

 eBook Gratuit

APPRENEZ

meteor

eBook gratuit non affilié créé à partir des
contributeurs de Stack Overflow.

#meteor

Table des matières

À propos.....	1
Chapitre 1: Démarrer avec meteor.....	2
Remarques.....	2
Versions.....	2
Exemples.....	3
Commencer.....	3
Installer Meteor.....	3
Sous OS X et Linux.....	3
Sous Windows.....	3
Créez votre application.....	3
Exécuter.....	3
Exemples d'applications.....	4
Gestion des packages.....	4
Comprendre le progrès de la construction.....	5
Exemple Linux / OSX.....	5
Exemple Windows.....	5
Vérification de la version des projets Meteor Tool et Meteor.....	5
Outil Meteor.....	5
Projets de météores.....	5
Site Web Meteor.....	6
Mise à jour des projets Meteor et des packages installés.....	6
Construire des applications mobiles.....	7
Chapitre 2: Accéder aux machines de construction Meteor à partir de Windows.....	8
Remarques.....	8
Exemples.....	8
Utiliser PuTTY (Advanced).....	8
Utilisation de Cygwin (outils Unix sous Windows).....	8
Chapitre 3: Agrégation MongoDB.....	10
Remarques.....	10

Exemples.....	10
Agrégation de serveurs.....	10
Agrégation dans une méthode serveur.....	11
Chapitre 4: Application mobile.....	12
Exemples.....	12
Mise en page sur différents appareils - CSS.....	12
Windows de taille fixe.....	12
Mise en cache hors ligne.....	12
Désactiver Scroll-Bounce.....	13
Multitouch et gestes.....	13
Créez vos icônes et vos ressources d'écran de démarrage.....	14
Pipeline d'Architecture Meteor Cordova.....	15
Développement IOS.....	16
Test de périphérique IOS.....	16
Configurez votre projet Cordova (config.xml).....	16
Détecter l'événement déviant.....	17
Chapitre 5: Applications hors ligne.....	18
Remarques.....	18
Exemples.....	18
Meteor.status ().....	18
Activer Appcache.....	18
Activer GroundDB.....	19
Choses à faire attention.....	19
Chapitre 6: Blaze Templating.....	20
Introduction.....	20
Exemples.....	20
Remplir un modèle à partir d'un appel de méthode.....	20
Contexte de données d'un modèle.....	20
Aide du modèle.....	21
Chapitre 7: Collections Mongo.....	23
Remarques.....	23
Exemples.....	23

Création d'enregistrements dans une base de données existante.....	23
Insérer des données dans un document.....	23
Obtenir le _id du document le plus récemment créé.....	23
Données de séries chronologiques.....	24
Filtrage avec des expressions régulières.....	24
Collections géospatiales - Apprendre plus.....	25
Auditer les requêtes de collection.....	26
Fonctions des observateurs et des travailleurs.....	26
Chapitre 8: Comptes utilisateurs Meteor.....	28
Exemples.....	28
Paquet de comptes Meteor.....	28
Comptes-mot de passe.....	28
Accès aux données utilisateur.....	29
Autres fonctions de comptes.....	29
Ne pas utiliser le champ de profil par défaut.....	29
Chapitre 9: Configuration de base de codeship pour le test automatisé.....	31
Exemples.....	31
Configuration du code d'accès.....	31
Préparer le projet.....	31
Chapitre 10: Déploiement avec Upstart.....	33
Exemples.....	33
Service de démarrage.....	33
Copier des fichiers sur votre serveur puis générer.....	33
Bundle puis copier sur le serveur.....	33
Écrire votre script de démarrage.....	33
Script de démarrage pour les jeux de réplicas.....	34
Exécuter votre script de démarrage.....	34
Configuration d'un serveur pour héberger plusieurs applications Meteor.....	35
Chapitre 11: Déploiement continu sur Galaxy de Codeship.....	36
Remarques.....	36
Exemples.....	36
Installer.....	36

Chapitre 12: Détection de l'environnement	37
Exemples	37
Configurations d'environnement avancées	37
Spécification des paramètres de l'application avec METEOR_SETTINGS	37
Détection d'environnement sur le serveur	38
Détection de l'environnement client à l'aide des méthodes Meteor	38
Détection de l'environnement client à l'aide de NODE_ENV	39
Chapitre 13: Electrify - Compiler Meteor comme une application localement installable	40
Exemples	40
Installer Electrify pour une application Meteor	40
Utiliser l'électrification sur une application Meteor	41
Chapitre 14: Enregistrement	43
Exemples	43
Basic Server Side Logging	43
Outils de journalisation côté client	43
Outils avancés de journalisation de serveur	43
Erreur de journalisation sur le volet de la base de données	44
Informations de journalisation sur le contexte de données dans un assistant de modèle	44
Journalisation des événements et des interactions utilisateur	44
Journalisation avec des variables de niveau journal	44
Désactiver la journalisation en production	45
Winston	45
Niveau de la logle	45
Chapitre 15: Enveloppant les méthodes asynchrones dans une fibre pour une exécution synchr	47
Syntaxe	47
Paramètres	47
Remarques	47
Exemples	47
Exécution synchrone de méthodes NPM asynchrones avec callbacks	47
Chapitre 16: ES2015 modules (Import & Export)	49
Remarques	49
Exemples	49

Importation dans des modules d'application.....	49
Importer dans les packages Meteor.....	49
Exportation de variables à partir de modules d'application.....	49
Exportation de symboles de packages Meteor.....	50
Chapitre 17: ESLint.....	51
Exemples.....	51
Ajout d'eslint à votre projet Meteor.....	51
Utiliser un script npm pour filtrer votre code.....	51
Chapitre 18: Guide d'initiation à l'installation de Meteor 1.4 sur AWS EC2.....	52
Exemples.....	52
Inscription au service AWS.....	52
Chapitre 19: Installation complète - Mac OSX.....	58
Exemples.....	58
Installation du noeud et du NPM.....	58
Procédure d'installation de Meteor.....	58
Installation Mongo.....	59
Autres outils de développement.....	59
Chapitre 20: Intégration continue et nuages de périphériques (avec Nightwatch).....	61
Remarques.....	61
Exemples.....	61
Travis.....	61
Cercle.....	62
SauceLabs.....	64
BrowserStack.....	64
Chapitre 21: Intégration d'API tiers.....	66
Exemples.....	66
Appel HTTP basique.....	66
Créer un package pour votre wrapper API.....	66
Créer un package d'atmosphère pour votre wrapper API.....	66
Inclure le package API dans votre application.....	67
Utilisation de l'objet wrapper API dans votre application.....	67
Chapitre 22: Jeux de répliques et de fragmentation.....	68

Remarques.....	68
Exemples.....	68
Jeu de répliques Quickstart.....	68
Configuration du jeu de réplicas.....	69
Chapitre 23: L'optimisation des performances.....	70
Remarques.....	70
Exemples.....	70
Conception et déploiement de logiciels prêts pour la production.....	70
Chapitre 24: Le débogage.....	72
Exemples.....	72
Débogueurs de navigateur.....	72
Ajouter des points d'arrêt du débogueur à votre application.....	72
Débogage côté serveur avec l'inspecteur de noeud.....	72
Débogage côté serveur avec débogage npm.....	73
Meteor Shell.....	73
Autres utilitaires de débogage.....	73
Chapitre 25: Le routage.....	74
Exemples.....	74
Routage avec routeur de fer.....	74
Avec FlowRouter.....	75
Installez FlowRouter.....	75
Rendu d'un modèle.....	75
Rendu d'un modèle avec des paramètres et / ou une requête.....	76
Chapitre 26: Les atouts.....	77
Exemples.....	77
Accès aux ressources sur le serveur.....	77
Fichiers texte.....	77
Fichiers binaires.....	77
Chapitre 27: Meteor + React + ReactRouter.....	78
Introduction.....	78
Exemples.....	78

