

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

Histoire, écosystème et gouvernance • Installer, mettre à jour et développer • Jouer avec JavaScript • Jouer avec Node.js • Jouer avec npm • Déployer notre code • Créer une application web • Créer un outil en ligne de commandes • Créer une application front-end • Sélection de modules npm

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, 2^e 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.

Retrouvez nos bundles (livres papier + e-book) et livres numériques sur

<http://izibook.eyrolles.com>

Thomas Parisot

Node.js

Apprendre par la pratique

EYROLLES

The logo for EYROLLES features the word "EYROLLES" in a bold, uppercase, sans-serif font. Below the text is a horizontal line with a small circle centered underneath it.

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

Mise en pages : Gaël Thomas

En application de la loi du 11 mars 1957, il est interdit de reproduire intégralement ou partiellement le présent ouvrage, sur quelque support que ce soit, sans l'autorisation de l'Éditeur ou du Centre français d'exploitation du droit de copie, 20, rue des Grands-Augustins, 75006 Paris.

© Éditions Eyrolles, 2019, ISBN : 978-2-212-13993-8

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 avons 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 sur-ingénierie 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 **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·es 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 1

Histoire, écosystème et gouvernance	1
Node.js : un environnement d'exécution JavaScript	1
Bref historique	3
Les raisons du succès	5
Pourquoi éviter Node.js ?	5
Pourquoi choisir Node.js ?	6
L'écosystème des acteurs	7
Joyent	7
npm	8
npm, Inc.	8
io.js	8
Node.js Foundation	9
Nodejitsu	9
Node Security Platform	10
Gouvernance du projet	10
Conclusion	11

CHAPITRE 2

Installer, mettre à jour et développer	13
Installer Node.js	14
Sans installation, dans un navigateur web	14
Plusieurs versions sur la même machine (nvm)	17
Distributions Linux : Debian/Ubuntu et RedHat/CentOS	18
Distributions Linux : les autres	19
macOS	19
Windows	20
Raspberry Pi	22
Compiler depuis les sources	22
Images Docker	23
Vérifier l'installation de Node depuis un terminal (shell)	24
Qu'est-ce qu'un terminal ?	24
Choisir un terminal	25
Choisir un éditeur de texte	26
Atom	26
Visual Studio Code	28

WebStorm	29
Visual Studio IDE	30
Quand mettre à jour Node.js ?	30
Mises à jour de sécurité	31
Versions mineures	32
Versions majeures	32
Comprendre le cycle de vie produit	32
Conclusion	33

CHAPITRE 3

Jouer avec JavaScript	35
Qu'est-ce que JavaScript ?	35
ECMAScript 5 (aka ES5)	37
ECMAScript 2015 (aka ES6 puis ES2015)	38
ECMAScript 2016, etc. (aka ES2016)	39
Éléments de base du langage	40
Les types de données	40
Les variables	42
Les instructions	43
La portée (scope)	44
Jongler avec du texte (chaînes de caractères)	46
Expressions régulières (RegExp)	48
Jongler avec des valeurs vraies ou fausses (booléens)	53
Jongler avec des valeurs numériques (Number, Math)	54
Opérations mathématiques	55
Les nombres qui n'en sont pas (NaN)	56
Convertir en nombre	57
Formater et arrondir des nombres	58
Créer et réutiliser des blocs de code (fonctions)	59
Les fonctions anonymes	60
Les fonctions de rappel (callback)	60
Paramètres du reste (rest parameters)	61
Lister, filtrer et trier des éléments (Array)	61
Créer des tableaux à partir d'autres valeurs	62
Combiner des tableaux	62
Itérer sur les valeurs avec des boucles	63
Trier les valeurs	65
Transformer les valeurs	66
Filtrer les valeurs	67
Identifier des valeurs	67
Décomposition de tableau (destructuring)	69
Représenter des structures d'objet et y accéder	71
Décomposition d'objet (destructuring)	72
Combiner des objets	73
Itérer sur des objets	73
Identifier des valeurs	75
Lire et écrire des données au format JSON	76

Interagir avec des dates	78
Formatage internationalisé (Intl.DateTimeFormat)	79
Partager une logique avec des objets de même nature (Class)	81
Méthodes d'instance	82
Méthodes statiques	83
Accesseurs et mutateurs	84
Héritage	85
Coordonner des actions asynchrones (Promise)	86
Collection de promesses	88
async/await	89
Conclusion	90

