Thomas Parisot

Node.js

Apprendre par la pratique

EYROLLES

Plus de 500 exemples pour apprendre en pratiquant

La plate-forme Node.js est passée du statut d'OVNI à celui d'incontournable. En 2018, elle fait fonctionner des applications web, de l'outillage front-end, de la distribution de fichiers pair-à-pair, des objets connectés et la NASA l'utilise même pour analyser les données télémétriques de ses sondes spatiales.

L'intention de cet ouvrage est d'être votre base de départ pour comprendre ce qu'il se passe dans cet écosystème en perpétuel mouvement. Il devrait vous permettre de faire des choix techniques durables et de vous rendre opérationnelle sur LA plate-forme JavaScript, le langage le plus populaire du monde selon GitHub.

Nous allons parcourir ensemble l'écosystème Node.js en partant de son historique et de sa gouvernance jusqu'au déploiement en continu de nos applications. Nous (re)partirons sur des bases solides de JavaScript pour ensuite découvrir petit à petit le potentiel des modules Node.js. Ces connaissances sont les fondations pour explorer la conception, le développement et le déploiement d'applications web, d'utilitaires en lignes de commandes et d'applications front-end modernes. Une sélection variée de modules complémentaires vous permettra d'y voir plus clair parmi les centaines de milliers de contributions de la communauté Node.js.

Plusieurs centaines d'exemples ponctuent l'ouvrage pour mieux cerner les concepts et leurs résultats. Ils sont exécutables dans un terminal grâce à un module npm ou dans un navigateur web grâce à une surcouche interactive.

À qui cet ouvrage s'adresse-t-il ?

- Aux développeurs et développeuses qui veulent se reconvertir depuis un autre langage de programmation comme Perl, PHP ou Ruby
- Aux développeurs et développeuses qui cherchent à tirer parti de l'outillage npm front-end
- À toute personne autodidacte et désireuse de renforcer ses compétences en programmation web

Au sommaire

Thomas Parisot est un autodidacte qui développe des applications web open source, et qui facilite des projets avec des pratiques de travail collaboratif ainsi qu'une approche centrée utilisateurs. Il utilise Node.js depuis la version 0.4 et baigne dans les technologies web depuis 1998.

Thomas façonne ses outils de travail au sein de la coopération d'innovation numérique dtc innovation depuis 2017. Auparavant, il a travaillé pendant quatre ans dans le département R&D de la BBC. Il a aussi co-fondé les conférences Sud Web et fait ses débuts professionnels à l'agence Clever Age Bordeaux.

Node.js

DANS LA MÊME COLLECTION

C. PIERRE DE GEYER, J. PAULI, P. MARTIN, É. DASPET. – **PHP 7 avancé**.
N°67720, 2e édition, 2018, 756 pages.

H. WICKHAM, G. GROLEMUND. – **R pour les data sciences**.
N°67571, 2018, 496 pages.

F. PROVOST, T. FAWCETT. – **Data science pour l'entreprise**.
N°67570, 2018, 370 pages.

J. CHOKOGOUE. – **Maîtrisez l'utilisation des technologies Hadoop**.
N°67478, 2018, 432 pages.

R. GOETTER. – **CSS 3 Flexbox**.
N°14363, 2016, 152 pages.

H. BEN REBAH, B. MARIAT. – **API HTML 5 : maîtrisez le web moderne !**
N°67554, 2018, 294 pages.

W. MCKINNEY. – **Analyse de données en Python**.
N°14109, 2015, 488 pages.

É. BIERNAT, M. LUTZ. – **Data science : fondamentaux et études de cas**.
N°14243, 2015, 312 pages.

SUR LE MÊME THÈME

É. SARRION. – **Programmation avec Node.js, Express.js et MongoDB**.
N°13994, 2014, 586 pages.

Thomas Parisot

Node.js

Apprendre par la pratique

EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

Mise en pages : Gaël Thomas

Avant-propos

Node.js – appelons-le **Node** dès à présent – est né dans le cerveau de Ryan Dahl, ancien étudiant sans le sou et au parcours informatique atypique. Son talent a été de s'obstiner à **résoudre un problème d'expérience utilisateur** fréquent sur le web : **l'attente devant un navigateur web figé**. Son but ? Rendre possible et facile la création de **barres de progression** dans les navigateurs web.

J'ai utilisé Node pour la première fois en 2010, par curiosité. Nous en étions alors à la version 0.4. Créer mon propre serveur HTTP était un concept qui semblait étrange, habitué que j'étais à écrire des applications PHP et à les mettre derrière un serveur Apache – nginx commençait à peine à décoller.

L'effet « wahou » était pourtant là : une installation en quelques secondes, quelques lignes de JavaScript et j'avais une API REST câblée avec une base de données *CouchDB* pour impressionner mes collègues.

Je bascule dans un **contexte startup** fin 2011 : nous sommes quatre développeurs avec des bagages différents. Nous voulions **un langage commun** pour le *back-end* et le *front-end*. Nous partons sur Node, installé sans encombre sur notre serveur dédié géré par l'hébergeur alwaysdata. Nous avions la **sensation de progresser rapidement**. Nous allions réellement vite.

Depuis, de nombreuses entreprises ont communiqué sur leur adoption de Node : Paypal, LinkedIn, eBay, Airbnb, British Gas, Allociné, The New York Times, Yahoo!, Microsoft, Mozilla, Flickr ou encore Twitter. Ces entreprises l'utilisent pour façonner leur outillage métier, gérer les transactions bancaires, leurs serveurs LDAP, des services ou sites web.

Pourquoi ce livre ?

Je trouve que **Node est un environnement élégant et agréable**. Je prends beaucoup de plaisir à l'utiliser au quotidien, mais je regrette l'accent placé sur la nouveauté permanente et la suringénierie des articles et ouvrages en langue française.

Node est mon outil de travail principal depuis 2011 et je fais quasiment tout avec – y compris l'outillage de publication de cet ouvrage. J'avais envie de communiquer sur la simplicité, l'architecture modulaire et la versatilité de Node pour que vous puissiez **progresser dans l'usage de JavaScript**, **façonner votre propre outillage** et pour mieux **utiliser les ressources des systèmes d'exploitation**.

Je vous propose donc un **contenu moins technique que d'ordinaire**, orienté sur l'apprentissage par la pratique et des concepts qui durent dans le temps. Cet ouvrage s'installe avec Node : tous les exemples sont réels et fonctionnent dans votre terminal.

Indirectement, je souhaite **déconstruire la posture d'expert et de lecteur** en me mettant à vos côtés pendant votre apprentissage. Une partie des contenus a été conçue et relue en ateliers collectifs pour que la progression se fasse de manière naturelle.

À qui s'adresse cet ouvrage ?

J'ai écrit cet ouvrage avec trois profils de lecteurs en tête : des personnes qui souhaitent **se mettre à Node sans savoir par où commencer**, celles qui font du *front-end* **et souhaitent mieux comprendre leur outillage** et celles qui veulent **renforcer leurs connaissances** en Node et JavaScript afin de changer de travail/métier.

Cet ouvrage s'adresse aussi à des lecteurs qui ne programment pas pour gagner leur vie mais qui veulent mieux comprendre de quoi parlent les développeurs et développeuses. Je pense que les interactions de travail sont de meilleure qualité quand nous comprenons les problématiques de celles et ceux avec qui nous travaillons.

Structure de l'ouvrage

Cet ouvrage se lit dans l'ordre de votre choix.

J'ai fait en sorte que la lecture soit progressive, du plus simple au plus complexe. Le contenu des chapitres explore une problématique et se termine souvent sur une section avancée, pour aller plus loin.

Cette édition est composée de neuf chapitres et d'une annexe :

1. **Histoire, écosystème et gouvernance**

   D'où vient Node et qui sont les acteurs participant à son histoire ?

2. **Installer, mettre à jour et développer**

   Être autonome pour installer Node.js sur notre ordinateur ou un serveur, se tenir au courant des mises à jour et jongler entre différentes versions. Découvrir des outils confortables pour écrire du code.

3. **Jouer avec JavaScript**

   Comprendre les différences entre JavaScript, ECMAScript, le DOM et Node.js. Revoir les bases du langage pour se sentir plus à l'aise.

4. **Jouer avec Node.js**
   S'exercer avec des scripts et des modules Node.js : les nôtres, ceux fournis par Node et ceux de la communauté.
5. **Jouer avec npm**
   Tirer parti de l'outil livré par défaut avec Node.js. S'amuser à créer nos propres outils dignes d'artisans du logiciel.
6. **Déployer notre code**
   Déployer notre code le plus tôt possible pour célébrer notre premier programme !
7. **Créer une application web**
   Créer une application web en partant de zéro, en utilisant un *framework* puis en parlant à une base de données. Bienvenue dans l'odyssée d'une requête HTTP !
8. **Créer un outil en ligne de commandes**
   Rendre du code métier fonctionnel dans un terminal ; la ligne de commandes ne paraîtra plus aussi austère.
9. **Créer une application** *front-end*
   Utiliser la richesse de l'écosystème npm et profiter des modules pour écrire des applications *front-end* de qualité.