Créer le projet.....	78
Remarque:	78
Ajouter React + ReactRouter.....	79
Remarque:	80
Étape 3 - Ajouter des comptes.....	80
Remarque:	81
Ajouter des rôles.....	81
Remarque:	84
Chapitre 28: Meteor + Réagir	85
Remarques.....	85
Exemples.....	85
Configuration et "Hello World".....	85
Créer un conteneur réactif à l'aide de createContainer.....	85
Affichage d'une collection MongoDB.....	86
Chapitre 29: Migrations du schéma Mongo	89
Remarques.....	89
Exemples.....	89
Ajouter un champ de version à tous les enregistrements d'une collection.....	89
Supprimer un tableau de tous les enregistrements d'une collection.....	89
Renommer la collection.....	89
Rechercher un champ contenant une chaîne spécifique.....	89
Créer un nouveau champ à partir de l'ancien.....	90
Sortez les objets d'un tableau et placez-les dans un nouveau champ.....	90
Blob Record d'une collection à une autre collection (par exemple, supprimer Join & Flatten.....	90
Assurez-vous que le champ existe.....	90
Assurez-vous que le champ a une valeur spécifique.....	90
Supprimer un enregistrement si un champ spécifique est une valeur spécifique.....	91
Changer la valeur spécifique du champ en nouvelle valeur.....	91
Champ spécifique non défini sur Null.....	91
Convertir ObjectId en chaîne.....	91
Convertir les valeurs de champ des nombres en chaînes.....	91
Convertir les valeurs de champs de chaînes en nombres.....	91

Créer un horodatage à partir d'un ObjectId dans le champ <code>_id</code>	92
Créer un ObjectId à partir d'un objet de date.....	92
Rechercher tous les enregistrements contenant des éléments dans un tableau.....	92
Chapitre 30: Mise à l'échelle horizontale.....	93
Exemples.....	93
Déploiement d'une application avec une base de données séparée (MONGO_URL).....	93
Configuration du jeu de réplicas.....	93
Configuration d'un jeu de réplicas pour utiliser l'oscillation.....	93
Script de démarrage Oplog.....	94
Éclatement.....	94
Chapitre 31: Mongo Database Management.....	95
Remarques.....	95
Exemples.....	95
Analyser une base de données héritée.....	95
Connectez-vous à une base de données sur <code>*.meteorapp.com</code>	95
Téléchargez une base de données à partir de <code>*.meteor.com</code>	96
Exporter des données depuis une instance de développement Meteor locale?.....	96
Restaurer des données à partir d'un fichier de vidage.....	96
Exporter une collection vers JSON.....	96
Importer un fichier JSON dans Meteor.....	96
Copie de données entre bases de données intermédiaires et locales.....	97
Compacter une base de données Mongo sur une boîte Ubuntu.....	97
Réinitialiser un jeu de répliques.....	98
Connectez-vous à distance à une instance Mongo sur <code>*.meteor.com</code>	98
Accès aux fichiers journaux Mongo sur une instance Meteor locale.....	98
Faire pivoter les fichiers journaux sur une boîte Ubuntu.....	98
Chapitre 32: MongoDB.....	100
Introduction.....	100
Exemples.....	100
Exportation d'une base de données Mongo distante, importation dans une base de données Met.....	100
Obtenez l'URL Mongo de votre base de données locale Meteor Mongo.....	100
Connectez votre application Meteor locale à une autre base de données Mongo.....	100

Exemple Linux / MacOS:.....	100
Exemple Windows.....	101
MNP.....	101
Running Meteor sans MongoDB.....	101
Commencer.....	101
Documents de requête.....	102
Insérer des documents.....	102
Mise à jour des documents.....	102
Suppression de documents.....	102
Chapitre 33: Nightwatch - Configuration et configuration.....	104
Remarques.....	104
Exemples.....	104
Configuration.....	104
Installation et utilisation.....	105
Configuration des scripts de lancement.....	106
Structure des dossiers.....	107
Tests basés sur les données.....	107
Chapitre 34: Node / NPM.....	109
Exemples.....	109
Version de nœud testée / prise en charge de Meteor.....	109
Chapitre 35: Outils de développement.....	110
Exemples.....	110
Environnements de développement intégrés.....	110
Outils de base de données.....	110
Utilitaires de collaboration à distance pour les développeurs distribués.....	110
Clients REST.....	111
Débogueurs.....	111
Mobile Coding sur iOS.....	111
Chapitre 36: Publication d'une piste de publication.....	113
Remarques.....	113
Exemples.....	113

Utilisation de base.....	113
Manifeste manifeste.....	113
Personnalisation de l'outil Meteor.....	114
Extraire un manifeste de version de .meteor / versions.....	114
Affichage du manifeste de version pour une version spécifique.....	114
Publication d'un communiqué de presse.....	114
Récupération des derniers commits pour chaque package dans une version.....	114
Chapitre 37: Publication de données.....	116
Remarques.....	116
Exemples.....	116
Abonnement de base et publication.....	116
Publications mondiales.....	117
Publications nommées.....	117
Abonnements à des modèles.....	117
Publier dans une collection nommée éphémère côté client.....	118
Créer et répondre à une erreur sur une publication.....	118
Réinscription réactive à une publication.....	119
Attendez dans la vue Blaze pendant l'extraction des données publiées.....	119
Validation du compte d'utilisateur sur Publier.....	120
Publier plusieurs curseurs.....	120
Simuler le retard dans les publications.....	120
Fusion de publications.....	121
Chapitre 38: Réactif (Vars et Dictionnaires).....	122
Exemples.....	122
Requête Réactive.....	122
Chapitre 39: Recettes de l'interface utilisateur Blaze (Bootstrap; No jQuery).....	124
Remarques.....	124
Exemples.....	124
Menu déroulant.....	124
Navbars.....	125
Modals.....	126
Le marquage.....	127

Alertes et erreurs.....	129
Flux de travail par onglets.....	131
Chapitre 40: Récupération des données d'un Meteor.call.....	134
Exemples.....	134
Les bases de Meteor.call.....	134
Utilisation de la variable de session.....	135
Du côté serveur.....	135
Côté client.....	135
Utiliser ReactiveVar.....	135
Du côté serveur.....	135
Côté client.....	136
Chapitre 41: Structure du répertoire.....	137
Introduction.....	137
Remarques.....	137
Exemples.....	138
Structures de répertoire classiques.....	138
Structure de répertoire uniquement pour les packages.....	138
Structure du répertoire des importations / modules.....	139
Structure de répertoire en mode mixte.....	139
Ordre de chargement du répertoire.....	139
Chapitre 42: Tâches d'arrière-plan.....	141
Remarques.....	141
Exemples.....	141
Cron simple.....	141
Chapitre 43: Téléchargement de fichier.....	142
Remarques.....	142
Exemples.....	142
Serveur / Client.....	142
Dropzone (avec fer: routeur).....	144
Filepicker.io.....	145
CollectionFS.....	145
Téléchargement de serveur.....	146

Chapitre 44: Test d'acceptation (avec Nightwatch)	148
Remarques	148
Exemples	148
Surface de l'application	148
Commandes personnalisées	149
Inspection des objets Meteor sur le client	150
Formulaires et types d'entrées	150
Composants et objets de page	152
Chapitre 45: Utilisation de polymère avec Meteor	154
Exemples	154
En utilisant le différentiel: vulcaniser	154
Chapitre 46: Utiliser des paquets de météorites privés sur Codeship	156
Remarques	156
Exemples	156
Installer MGP	156
Configurer Codeship pour installer des packages Github privés	156
Chapitre 47: Utiliser Meteor avec un serveur proxy	158
Exemples	158
En utilisant la variable d'envoi HTTP [S]_PROXY`	158
Configuration d'un niveau de proxy	158
Chapitre 48: Variables d'environnement	159
Paramètres	159
Exemples	161
Utilisation de variables d'environnement avec Meteor	161
Paramétrage du serveur SMTP Meteor	161
Crédits	162

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [meteor](#)

It is an unofficial and free meteor ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official meteor.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec meteor

Remarques

Meteor est une plate **-forme** JavaScript **complète** pour le développement d'applications Web et mobiles modernes.

