
npm install nodemon --save-dev
npm install pm2 -g
npm install npm-check -g

npm help install
```

### [express](#)

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

#### server.js

```
const express = require('express')
const app = express()
```

```
. . .
app.use('/task', taskRoute)
app.use('/user', userRoute)
. . .
app.listen(port, () => {
  console.log(`Application started on port ${port}`)
})
```

## MongoDB

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas.

The flexibility of MongoDB as a schemaless database is one of its strengths. If we would like to be more restrictive and have a really fixed schema for the collection we need to add the `additionalProperties: false` parameter in the `createCollection` command.

```
db.createCollection( "people" , {
  validator: {
    $jsonSchema:
    {
      bsonType: "object",
      additionalProperties: false,
      required: ["name","age"],
      properties: {
        _id : {
          bsonType: "objectId" },
        name: {
          bsonType: "string",
          description: "required and must be a string" },
        age: {
          bsonType: "int",
          minimum: 0,
          maximum: 100,
          description: "required and must be in the range 0-100" }
      }
    }
  })
```

In this case, we don't have flexibility, and that is the main benefit of having a NoSQL database like MongoDB. It's up to you to use it or not. It depends on the nature and goals of your application. I wouldn't recommend it in most cases.

## Redis

Redis (Remote Dictionary Server) is an in-memory data structure store, used as a distributed, in-memory key-value database, cache and message broker, with optional durability.

The default server-side session storage, MemoryStore, is purposely not designed for a production environment. It will leak memory under most conditions, does not scale past a single process, and is meant for debugging and developing.

```
const session = require("express-session");
const Redis = require("ioredis");
. . .
// Local session
let sess = {
  secret: 'keyboard cat',
  resave: false,
  saveUninitialized: true,
  cookie: {
    maxAge: 30 * 60 * 1000 // in milliseconds
  }
}
// if Redis is defined, it should be Redis session
if (process.env.REDIS_URI)
{
  let RedisStore = require("connect-redis")(session);
  let RedisClient = new Redis(process.env.REDIS_URI);
  sess.store = new RedisStore({ client: RedisClient })
  RedisClient.on('connect', () => {
    console.log('Connected to Redis Server.')
    RedisClient.set('hello', format(new Date(), 'yyyy-MM-dd hh:mm:ss'));
  })
}
app.use(session(sess))
```

### III. [Mongoose](#) - The importance of being Model



Mongoose provides a straight-forward, schema-based solution to model your application data. It includes built-in type casting, validation, query building, business logic hooks and more, out of the box.

task.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

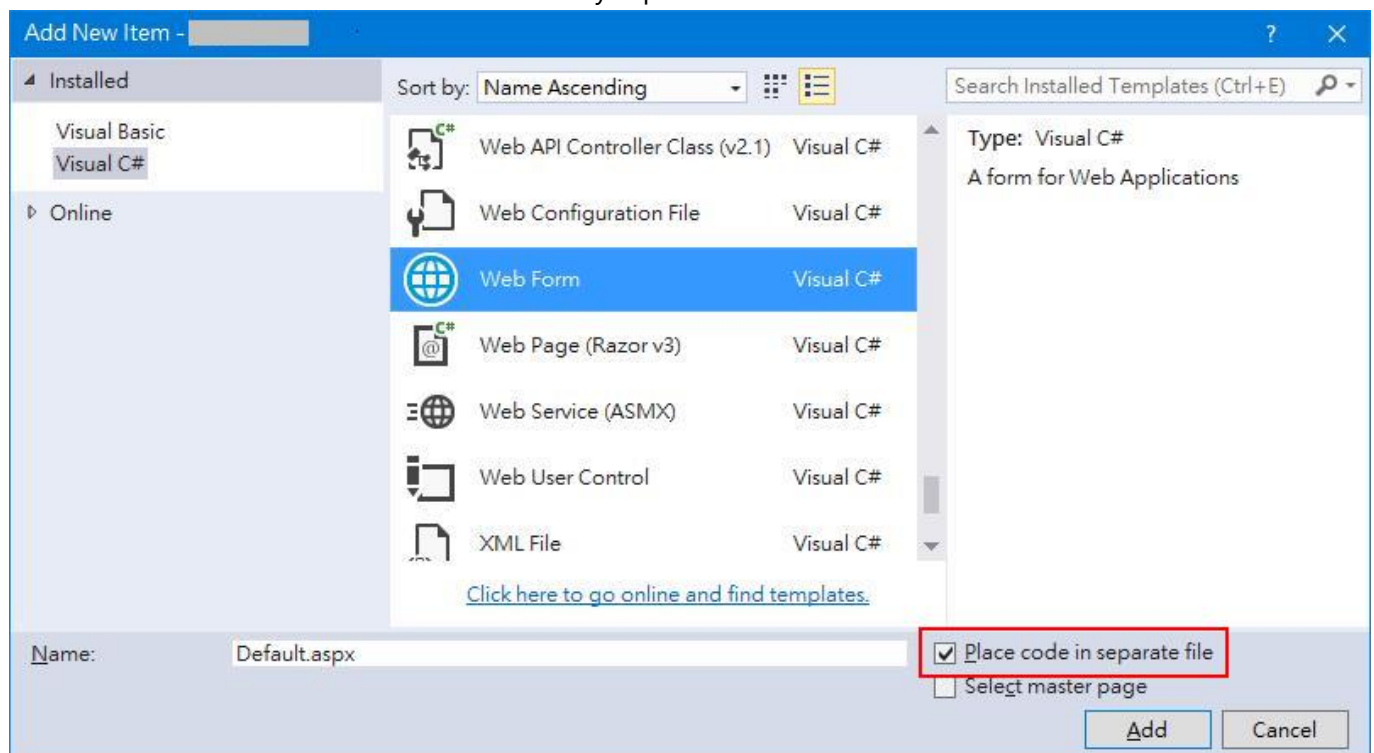
const TaskSchema = new Schema({
  taname: {
    type: String,
    required: true,
    index: true,
    lowercase: true
  },
  tatype: {
    type: String,
    required: true
  },
  tadate: {
    type: Number,
    required: true
  },
  taappnum: String,
  tafamrep: String,
  taremark: String
```

```
    },  
    {  
      timestamps: true  
    }  
  });  
  
module.exports = mongoose.model('Task', TaskSchema);
```

