Instance Node.js

Simple Hosting Node.js vous permet d'héberger des applications [Node.js](https://nodejs.org/) qui montent en échelle simplement et de manière économique.

Prérequis

* Connaissances basiques de Node.js et du protocole HTTP
* Votre site doit utiliser NPM pour gérer ses dépendances

Support des versions

Version des instances

Deux versions de l'instance Node.js sont disponibles actuellement :

* Node.js recommandé, avec PostgreSQL 9.4, MySQL 5.6 (Percona) ou MongoDB 2.4
* Node.js avec PostgreSQL 9.2, MySQL 5.5 ou MongoDB 2.4

**Node.js recommandé**

Pour [créer une nouvelle instance via le site Web](https://www.gandi.net/hosting/simple/create), nous vous recommandons de choisir entre :

* Node.js / MongoDB 2.4 (recommandé)
* Node.js / PostgreSQL 9.4 (recommandé)
* Node.js / MySQL 5.6 (Percona) (recommandé)

Via [Gandi CLI](https://github.com/Gandi/gandi.cli) ou l' [API](http://doc.rpc.gandi.net/paas/usage.html) :

* nodejsmongodb2.4
* nodejspgsql9.4
* nodejsmysql5.6

**Node.js legacy**

Nous ne recommandons plus la sélection les instances suivantes à la création :

* Node.js / MongoDB 2.4 (CLI / API : nodejsmongodb)
* Node.js / PostgreSQL 9.2 (CLI / API: nodejspgsql)
* Node.js / MySQL 5.5 (CLI / API: nodejsmysql)

Versions supportées

**Node.js 4 LTS** est installé par défaut et supporté sur les nouvelles instances Node.js

Vous pouvez néanmoins choisir votre propre version de Node.js et de npm en configuration le fichier package.json.

Pour choisir la version de Node.js à utiliser pour exécuter votre application, voir [Sélection de la version de Node.js](https://wiki.gandi.net/fr/simple/instance/nodejs#selection_de_la_version_de_nodejs).

Sur les instances précédentes, nous supportons les versions 0.12.x (version par défaut), 0.10.x et 0.8.x de Node.js et vous pouvez également utiliser la version de votre choix (mais pas de npm).

Version de votre choix

Vous pouvez installer d'autres versions de Node.js, à partir du moment où elle est supportée et distribuée par [Node Version Manager ("nvm")](https://github.com/creationix/nvm).

Veuillez consulter la section [Sélection de la version de Node.js](https://wiki.gandi.net/fr/simple/instance/nodejs#selection_de_la_version_de_nodejs) pour en savoir plus.

Fonctionnement général

Votre application Node.js doit être composée au minimum d'un fichier nommé server.js. Ce fichier sera exécuté par l'interpréteur Node.js lors du démarrage de votre instance ou lors d'un déploiement avec l'accés Git. Afin de recevoir les requêtes HTTP et HTTPS, votre application doit écouter sur le port spécifié par la variable d'environnement *PORT* (process.env.PORT).

Vhosts

Contrairement aux instances PHP, chaque création de vhost ne crée pas un dossier spécifique sur votre disque. Le dossier nommé default présent dans vhosts permet de gérer les données pour votre instance Node.js. Une seule instance de votre application est exécutée, et doit gérer tous les vhosts que vous avez associé à votre instance Simple Hosting.

Exemples

Exemple d'une simple application Node.js affichant le nom du vhost

*server.js*

var http = require('http');

http.createServer(function (req, res) {

res.writeHead(200, {'Content-Type': 'text/plain'});

res.write('Hello Node.js!\n');

if ('host' in req.headers) {

var host = req.headers['host'];

res.write('Vhost is ' + host.split(':')[0] + '\n');

} else {

res.write('No vhost specified\n');

}

res.end();

}).listen(process.env.PORT);

Exemple vhost avec express.js : <https://github.com/visionmedia/express/blob/master/examples/vhost/index.js> [Annexe1]

Démarrage

Simple Hosting supporte 3 façons de démarrer une application Node.js.

La plateforme essayera de démarrer votre application suivant cette séquence :

1. Depuis le “start script” défini dans le fichier *package.json*
2. Depuis le champs “main” du fichier *package.json*
3. Depuis un fichier *server.js* s'il existe à la racine de l'application

L'ordre de démarrage a changé en Décembre 2015. Avant, les applications étaient démarrées en priorité à partir du fichier *server.js*.

Avec package.json (recommandé)

La manière “standard” de démarrer une application Node.js est d'utiliser les instructions définies dans package.json.

Alors que Simple Hosting ne supportait pas cette fonctionnalité originalement, elle est désormais la manière recommandée de démarrer votre application.