Dans *un* projet, vous pouvez créer votre client (navigateur et / ou application mobile hybride pour Android et / ou iOS) et les côtés du serveur.

Pages de référence:

- [Guide des météores](#)
- [Documents de Meteor API](#)
- [Tutoriels Meteor](#)
- [Forums Meteor](#)

Versions

Version	Date de sortie
0.4.0	2012-08-30
0.5.0	2013-10-17
0.6.0	2013-04-04
0.7.0	2013-12-20
0.8.0	2014-04-21
0.9.0	2014-08-26
0.9.1	2014-09-04
0.9.2	2014-09-15
0.9.3	2014-09-25
0.9.4	2014-10-13
1.0.1	2014-12-09
1.0.2	2014-12-19
1.0.3.1	2014-12-09
1.1.0	2015-03-31

Version	Date de sortie
1.2.0	2015-09-21
1.3.0	2016-03-27
1.4.0	2016-07-25
1.5.0	2017-05-30

Exemples

Commencer

Installer Meteor

Sous OS X et Linux

Installez la dernière version officielle de Meteor depuis votre terminal:

```
$ curl https://install.meteor.com/ | sh
```

Sous Windows

[Téléchargez l'installateur officiel de Meteor ici](#) .

Créez votre application

Une fois que vous avez installé Meteor, créez un projet:

```
$ meteor create myapp
```

Exécuter

Exécutez-le localement:

```
$ cd myapp  
$ meteor npm install  
$ meteor
```


Remarque: serveur Meteor exécuté sur: <http://localhost:3000/>

Ensuite, rendez-vous sur <http://localhost:3000> pour voir votre nouvelle application Meteor.

- En savoir plus sur les débuts avec Meteor au [\[Meteor Guide\]](#) .
- Explorez Meteor Packages à l' [atmosphère](#) - un gestionnaire de paquets moderne, rapide et bien conçu.

Exemples d'applications

Meteor a plusieurs exemples d'applications intégrées. Vous pouvez créer un projet avec l'un d'entre eux et apprendre comment il a été construit. Pour créer un exemple d'application, installez Meteor (voir [Mise en route](#)), puis tapez:

```
meteor create --example <app name>
```

Par exemple, pour créer un exemple d'application `todos` , écrivez:

```
meteor create --example todos
```

Pour obtenir une liste de toutes les applications exemples, tapez:

```
meteor create --list
```

Gestion des packages

Meteor a son propre référentiel de paquets sur atmospherejs.com

Vous pouvez ajouter de nouveaux paquets à partir de l'atmosphère en exécutant:

```
meteor add [package-author-name:package-name]
```

Par exemple:

```
meteor add kadora:flow-router
```

De même, vous pouvez supprimer le même package en:

```
meteor remove kadora:flow-router
```

Pour voir les packages actuels dans votre projet, tapez:

```
meteor list
```

La liste des paquets peut également être trouvée dans le fichier `./meteor/packages` . Pour ajouter un package, ajoutez le nom du package dans ce fichier et supprimez-le.

Pour ajouter un package localement (par exemple, des packages non publiés ou une version modifiée des packages publiés), enregistrez le package dans le dossier `packages` dans la racine.

À partir de la version 1.3, Meteor a **ajouté un support pour les paquets npm** .

Vous pouvez utiliser la commande `npm` dans le répertoire du projet Meteor comme vous le feriez normalement sans Meteor, ou avec la commande `meteor npm` , qui utilisera la version intégrée de npm.

Comprendre le progrès de la construction

Parfois, les builds prennent plus de temps que prévu. Vous pouvez définir quelques variables d'environnement pour mieux comprendre ce qui se passe pendant le processus de construction.

```
METEOR_DEBUG_BUILD=1      (logs progress)
METEOR_PROFILE=<n>        (logs time spent)
METEOR_DEBUG_SPRINGBOARD=1 (?)
METEOR_DEBUG_SQL=1       (logs SQLITE calls)
METEOR_PROGRESS_DEBUG=1  (? looks like it might be useful, but seems confusing)
```

Où `<n>` est un nombre de ms. Tout processus nécessitant plus de temps sera enregistré.

Exemple Linux / OSX

```
export METEOR_DEBUG_BUILD=1
export METEOR_PROFILE=100
meteor
```

Exemple Windows

```
set METEOR_DEBUG_BUILD=1
set METEOR_PROFILE=100
meteor
```

Vérification de la version des projets Meteor Tool et Meteor

Outil Meteor

Pour vérifier la version installée de l'outil Meteor, exécutez simplement la commande suivante en dehors des projets Meteor:

```
meteor --version
```

Pour obtenir une liste de toutes les versions officielles (recommandées) de Meteor, lancez:

```
meteor show METEOR
```

Projets de météores

Si vous voulez vérifier la version du projet de Meteor, vous pouvez également exécuter la commande suivante dans un projet:

```
meteor --version
```

ou simplement imprimer le contenu du fichier `.meteor/release` :

```
cat .meteor/release
```

Si vous souhaitez vérifier la version des packages actuellement installés dans votre projet Meteor, imprimez le contenu du fichier `.meteor/versions` :

```
cat .meteor/versions
```

Site Web Meteor

Pour voir quelle version de Meteor un site Web basé sur Meteor fonctionne, vider le contenu de `Meteor.release` dans la console de votre navigateur tout en visitant le site Web:

```
Meteor.release
```

Mise à jour des projets Meteor et des packages installés

L'outil Meteor vous avertit lorsqu'une nouvelle version est disponible.

Pour mettre à jour les projets Meteor vers la dernière version, exécutez la commande suivante dans un projet Meteor:

```
meteor update
```

Si vous souhaitez mettre à jour votre projet Meteor vers une version spécifique de Meteor, exécutez la commande suivante dans le projet:

```
meteor update --release <release>
```

Si vous souhaitez mettre à jour tous les packages non-core, exécutez:

```
meteor update --packages-only
```

Vous pouvez également mettre à jour des packages spécifiques en transmettant leurs noms en tant qu'argument de ligne de commande à `meteor update` , par exemple:

```
meteor update [packageName packageName2 ...]
```

Construire des applications mobiles

Meteor utilise [Cordova](#) pour intégrer votre application dans une application mobile *hybride* . Une fois empaquetée, l'application peut être distribuée comme des applications natives (via Apple App Store, Google Play Store, etc.)

1. **Ajoutez** la ou les plates-formes cibles à votre projet Meteor:

```
meteor add-platform android
meteor add-platform ios # Only available with Mac OS
```

2. **Installez** le SDK Android et / ou Xcode (pour iOS, nécessite Mac OS).

3. **Exécutez** votre projet (commencez avec le mode de développement):

```
meteor run android # You may need to configure a default Android emulator first
```

Pour iOS (uniquement disponible avec Mac OS):

```
meteor run ios # This will auto start an iOS simulator
```

4. **Construisez** votre package d'application pour la distribution:

```
meteor build <output_folder> --server <url_app_should_connect_to>
```

Cela créera un ou plusieurs dossiers `android` et / ou `ios` côté de votre ensemble de serveurs.

- Le dossier `android` contient le fichier `release-unsigned.apk` que vous devez signer et zip aligner.
- Le dossier `ios` contient le projet Xcode que vous devez signer.

Voir aussi la rubrique Meteor [Mobile Apps](#) .

Page de référence: [Meteor Guide](#)> [Build](#)> [Mobile](#)

Lire [Démarrer avec meteor en ligne](#): <https://riptutorial.com/fr/meteor/topic/439/demarrer-avec-meteor>

Chapitre 2: Accéder aux machines de construction Meteor à partir de Windows

Remarques

Sous Mac et Linux, l'outil de ligne de commande `meteor` suppose que l'outil de ligne de commande `ssh`, utilisé pour établir des connexions sécurisées à d'autres ordinateurs, est toujours présent. Sous Windows, cet outil doit être installé. Vous trouverez ci-dessous deux options pour le configurer et l'utiliser.