L'annexe complète l'ouvrage avec une **sélection de modules** npm, pour démarrer plus vite et mieux cerner ce qu'on peut faire avec Node.

> **RESSOURCES** **Contenu en libre accès**
>
> Le contenu de cet ouvrage est actualisé en permanence sur https://oncletom.io/node.js/.
> L'intégralité du code source du livre et des exemples sont hébergés sur https://github.com/oncletom/nodebook.
> J'utilise la plate-forme collaborative GitHub pour recueillir vos avis, corrections et suggestions.

Remerciements

Cet ouvrage a été rédigé sur près de quatre années – longue est la liste des personnes à remercier. Longue aussi est la liste de celles et ceux à qui je demande pardon pour mes sautes d'humeur, mes absences ou mes passages à vide. L'écriture sur une longue durée est une activité que je trouve éprouvante. Elle nécessite un ajustement permanent vis-à-vis de moi-même pour ne pas (trop) en souffrir.

Mes premiers remerciements vont à Noémie. Ta joie, tes encouragements et ta présence sont essentiels à mon équilibre. Ils m'emplissent de confiance, m'inspirent et atténuent les peurs qui me paralysent.

Je remercie Karine, Laurène, Elsa et Alexandre des Éditions Eyrolles pour cette opportunité. Vous avez fait preuve d'une patience et d'une confiance infinies malgré le délai d'écriture.

Je tiens à remercier mes anciens collègues de BBC R&D pour leur soutien, leur confiance et leur tolérance aux jeux de mots. Olivier, Sean, Katie, Chris N et Chris L : c'était un plaisir et un honneur d'être à vos côtés.

Certain·e·s d'entre vous m'ont beaucoup aidé par leurs conseils, leurs relectures et leurs encouragements. Alors un grand merci pour cela à Béa, Claire, Clémentine, David, Efi, Fabien, Frank, Philippe, Stéphane et Thierry.

Merci à l'équipe d'Asciidoctor pour leur travail précieux et leurs réponses à mes innombrables questions. Je pense notamment à Guillaume et à Dan.

Merci à Antoine pour notre travail en duo sur l'écriture numérique. Je suis fier d'apporter ce mode d'écriture dans les communautés académiques, du design et des archivistes.

Enfin, je remercie toutes les personnes qui m'ont accueilli chez elles, dans leur maison, dans leur bureau ou dans leur communauté/*meet-up* pendant mon écriture itinérante en 2017 et 2018.

Bonne lecture,

Thomas

Table des matières

CHAPITRE 7
Créer une application web 249

CHAPITRE 8
Créer un outil en ligne de commandes 313

ANNEXE

Sélection de modules npm 417

1

Histoire, écosystème et gouvernance

Faisons le point sur ce qu'est Node pour mieux comprendre les discussions qui animent la communauté et d'où vient cette plate-forme qui fait tant parler d'elle.

- Bref historique
- Les raisons du succès
- Pourquoi éviter Node
- Pourquoi choisir Node
- L'écosystème des acteurs
- Gouvernance du projet

Node.js est-il un langage de programmation ? Node.js est-il un *framework* JavaScript ? Qu'en restera-t-il une fois la frénésie retombée ? Ce chapitre permet de **comprendre pourquoi Node a émergé** et comment. Surtout, il vous permettra de comprendre les choix techniques à l'origine des fondations de Node et ce que l'utiliser peut vous apporter, que ce soit dans un contexte personnel ou professionnel.

Node.js : un environnement d'exécution JavaScript

Node.js n'est pas un langage de programmation. Ce n'est pas non plus un *framework* JavaScript. **Node.js est un environnement d'exécution JavaScript**.

La différence entre ces trois désignations peut sembler subtile, futile voire inutile, mais le terme « environnement » est la véritable nature de Node.

Exécuter du JavaScript côté serveur n'est pas une révolution. Netscape Enterprise Server s'y est déjà essayé au début des années 1990, juste après son introduction dans le navigateur web Netscape Navigator.

En 1997, la société Netscape s'est attelée à créer Rhino (https://www.mozilla.org/rhino/), un environnement d'exécution JavaScript tournant sous Java et disponible sous licence libre. C'était un des projets liés à la réécriture de Netscape Navigator en Java. Si la société a depuis fermé ses portes, Rhino a entraîné l'émergence de projets utiles aux développeurs web.

Entre-temps, le langage JavaScript évolue, le Web 2.0 émerge des cendres de la première bulle Internet et d'autres initiatives voient le jour dans les années 2000 comme APE (*Ajax Push Engine*, http://ape-project.org/). Elles mettent également en œuvre JavaScript côté serveur. JavaScript était surtout un choix logique de partage de code entre client et serveur pour Comet, le précurseur des WebSockets.

GLOSSAIRE Comet

Comet est un terme regroupant les différentes tentatives techniques permettant à un serveur web d'envoyer des données à un client sans que celui-ci ne les ait demandées initialement.
Parmi ces techniques, on retrouve le *long polling*, consistant à conserver une connexion Ajax ouverte pendant la durée de vie d'une page web.

GLOSSAIRE WebSockets

WebSockets est un protocole basé sur TCP.
Il maintient une connexion HTTP active entre un client et un serveur et y fait transiter les données de manière bidirectionnelle.
Ce protocole sera probablement rendu obsolète par HTTP/2, le successeur de HTTP/1.1. HTTP/2 a été lancé par Google sous le nom de protocole SPDY (prononcer *speedy*).

Cas d'utilisation modernes de Rhino

Rhino est toujours utile dès qu'un projet Java implique du JavaScript.
Google l'utilise comme environnement d'exécution de ses https://gsuite-developers.googleblog.com/2012/11/using-open-source-libraries-in-apps.html. Ces scripts sont destinés à développer des extensions et des interactions supplémentaires pour les documents Google Drive.
Rhino est également employé dans YUI Compressor (https://yui.github.io/yuicompressor/), un optimiseur CSS et JavaScript créé par Yahoo, désormais surpassé par Closure Compiler (https://developers.google.com/closure/compiler/) et UglifyJS (https://npmjs.com/uglify-js). Ce dernier est écrit en JavaScript et repose sur Node. La boucle est bouclée.

Node représente un **environnement d'exécution** *(runtime)*, un **ensemble d'API JavaScript** ainsi qu'une **machine virtuelle (VM) JavaScript** performante (*parseur*, interpréteur et compi-

lateur) capable d'accéder à des ressources système telles que des fichiers *(filesystem)* ou des connexions réseau *(sockets)*.

Typiquement, une personne développant en Node écrit du code se basant sur les API à disposition. Ce code est lu par le *runtime* Node, qui le transmet à la VM JavaScript. Enfin, cette dernière traduit le programme en langage machine *(bytecode)* avant que le programme soit effectivement exécuté par le processeur.

Pour comprendre comment Node a opté pour cette approche, retournons en 2009, lorsque son créateur Ryan Dahl cherche à résoudre élégamment un problème de performance de programmation.

Bref historique

En 2006, Ryan Dahl est un étudiant américain en troisième année de doctorat de mathématiques. Son but initial était de devenir professeur dans cette matière, mais il prend la décision de ne pas terminer sa thèse et d'entreprendre un voyage au Chili.

Alors qu'il cherche à effectuer des petits boulots, il y rencontre une autre personne développant des sites web. Ruby on Rails connaît un succès grandissant et attire son attention. Alors que Ryan envisage d'utiliser Rails, il découvre avec horreur la lenteur du framework et cherche à en découvrir les causes.

Ryan débute alors sa quête des applications web performantes et découvre Mongrel, un serveur HTTP écrit en Ruby. Il est séduit par deux choses :

- la possibilité d'inclure un serveur HTTP comme bibliothèque applicative ;
- la simplicité de fonctionnement : recevoir une requête HTTP et décider soi-même de la réponse à apporter.

La quête initiale le dirige alors vers la possibilité de **créer un serveur web non bloquant** ; en d'autres termes, un serveur capable, dans un même processus, de traiter d'autres requêtes en attendant de renvoyer la réponse initiale.

Nous sommes alors en 2008 et le site de partage de photos Flickr innove avec un nouveau système de téléversement d'images : une **barre de progression** représentant le statut du téléversement remplace alors la page figée – effet inhérent à l'envoi de fichiers depuis un formulaire HTML.

C'est le déclic pour Ryan : bien que Mongrel ait déjà un *plug-in* pour cette fonctionnalité, il souhaite simplifier davantage le travail pour les développeurs. Il reproduit le mécanisme avec succès en C. Les développeurs web jugeant la solution trop complexe, Ryan tente la même approche avec d'autres langages, comme Python, Lua ou même Haskell. Il se heurte au sempiternel problème des ressources bloquantes des différents interpréteurs.