CHAPITRE 4

Jouer avec Node.js..... 91

Interagir avec l'interpréteur Node	92
Afficher la version	92
Avec un script	93
Avec l'invite de commandes interactive (REPL)	93
Les modules de base	95
console : déboguer rapidement des variables	97
path : manipuler des chemins de fichiers	99
url : manipuler des URL	102
fs : manipuler le système de fichiers	105
events : programmer des événements	109
util : transformer des fonctions de rappel en promesses	112
http : créer et interroger des ressources via le protocole HTTP	114
os : en savoir plus sur les capacités de l'ordinateur	124
child_process : appeler un exécutable système	126
process : en savoir plus sur le processus en cours	130
stream : manipuler des flux de données	141
D'autres modules pour aller plus loin	145
Créer ses propres modules Node	146
Importer et exporter des valeurs	146
Aller plus loin avec require()	148
Le futur : les modules ECMAScript	150
S'en sortir quand ça ne se passe pas comme prévu	152
Une erreur est nichée dans notre code	152
Une erreur est retournée dans une fonction de rappel	154
Une erreur est retournée dans une promesse	155
Une erreur est retournée dans un événement	157
Une erreur est renvoyée par le système d'exploitation	158
Le programme ne se termine pas	159
Une alerte de dépréciation s'affiche	159
Les différences d'ECMAScript entre Node et les navigateurs web	160
L'absence du DOM et des variables window et document	160
Il n'y a pas d'interface graphique	160
Le mécanisme de modules	160

L'interfaçage avec le système d'exploitation	161
Node est un processus système	161
Options utiles pour démarrer Node	161
Afficher le résultat d'une expression, sans script	161
Précharger un module	162
Inspecter notre code avec Google Chrome	162
Ajuster les options de compatibilité et de traçabilité de V8	164
Conclusion	165

CHAPITRE 5

Jouer avec npm..... 167

Créer un fichier package.json	168
Installer des modules npm	169
Depuis le registre npm	170
Trouver son bonheur dans le registre npm	171
Désinstaller un module	175
Depuis un fichier package.json	175
Spécifier une version	176
Comprendre les numéros de versions (Semantic Versioning)	178
Mises à jour	179
Autres manières d'installer et d'utiliser des modules npm	180
Depuis GitHub, GitLab ou un dépôt Git	180
Dépendances de développement	181
Exécutable système (installation globale)	182
Outiller un projet avec les scripts npm	183
Démarrer l'application	183
Exécuter des tests	184
Créer un script npm personnalisé	185
Exécuter des commandes avant et après des scripts npm	187
Automatiser tout l'outillage projet	188
Anatomie du fichier package.json	189
Quelques commandes pour aller plus loin	191
npm view : voir les informations d'un module	191
npm x : exécuter un module sans l'installer	193
npm home : visiter le site web d'un module	193
npm audit : vérifier la sécurité des dépendances	194
npm ci : installer à toute vitesse	195
npm doctor : vérifier l'état du système	195
npm config : changer les réglages de l'exécutable npm	196
npm publish : publier un module npm	198
npm version : déterminer une nouvelle version sans se tromper	199
Questions et mystères autour de npm	200
Quand mettre à jour l'exécutable npm ?	200
Je ne vois pas l'intérêt du fichier package-lock.json	201
npm c'est pour le back-end et bower pour le front-end	202
Est-ce que je dois versionner le répertoire node_modules ?	202
Il paraît que Yarn, c'est mieux	203

npm est lent, il installe la moitié d'Internet à chaque fois	203
Que signifient les erreurs affichées pendant npm install ?	205
<i>Module déprécié.</i>	205
<i>Problème avec une dépendance optionnelle.</i>	205
<i>Module introuvable.</i>	206
<i>Caractère de fin de ligne sous Windows</i>	206
<i>Fichier package.json incomplet.</i>	206
<i>Dépendance complémentaire à installer</i>	206
Conclusion	207