Exemples

Utiliser PuTTY (Advanced)

Si vous ne souhaitez pas ajouter de commandes Unix à votre PATH sous Windows, vous pouvez télécharger un client SSH autonome comme PuTTY. [Téléchargez PuTTY ici](#), puis suivez les instructions ci-dessous pour obtenir une machine de génération.

1. Appelez `meteor admin get-machine <os-architecture> --json`
2. Copier et enregistrer la clé privée à partir des données JSON renvoyées
3. Suivez les instructions [ici](#) pour convertir la clé privée dans un format accepté par PuTTY
4. Entrez le nom d'hôte, le nom d'utilisateur et la clé privée dans PuTTY, et vous êtes prêt à partir!

Utilisation de Cygwin (outils Unix sous Windows)

La manière la plus simple de démarrer est d'installer Git for Windows à partir de [cette page de téléchargement](#) et de sélectionner «Utiliser les outils Git et Unix facultatifs à partir de l'invite de commande Windows», comme dans la capture d'écran ci-dessous.

Adjusting your PATH environment

How would you like to use Git from the command line?

Use Git from Git Bash only

This is the safest choice as your PATH will not be modified and you will be able to use the Git command line tools from Git Bash.

Use Git from the Windows Command Prompt

This option is considered safe as it only adds some minimal Unix tools to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from both Git Bash and the Windows Command Prompt.

Use Git and optional Unix tools from the Windows Command Prompt

Both Git and the optional Unix tools will be added to your PATH.

Warning: This will override Windows tools like "find" and "dir". Use this option if you understand the implications.

<http://msysgit.github.io/>

< Back

Après cela, `met-meteor admin get-machine <os-architecture>` fonctionnera exactement comme sur Linux et Mac. N'oubliez pas que vous devrez peut-être redémarrer votre terminal pour obtenir les nouvelles commandes.

Lire [Accéder aux machines de construction Meteor à partir de Windows en ligne](https://riptutorial.com/fr/meteor/topic/518/accéder-aux-machines-de-construction-meteor-a-partir-de-windows):

<https://riptutorial.com/fr/meteor/topic/518/accéder-aux-machines-de-construction-meteor-a-partir-de-windows>

Chapitre 3: Agrégation MongoDB

Remarques

Agrégation de serveurs

[Requêtes d'agrégation moyenne dans Meteor](#)

est-il possible de conditionner une vraie bibliothèque mongodb à utiliser du côté * server * uniquement dans meteor 0.6

Agrégation de clients (Minimongo)

<https://github.com/utunga/pocketmeteor/tree/master/packages/mongowrapper>

Exemples

Agrégation de serveurs

La solution d'Andrew Mao. [Requêtes d'agrégation moyenne dans Meteor](#)

```
Meteor.publish("someAggregation", function (args) {
  var sub = this;
  // This works for Meteor 0.6.5
  var db = MongoInternals.defaultRemoteCollectionDriver().mongo.db;

  // Your arguments to Mongo's aggregation. Make these however you want.
  var pipeline = [
    { $match: doSomethingWith(args) },
    { $group: {
      _id: whatWeAreGroupingWith(args),
      count: { $sum: 1 }
    }}
  ];

  db.collection("server_collection_name").aggregate(
    pipeline,
    // Need to wrap the callback so it gets called in a Fiber.
    Meteor.bindEnvironment(
      function(err, result) {
        // Add each of the results to the subscription.
        _.each(result, function(e) {
          // Generate a random disposable id for aggregated documents
          sub.added("client_collection_name", Random.id(), {
            key: e._id.somethingOfInterest,
            count: e.count
          });
        });
        sub.ready();
      },
      function(error) {
        Meteor._debug( "Error doing aggregation: " + error);
      }
    )
  );
});
```

```
    )  
  );  
});
```

Agrégation dans une méthode serveur

Une autre façon de faire des agrégations est d'utiliser `Mongo.Collection#rawCollection()`

Cela ne peut être exécuté que sur le serveur.

Voici un exemple que vous pouvez utiliser dans Meteor 1.3 et supérieur:

```
Meteor.methods({  
  'aggregateUsers'(someId) {  
    const collection = MyCollection.rawCollection()  
    const aggregate = Meteor.wrapAsync(collection.aggregate, collection)  
  
    const match = { age: { $gte: 25 } }  
    const group = { _id:'$age', totalUsers: { $sum: 1 } }  
  
    const results = aggregate([  
      { $match: match },  
      { $group: group }  
    ])  
  
    return results  
  }  
})
```

Lire Agrégation MongoDB en ligne: <https://riptutorial.com/fr/meteor/topic/4199/agregation-mongodb>

Chapitre 4: Application mobile

Exemples

Mise en page sur différents appareils - CSS

Si votre application doit s'exécuter sur différents périphériques, il faudra la rendre dans différents ViewPorts, en fonction de la taille de l'appareil. Vous pouvez y faire face de deux manières: avec des règles javascript ou des styles de support CSS. Si vous avez utilisé une bibliothèque MVC ou MVVM, telle que Angular ou Ember (ou Blaze, par exemple) et que vous ne ciblez qu'un seul périphérique ou une seule plate-forme matérielle, vous devrez peut-être repenser votre modèle MVC. présenté à votre application.

```
// desktop
@media only screen and (min-width: 960px) {
}

// landscape orientation
@media only screen and (min-width: 768px) {
}

// portrait orientation
@media only screen and (min-width: 480px) {
}
```

Vous devrez déterminer si vous voulez casser les styles à 768px (mode portrait) ou à 1024 pixels (paysage). En supposant que votre appareil mobile cible soit l'iPad, qui utilise un ratio de 3: 4. Sinon, vous devrez déterminer les proportions des périphériques que vous souhaitez cibler et déterminer les niveaux de seuil à partir de là.

Windows de taille fixe

Si vous envisagez de concevoir des mises en page avec des écrans de taille fixe pour différents appareils mobiles, vous souhaitez peut-être reproduire cette conception lors de l'exécution de votre application sur un bureau. La méthode suivante corrige la taille de la fenêtre OUTSIDE de PhoneGap, donnant une fenêtre de taille fixe sur le bureau. Parfois, il est plus facile de gérer les attentes des utilisateurs et la conception de l'interface utilisateur en limitant les options!

```
// create a window of a specific size
var w=window.open('', '', 'width=100,height=100');
w.resizeTo(500,500);

// prevent window resize
var size = [window.width,window.height]; //public variable
$(window).resize(function() {
    window.resizeTo(size[0],size[1]);
});
```

Mise en cache hors ligne

Pour que tout cela fonctionne, vous aurez probablement besoin d'une assistance hors ligne, ce qui signifie la mise en cache des données d'application et des données utilisateur.

```
meteor add appcache
meteor add grounddb
```

Désactiver Scroll-Bounce

Sur les applications de bureau, vous pouvez désactiver le défilement-défilement pour donner à votre application un aspect plus natif. Vous pouvez le faire avec JavaScript, en désactivant la façon dont le navigateur contrôle le DOM:

```
// prevent scrolling on the whole page
// this is not meteorish; TODO: translate to meteor-centric code
document.ontouchmove = function(e) {e.preventDefault()};

// prevent scrolling on specific elements
// this is not meteorish; TODO: translate to meteor-centric code
scrollableDiv.ontouchmove = function(e) {e.stopPropagation()};
```

Vous pouvez également utiliser css, ainsi que les styles de débordement et de défilement.

```
#appBody {
  overflow: hidden;
}

#contentContainer {
  .content-scrollable {
    overflow-y: auto;
    -webkit-overflow-scrolling: touch;
  }
}
```

Le modèle d'objet nécessaire au fonctionnement ci-dessus ressemble à ceci:

```
<div id="appBody">
  <div id="contentContainer">
    <div class="content-scrollable">
      <!-- content -->
    </div>
  </div>
</div>
```