## IV. EJS - A Taste of the Past

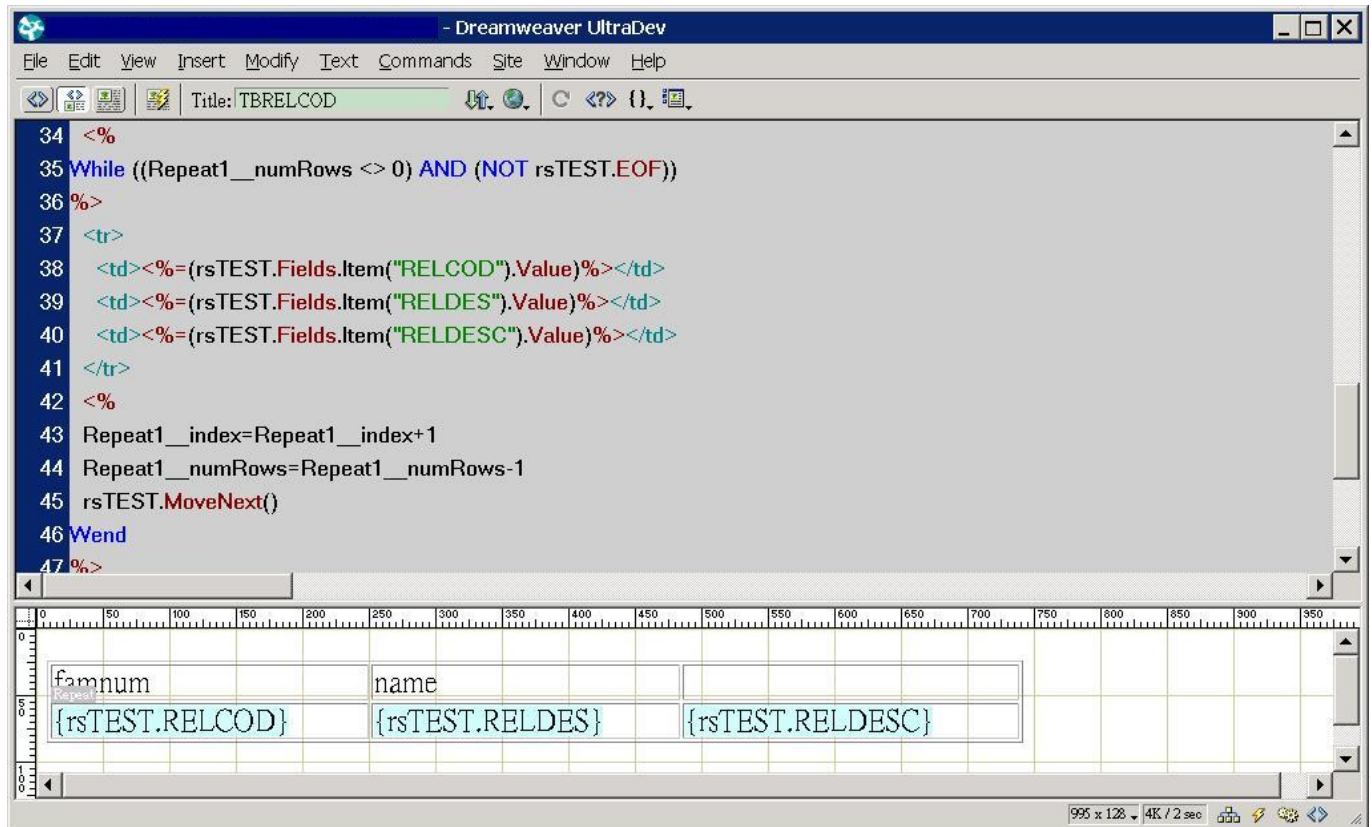
EJS is a simple templating language that lets you generate HTML markup with plain JavaScript. No religiousness about how to organize things. No reinvention of iteration and control-flow. It's just plain JavaScript.