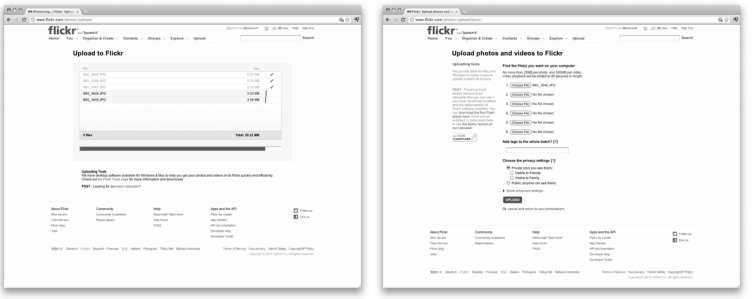


Figure 1–1
Interface du service Flickr après et avant l'introduction du téléversement progressif

> **LIEN Annonce du nouveau Flickr Uploadr**
>
> L'équipe d'ingénierie de Flickr explique dans un article (https://wp.me/p2DMyG-ok) comment elle a contourné le problème de l'*upload* via un formulaire classique. Ce problème gelait la fenêtre du navigateur pendant la durée du téléversement.

Le deuxième déclic se produit en janvier 2009, lorsque JavaScript émerge dans une discussion entre développeurs. *Eurêka !* La machine virtuelle JavaScript V8 de Google a été libérée en *open source* depuis quelques mois et Apple, Microsoft, Mozilla et Google se livrent à une course à la performance de leurs machines virtuelles respectives. En ligne de mire, il faut rendre la navigation sur mobile et sur ordinateur plus rapide et moins gourmande en ressources.

Ryan admet que JavaScript dispose des caractéristiques idéales, même s'il n'est pas un adepte du langage : fonctions anonymes, *closures* et l'*event loop* (dans le DOM en tout cas). Il manque juste aux machines virtuelles JavaScript la capacité d'accéder à des *sockets*, au système de fichiers et à d'autres fonctions système.

Ryan quitte alors son travail, s'inspire de ses travaux de modules non bloquants pour Nginx et s'affaire pendant six mois à marier JavaScript V8 et l'environnement du système d'exploitation. De ses efforts naît Node.js.

Il présente alors officiellement son travail (https://gist.github.com/ry/a3d0bbbff196af633995 et https://www.youtube.com/watch?v=ztspvPYybIY) qui suscite l'enthousiasme et attire l'attention.

L'entreprise américaine Joyent l'embauche à plein temps pour continuer le développement de Node. Ils pressentent que cet outil répondra à leurs projets de *datacenter* et d'hébergement. **Node et sa communauté se constituent** et continuent leur chemin depuis lors ; pour une simple affaire de barre de progression et une obsession pour la perception de rapidité.

En 2018, l'avenir de Node est au beau fixe avec plus de 2 000 contributeurs et plus de 782 000 modules publics hébergés sur le registre `npm`.

> VIDÉO **History of Node.js**
>
> La vidéo https://youtube.com/watch?v=SAc0vQCC6UQ est une présentation donnée par Ryan Dahl, le créateur de Node, au cours de l'année 2011. C'est la première fois qu'il intervient pour expliquer son parcours et la genèse du projet.
> Voilà peut-être un élément qui figurera dans les livres d'histoire !

Les raisons du succès

La tension et l'attention autour de JavaScript sont énormes en 2009. La mode du tout Ajax et des *mashups* s'est estompée, mais une chose en est ressortie : JavaScript n'a plus à rougir ni à être relégué au rang de sous-langage. Les initiatives JSLint, CommonJS et les *good parts* de Douglas Crockford sont pour beaucoup dans la création de code élégant.

D'un autre côté, les entreprises développant des navigateurs web se livrent à une féroce compétition d'optimisation. Google, Mozilla et Apple ont besoin de navigateurs rapides pour améliorer leurs parts de marché sur les ordinateurs, mais aussi les téléphones et tablettes. On peut considérer que JavaScript est à cette époque le langage de programmation bénéficiant du plus grand investissement financier et humain en R&D.

La communauté JavaScript accueille avec ferveur Node lors de la conférence JSConf Europe en 2009. Elle contribue à son amélioration et à la création d'un écosystème de modules réutilisables.

Il faudra attendre la création de `npm` au tout début 2010, qui a pour but d'héberger des modules Node et de faciliter leur installation. Dès lors, une simple commande suffit pour inclure dans nos projets du code écrit par d'autres personnes.

`npm` devient une pierre angulaire, à tel point qu'il est inclus dans l'installation de Node à partir de la version 0.6.3 en novembre 2011. La communauté Node fait le reste du travail en constituant un **écosystème de modules riches et variés** : frameworks web, pilotes de bases de données, serveurs HTTP, serveurs WebSockets, préprocesseurs CSS, CoffeeScript, *parseurs*, *proxy*, serveurs de log, modules de tests, langages de *templating*, etc.

Malgré ses défauts de jeunesse, Node réussit le tour de force de la performance. La recette de l'accès non-bloquant a-t-elle fonctionné ? À en croire les personnes ayant migré vers Node pour ces raisons, la réponse est « oui ».

Pourquoi éviter Node.js ?

Il est tentant de vouloir suivre un chemin populaire, d'adopter le dernier outil ou framework à la mode.

J'ai pourtant envie d'écrire qu'il n'est pas forcément nécessaire de passer à Node.

Si votre équipe dispose déjà de fortes compétences, d'aisance et de productivité dans un autre langage, il n'y a pas de raison de passer à Node. Cette équipe a tout intérêt à capitaliser sur ses connaissances pour être efficace et minimiser la dette technique de ses applications.

L'**offre logicielle est également à prendre en compte** : CMS, systèmes e-commerce ou autre application prête à l'emploi que la communauté Node n'offrirait pas à ce jour.

Un facteur important et souvent oublié est **l'acceptation et la compréhension de l'utilisation de Node** par une équipe. Il est alors plus intéressant de comprendre les raisons d'un blocage que de forcer ou d'imposer cet outil. La solution peut être simple : balayer des idées reçues, animer un atelier technique ou inviter un expert pour répondre aux questions, interrogations et utilité d'un tel changement.

Node ne vous aidera probablement pas si vous cherchez à **réaliser des choses contre-productives pour JavaScript**. Je pense à des opérations mathématiques de très haute précision, de l'apprentissage automatique avancé (*machine learning*) ou du calcul scientifique poussé par exemple. Il est difficile d'égaler la richesse fonctionnelle de Python et de ses bibliothèques (SciPy, NumPy, Scikit-learn) ou la finesse de gestion de mémoire de C++ ou de Rust.

Node ne résout pas les problèmes par magie. Cela reste avant tout une affaire de compétences et d'expérience.

Pourquoi choisir Node.js ?

Node est un choix de langage principal tout à fait viable.

Il est préférable que ce choix soit une **volonté partagée** entre membres d'une équipe, qu'il soit motivé par ce que Node apporte et simplifie pour vous.

Node est tout désigné pour créer des **applications à nombreuses actions concurrentes** ; autrement dit, dès qu'une application ou programme fait appel à des accès réseau, aux fichiers ou au système.

Node est également adapté pour **transformer des flux importants de données** en économisant la mémoire. Cela concerne aussi bien la lecture de fichiers CSV, JSON ou XML de plusieurs gigaoctets.

Le mécanisme de modules de Node encourage à respecter le *principe de responsabilité unique*. Nos applications seront **modulaires et autonomes** au lieu d'être lourdes et monolithiques.

Les développeurs et développeuses verront dans Node leur compagnon idéal pour **compiler, générer, assembler et** *minifier* leurs applications *front-end*. Le bénéfice évident est le partage d'un outillage CSS, JavaScript et HTML entre équipes, par projet.

Conséquence directe, Node offre l'opportunité d'**unifier vos équipes de développement** *front-end* **et** *back-end*. Il devient un environnement commun, un langage partagé entre les individus, qui peuvent se focaliser sur des fonctionnalités quelle que soit la cible, aussi bien le Web, un serveur ou une API.

Node est un environnement adapté à **l'apprentissage et l'amélioration de nos connaissances en JavaScript**. On peut désormais penser sur du long terme, en se souciant moins de devoir supporter de vieilles versions de Node ou de navigateurs web tant il est facile de passer d'une version du langage à une autre.

L'écosystème des acteurs

Les développements initiaux de Node sont en majorité financés par Joyent, à commencer par l'embauche de Ryan Dahl.

L'écosystème des acteurs se diversifie avec le temps. Avec de nouveaux employés chez Joyent, puis avec des contributeurs externes qui voient un avenir prometteur dans le projet. S'ensuivent des entreprises privées comme IBM ou PayPal, qui sponsorisent le projet ou le salaire de contributeurs.

Joyent

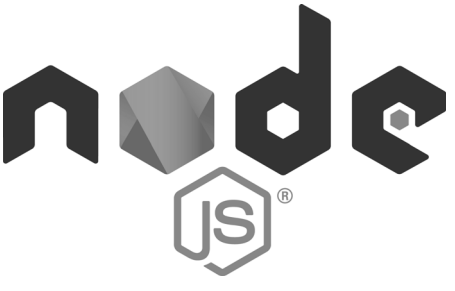
Joyent est une entreprise américaine fondée en 2004. Elle propose à l'origine des services de collaborations en ligne : documents, calendriers, courriels, etc.