Multitouch et gestes

Les appareils mobiles n'ont généralement pas de clavier, vous devrez donc ajouter des contrôleurs haptiques à votre application. Les deux paquets populaires que les gens semblent utiliser sont FastClick et Hammer. L'installation est facile.

```
meteor add fastclick
meteor add hammer:hammer
```

FastClick ne nécessite presque aucune configuration, tandis que Hammer nécessite un peu de travail pour se connecter. L'exemple cononical de l'application Todos ressemble à ceci:

```
Template.appBody.onRendered(function() {
  if (Meteor.isCordova) {
    // set up a swipe left / right handler
    this.hammer = new Hammer(this.find('#appBody'));
    this.hammer.on('swipeleft swiperight', function(event) {
      if (event.gesture.direction === 'right') {
        Session.set(MENU_KEY, true);
      } else if (event.gesture.direction === 'left') {
        Session.set(MENU_KEY, false);
      }
    });
  }
});
```

Créez vos icônes et vos ressources d'écran de démarrage

Avant de compiler votre application et de l'exécuter sur votre appareil, vous devez créer des icônes et des écrans de démarrage et ajouter un fichier `mobile-config.js` à votre application.

```
App.icons({
  // iOS
  'iphone': 'resources/icons/icon-60x60.png',
  'iphone_2x': 'resources/icons/icon-60x60@2x.png',
  'ipad': 'resources/icons/icon-72x72.png',
  'ipad_2x': 'resources/icons/icon-72x72@2x.png',

  // Android
  'android_ldpi': 'resources/icons/icon-36x36.png',
  'android_mdpi': 'resources/icons/icon-48x48.png',
  'android_hdpi': 'resources/icons/icon-72x72.png',
  'android_xhdpi': 'resources/icons/icon-96x96.png'
});

App.launchScreens({
  // iOS
  'iphone': 'resources/splash/splash-320x480.png',
  'iphone_2x': 'resources/splash/splash-320x480@2x.png',
  'iphone5': 'resources/splash/splash-320x568@2x.png',
  'ipad_portrait': 'resources/splash/splash-768x1024.png',
  'ipad_portrait_2x': 'resources/splash/splash-768x1024@2x.png',
  'ipad_landscape': 'resources/splash/splash-1024x768.png',
  'ipad_landscape_2x': 'resources/splash/splash-1024x768@2x.png',

  // Android
  'android_ldpi_portrait': 'resources/splash/splash-200x320.png',
  'android_ldpi_landscape': 'resources/splash/splash-320x200.png',
  'android_mdpi_portrait': 'resources/splash/splash-320x480.png',
  'android_mdpi_landscape': 'resources/splash/splash-480x320.png',
  'android_hdpi_portrait': 'resources/splash/splash-480x800.png',
  'android_hdpi_landscape': 'resources/splash/splash-800x480.png',
  'android_xhdpi_portrait': 'resources/splash/splash-720x1280.png',
  'android_xhdpi_landscape': 'resources/splash/splash-1280x720.png'
});
```

Pipeline d'Architecture Meteor Cordova

Maintenant, il est temps de passer par la documentation de [Meteor Cordova Phonegap Integration](#)

Depuis que cette documentation a été écrite, XCode et Yosemite ont été publiés, ce qui a causé quelques problèmes lors de l'installation. Voici les étapes à suivre pour compiler Meteor sur un appareil iOS.

- Passez à Yosemite.
- Supprimer XCode (faites glisser depuis le dossier Applications vers la corbeille)
- Installez XCode 6.1 à partir de l'App Store.
- Acceptez les différents termes et conditions.

```
# 5. clone and rebuild the ios-sim locally
# (this step will not be needed in future releases)
git clone https://github.com/phonegap/ios-sim.git
cd ios-sim
rake build

# 6. make sure we can update the .meteor/packages locations
# (this step will not be needed in future releases)
sudo chmod -R 777 ~/.meteor/packages

# 7. copy the new build into Meteor locations
# (this step will not be needed in future releases)
for i in `find ~/.meteor/packages/meteor-tool/ -name ios-sim -type f`; do
  cp -R ./build/Release/ios-sim "$i"
done

# 8. install the ios platform to your app
cd myapp
meteor list-platforms
meteor add-platform ios
meteor list-platforms

# 9. and that there aren't dead processes
ps -ax
kill -9 <pid>
# /Users/abigailwatson/.meteor/packages/meteor-
tool/.1.0.35.wq14jh++os.osx.x86_64+web.browser+web.cordova/meteor-tool-
os.osx.x86_64/dev_bundle/mongodb/bin/mongod
# tail -f /Users/abigailwatson/Code/Medstar/dart/webapp/.meteor/local/cordova-
build/platforms/ios/cordova/console.log

# 10. make sure there are correct permissions on the application (important!)
sudo chmod -R 777 .meteor/local/

# 11. run app
meteor run ios

# 12. if that doesn't work, clear the directory
sudo rm -rf .meteor/local

# 13a. run meteor again to create the default browser build
meteor
```

```
# 13b. run it a second time so bootstrap and other packages get downloaded into the browser
build
ctrl-x
meteor

# 14. then run the ios version
ctrl-x
meteor run ios
```

XCode doit être lancé pendant le processus. Sélectionnez votre simulateur et appuyez sur le bouton 'Play'.

Développement IOS

- Enregistrez votre compte de développeur Apple
- Enregistrer un identifiant d'application pour votre application
- Enregistrez l'UUID de vos appareils de test
- Générer un profil d'approvisionnement iOS App Development
 - Générer un CertificateSigningRequest à partir de KeychainAccess
 - Soumettez CertificateSigningRequest à <https://developer.apple.com/account/ios/profile/profileCreate.action>
 - Téléchargez et double-cliquez sur le certificat à importer dans Keychain
- Accédez à XCode> Préférences> Comptes et enregistrez votre compte de développeur Apple.

Test de périphérique IOS

- Assurez-vous que votre poste de travail de développement et votre iPhone sont connectés au même réseau WiFi. Le partage de connexion, les zones sensibles et d'autres réseaux ad hoc ne fonctionneront pas.
- Exécuter `sudo meteor run ios-device`
- Déployez sur votre appareil!

Configurez votre projet Cordova (config.xml)

Meteor lit un fichier `mobile-config.js` à la racine de votre répertoire d'application pendant la génération et utilise les paramètres spécifiés pour générer le `config.xml` de Cordova.

```
Project_folder
├── /.meteor
└── mobile-config.js
```

La plupart des configurations peuvent être réalisées avec `mobile-config.js` (métadonnées d'application, préférences, icônes et écrans de lancement, ainsi que les paramètres d'installation des plug-ins Cordova).

```
App.info({
  id: 'com.example.matt.uber',
  name: 'über',
```

```

description: 'Get über power in one button click',
author: 'Matt Development Group',
email: 'contact@example.com',
website: 'http://example.com'
});

// Set up resources such as icons and launch screens.
App.icons({
  'iphone': 'icons/icon-60.png',
  'iphone_2x': 'icons/icon-60@2x.png',
  // ... more screen sizes and platforms ...
});

App.launchScreens({
  'iphone': 'splash/Default~iphone.png',
  'iphone_2x': 'splash/Default@2x~iphone.png',
  // ... more screen sizes and platforms ...
});

// Set PhoneGap/Cordova preferences
App.setPreference('BackgroundColor', '0xff0000ff');
App.setPreference('HideKeyboardFormAccessoryBar', true);
App.setPreference('Orientation', 'default');
App.setPreference('Orientation', 'all', 'ios');

// Pass preferences for a particular PhoneGap/Cordova plugin
App.configurePlugin('com.phonegap.plugins.facebookconnect', {
  APP_ID: '1234567890',
  API_KEY: 'supersecretapikey'
});

```

Ne modifiez pas manuellement le fichier `./.meteor/local/cordova-build/config.xml` , car il sera régénéré lors de chaque `meteor build meteor run ios/android` **ou** `meteor build` , vous perdrez ainsi toutes vos modifications.