One of the best feature of ASP.NET Web Forms is to enable user to place code in separate file, i.e. **code behind**. Source code and HTML code are cleanly separated.



While legacy ASP writes source code and HTML code side-by-side, i.e. **code beside**. Source code and HTML code can mess up easily.





index.ejs

```

. . .
<body>
  <%- include('header.ejs'); %>

  <%- include('create_edit.ejs'); %>
  <hr>
  <h3><a href="/task">所有工作</a></h3>
  <% if (locals.tasks.length > 0 ) { %>
    <table border="1">
      <thead>
        <tr>
          <th>工作類型</th>
          <th>工作日期</th>
          <th>申請表編號</th>
          <th>家團代表</th>
          <th>工作備註</th>
          <th></th>
          <th></th>
        </tr>
      </thead>
      <tbody><% locals.tasks.forEach( t => { %>
        <tr>
          <td><%= t.tatype %></td>
          <td><%= t.tadate %></td>
          <td><%= t.taappnum %></td>
          <td><%= t.tafamrep %></td>
          <td><%= t.taremark %></td>
          <td>

```

```

        <form action="/task?id=<%= t._id %>" method="GET">
            <input type="hidden" name="id" value="<%= t._id
%>">

            <button type="submit">Edit</button>
        </form>
    </td>
    <td>
        <form action="/task/delete/<%= t._id %>?
_method=DELETE" method="POST">
            <button type="submit" onclick="return confirm('Are
you sure?')">Delete

            </button>
        </form>
    </td>
</tr>
<% }) %>
</tdody>
</table>
<% } else { %>
    <p>沒有工作</p>
<% } %>

<%- include('footer.ejs'); %>
</body>
. . .

```

## V. PM2 - Get Job Done!

Since Node.js is built on top of the V8 JavaScript engine, it is single-threaded. So our app logic runs on one thread and hence doesn't fully utilize the available system resources.

This might not be a serious problem for event-based I/O calls, because the event loop is efficient enough for most use cases at moderate traffic loads. The problem occurs when we have a computationally intensive logic that might potentially block the thread or when we have a sudden spike in our traffic that could potentially increase the latency of our services.

http.js

```

const http = require('http')

const server = http.createServer((req, res) => {
  console.log(req.url)
  if (req.url === '/') {
    res.end('Home page')
  }
  else
  if (req.url === '/about') {
    console.time()
    for (let i=0; i< 1000; i++)
      for (let j=0; j < 1000; j++)
        console.log(`i=${i}, j=${j}`)
  }
})