Le fichier *package.json* doit être placé à la racine de votre application. Il doit contenir un champ *package.json[“main”]* pour définir un point d'entrée, ou contenir une commande de démarrage dans le champ *package.json[“scripts”][“start”]*.

La plupart des applications Node.js pré-packagées, comme par exemple le moteur de blogs [Ghost](http://wiki.gandi.net/tutorials/simple/nodejs/ghost), utilisent déjà cette convention et il est donc plus facile de les installer.

**Définir un point d'entrée**

Vous pouvez définir un point d'entrée dans le champs [*main*](https://docs.npmjs.com/files/package.json#main) de *package.json*.

Il vous suffit d'introduire le nom d'un fichier placé à la racine de votre application pour que Simple Hosting l'utilise pour démarrer votre application.

Par exemple, pour démarrer à partir du fichier “index.js”:

*package.json*

{

"name" : "foo",

"version" : "1.2.3",

"description" : "A packaged foo fooer for fooing foos",

"main" : "index.js",

}

**Définir une commande de démarrage**

Vous pouvez définir vous-même la commande de démarrage dans le champs [*['scripts']['start']*](https://docs.npmjs.com/misc/scripts#default-values).

Simple Hosting va simplement éxecuter cette commande pour démarrer votre application.

L'exemple suivant est équivalent à l'exemple de “point d'entrée” décrit plus haut :

*package.json*

{

"name" : "foo",

"version" : "1.2.3",

"description" : "A packaged foo fooer for fooing foos",

"scripts": {

"start": "node index.js"

}

}

Le script de démarrage est très flexible et vous permet de personnaliser la manière de démarrer votre application.

Par exemple, vous pouvez activer le support ES6 en lançant Node.js en [mode harmony](https://github.com/nodejs/node-v0.x-archive/wiki/es6-(a.k.a.-harmony)-features-implemented-in-v8-and-available-in-node).

*package.json*

{

"name" : "foo-ecma-script",

"version" : "6",

"description" : "ES6 on Simple Hosting",

"engines": {

"node": ">=0.12"

},

"scripts": {

"start": "node --harmony index.js"

}

}

Vous pouvez également utiliser cette fonctionnalité avec des variables d'environnement pour démarrer votre application avec des gestionnaires de processus tels que [pm2](https://github.com/Unitech/pm2). Ou encore pour expérimenter avec les threads grâce à [libuv](https://github.com/libuv/libuv).

Avec server.js

La manière la plus simple de créer une application Node.js sur Simple Hosting est de placer un fichier nommé *server.js* à la racine de votre projet.

Cette méthode était utilisée par défaut jusqu'en Décembre 2015.

Variables d'environnement

Vous pouvez utiliser les variables d'environnement pour lire ou définir des options de configuration, au niveau de la plateforme ou de votre application.

Voici quelques exemples de variables d'environnement définies par Simple Hosting :

* PORT=8080
* NODE\_ENV=production
* WEB\_MEMORY=128
* WEB\_CONCURRENCY=2

Les valeurs de *WEB\_MEMORY* et *WEB\_CONCURRENCY* dépendent de la taille de votre instance (S, M, L, XL, XXL). La première correspond à la valeur maximale. La dernière peut être modifiée selon vos critères, la valeur par défaut étant celle que nous recommandons.

Lire les variables d'environnement

Vous pouvez accéder aux variables d'environnement normalement, c'est-à-dire depuis *process.env*.

Par exemple :

var port = process.env.PORT;

console.log(process.env.PORT);

// 8080

Définir des variables d'environnement

Vous pouvez définir ou réecrire des variables d'environnement depuis votre application ou depuis la commande de démarrage.

Depuis votre application, il suffit de définir les variables en utilisant *process.env*. Par exemple :

process.env.NODE\_ENV = "staging";

console.log(process.env.NODE\_ENV); // => "staging"

Au démarrage, pour pouvez utiliser le champs *package.json['scripts']['start']* pour définir une ou plusieurs variables d'environnement au style shell.

Par exemple :

*package.json*

{

...

"scripts": {

"start": "NODE\_ENV=staging node index"

}

}

Assurez-vous de ne pas écraser la variable d'environnement *PORT*, puisque vous pourriez perdre la connectivité avec la plateforme. De manière générale, veillez à ne pas écraser des variables d'environnement importantes avant de lancer votre application.

Installation des dépendances de votre site

La plateforme Simple Hosting Node.js se charge lors d'un déploiement via l’accès Git, d'installer les dépendances de votre site. Pour cela vous devez spécifier les dépendances dans le fichier “package.json” à la racine du vhost, soit dans 'vhost/default' :

*package.json*

{

"name": "website",

"version": "0.0.1",

"dependencies": {

"express": "3.x"

}

}

Lors de l'installation, 'npm install' est lancé avec le paramètre '–production' par défaut. Il n'est pas possible de modifier ce paramètre

Sélection de la version de Node.js

Vous pouvez utiliser n'importe quelle version Node.js et de npm supportées par [Node Version Manager (nvm)](https://github.com/creationix/nvm).