Page de référence: [Meteor Guide](#)> [Build](#)> [Mobile](#)> [Configurer votre application](#)

Détecter l'événement déviant

Bien entendu, le meilleur moyen de détecter le mobile est que le matériel vous avertisse directement. Cordova PhoneGap expose un événement «deviceready», auquel vous pouvez ajouter un écouteur d'événement.

```

document.addEventListener('deviceready', function(){
  Session.set('deviceready', true);
}, false);

```

Lire Application mobile en ligne: <https://riptutorial.com/fr/meteor/topic/3705/application-mobile>

Chapitre 5: Applications hors ligne

Remarques

Recherche supplémentaire sur Appcache

<http://www.html5rocks.com/en/tutorials/indexeddb/todo/>

<http://grinninggecko.com/2011/04/22/increasing-chromes-offline-application-cache-storage-limit/>

<http://www.html5rocks.com/en/tutorials/offline/quota-research/>

https://developers.google.com/chrome/apps/docs/developers_guide?csw=1#installing

https://developers.google.com/chrome/apps/docs/developers_guide?csw=1#manifest

Exemples

Meteor.status ()

La première chose à faire lorsque vous déconnectez votre application Meteor est de créer une indication visuelle de la connexion ou non de l'application client locale. Il y a plusieurs façons de faire cela, mais le plus simple est de faire quelque chose comme ceci:

```
Template.registerHelper('getOnlineStatus', function(){
  return Meteor.status().status;
});
```

```
Template.registerHelper('getOnlineColor', function(){
  if(Meteor.status().status === "connected"){
    return "green";
  }else{
    return "orange";
  }
});
```

```
<div id="onlineStatus" class="{{getOnlineColor}}">
  {{getOnlineStatus}}
</div>
```

```
.green{
  color: green;
}
.orange{
  color: orange;
}
```

Activer Appcache

L'une des étapes les plus faciles consiste à ajouter l'appcache. Appcache permettra au contenu de votre application de se charger même s'il n'y a pas d'accès à Internet. Vous ne pourrez pas obtenir de données de vos serveurs mongo, mais le contenu statique et les ressources seront

disponibles hors connexion.

```
meteor add appcache
```

Activer GroundDB

Enfin, nous voulons que certaines de nos données dynamiques soient stockées hors ligne.

```
meteor add ground:db
```

```
Lists = new Meteor.Collection("lists");  
GroundDB(Lists);  
  
Todos = new Meteor.Collection("todos")  
GroundDB(Todos);
```

Choses à faire attention

- L'appcache entraînera une certaine confusion dans votre flux de travail de développement, car il masque les fonctionnalités de mise à jour automatique de Meteor. Lorsque vous désactivez le composant serveur de votre application, la partie client de votre navigateur continue de fonctionner. C'est une bonne chose! Mais, vous ne recevez pas le retour immédiat que votre application a été désactivée ou qu'il y a eu des mises à jour.
- Essayez d'utiliser le mode navigation privée de Chrome lors du développement de votre application, car elle n'utilise pas l'appcache.
- GroundDB ne fonctionne pas particulièrement bien avec IronRouter.

Lire Applications hors ligne en ligne: <https://riptutorial.com/fr/meteor/topic/3375/applications-hors-ligne>

Chapitre 6: Blaze Templating

Introduction

Blaze est une bibliothèque puissante pour créer des interfaces utilisateur en écrivant des modèles HTML réactifs et dynamiques. La création de modèles Blaze permet d'utiliser les boucles et la logique conditionnelle directement dans le balisage HTML. Cette section explique et démontre l'utilisation correcte de la modélisation dans Meteor.js avec Blaze.

Exemples

Remplir un modèle à partir d'un appel de méthode

```
<template name="myTemplate">
  {{#each results}}
    <div><span>{{name}}</span><span>{{age}}</span></div>
  {{/each}}
</template>
```

```
Template.myTemplate.onCreated(function() {
  this.results = new ReactiveVar();
  Meteor.call('myMethod', (error, result) => {
    if (error) {
      // do something with the error
    } else {
      // results is an array of {name, age} objects
      this.results.set(result);
    }
  });
});

Template.myTemplate.helpers({
  results() {
    return Template.instance().results.get();
  }
});
```

Contexte de données d'un modèle

Chaque fois qu'un modèle est appelé, le contexte de données par défaut du modèle est implicitement obtenu à partir de l'appelant, par exemple, le `childTemplate` gagne le contexte de données du `parentTemplate`, c'est-à-dire le modèle de l'appelant.

```
<template name="parentTemplate">
  {{#with someHelperGettingDataForParentTemplate}}
  <h1>My name is {{firstname}} {{lastname}}</h1>
  //some stuffs here
  {{> childTemplate}}
  {{/with}}
</template>
```

Dans la situation ci-dessus, quelles que soient les données extraites automatiquement par le composant parent par `childTemplate`. For exemple, `{{firstname}}` et `{{lastname}}` sont accessibles depuis `childTemplate`, comme indiqué ci-dessous.

```
<template name="childTemplate">
  <h2>My name is also {{firstname}} {{lastname}}</h2>
</template>
```

Nous pouvons même définir explicitement le contexte de données du `childTemplate` en transmettant des arguments au modèle, comme dans l'exemple ci-dessous.

```
<template name="parentTemplate">
  {{#with someHelperGettingDataForParentTemplate}}
  <h1>My name is {{firstname}} {{lastname}}</h1>
  //some stuffs here
  {{> childTemplate childData=someHeplerReturningDataForChild}}
  {{/with}}
</template>
```

En supposant que l'assistant **`someHelperReturningDataForChild`** renvoie un objet tel que `{profession: "Meteor Developer", hobby: "stackoverflow"}`, cet objet particulier sera le contexte de données explicite pour le `childTemplate`. Maintenant, dans le modèle enfant, nous pouvons faire quelque chose comme

```
<template name="childTemplate">
  <h2>My profession is {{profession}}</h2>
  <h3>My hobby is {{hobby}}</h3>
</template>
```

Aide du modèle

[Les aides de modèle](#) sont une partie essentielle de Blaze et fournissent à la fois une logique métier et une réactivité à un modèle. Il est important de se rappeler que les assistants Template sont en réalité [des calculs réactifs](#) qui sont réexécutés chaque fois que leurs dépendances changent. Selon vos besoins, les assistants de modèle peuvent être définis globalement ou définis sur un modèle spécifique. Des exemples de chaque approche de définition d'assistant de modèle sont fournis ci-dessous.

1. Exemple d'assistance de modèle limitée à un modèle unique.

Définissez d'abord votre modèle:

```
<template name="welcomeMessage">
  <h1>Welcome back {{fullName}}</h1>
</template>
```

Ensuite, définissez l'aide du modèle. Cela suppose que le contexte de données du modèle contient une propriété `firstName` et `lastName`.

```
Template.welcomeMessage.helpers({
```

```
fullName: function() {
  const instance = Template.instance();
  return instance.data.firstName + ' ' + instance.data.lastName
},
});
```

2. Exemple d'une aide globale de modèle (cette aide peut être utilisée depuis n'importe quel modèle)

Commencez par enregistrer l'aide:

```
Template.registerHelper('equals', function(item1, item2) {
  if (!item1 || !item2) {
    return false;
  }

  return item1 === item2;
});
```

Avec l'assistant d' `equals` défini, je peux maintenant l'utiliser dans n'importe quel modèle:

```
<template name="registration">
  {{#if equals currentUser.registrationStatus 'Pending'}}
  <p>Don't forget to complete your registration!<p>
  {{/if}}
</template>
```

Lire **Blaze Templating** en ligne: <https://riptutorial.com/fr/meteor/topic/2434/blaze-templating>

Chapitre 7: Collections Mongo

Remarques

Une manière utile de penser aux collections Mongo est en termes de qui, quoi, quand, où, pourquoi et comment. Mongo dispose des optimisations suivantes pour différents types de données:

Où - GeoJSON

When - Horodatage ObjectID

Who - Les chaînes de compte Meteor

How - JSON pour les arbres de décision

Ce qui laisse le document par défaut dans Mongo représentant grosso modo un «quoi».