Elle se lance sur le marché de l'hébergement fin 2005 par le biais d'une acquisition-fusion. Parmi ses clients, on dénombre le site de Ruby on Rails (société Basecamp), WordPress.com (société Automattic) ou encore le site historique A List Apart (https://alistapart.com).

En 2009, Joyent se spécialise dans les infrastructures et plates-formes à la demande et à haute performance. L'entreprise se concentre sur des solutions dites temps réel pour les réseaux sociaux, applications mobiles et compagnies de jeux vidéo en ligne.

En avril 2011, Joyent dépose la marque Node.js ainsi que son logo.

Figure 1–2
Logo officiel de Node.js



En février 2015, Joyent transfère la gestion de Node à la Node.js Foundation, mais reste propriétaire de la marque et de son logo.

> LIEN **Annonce du dépôt de marque**
>
> Ryan Dahl, alors développeur actif de Node, annonce le dépôt de marque par l'entreprise Joyent sur le blog officiel du projet (https://nodejs.org/en/blog/uncategorized/trademark/).

npm

npm est une dénomination qui abrite plusieurs concepts : un outil en ligne de commandes, un registre de modules ainsi qu'une entreprise privée à but lucratif (npm, Inc.).

À l'origine, c'est un module Node créé par Isaac Schlueter, qui sert à installer des modules tiers et à les lier sous forme d'un arbre de dépendances. Il est l'équivalent de RubygGems (https://rubygems.org/) pour Ruby, de CPAN (http://www.cpan.org/) pour Perl ou encore de PyPi (https://pypi.python.org/pypi) pour Python.

Nous reviendrons plus en détail sur son utilisation dans le chapitre 5.

`npm` désigne également le registre principal qui héberge les modules des communautés Node : https://npmjs.com.

> **LIEN Annonce de l'inclusion de npm dans Node**
>
> `npm` est installé par défaut avec Node depuis la version 0.6.3, sortie en novembre 2011. Vous retrouverez son annonce sur https://nodejs.org/en/blog/release/v0.6.3/.
> Auparavant, il fallait l'installer séparément.

npm, Inc.

Isaac Schlueter est embauché par Joyent en septembre 2010. Il succède à Ryan Dahl dans la gestion du projet Node de janvier 2012 jusqu'à janvier 2014, date à laquelle il quitte Joyent pour fonder npm, Inc. Cette entreprise a pour but de fournir des solutions professionnelles basées sur `npm` et soutient en parallèle l'effort open source et les coûts d'infrastructure du registre.

Elle lève 2,6 millions de dollars en février 2014 pour élaborer une nouvelle architecture du registre `npm`. Ce financement a également pour vocation la mise en place d'une stratégie commerciale basée sur les modules privés et les solutions professionnelles.

La société npm, Inc détient les marques npm et npm, Inc ainsi que le logo npm.

Figure 1–3
Logo officiel de npm, Inc

io.js

io.js est un *fork* de Node initié par la communauté en décembre 2014 en raison de la mainmise de Joyent sur les développements et de la communication erratique sur le projet.

Les objectifs initiaux du projet io.js sont doubles :

- offrir à la communauté Node une gestion transparente, inclusive et ouverte ;
- fournir un environnement technique plus moderne, une version de V8 plus récente, ainsi qu'une intégration rapide des nouvelles fonctionnalités ECMAScript.

Le projet io.js connaît une fin heureuse en 2015 : les efforts du projet et de sa communauté auront abouti à la création de la Node.js Foundation et du Node.js Advisory Board, respectivement l'organe de gestion du projet et le groupe d'individus qui en a la charge.

> **LIEN Clap de fin**
>
> L'annonce de la sortie de Node v4 et de la création de la Node.js Foundation est consultable à l'adresse suivante :
>
> ▸ https://nodejs.org/en/blog/announcements/foundation-v4-announce/

Node.js Foundation

La Node.js Foundation (https://foundation.nodejs.org) est l'un des organes officiels de gouvernance du projet Node depuis juin 2015. C'est une organisation à but non lucratif. Elle fait elle-même partie de la Linux Foundation (http://collabprojects.linuxfoundation.org/), au même titre que des projets comme Open Container, Let's Encrypt ou Xen.

La tâche première de la fondation est d'opérer la fusion entre les bases de code de Node et d'io.js en septembre 2015. Cela donne lieu à la sortie de Node v4.0.0.

La fondation fait partie intégrante de la gouvernance du projet Node.

Nodejitsu

Nodejitsu est une entreprise privée américaine fondée en 2010. Elle vise à fournir des solutions professionnelles autour de Node en tant que *Platform as a Service* (PaaS), ainsi qu'avec des dépôts `npm` privés. Son activité en fait un concurrent direct de Joyent et de *npm, Inc.*

Nodejitsu démontre un investissement fort dans la communauté Node en contribuant à plusieurs centaines de modules. La société prend en charge l'hébergement du registre `npm` de 2010 jusqu'en décembre 2013.

En 2013, l'entreprise lance l'initiative controversée #scalenpm (voir l'encadré ci-après). Elle vise à collecter des fonds pour améliorer la performance et la stabilité du registre `npm` (https://www.npmjs.com/). Nodejitsu attise les tensions avec la compagnie *npm, Inc.* en tentant de lui couper l'herbe sous le pied. Ce ne sera pas un succès.

En février 2015, la société américaine GoDaddy rachète Nodejitsu, absorbe son équipe et met fin à ses activités commerciales.

> **LIEN La controverse #scalenpm**
>
> L'initiative #scalenpm réunit quelque 326 000 $ auprès d'entreprises privées et de la communauté Node. Son effort se poursuit dans le but de fournir une meilleure instrumentation et une architecture résistant à la montée en puissance de l'utilisation des modules `npm`.
> Cette initiative a suscité une controverse dans la mesure où l'opération s'est déroulée lors de la naissance de npm, Inc et du dépôt de marque associé, mais sans entente apparente entre les deux parties.
> Le contenu est depuis archivé sur :
> http://web.archive.org/web/20160506191542/https://scalenpm.nodejitsu.com/.

Node Security Platform

La Node Security Platform (https://nodesecurity.io/ – anciennement Node Security Project) est un projet à but lucratif soutenu par la société américaine &yet (https://andyet.com/). Il a été initié au début de l'année 2013. La société, les employés et les logiciels ont été rachetés par npm, Inc. en 2018.

Son but est triple :

1. auditer la sécurité de tous les modules `npm` ;
2. communiquer les failles auprès des auteurs de modules ;
3. indiquer à tous si un module donné dépend de modules vulnérables.

Le projet met à disposition des services et des modules, tout en cherchant à impliquer la communauté Node dans le processus. Cela concerne aussi bien la déclaration des vulnérabilités que leur résolution ou l'éducation des développeurs à la sécurité.

Nous aborderons le sujet de la sécurité tout au long de cet ouvrage :

- au chapitre 4 : mettre à jour Node en cas de failles de sécurité ;
- au chapitre 6 : surveiller la santé d'une application en production ;
- au chapitre 7 : identifier les opérations à risque dans une application web.

Gouvernance du projet

Node a connu une forte croissance depuis 2012. La gouvernance du projet open source a été effectuée par la société Joyent. Ses agissements et la direction donnée au projet ont régulièrement fait grincer des dents, notamment en entretenant un climat d'incertitude sur la pérennité à long terme, si Node venait à ne plus répondre aux intérêts commerciaux de Joyent.

Des voix se sont élevées pour critiquer l'absence d'une organisation ouverte, commercialement neutre et ouverte aux contributeurs externes. Cela a mené à la création d'un *fork* de Node : io.js.

La réconciliation entre les projets Node et io.js a sérieusement assaini les rapports de gouvernance. Cela a également apporté une direction et des opportunités plus claires de contribuer à la direction du projet.

Depuis juin 2015, la gérance du projet est garantie par plusieurs entités :

- Technical Steering Committee (TSC) : planning, décisions techniques, direction du projet, documentation et qualité du projet ;
- Node.js Foundation Board : promotion du projet, relations commerciales ;
- Node.js Foundation Community Committee (CommComm) : relation avec la communauté, *onboarding*.

Le Technical Steering Committee (https://github.com/nodejs/TSC) réfère ses intentions d'actions au *Board*. Son fonctionnement est régi par une charte co-signée avec ce dernier. Ce comité est composé de contributeurs et de collaborateurs individuels.

ANECDOTE **Une fois n'est pas coutume**

Le TSC était autrefois dissocié d'une autre entité, le Core Technical Committee (https://github.com/nodejs/CTC). Les conflits de gouvernance ayant résulté dans la création du *fork* ayo.js ont mené à la fusion des deux comités.
L'objectif recherché était de rapprocher gouvernance et décisions techniques tout en réduisant les opportunités d'abus de pouvoir par les membres techniques éminents.