```

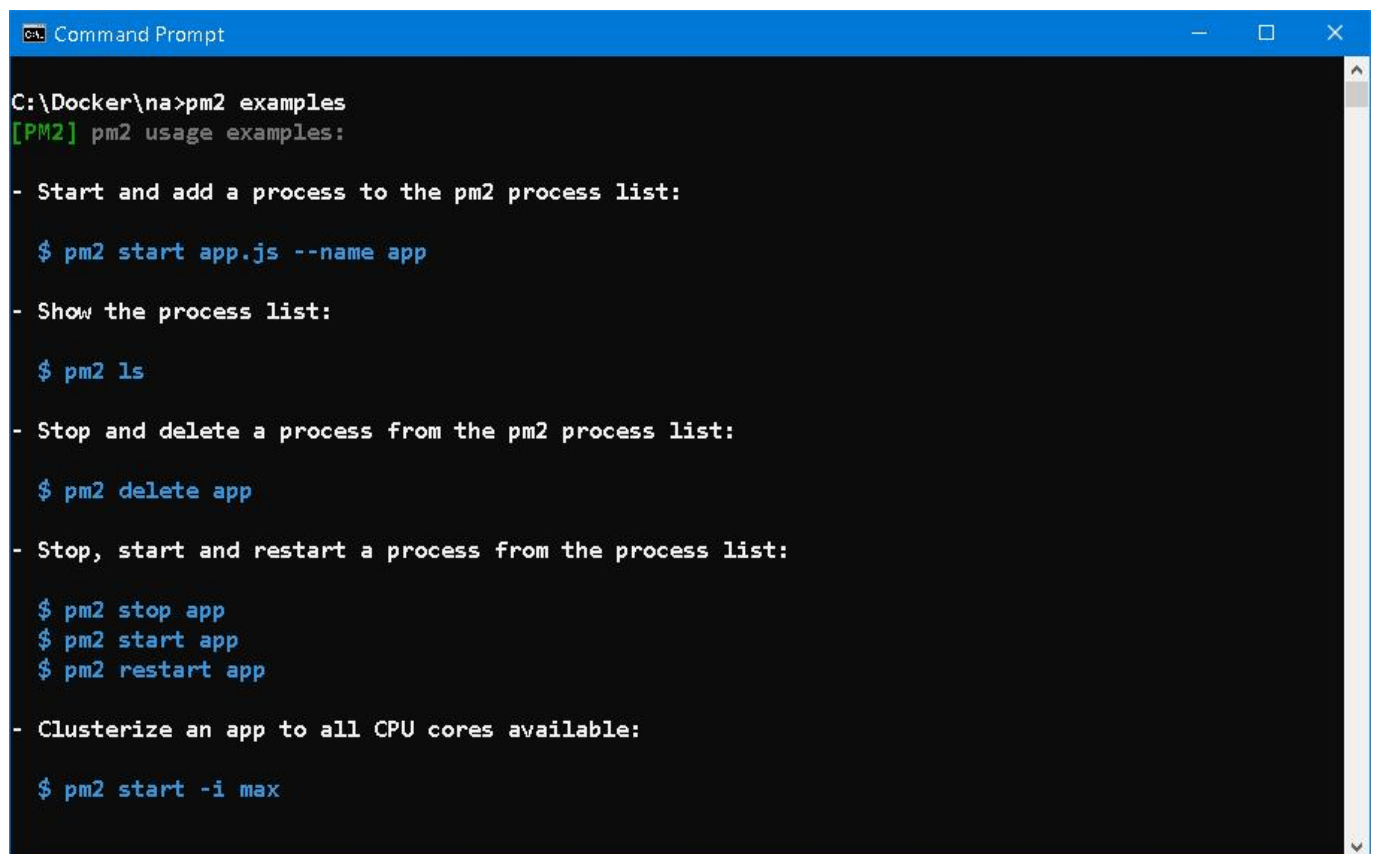


```
        console.timeEnd()  
        res.end('About page')  
    }  
    else  
        res.end('Error page')  
    })  
  
    server.listen(5000)
```

A request to `/about` will block subsequent `/` request.

PM2 internally uses the Node.js cluster module, but everything including the edge cases is handled for us and we don't even have to touch our existing code to get this working.

## Local server



```
Command Prompt  
C:\Docker\na>pm2 examples  
[PM2] pm2 usage examples:  
  
- Start and add a process to the pm2 process list:  
  $ pm2 start app.js --name app  
  
- Show the process list:  
  $ pm2 ls  
  
- Stop and delete a process from the pm2 process list:  
  $ pm2 delete app  
  
- Stop, start and restart a process from the process list:  
  $ pm2 stop app  
  $ pm2 start app  
  $ pm2 restart app  
  
- Clusterize an app to all CPU cores available:  
  $ pm2 start -i max
```

Start and daemonize an app, with a name and keeping watching on source changes (default).

```
pm2 start server.js --name na --watch
```