Il vous suffit de définir les numéros des versions désirées dans les [propriétés engines["node"] et engines["npm"]](https://npmjs.org/doc/json.html#engines) .

{

"engines": {

"node": "~ 4.0",

"npm": "\*"

}

}

Les versions demandées seront installées pendant [la phase de build](https://wiki.gandi.net/fr/simple/git), après avoir poussé le code de votre projet sur votre instance avec git et avoir lancé la commande “deploy” via SSH.

Node.js legacy

**Fichier .nvmrc**

Simple Hosting peut installer la version de Node.js de votre choix grâce à Node Version Manager (nvm).

Créez un fichier nommé ”.nvmrc” à la racine de votre projet et écrivez-y la version désirée. **Quand vous déploierez votre code**, Simple Hosting détectera l'existence de ce fichier (cela ne fonctionnera pas si vous créez simplement le fichier via sFTP ou SSH). Si nécessaire, la plateforme installera la version de Node.js détectée avec “nvm” avant d'installer les dépendances de votre application.

Exemple :

$ echo "4.4" > .nvmrc

$ cat .nvmrc

4.4

La présence du fichier ”.nvmrc” est prioritatire par rapport à l'utilisation du champ “engines” dans package.json, qui sera ignoré si le fichier existe.

Les binaires Node.js téléchargés avec **nvm** ne sont pas vérifiés ou supportés par Gandi. Nous vous prions de faire vous-mêmes les validations de stabilité et sécurité que vous jugerez nécessaires.

**Package.json**

Pour utiliser une des [versions de Node.js qui sont pré-installées sur votre instance](https://wiki.gandi.net/fr/simple/instance/nodejs#support_des_versions), il suffit de spécifier dans le fichier “package.json“ la version vous souhaitez utiliser dans le champ “engines“.

Par exemple :

*package.json*

{

"name": "website",

"version": "0.0.1",

"engines": {

"node": "< 0.10"

}

}

L'exemple ci-dessus forcera l'utilisation de la version 0.8.x de Node.js. Pour plus d'information sur le champ “engines“, voir <https://npmjs.org/doc/json.html#engines> [en].

Websockets

Il est désormais possible d'utiliser les Websockets sur les instances NodeJS (en Beta) !

Reportez vous à la page <http://wiki.gandi.net/fr/simple/websockets> pour plus d'informations.

Logs

Les sorties standard et d'erreur de votre site sont redirigées vers un fichier de log sur votre disque :

* via la console SSH : /srv/data/var/log/www/nodejs.log
* via SFTP : /lamp0/var/log/www/nodejs.log

Ce fichier de log contient également la sortie du programme NPM chargé d'installer les dépendances de votre site. En complément vous pouvez consulter l'historique des actions effectuées par le programme chargé de démarrer votre site :

* via la console SSH : /srv/data/var/log/www/nodejs-watchd.log
* via SFTP : /lamp0/var/log/www/nodejs-watchd.log

Vous pourrez surveiller si votre application Node.js a correctement démarré.

Base de données

MySQL

connection: {

host: '127.0.0.1',

socketPath: '/srv/run/mysqld/mysqld.sock',

user: 'user',

password: 'pwd',

database: 'db',

charset: 'utf8'

}

PostgreSQL

Pour le moment l'instance Node.js vient avec la base de données PostgreSQL. Notez que PostgreSQL9.2 ajoute le support le [type de données JSON](http://wiki.postgresql.org/wiki/What%27s_new_in_PostgreSQL_9.2#JSON_datatype); c'est à dire que vous pouvez désormais directement stocker des données au format JSON. Vous pouvez aussi manipuler des tables existantes et obtenir le résultat au format JSON grâce aux fonctions array\_to\_json and row\_to\_json.

Exemple complet avec utilisation de la base de données :

*server.js*

var http = require('http');

var pg = require('pg');

var conString = "tcp://pguser:password@localhost/mycms";

function list\_articles(callback) {

var client = new pg.Client(conString);

client.connect(function(err) {

if (err) {

callback(null, err);

} else {

client.query('SELECT \* FROM articles ORDER BY art\_id DESC;',

callback);

}

});

}

function handle\_error(res) {

res.writeHead(500, {'Content-Type': 'text/plain'});

res.end('Internal Server Error\n');

}

function write\_response(res, result) {

res.writeHead(200, {'Content-Type': 'text/plain'});

res.write('Number of articles: ' + result.rowCount + '\n');

result.rows.forEach(function(row) {

res.write(row.art\_id + ' | ' + row.art\_title + '\n');

});

res.end();

}

server = http.createServer(function(req, res) {

list\_articles(function(err, result) {

if (err) {

console.log(err);

handle\_error();

} else {

write\_response(res, result);

}

});

});

server.listen(process.env.PORT);

*package.json*

{

"name": "website",

"version": "0.0.1",

"dependencies": {

"pg": "0.x"

}

}

MongoDB

Il est désormais possible de créer des instances Node.js/mongoDB.