CHAPITRE 6

Déployer notre code 209

Déployer une application Node	210
En codant dans un navigateur web	210
En transférant des fichiers via SSH	212
En important du code depuis GitHub	213
Avec l'outil en ligne de commandes de l'hébergeur	215
En faisant git push depuis sa machine	216
En faisant git pull lors d'une session SSH	217
Avec une recette de déploiement (Ansible, Chef, etc.)	218
En publiant une image Docker	220
En paramétrant un logiciel d'intégration continue	222
Choisir son hébergement	224
Plate-forme de services (Platform as a Service, PaaS)	224
Hébergement mutualisé	226
Serveur virtualisé, dédié ou cloud	228
Fonction événementielle (Serverless, Lambda)	229
Améliorer la portabilité applicative	231
Utiliser la bonne version de Node	231
L'application tourne mais elle est injoignable	232
S'affranchir des chemins et configurations écrits « en dur »	233
Faire persister les fichiers en dehors de notre application	235
Versionner les schémas de base de données	236
Démarrer automatiquement une application	238
L'hébergeur s'en occupe à notre place	238
Avec un gestionnaire de processus	238
En créant un service système	239
Avec un serveur d'applications web	240
À quoi penser après la mise en ligne ?	242
L'application a planté	242
S'informer des erreurs applicatives	242
Notre version de Node fait l'objet d'une faille de sécurité	244
Un des modules npm fait l'objet d'une faille de sécurité	245
Conclusion	247

CHAPITRE 7

Créer une application web..... 249

Composer son application web	250
Démarrer un serveur HTTP	252
Répondre à un chemin (routing)	253
Répondre avec des fichiers statiques	258
Réagir aux arguments d'URL	261
Recevoir des données de formulaire (POST)	264
Téléverser des fichiers	268
Garder un lien avec les cookies	271
Structurer l'affichage avec les gabarits de présentation	274
Pendant le développement : relancer le serveur automatiquement	277
Organiser une application avec le framework Express	278
Configuration du framework	278
Greffer des extensions (middlewares)	279
Brancher les gabarits de présentation	282
Intégrer les ressources front-end (CSS, images, JavaScript)	284
Brancher une base de données	286
Sessions utilisateur	288
Tracer les actions (logs)	290
Vers un code réutilisable et testable	291
Modulariser le code des routes	291
Un code testable est un code indépendant du framework	295
Déployer automatiquement	303
Pour aller plus loin	304
Pourquoi lancer un serveur ?	304
Comprendre le modèle HTTP	304
Quel(s) moteur(s) de bases de données choisir ?	306
Protéger nos applications	307
Une application minimaliste avec les Lambda	309
Conclusion	310

CHAPITRE 8

Créer un outil en ligne de commandes..... 313

Créer un script exécutable	314
Au départ, un simple script Node	314
Modifier les permissions du script	315
Préciser le contexte d'exécution (shebang)	316
Faire le lien avec un module npm	317
Du script au programme interactif	318
Utiliser des arguments et des options	318
Améliorer la lisibilité grâce aux couleurs	325
Demander une série d'informations	328
Informé de la progression	331
Afficher des informations sous forme de tableau	335
Inviter à mettre à jour le module	336
Vers un code réutilisable et testable	338

Modulariser le code du fichier exécutable	338
Tester le code partagé	339
Présenter les messages en contexte	341
Tester l'exécutable	342
Documenter notre programme	343
Pour aller plus loin	346
Utilisation d'un framework d'application en ligne de commandes	346
Stratégies pour gérer les chemins d'accès	351
Utiliser les flux de données (stdin, stdout et stderr)	352
Activer l'autocomplétion des commandes	357
Rendre le programme indépendant de Node	359
Conclusion	361

CHAPITRE 9

Créer une application front-end..... 363

Quel rapport entre Node et les navigateurs web ?	364
Écrire dès à présent le code du futur	365
La fin de l'approche par le dénominateur commun	365
Écrire au plus proche des standards	365
Comblar les manques avec des polyfills	368
Importer des modules	371
La balise <script>	371
Les modules ECMAScript	372
Importer des modules npm pour le Web	374
Récapitulatif	376
Conception modulaire	376
Le syndrome du plug-in jQuery	377
Vers une approche jQuery composite	379
Partager le code métier avec Node	380
Séparation du fond et de la forme : données, rendu et interactions	381
Rapprocher données, rendu et interactions avec React	381
Des requêtes Ajax vers du temps réel	384
Échange ponctuel de données avec fetch()	385
Approche unidirectionnelle avec EventSource	387
Échanges en temps réel avec WebSocket	389
Développer au quotidien	392
Reconstruire en continu avec watchify	392
Changements en temps réel dans le navigateur	393
Modulariser ses feuilles de styles avec Sass	397
Lier composants visuels et feuilles de styles	399
Optimiser ses ressources graphiques	400
Tester son code	402
Que tester ?	402
S'outiller pour écrire des assertions	404
Tester ses composants React sans navigateur	405
Tester code et composants dans les navigateurs	408
Intégration continue et compatibilité navigateurs	412
Conclusion	415