À l'inverse, le *Board* géré par la Node.js Foundation est composé essentiellement d'acteurs de l'industrie – dont Google, IBM, Joyent et PayPal. Certains membres émérites ou choisis par le *Board* (https://github.com/nodejs/board) sont des individus agissant en leur nom propre.

Les participant du TSC ont une obligation de régularité, de présence et de vote aux différents rendez-vous organisés par le comité. Ce mécanisme a été choisi afin de préserver la vitalité du projet.

Un quota d'appartenance à une même entreprise a été mis en place pour maintenir une diversité de représentation. Dans une moindre mesure, ce mécanisme vise à réduire les possibles conflits d'intérêts ou une prise en otage du projet comme a pu le faire Joyent avant l'apparition de ce modèle de gouvernance.

LIENS **Documentation**

Les documents clés régissant le fonctionnement du projet Node sont répartis dans les dépôts GitHub des différents acteurs :

- https://github.com/nodejs/node/blob/master/GOVERNANCE.md
- https://github.com/nodejs/TSC/blob/master/TSC-Charter.md
- https://github.com/nodejs/admin/blob/master/CODE_OF_CONDUCT.md
- https://github.com/nodejs/node/blob/master/COLLABORATOR_GUIDE.md

Conclusion

Nous venons d'en apprendre davantage sur les origines de Node, les différentes parties prenantes dans son développement, mais aussi sa philosophie – de conception, de distribution et d'évolution.

Nous allons voir dans le prochain chapitre comment installer un environnement fonctionnel pour développer et exécuter des programmes Node.

2

Installer, mettre à jour et développer

Installons Node et l'outillage de développement qui nous correspond le mieux, peu importe notre niveau de familiarité avec le développement logiciel.

- Installer Node.js sur notre ordinateur
- Choisir un éditeur de code pour écrire nos programmes
- Déterminer quand migrer vers une nouvelle version de Node.js

Certains systèmes d'exploitation embarquent l'environnement d'exécution Node tandis que d'autres ne le font pas, ou dans une version trop ancienne. Idéalement, nous voudrions pouvoir **installer la version de Node de notre choix**.

Les **éditeurs de code** nous facilitent la vie en rendant le code lisible, en ajoutant de la couleur et des repères visuels. Ces logiciels nous invitent à piocher dans leur bibliothèque d'extensions pour en faire un outil qui nous ressemble.

> **REMARQUE** **Versions de Node et npm**
>
> Le contenu de ce chapitre utilise les versions **Node v10** et **npm v6**. Ce sont les versions stables recommandées en 2018.

Il n'est pas nécessaire d'avoir suivi des études d'informatique pour vouloir s'essayer à la programmation. Cela n'implique pas non plus d'en faire son métier.

Que l'on se qualifie de débutant, confirmé ou expert, il y a un petit rituel auquel on coupera difficilement :

1. installer Node.js pour **voir le résultat de nos programmes** écrits en JavaScript ;
2. installer un éditeur de code pour **écrire nos programmes JavaScript** plus confortablement.

Si l'idée est de jouer rapidement avec du code, sans rien installer et avec le premier ordinateur qui vous passe sous la main, je vous invite à aller directement à la section « Sans installation, dans un navigateur web » ci-après.

Les sections qui suivent vont vous aider à créer un environnement Node à jour sur votre ordinateur. Ce contenu s'applique également à un serveur destiné à héberger vos applications.

Installer Node.js

Il y a plusieurs manières d'installer Node sur une machine. Elles sont *toutes* correctes. Certaines sont plus adaptées que d'autres, selon votre aisance avec un terminal et selon le besoin de jongler rapidement entre différentes versions de Node.

- Si vous vous êtes déjà servi d'un terminal : je recommande d'utiliser `nvm`.
- Si vous ne vous êtes jamais servi d'un terminal : il est plus simple d'utiliser un *installeur* depuis le site officiel de Node.
- Si vous souhaitez maîtriser les options d'installation : il serait logique de compiler depuis les *sources* et/ou d'utiliser Docker.
- Si rien de tout ça ne vous parle : des services en ligne rendent Node accessible depuis un simple navigateur web.

> QUESTION **Pourquoi utiliser un installeur ?**
>
> Les installeurs facilitent l'installation de Node, en quelques clics et sans toucher à un terminal.
> Si vous utilisez un installeur correspondant à une version plus récente de Node, c'est celle-ci qui sera utilisée dans *tous* vos projets.
> C'est la solution la plus simple pour installer Node.

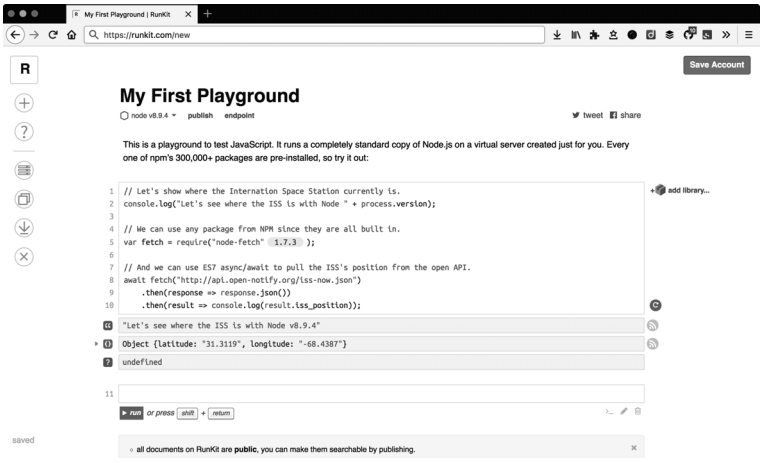
Sans installation, dans un navigateur web

Il est facile de s'essayer à Node avec un simple navigateur web moderne comme Firefox, Edge ou Chrome. Des services en ligne combinent un éditeur de texte et un environnement d'exécution Node à distance.

Nous brosserons le portrait de trois services qui diffèrent par leurs fonctionnalités et leur rapidité de prise en main : RunKit, Codenvy et Cloud9.

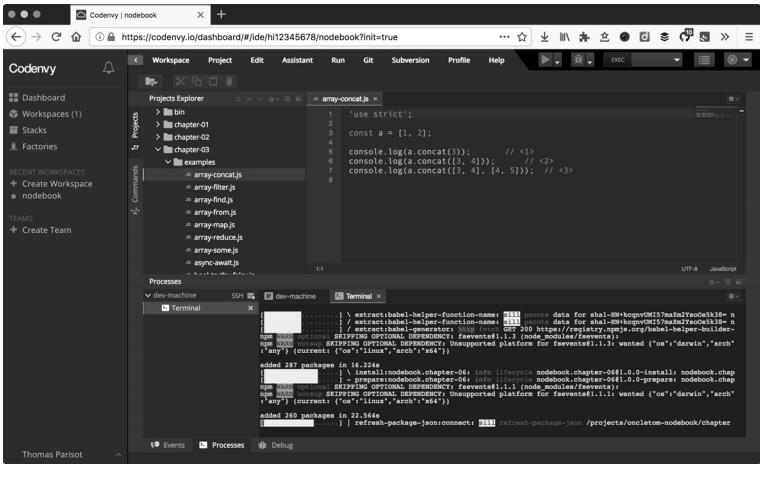
RunKit (https://runkit.com) est de loin le service le plus simple. Il s'articule autour d'un concept de *bac à sable* ; notre code ne sera pas plus long qu'un fichier. Pour créer un nouveau bac à sable, il suffit de se rendre sur https://runkit.com/new.

Figure 2–1
Bac à sable sur RunKit



Codenvy (https://codenvy.io) est un service basé sur le logiciel open source Eclipse et édité par la compagnie Red Hat. L'interface est organisée exactement comme un des éditeurs de texte abordé plus loin dans ce chapitre.

Figure 2–2
Espace de travail sur Codenvy



Le processus de création de projet est relativement intuitif et il est très facile d'importer du code hébergé en ligne, sur GitHub notamment. La configuration minimale d'un projet est gratuite tandis que les fonctionnalités avancées et le travail en équipe sont payantes.

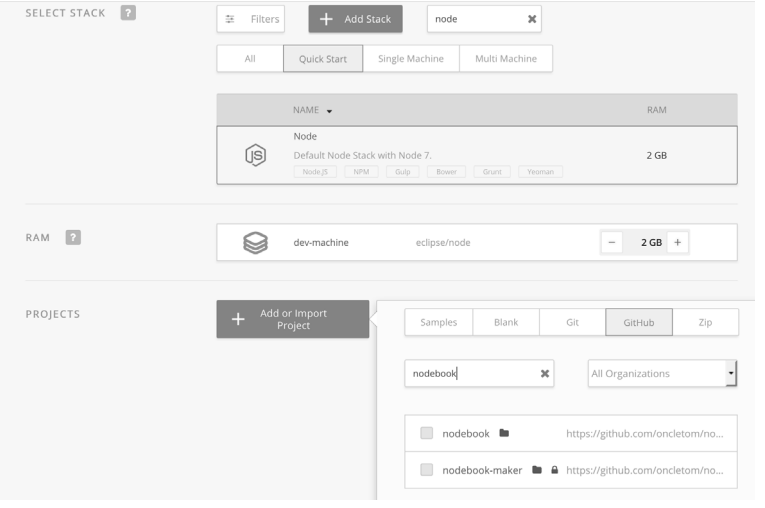


Figure 2–3
Processus de création d'un espace de travail sur Codenvy

Enfin, Cloud9 (https://c9.io) est un service propriétaire édité par la compagnie Amazon Web Services (AWS, https://aws.amazon.com). Il intègre des fonctionnalités similaires à celles de Codenvy (éditeur en ligne, configuration de machine) et s'intègre de manière poussée avec les autres services d'AWS, dont *EC2* et *Lambda* (chapitre 6). Le service est entièrement gratuit ; c'est la consommation de ressources qui est payante, à l'heure, en fonction du dimensionnement des ressources demandées.