Problèmes connus

PostgreSQL

L'installation du module *pg* ne fonctionne pas en utilisant la version de npm livrée avec Node.js 0.10.x. Il est pour l'instant conseillé de sélectionner la version 0.8.x de Node.js, en suivant les instructions de la section [Sélection de la version de Node.js](https://wiki.gandi.net/fr/simple/instance/nodejs#selection_de_la_version_de_nodejs).

Lors de l'installation du module *pg*, les messages d'erreurs suivants apparaissent :

gyp WARN EACCES user "root" does not have permission to access the dev dir "/srv/data/.node-gyp/0.8.22"

gyp WARN EACCES attempting to reinstall using temporary dev dir "/srv/data/tmp/.node-gyp"

You need to install postgresql-server-dev-X.Y for building a server-side extension or libpq-dev for building a client-side application.

gyp: Call to 'pg\_config --libdir' returned exit status 1. while trying to load binding.gyp

gyp ERR! configure error

gyp ERR! stack Error: `gyp` failed with exit code: 1

gyp ERR! stack at ChildProcess.onCpExit (/usr/lib/node\_modules/npm/node\_modules/node-gyp/lib/configure.js:416:16)

gyp ERR! stack at ChildProcess.EventEmitter.emit (events.js:99:17)

gyp ERR! stack at Process.\_handle.onexit (child\_process.js:678:10)

gyp ERR! System Linux 3.4.7-grsec-paas-19c573d

gyp ERR! command "node" "/usr/lib/node\_modules/npm/node\_modules/node-gyp/bin/node-gyp.js" "rebuild"

gyp ERR! cwd /srv/data/web/vhosts/default/node\_modules/pg

gyp ERR! node -v v0.8.22

gyp ERR! node-gyp -v v0.8.5

gyp ERR! not ok

Le module *pg* est quand même installé, mais limité à l'implémentation en Javascript. La correction est en cours.

Voir aussi

* [NPM](https://npmjs.org/)
* [Node.js](http://nodejs.org/)
* [express.js](http://expressjs.com/)
* [le format JSON](https://npmjs.org/doc/json.html)
* [accès Git pour les instances Node.js](https://wiki.gandi.net/fr/simple/git#les_instances_nodejs)
* [le type de données JSON dans PostgreSQL](http://wiki.postgresql.org/wiki/What%27s_new_in_PostgreSQL_9.2#JSON_datatype)

ANNEXE1

|  |  |
| --- | --- |
|  | /\*\* |
|  | \* Module dependencies. |
|  | \*/ |
|  |  |
|  | var express = require('../..'); |
|  | var logger = require('morgan'); |
|  | var vhost = require('vhost'); |
|  |  |
|  | /\* |
|  | edit /etc/hosts: |
|  |  |
|  | 127.0.0.1 foo.example.com |
|  | 127.0.0.1 bar.example.com |
|  | 127.0.0.1 example.com |
|  | \*/ |
|  |  |
|  | // Main server app |
|  |  |
|  | var main = express(); |
|  |  |
|  | if (!module.parent) main.use(logger('dev')); |
|  |  |
|  | main.get('/', function(req, res){ |
|  | res.send('Hello from main app!'); |
|  | }); |
|  |  |
|  | main.get('/:sub', function(req, res){ |
|  | res.send('requested ' + req.params.sub); |
|  | }); |
|  |  |
|  | // Redirect app |
|  |  |
|  | var redirect = express(); |
|  |  |
|  | redirect.use(function(req, res){ |
|  | if (!module.parent) console.log(req.vhost); |
|  | res.redirect('http://example.com:3000/' + req.vhost[0]); |
|  | }); |
|  |  |
|  | // Vhost app |
|  |  |
|  | var app = module.exports = express(); |
|  |  |
|  | app.use(vhost('\*.example.com', redirect)); // Serves all subdomains via Redirect app |
|  | app.use(vhost('example.com', main)); // Serves top level domain via Main server app |
|  |  |
|  | /\* istanbul ignore next \*/ |
|  | if (!module.parent) { |
|  | app.listen(3000); |
|  | console.log('Express started on port 3000'); |
|  | } |