Stop a process

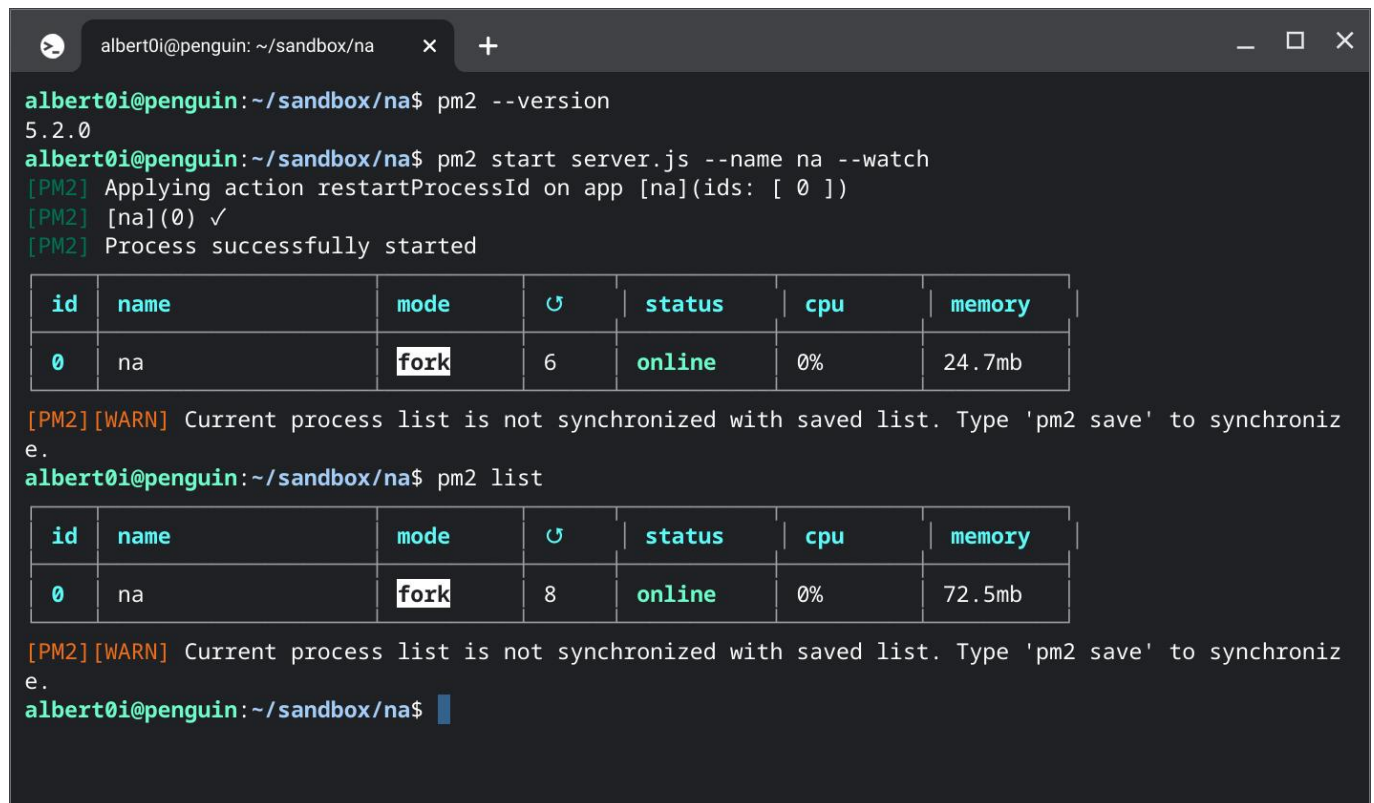
```
pm2 stop na
```

Restart a process

```
pm2 restart na
```

List all processes

```
pm2 list
```



```
albert0i@penguin: ~/sandbox/na x +
albert0i@penguin:~/sandbox/na$ pm2 --version
5.2.0
albert0i@penguin:~/sandbox/na$ pm2 start server.js --name na --watch
[PM2] Applying action restartProcessId on app [na](ids: [ 0 ])
[PM2] [na](0) ✓
[PM2] Process successfully started
```

id	name	mode	↻	status	cpu	memory
0	na	fork	6	online	0%	24.7mb

```
[PM2] [WARN] Current process list is not synchronized with saved list. Type 'pm2 save' to synchroniz
e.
albert0i@penguin:~/sandbox/na$ pm2 list
```