Exemples

Création d'enregistrements dans une base de données existante

Vous pouvez utiliser le format Mongo normal par défaut en définissant vos collections avec le champ `idGeneration`.

```
MyCollection = new Meteor.Collection('mycollection', {idGeneration : 'MONGO'});
```

Insérer des données dans un document

De nombreux débutants à Mongo ont des difficultés avec les bases, telles que l'insertion d'un tableau, d'une date, d'une valeur booléenne, d'une variable de session, etc. dans un enregistrement de document. Cet exemple fournit des indications sur les entrées de données de base.

```
Todos.insert({
  text: "foo", // String
  listId: Session.get('list_id'), // String
  value: parseInt(2), // Number
  done: false, // Boolean
  createdAt: new Date(), // Dimestamp
  timestamp: (new Date()).getTime(), // Time
  tags: [] // Array
});
```

Obtenir le `_id` du document le plus récemment créé

Vous pouvez l'obtenir soit de manière synchrone:

```
var docId = Todos.insert({text: 'foo'});
```

```
console.log(docId);
```

Ou de manière asynchrone:

```
Todos.insert({text: 'foo'}, function(error, docId){  
  console.log(docId);  
});
```

Données de séries chronologiques

Utiliser MongoDB pour les séries chronologiques est un document très utile et un cas d'utilisation établi, avec des livres blancs et des présentations officiels. Lisez et regardez la documentation officielle de MongoDB avant d'essayer d'inventer vos propres schémas pour les données de séries chronologiques.

[MongoDB pour les données de séries chronologiques](#)

En général, vous voudrez créer des "buckets" pour les données de vos séries de temps:

```
DailyStats.insert({  
  "date" : moment().format("MM-DD-YYYY"),  
  "dateIncrement" : moment().format("YYYYMMDD"),  
  "dailyTotal" : 0,  
  'bucketA': 0,  
  'bucketB': 0,  
  'bucketC': 0  
});
```

Et ensuite, incrémentez ces compartiments au fur et à mesure que les données alimentent votre application. Cet incrément peut être placé dans une méthode Meteor, un observateur de collection, un point de terminaison API REST et divers autres endroits.

```
DailyStats.update({_id: doc._id}, {$inc: {bucketA: 1}});
```

Pour un exemple plus complet de Meteor, consultez les exemples de la piste Clinical Meteor:

[Pipeline d'analyse en temps réel](#)

[Clinical Meteor - Graphes - Dailystats](#)

Filtrage avec des expressions régulières

Modèle simple pour filtrer les abonnements sur le serveur, en utilisant des expressions rationnelles, des variables de session réactives et des autoruns de deps.

```
// create our collection  
WordList = new Meteor.Collection("wordlist");  
  
// and a default session variable to hold the value we're searching for  
Session.setDefault('dictionary_search', '');  
  
Meteor.isClient(function(){
```

```

// we create a reactive context that will rerun items when a Session variable gets updated

Deps.autorun(function(){
  // and create a subscription that will get re-subscribe to when Session variable gets
  updated
  Meteor.subscribe('wordlist', Session.get('dictionary_search'));
});

Template.dictionaryIndexTemplate.events({
  'keyup #dictionarySearchInput': function(evt,tmpl){
    // we set the Session variable with the value of our input when it changes
    Session.set('dictionary_search', $('#dictionarySearchInput').val());
  },
  'click #dictionarySearchInput':function(){
    // and clear the session variable when we enter the input
    Session.set('dictionary_search', '');
  },
});
});
Meteor.isServer(function(){
  Meteor.publish('wordlist', function (word_search) {
    // this query gets rerun whenever the client subscribes to this publication
    return WordList.find({
      // and here we do our regex search
      Word: { $regex: word_search, $options: 'i' }
    },{limit: 100});
  });
});
});

```

Et le HTML qui est utilisé sur le client:

```
<input id="dictionarySearchInput" type="text" placeholder="Filter..." value="hello"></input>
```

Ce modèle lui-même est assez simple, mais les expressions rationnelles peuvent ne pas l'être. Si vous n'êtes pas familier avec les regex, voici quelques tutoriels et liens utiles:

[Tutoriel d'expression régulière](#)

[Feuille de triche d'expression régulière](#)

[Expressions régulières en Javascript](#)

Collections géospatiales - Apprendre plus

Les collections géospatiales impliquent généralement de stocker GeoJSON dans la base de données Mongo, de transmettre ces données au client, d'accéder à `window.navigator.geolocation` du navigateur, de charger une API Map, de convertir GeoJSON en LatLngs et de tracer sur la carte. De préférence tous en temps réel. Voici une liste de ressources pour vous aider à démarrer:

- [mongodb stocke de manière optimale ses données dans geoJSON](#)
- [geojson.org](#)
- [window.navigator.geolocation](#)
- [Géolocalisation HTML](#)
- [Sélecteur d'API Maps](#)
- [google.maps.LatLng](#)

- [Google map.data.loadGeoJson](#)
- [fond météo-cordoue-géolocalisation](#)
- [phonegap-googlemaps-plugin](#)
- [LatLng](#)
- [maps.documentation](#)
- [google.maps.LatLng](#)
- [Index 2dsphere](#)
- [créer un index 2dsphere](#)
- [interroger un index 2dsphere](#)
- [index géospatiaux et requêtes](#)

Auditer les requêtes de collection

L'exemple suivant enregistre toutes vos requêtes de collecte sur la console du serveur en temps réel.

```
Meteor.startup(
  function () {
    var wrappedFind = Meteor.Collection.prototype.find;

    // console.log('[startup] wrapping Collection.find')

    Meteor.Collection.prototype.find = function () {
      // console.log(this._name + '.find', JSON.stringify(arguments))
      return wrappedFind.apply(this, arguments);
    }
  },

  function () {
    var wrappedUpdate = Meteor.Collection.prototype.update;

    // console.log('[startup] wrapping Collection.find')

    Meteor.Collection.prototype.update = function () {
      console.log(this._name + '.update', JSON.stringify(arguments))
      return wrappedUpdate.apply(this, arguments);
    }
  }
);
```

Fonctions des observateurs et des travailleurs

Si la boucle d'événement de noeud agit comme une chaîne de bicyclette, l'observateur de collection côté serveur est comme un dérailleur. C'est un mécanisme d'engrenage qui va se retrouver sur la collecte de données à mesure que les données entrent en jeu. Il peut être très performant, car tous les vélos de course ont des dérailleurs. Mais c'est aussi une source de rupture de tout le système. C'est une fonction réactive à haute vitesse qui peut exploser sur vous. Être averti.

```
Meteor.startup(function() {
  console.log('starting worker....');
```

```
var dataCursor = Posts.find({viewsCount: {$exists: true}}, {limit:20});

var handle = dataCursor.observeChanges({
  added: function (id, record) {
    if(record.viewsCount > 10){
      // run some statistics
      calculateStatistics();

      // or update a value
      Posts.update({_id: id}, {$set:{
        popular: true
      }});
    }
  },
  removed: function () {
    console.log("Lost one.");
  }
});
```

Notez que la limite de 20 est la taille du dérailleur combien de dents il a; ou, plus précisément, combien d'éléments se trouvent dans le curseur lorsqu'il parcourt la collection. Faites attention à l'utilisation du mot clé 'var' dans ce type de fonction. Écrivez le moins d'objets possible en mémoire et concentrez-vous sur la réutilisation des objets dans la méthode ajoutée. Lorsque le journal des opérations est activé, et que cette opération est en cours, il est un candidat idéal pour exposer les fuites de mémoire nuisibles si elle écrit des objets sur le tas de mémoire plus rapidement que le ramasse-miettes Node ne peut les nettoyer.