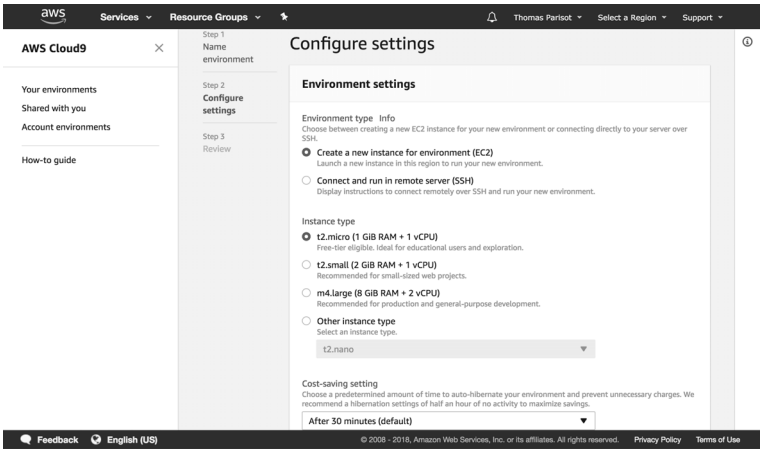
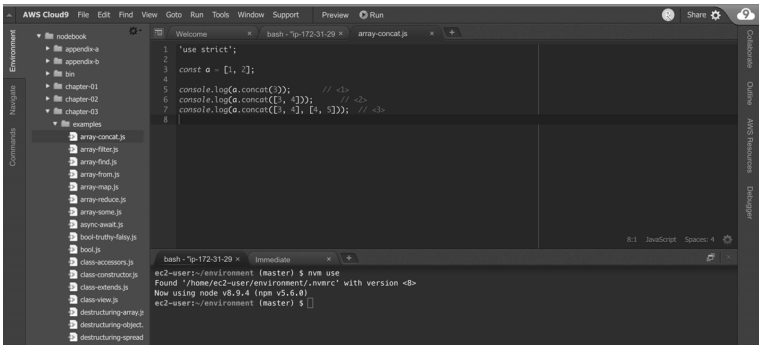


Figure 2–4
Configuration d'un espace de travail Cloud9

Cloud9 a peu d'opinions sur la manière dont votre environnement doit être configuré. Tout devra être configuré, des accès à votre compte GitHub à la version de Node à installer.

Figure 2–5
Espace de travail sur Cloud9



Cela tombe bien, `nvm` est préinstallé sur chaque espace de travail.

Plusieurs versions sur la même machine (nvm)

`nvm` est l'acronyme de *Node Version Manager*. Ce logiciel permet d'**installer plusieurs versions de Node sur un même ordinateur**. Si vous êtes sous Windows, `nvm-windows` offre exactement les mêmes fonctionnalités (voir encadré).

Si vous venez d'un autre univers de programmation, `nvm` est l'équivalent de `rvm` ou `rbenv` dans le monde Ruby, de `phpenv` dans le monde PHP ou encore de `virtualenv` pour Python.

Le programme s'obtient à l'adresse suivante : https://github.com/creationix/nvm.

> **ALTERNATIVES** **nvm pour Windows**
>
> `nvm` ne fonctionne pas sur les ordinateurs équipés de Windows. Il existe trois alternatives :
>
> - `nvm-windows` (https://github.com/coreybutler/nvm-windows) offre les mêmes fonctionnalités que son équivalent pour Linux et macOS. Il est facile à installer et ne demande pas de privilège système particulier pour fonctionner.
> - Deux autres solutions sont à essayer si `nvm-windows` pose problème : `nvmw` (https://github.com/hakobera/nvmw) et `nodist` (https://github.com/marcelklehr/nodist).

Commande d'installation du logiciel nvm v0.33.11

```
$ curl -L https://github.com/creationix/nvm/raw/v0.33.11/install.sh | bash
```

> **ALTERNATIVE** **nvm sous macOS**
>
> `nvm` et Node s'installent encore plus facilement sous macOS si vous utilisez le logiciel Homebrew. Rendez-vous dans la section « macOS » pour en savoir plus.

Nous sommes libres d'installer toutes les versions de Node qui nous intéressent en appelant la commande `nvm install` dans un terminal.

Installation de deux versions différentes de Node

```
$ nvm install v10
$ nvm install v9
```

Dans cet exemple, nous installons deux versions de Node sur le même ordinateur. La version active est la dernière installée. On retrouve à tout moment les versions installées avec la commande `nvm ls`.

Liste des versions installées de Node avec la commande nvm ls

```
$ nvm ls
->       v10.9.0
         v9.0.0
         system
default -> v10 (-> v10.9.0) (default)
```

Cet exemple détaille la liste des versions de Node installées et celle des *alias* – associations entre un libellé et une version.

La commande `nvm use` nous fait naviguer entre des versions différentes de Node. L'utilisation répétée de `nvm use v10` et de `nvm use v9` nous fera aller et venir entre ces deux versions, sans rien perdre de ce que nous étions en train de faire.

La commande `nvm --help` affiche de l'aide à propos des commandes disponibles. J'utilise principalement les suivantes :

- `nvm install` : installe ou met à jour une version de Node.
- `nvm ls` : liste les versions installées.
- `nvm use` : bascule vers une version donnée.
- `nvm alias <nom> <version>` : crée un alias nommé vers une version.
- `nvm run <version> <script>` : exécute un script Node dans une version donnée.

Quand nous avons fixé une version de Node à utiliser au quotidien, l'alias spécial `default` en fait la version qui est systématiquement activée :

```
$ nvm alias default 10
```

La commande `nvm use` nous sert alors à activer une autre version pendant la durée nécessaire à notre expérimentation.

Distributions Linux : Debian/Ubuntu et RedHat/CentOS

Ces distributions Linux listent Node dans leurs paquets officiels. Un simple `apt-get install nodejs` et ça fonctionne ! Ou presque, car bien souvent on écopera d'une vieille version de Node.

Des paquets officiels pour Debian (≥ jessie), Ubuntu (≥ 12.04), Mint, RedHat Enterprise Linux (RHEL) et CentOS sont maintenus sur https://github.com/nodesource/distributions.

L'installation de Node 10.x sous Ubuntu et Mint se résume aux commandes suivantes :

```
$ sudo su
$ curl -sL https://deb.nodesource.com/setup_10.x | bash -
$ sudo apt-get install -y nodejs
```

Pour Debian, il faudra se mettre en `root` et ensuite saisir ces commandes :

```
$ curl -sL https://deb.nodesource.com/setup_10.x | bash -
$ apt-get install -y nodejs
```

Pour RHEL et CentOS, il faudra se mettre en `root` et saisir la commande suivante :

```
$ curl -sL https://rpm.nodesource.com/setup_8.x | bash -
```

Distributions Linux : les autres

Des paquets officiels pour d'autres distributions que Debian, Ubuntu et Mint sont maintenus et accessibles par le biais des gestionnaires de paquets habituels :

- Gentoo : `emerge nodejs`
- Fedora (≥ 18) : `sudo yum install nodejs npm`
- Arch Linux : `pacman -S nodejs`
- FreeBSD, OpenBSD : `pkg install node`

La liste à jour est maintenue sur le site officiel de Node : https://nodejs.org/fr/download/package-manager.

Notez que vous pouvez aussi utiliser `nvm` pour gérer vos versions de Node indépendamment du cycle de vie de votre système d'exploitation Linux.

macOS

Un installeur officiel pour macOS est fourni sur le site de Node, à l'adresse https://nodejs.org/fr/download/.

`nvm` est la voie alternative recommandée pour gérer plusieurs versions de Node en parallèle.

Il est toutefois possible d'installer Node et `nvm` via le gestionnaire de paquets Homebrew (https://brew.sh/). Il aide à installer des logiciels qui ne sont pas distribués via le *Mac App Store*.

1. Installer les *Command Line Tools* pour compiler des logiciels depuis leur code source.
2. Installer Homebrew.

Installation des Command Line Tools et de Homebrew sous macOS

```
$ xcode-select --install
$ BREWRL=https://rawgit.com/Homebrew/install/master/install \
   /usr/bin/ruby -e "$(curl -fsSL $BREWRL)"
```

Pour installer `nvm`, il suffit alors de lancer la commande suivante :

```
$ brew install nvm
```

Ou, pour installer une seule version de Node, la plus récente :

```
$ brew install node
```

Vous obtiendrez des options d'installation et de configuration en tapant `brew info nvm` et/ou `brew info node`.