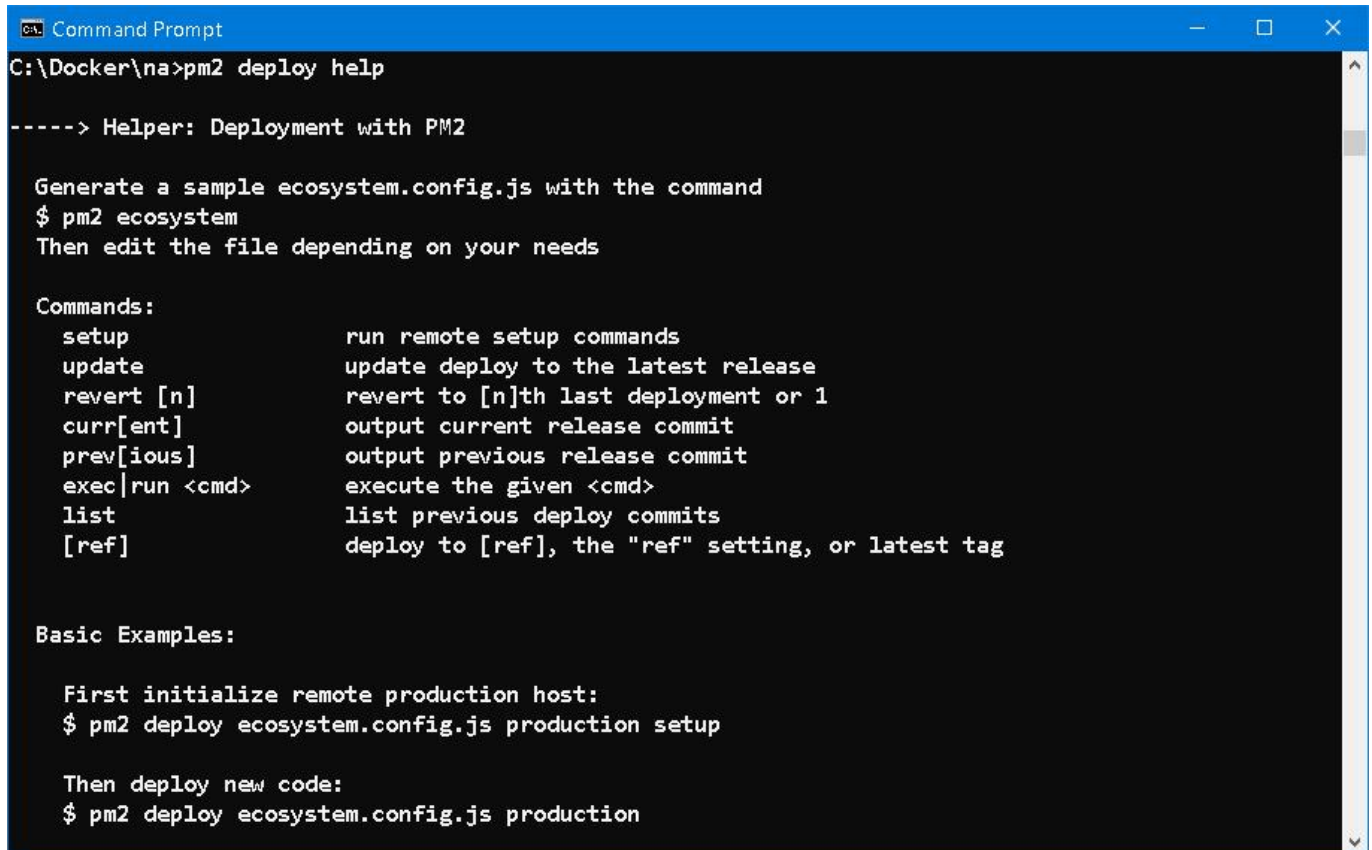
id	name	mode	↻	status	cpu	memory
0	na	fork	8	online	0%	72.5mb

```
[PM2] [WARN] Current process list is not synchronized with saved list. Type 'pm2 save' to synchroniz
e.
albert0i@penguin:~/sandbox/na$
```

Stop and delete a process from pm2 process list

```
pm2 delete na
```

Remote server



```
C:\Docker\na>pm2 deploy help

-----> Helper: Deployment with PM2

Generate a sample ecosystem.config.js with the command
$ pm2 ecosystem
Then edit the file depending on your needs

Commands:
  setup          run remote setup commands
  update         update deploy to the latest release
  revert [n]     revert to [n]th last deployment or 1
  curr[ent]      output current release commit
  prev[ious]     output previous release commit
  exec|run <cmd> execute the given <cmd>
  list          list previous deploy commits
  [ref]         deploy to [ref], the "ref" setting, or latest tag

Basic Examples:

First initialize remote production host:
$ pm2 deploy ecosystem.config.js production setup

Then deploy new code:
$ pm2 deploy ecosystem.config.js production
```

## Setup

```
pm2 deploy production setup
```

## Edit `ecosystem.config.js`

```
module.exports = {
  apps : [{
    name: "na",
    script: 'server.js',
    instances: 4,
    exec_mode: "cluster",
    watch: '.',
    env_production: {
      NODE_ENV: "production"
    }
  }],
  deploy : {
    production : {
      key: "../ssh-key-2022-05-09(2).pem",
      user : 'ubuntu',
      host : '140.238.40.147',
      ssh_options: "StrictHostKeyChecking=no",
      ref : 'origin/main',
      repo : 'https://github.com/Albert0i/na.git',
      path : '/home/ubuntu/na',
      'pre-setup' : '',
    }
  }
}
```

```
    'post-setup' : '',
    'pre-deploy-local': '',
    'pre-deploy' : '',
    'post-deploy' : 'npm install && pm2 reload ecosystem.config.js --env
production '
  }
}
};
```