ANNEXE

Sélection de modules npm	417
Boîte à outils du quotidien	418
lodash	418
he	418
chance	419
date-fns	419
tcomb	419
eventemitter3	419
Pendant le développement	420
debug	420
nodemon	421
npm-run-all	421
husky	422
onchange	422
tiny-lr	422
livereactload	423
Protéger nos applications	423
sanitize-filename	423
helmet	423
dompurify	424
filenamify	424
retire.js	424
snyk	424
Vérifier la syntaxe de notre code	424
htmlhint	424
eslint	425
csslint	426
doouse	427
Optimiser notre code	428
uglify-es	428
postcss	428
autoprefixer	429
uncss	429
csswring	429
google-closure-compiler-js	429
csso	430
svgo	430
wawoff2	430
Passer d'un langage à un autre	430
less	430
sass	431
browserify	431
babel	431
typescript	431
Gérer des fichiers	432
mkdirp	432

rimraf	432
glob	432
fs-extra	432
graceful-fs	433
chokidar	433
Stocker dans des bases de données	433
db-migrate	433
knex	433
bookshelf	434
sequelize	434
mongoose	435
levelup	435
redis	435
elasticsearch	436
ligne de commandes	436
yargs-parser	436
args	436
caporal	437
promptly	437
supports-color	437
Applications web	437
fastify	437
passport	437
restify	438
faye	438
swagger-client	439
Modules front-end et templating	439
nunjucks	440
handlebars	440
pug	441
ejs	441
react	441
storybook	442
Tester notre code	442
tape	442
chai	443
sinon	443
nyc	444
karma	445
supertest	445
Objets connectés	446
nitrogen	446
serialport	446
firmata	446
johnny-five	446
node-red	446
Accessibilité	447
a11y	447

a11y.css	447
lighthouse	447
storybook-addon-a11y	447
Travailler avec des images, des sons et des vidéos	448
sharp	448

Index	449
--------------------	------------

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 compili-

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 :

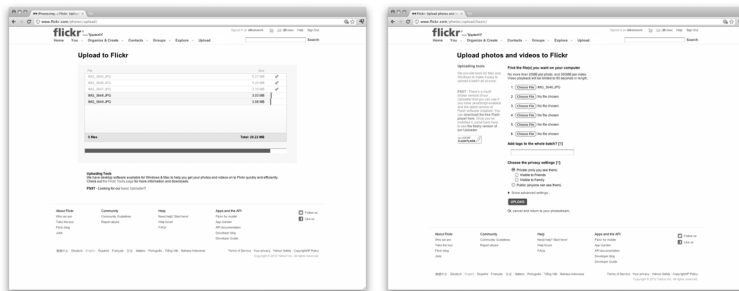
- la possibilité d'inclure un serveur HTTP comme bibliothèque applicative ;
- la simplicité de fonctionnement : recevoir une requête HTTP et décider soi-même de la réponse à apporter.

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 dé clic se produit en janvier 2009, lorsque JavaScript émerge dans une discussion entre développeurs. *Eureka !* 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/a3d0bbbf196af633995> et <https://www.youtube.com/watch?v=ztspvPYyb1Y>) 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'égaliser 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

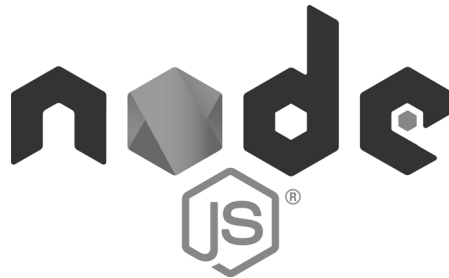
Joyent est une entreprise américaine fondée en 2004. Elle propose à l'origine des services de collaborations en ligne : documents, calendriers, courriels, etc.

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 RubyGems (<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/annoncements/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 résistante 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 participants 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.