Windows

Un installeur officiel pour Windows est fourni sur le site de Node à l'adresse https://nodejs.org/fr/download/.

`nvm-windows` est la voie alternative recommandée pour gérer plusieurs versions de Node en parallèle.

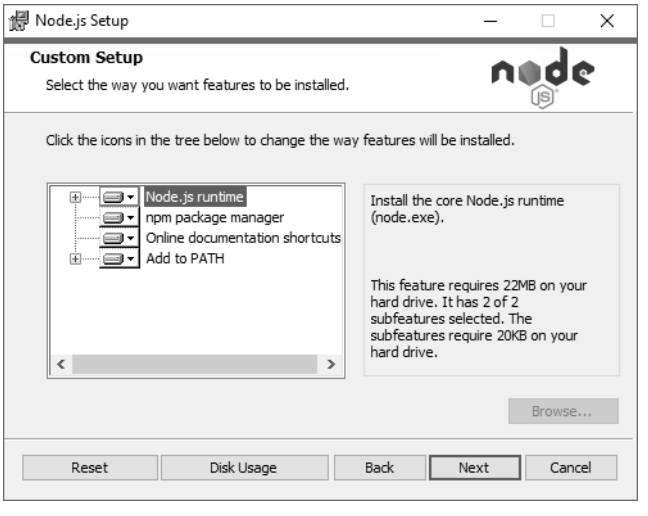


Figure 2–6
Un des écrans d'installation de Node sous Windows 10

L'installeur officiel créera plusieurs raccourcis dans le dossier *Node.js* du menu *Démarrer* :

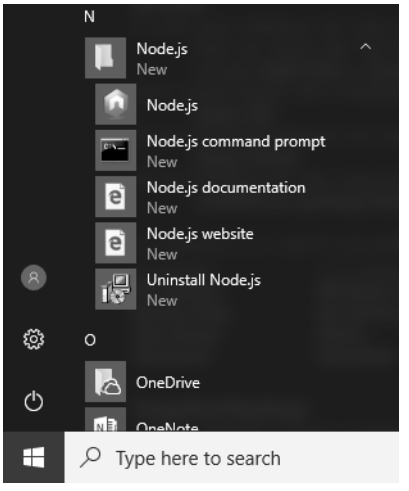


Figure 2–7
Node.js dans le menu Démarrer sous Windows 10

Le menu créé par l'installeur contient deux icônes que nous utiliserons tout au long de la lecture de cet ouvrage :

- *Node.js* : un terminal pour jouer avec JavaScript et voir les résultats que Node va interpréter ;
- *Node.js command prompt* : un terminal pour exécuter nos programmes écrits en JavaScript.

Si vous utilisez déjà un gestionnaire de paquets comme Scoop (http://scoop.sh/) ou Chocolatey (https://chocolatey.org/), le chemin d'installation vers Node s'en trouvera réduit à une simple commande :

Avec Scoop

```
$ scoop install nodejs
```

Avec Chocolatey

```
$ choco install nodejs
```

> ATTENTION **Versions de Windows**
>
> Node n'est pas compatible avec les versions antérieures à Vista. Cela inclut Windows XP.
> Mieux vaudra utiliser un service en ligne pour essayer Node sur votre ordinateur, ou passer à Linux pour lui redonner une seconde vie en toute sécurité !

Raspberry Pi

Des binaires sont disponibles pour les microcontrôleurs fonctionnant avec des processeurs ARM v6/7/8, au cœur de ce que l'on appelle *Internet des Objets (Internet of Things, IoT)*. Ces petits ordinateurs consomment peu d'énergie, disposent d'une connectique pour se relier à Internet et s'interfacent avec toutes sortes de capteurs.

Un paquet `deb` est également proposé pour les utilisateurs du système d'exploitation Raspbian (https://www.raspbian.org/). Les instructions d'installation sont identiques à celles décrites dans la section relative aux distributions Linux Debian.

Installation de Node v10 sur Raspberry Pi Model 3 (CPU ARMv8)

```
$ curl -SLO https://nodejs.org/dist/v10.9.0/node-v10.9.0-linux-arm64.tar.xz
$ tar -xJf "node-v10.9.0-linux-arm64.tar.xz" -C /usr/local --strip-components=1
$ ln -s /usr/local/bin/node /usr/local/bin/nodejs
```

Tous les binaires et les instructions d'installation sont disponibles sur https://nodejs.org/fr/download/.

Compiler depuis les sources

Certaines situations exigeront que vous compiliez Node. C'est le cas si vous cherchez à travailler au plus près du système sur lequel vous comptez déployer vos applications. Je pense par exemple à des architectures à processeur ARM, PowerPC, IBM System/390 ou bien à des systèmes Android, OpenBSD ou AIX.

La compilation manuelle est également intéressante pour régler plus finement certains aspects de Node : rendre le binaire indépendant des bibliothèques système (statique), module `http/2`, langues et fuseaux horaires fonctionnels avec l'API ECMA 402 Intl, options de sécurité liées à OpenSSL, mais aussi intégration avec des profileurs et débogueurs externes (type XCode, GNU Debugger, Intel VTune).

Étapes de compilation de Node v10.9.0

```
$ curl -sS https://nodejs.org/dist/v10.9.0/node-v10.9.0.tar.gz \
  | tar -zxf -
$ cd node-v10.9.0
$ ./configure && make && make install
```

La compilation manuelle requiert la présence de GCC ≥ 4.9, de Python ≥ 2.6 et de GNU Make ≥ 3.81.

Les instructions de compilation varient d'un système d'exploitation à l'autre. Consultez les dépendances et instructions complètes à l'adresse suivante : https://github.com/nodejs/node/blob/master/BUILDING.md.

Images Docker

Docker (https://docker.com) est un outil open source dit d'*applications en conteneur*. Une de ses qualités principales est d'**isoler l'exécution d'applications de leur environnement**. Une image Docker décrit la recette d'installation d'une application. L'environnement d'exécution Docker fait office de « passe-plat » avec le système d'exploitation. Une image Docker fonctionne ainsi de la même manière, qu'elle soit exécutée sous Linux, macOS ou encore Windows.

Les installeurs et instructions d'installation de Docker se trouvent à l'adresse https://docker.com/community-edition.

Une fois Docker installé, on peut exécuter une image officielle pour Node v10.

Affiche la version de Node

```
$ docker run -ti --rm node:10 node --version
v10.9.0
```

Ici, Docker télécharge l'image `node:10` et exécute la commande `node --version` dans un contexte isolé du système d'exploitation.

Pour lancer un terminal Node dans Docker, il faudrait saisir la commande suivante :

```
$ docker run -ti --rm node:10 node
> 2+2
4
```

Plusieurs variantes d'images Node se trouvent à notre disposition :

- *standard* (`node:10`)

  Base Linux Debian pour tout type d'applications Node.
- *Debian* (`node:10-wheezy`)

  Comme *Standard* mais sur des bases Debian différentes, comme Wheezy, Stretch, etc.
- *Alpine* (`node:10-alpine`)

  Distribution spécialement créée pour Docker (http://alpinelinux.org), qui pèse à peine quelques Mo.
- *Allégée* (`node:10-slim`)

  Base Linux Debian sans outillage parfois nécessaire à des modules Node – utile si vous souhaitez économiser de l'espace disque.

L'intégralité des versions et architectures prises en charge est disponible sur le catalogue Docker Hub : https://hub.docker.com/_/node/.

Nous reviendrons sur ce sujet dans la section « Déploiement d'une image Docker » du chapitre 6.

Vérifier l'installation de Node depuis un terminal (shell)

Nous avons installé un environnement d'exécution Node dans la section précédente. Pour vérifier que tout s'est bien déroulé, ouvrez un terminal et saisissez la commande suivante :

```
$ node --version
v10.9.0
```

Le numéro de version du Node fraîchement installé devrait apparaître. Si le mot *terminal* ne vous parle pas, la suite de cette section va vous éclairer – vous pourrez ensuite revenir essayer cette commande.

Qu'est-ce qu'un terminal ?

Le terminal est notre compagnon pour dialoguer avec le système d'exploitation. L'*invite de commandes* est son nom véritable. Ce nom nous donne un indice sur la fonction de ce type de logiciel : inviter l'utilisateur à saisir des commandes pour obtenir des résultats calculés par un ordinateur.