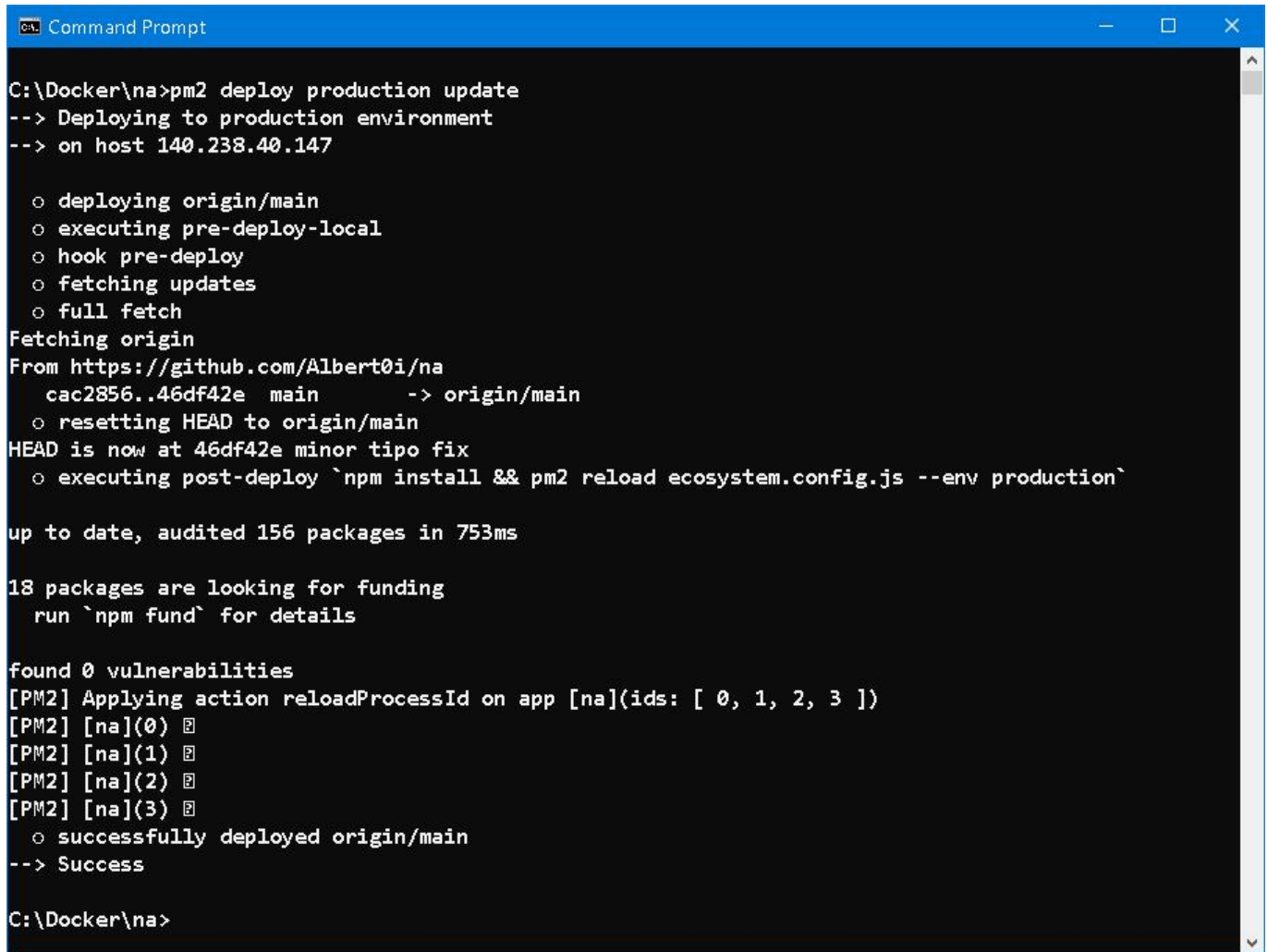
1. `exec_mode: "cluster"` tells PM2 to use the Node.js cluster module to execute the code.
2. `instances:0` when this is set to 0, PM2 will automatically spawn a number of child processes that are equal to the available number of cores. Respawning on termination is handled out of the box. It is important to note that, we could spawn more child processes than the number of CPU cores available, but that wouldn't be a good idea, because it would create a scheduling overhead and in that way, we might end up doing worse than better.
3. `"post-deploy": "npm install && pm2 reload ecosystem.config.js -env production"` the key thing to note here is pm2 reload which will make a zero-downtime deployment by adopting a rolling deployment approach where the instances are stoped and launched with the latest changes one after the other.
4. It is also interesting to note that the config can have multiple Node applications within the app array (think of a **microservices** scenario).