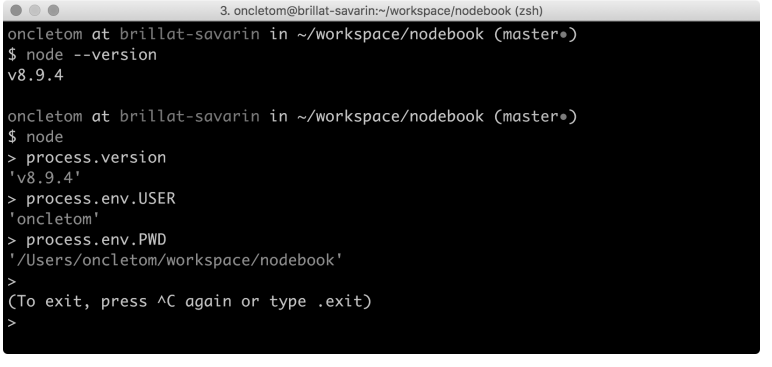


Figure 2–8
Terminal iTerm2 sous macOS

> HISTOIRE **Terminal physique**
>
> L'histoire des invites de commandes remonte au temps où les ordinateurs étaient plus volumineux que nos logements. Une époque lointaine où les ordinateurs étaient véritablement et physiquement distants des claviers qui les interrogeaient.
> Pour en savoir plus : https://fr.wikipedia.org/wiki/Terminal_informatique.

Un terminal est notre **moyen privilégié pour interagir avec Node** lorsqu'il est installé sur un ordinateur. Les systèmes d'exploitation en ont pour la plupart un installé par défaut. Cela vaut également pour la majorité des services en ligne.

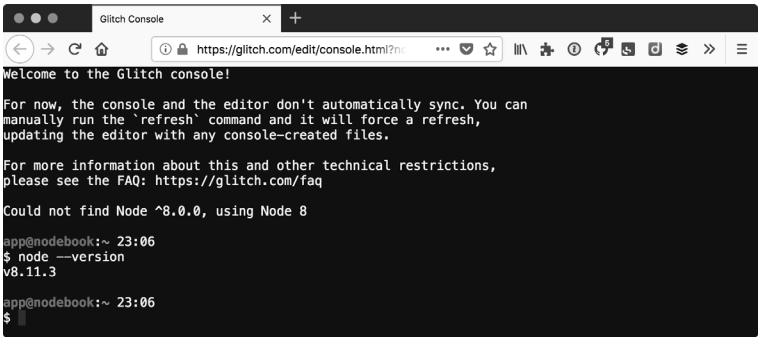


Figure 2–9
Terminal du service en ligne Glitch

Choisir un terminal

Voici une liste non exhaustive d'applications de type terminal :

- macOS

  Terminal.app : fourni par défaut (dans *Applications/Utilitaires*) ;

  iTerm2 : une version améliorée disponible sur https://iterm2.com (ou `brew cask install iterm2`) ;
- Linux

  GNOME Terminal : fourni par défaut sous Debian, Ubuntu et les distributions utilisant le bureau GNOME (https://wiki.gnome.org/Apps/Terminal) ;

  Terminator : un autre terminal populaire (https://gnometerminator.blogspot.com) ;
- Windows

  Node.js Command Prompt : fourni avec l'installeur Windows ;

  PowerShell : fourni par défaut depuis Windows 7, sinon disponible sur https://github.com/PowerShell/PowerShell – également disponible pour macOS et Linux.

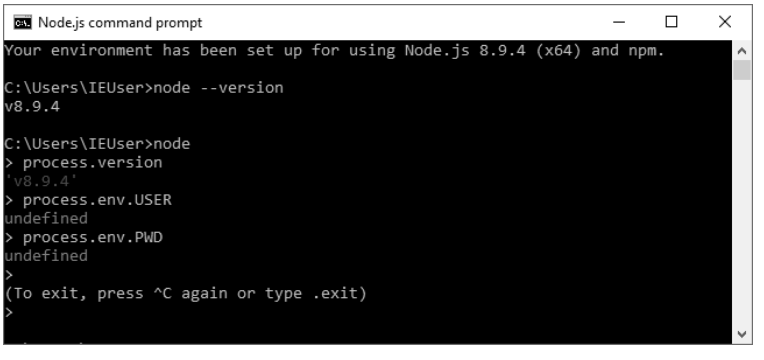


Figure 2–10
Node.js Command Prompt sous Windows 10

Maintenant que nous avons installé Node et compris comment y accéder depuis notre système d'exploitation ou navigateur web, attaquons-nous au dernier morceau du puzzle : **avec quel logiciel écrire du code JavaScript pour nos applications Node ?**

Choisir un éditeur de texte

Programmer pour Node revient dans la majorité des cas à écrire du JavaScript. À cela s'ajoutent le HTML et le CSS dans le cas d'applications ou de sites web.

À la base, si un éditeur de texte suffit pour écrire du code, prenons le temps de regarder ce qui pourrait nous apporter un peu de confort dans le processus d'écriture.

Les logiciels présentés ci-après couvrent bon nombre de fonctionnalités qui améliorent de près ou de loin notre capacité à écrire du code de qualité. Parmi elles, on retrouve la coloration syntaxique, l'inspection dynamique, le débogage, des astuces de productivité et d'intégration à l'écosystème Node.

Cette sélection a pour but de vous aider à piocher au plus près de vos goûts. Le meilleur logiciel sera celui qui vous plaira. Rien n'empêche d'en changer par la suite.

Atom

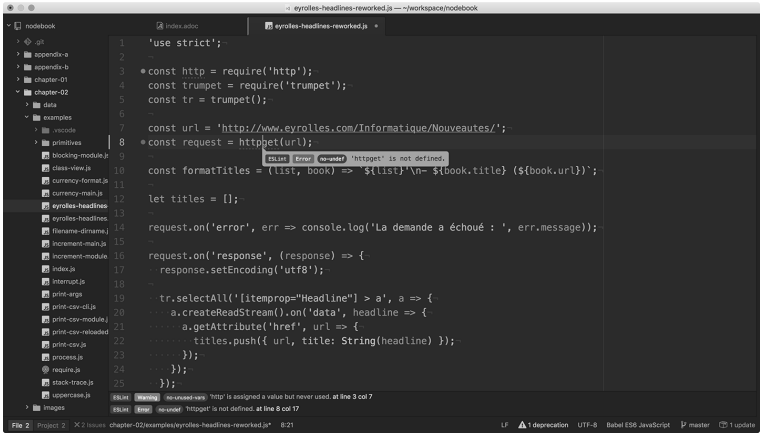


Figure 2–11
Atom

Atom (https://atom.io) est un éditeur de code open-source, multilingue et multi plate-forme, dont le développement a été lancé par la société commerciale GitHub (http://github.com). Le logiciel est basé sur Electron, un environnement d'exécution d'applications de bureau reposant sur les technologies web et sur Node !

Atom offre un écosystème d'extensions pour étendre les fonctionnalités de l'éditeur. On retrouvera des extensions dédiées à l'auto-complétion, un débogueur Node intégré (pour exécuter nos programmes sans changer de fenêtre), mais aussi une vérification syntaxique sur mesure.

Toutes les extensions d'Atom sont disponibles sur https://atom.io/packages ; voici une liste de celles que j'utilise au quotidien :

- minimap (https://atom.io/packages/minimap)

  Une prévisualisation de l'intégralité du code source d'un fichier.

- file-icons (https://atom.io/packages/file-icons)
  Une manière plus agréable de visualiser les différents types de fichiers en fonction de leur icône.
- pigments (https://atom.io/packages/pigments)
  Affiche les couleurs en marge et dans le code source.
- language-babel (https://atom.io/packages/language-babel)
  Coloration syntaxique de tous les langages compris par l'outil Babel (https://babeljs.io), dont les versions modernes de JavaScript, JSX, GraphQL, etc.
- emmet (https://atom.io/packages/emmet)
  Génère du HTML à partir d'une écriture sous forme de sélecteur CSS.
- linter-eslint (https://atom.io/packages/linter-eslint)
  Vérification syntaxique basée sur les règles projet du module ESLint (https://eslint.org, voir l'annexe).
- atom-ternjs (https://atom.io/packages/atom-ternjs)
  Autocomplétion pour JavaScript, Node et d'autres bibliothèques populaires comme jQuery, chai et underscore.
- editorconfig (https://atom.io/packages/editorconfig)
  Adapte les réglages d'indentation et autres styles d'écriture de code documentés dans chaque projet.
- language-sass (https://atom.io/packages/language-sass)
  Prise en charge du langage Sass.
- markdown-writer (https://atom.io/packages/markdown-writer)
  Raccourcis et aides pour formater du texte au format Markdown.
- autocomplete-modules (https://atom.io/packages/autocomplete-modules)
  Étend l'autocomplétion lors des appels aux modules `npm` (chapitre 5).
- linter-sass-lint (https://atom.io/packages/linter-sass-lint)
  Vérification syntaxique des fichiers Sass.
- node-debugger (https://atom.io/packages/node-debugger)
  Intégration du débogueur Node.
- tablr (https://atom.io/packages/tablr)
  Éditeur de fichiers CSV.
- linter-markdown (https://atom.io/packages/linter-markdown)
  Vérification syntaxique des fichiers Markdown.
- node-resolver (https://atom.io/packages/node-resolver)
  Navigation au sein des modules `npm` en cliquant sur leurs méthodes ou propriétés.