## Deploy

```
pm2 deploy production
```

## Update

```
pm2 deploy production update
pm2 deploy production update --force
```



```
C:\Docker\na>pm2 deploy production update
--> Deploying to production environment
--> on host 140.238.40.147

  ◦ deploying origin/main
  ◦ executing pre-deploy-local
  ◦ hook pre-deploy
  ◦ fetching updates
  ◦ full fetch
Fetching origin
From https://github.com/Albert0i/na
   cac2856..46df42e  main    -> origin/main
  ◦ resetting HEAD to origin/main
HEAD is now at 46df42e minor tipo fix
  ◦ executing post-deploy `npm install && pm2 reload ecosystem.config.js --env production`

up to date, audited 156 packages in 753ms

18 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
[PM2] Applying action reloadProcessId on app [na](ids: [ 0, 1, 2, 3 ])
[PM2] [na](0) 𐀀
[PM2] [na](1) 𐀀
[PM2] [na](2) 𐀀
[PM2] [na](3) 𐀀
  ◦ successfully deployed origin/main
--> Success

C:\Docker\na>
```

List previous deploy commits

```
pm2 deploy production list
```

Execute the given command

```
pm2 deploy production exec pm2 list
```

## VI. Summary

PM2 is a daemon process manager that will help you manage and keep your application online. Getting started with PM2 is straightforward, it is offered as a simple and intuitive CLI, installable via NPM.

Previously, we've talked about how to setup [Docker swarm on Oracle Cloud \(Free Tier\)](#) and [deploy](#) a NodeJS app on it. As a lightweight alternative, PM2 can be used to launch your app or deploy your code to remote host.

Links online

[http](#)

[https](#)

## VII. Reference

1. [Web Forms in the Age of .NET 5/6+: Planning for the Long Term](#)
2. [MVC vs Web Forms](#)
3. [ASP.NET Web API - PUT & DELETE Verbs Not Allowed - IIS 8](#)
4. [MongoDB Data Validator: How to Use the JSON Schema Validator](#)
5. [Schema Validation icons](#)
6. [Scaling Node.js Applications With PM2 Clusters](#)
7. [PM2 one click, multiple servers deploy and publish Node.js project at the same time!](#)
8. [PM2 | Listening on port 80 w/o root](#)
9. [Enabling HTTPS on express.js](#)
10. [Limiting Node.js API calls with express-rate-limit](#)
11. [Node.js vs .NET Core: What to Choose in 2022](#)
12. [Web server implementations in ASP.NET Core](#)
13. [Get started with Bootstrap](#)
14. [Internet Explorer 11 compatibility solution for Bootstrap 5](#)
15. [Markdown Guide | Basic Syntax](#)

## VIII. Appendix

.env file

```
# Server port
PORT = 3000

# Server port (optional https)
PORT_HTTPS = 3443

# Server limit
MAX_REQUEST_PER_MINUTE = 100

# MongoDB
MONGODB_URI = "mongodb+srv://your_mongodb_server_url"

# Redis (optional)
REDIS_URI = "redis://your_redis_server_url"
```

source tree

```
tree -L 3 -I node_modules

.
├── README.md
└── controller
```



```
|   ├── taskController.js
|   └── userController.js
├── ecosystem.config.js
├── img
|   ├── Naomi_Party_in_TAIPEI.jpg
|   ├── asp-era-1.JPG
|   ├── code_behind_beside.JPG
|   ├── npm.JPG
|   ├── pm2_deploy_help.JPG
|   ├── pm2_deploy_production_update.JPG
|   ├── pm2_examples.JPG
|   ├── pm2_list.JPG
|   └── ta2.JPG
├── middleware
|   └── auth.js
├── model
|   ├── Option.js
|   ├── Task.js
|   └── User.js
├── package-lock.json
├── package.json
├── public
|   ├── 404.png
|   ├── favicon.ico
|   └── styles.css
├── routes
|   ├── taskRoute.js
|   └── userRoute.js
├── server.js
├── ssl
|   ├── cert.pem
|   └── key.pem
└── views
    ├── 404.ejs
    ├── partials
    |   ├── footer.ejs
    |   └── header.ejs
    ├── task
    |   ├── create_edit.ejs
    |   ├── index.ejs
    |   └── options.ejs
    └── user
        ├── login.ejs
        └── signup.ejs
```

promise.js

```
const myFunc = (param) => {
  return new Promise((resolve, reject) => {
    if (param)
      resolve('success');
    else
```

```
        reject('fail');
    });
}

myFunc(true)
  .then(data => {console.log(data)})
  .catch(err => {console.log(err)})

const myFuncSync = async (param) => {
  try { return await myFunc(param) }
  catch (err) { console.error(err); }
}
myFuncSync(false)
```

output

```
success
fail
```

EOF (2022/08/31)