La solution ci-dessus ne sera pas bien mise à l'échelle car chaque instance Meteor essaiera de mettre à jour le même enregistrement. Donc, une sorte de détection de l'environnement est nécessaire pour que cela évolue horizontalement.

Voir le paquetage `percolatestudios:synced-cron` pour un excellent exemple de synchronisation des travailleurs du service sur plusieurs machines d'un cluster.

[météore-sync-cron](#)

Lire Collections Mongo en ligne: <https://riptutorial.com/fr/meteor/topic/5120/collections-mongo>

Chapitre 8: Comptes utilisateurs Meteor

Exemples

Paquet de comptes Meteor

Vous avez quelques options pour vous connecter à Meteor. La méthode la plus courante consiste à utiliser des `accounts` pour Meteor.

Comptes-mot de passe

Si vous souhaitez que les utilisateurs puissent créer et enregistrer sur votre site, vous pouvez utiliser `accounts-password`.

Installez le paquet en utilisant `meteor add accounts-password`.

Pour créer un utilisateur, vous devez utiliser `Accounts.createUser(options, [callback])`

`options` doit être un objet avec les propriétés suivantes:

- `username` : le nom d'utilisateur de l'utilisateur sous forme de chaîne ..
- `email` : email utilisateur sous forme de chaîne.
- `password` : password utilisateur (non chiffré) en tant que chaîne.
- `profile` : données supplémentaires facultatives de l'utilisateur en tant qu'objet. Cela peut être par exemple le prénom et le nom de l'utilisateur. `profile` est facultatif, cependant.

Le rappel renvoie 1 variable s'il y a une erreur, qui est un objet `Meteor.Error`.

Vous devez uniquement utiliser le `username` ou le `email` pour pouvoir créer un utilisateur avec un nom d'utilisateur, mais pas d'email, et vice versa. Vous pouvez également utiliser les deux.

Il renvoie l'ID utilisateur nouvellement créé si tout s'est bien passé.

Ainsi, vous pouvez par exemple utiliser ceci:

```
// server side
var id = Accounts.createUser({
  username: "JohnDoe",
  email: "JohnDoe@gmail.com",
  password: "TheRealJohn123",
  profile: {
    firstName: "John",
    lastName: "Doe"
  }
}, function(err) {
  console.log(err.reason);
});
```

Il vous connectera automatiquement si l'utilisateur a été créé avec succès.

C'est la partie créatrice. Pour vous connecter, vous devez utiliser

`Meteor.loginWithPassword(identifiant, password, [callback])` du côté client.

`identifiant` est le `username`, l' `email` ou `userId` tant que chaîne de votre utilisateur. `password` de `password` est le (non crypté) le `password` de `password` de l'utilisateur.

Le rappel renvoie une variable s'il y a une erreur, qui est un objet `Meteor.Error`.

Exemple:

```
// client side
Meteor.loginWithPassword("JohnDoe", "TheRealJohn123", function(err) {
  console.log(err.reason);
});
```

Et c'est tout pour la création de base de comptes et la connexion.

Accès aux données utilisateur

Vous pouvez vérifier sur le côté client si l'utilisateur est connecté en appelant `Meteor.userId()` qui renverra son `userId` s'il est connecté et `undefined` s'il n'est pas connecté.

Vous pouvez obtenir des informations sur `Meteor.user()`. Il ne sera pas défini si l'utilisateur n'est pas connecté et si certaines données utilisateur le sont. Il ne vous fournira aucun mot de passe par défaut, il affichera par défaut l'identifiant de l'utilisateur, le nom d'utilisateur et l'objet du profil.

Si vous souhaitez vérifier si un utilisateur est connecté sur une page, vous pouvez également utiliser l'assistant `currentUser`. Il retournera le contenu de `Meteor.user()`. Exemple:

```
{{#if currentUser}}
  <h1>Hello there, {{currentUser.username}}!</h1>
{{else}}
  <h1>Please log in.</h1>
{{/if}}
```

Autres fonctions de comptes

Il existe d'autres fonctions qui fonctionnent pour tous les packages de comptes.

Vous pouvez vous déconnecter en utilisant `Meteor.logout()`

Ne pas utiliser le champ de profil par défaut

Il existe un champ existant tentant appelé `profile` qui est ajouté par défaut lorsqu'un nouvel utilisateur s'inscrit. Ce champ était historiquement destiné à être utilisé comme bloc-notes pour les

données spécifiques à l'utilisateur - peut-être leur avatar d'image, leur nom, texte d'introduction, etc. De ce fait, **le champ de profile de chaque utilisateur** Il est également publié automatiquement sur le client pour cet utilisateur particulier.

Il s'avère que le fait d'avoir un champ accessible en écriture par défaut sans que cela soit trop évident pourrait ne pas être la meilleure idée. Il y a beaucoup d'histoires de nouveaux développeurs Meteor stockant des champs tels que `isAdmin` sur le `profile` ... puis un utilisateur malveillant peut facilement définir cela comme vrai à tout moment, se faisant un administrateur. Même si cela ne vous préoccupe pas, il est déconseillé de laisser des utilisateurs malveillants stocker des quantités arbitraires de données dans votre base de données.

Plutôt que de traiter des spécificités de ce domaine, il peut être utile d'ignorer entièrement son existence. Vous pouvez le faire en toute sécurité tant que vous refusez toutes les écritures du client:

```
// Deny all client-side updates to user documents
Meteor.users.deny({
  update() { return true; }
});
```

Même en ignorant les implications de sécurité du profil, il n'est pas judicieux de placer toutes les données personnalisées de votre application sur un seul champ. Le protocole de transfert de données de Meteor ne permet pas de différencier profondément les champs. Il est donc recommandé d'aplatir vos objets dans de nombreux champs de niveau supérieur du document.

Lire Comptes utilisateurs Meteor en ligne: <https://riptutorial.com/fr/meteor/topic/6219/comptes-utilisateurs-meteor>

Chapitre 9: Configuration de base de codeship pour le test automatisé

Exemples

Configuration du code d'accès

- Allez sur [Codeship.com](https://codeship.com) et créez un compte (ou connectez-vous)
- Créer un nouveau projet
- Importez votre projet via Github ou Bitbucket
- Sur l'écran "Configure Your Tests", utilisez ces commandes:

- Sélectionnez "Je veux créer mes propres commandes personnalisées" dans le menu déroulant "Sélectionnez votre technologie pour pré-remplir les commandes de base".

- Entrez les commandes suivantes:

```
curl -o meteor_install_script.sh https://install.meteor.com/
chmod +x meteor_install_script.sh
sed -i "s/type sudo >\/dev\/null 2>&1\/ false /g" meteor_install_script.sh
./meteor_install_script.sh
export PATH=$PATH:~/.meteor/
meteor --version
meteor npm install
```

- Laissez les commandes de test comme ceci:

```
npm test
```

- Poussez un nouvel engagement vers Github / Bitbucket
- C'est tout

Préparer le projet

- [Ecrivez quelques tests](#)
- Installation de l' [expédition: mocha-phantomjs](#) :

```
meteor add dispatch:mocha-phantomjs
```

- Ajoutez une commande de test à votre package.json.

```
{
  "name": "awesome meteor package",
  "scripts": {
    "test": "meteor test --driver-package dispatch:mocha-phantomjs --once"
  }
}
```

- Assurez-vous que vous pouvez exécuter `npm test` dans la racine de votre projet.

Lire Configuration de base de codeship pour le test automatisé en ligne:

<https://riptutorial.com/fr/meteor/topic/6741/configuration-de-base-de-codeship-pour-le-test-automatise>

Chapitre 10: Déploiement avec Upstart

Exemples

Service de démarrage

Ce guide de déploiement suppose que vous utilisez un serveur Ubuntu et que vous utilisez soit un hébergement autonome, soit un fournisseur IaaS (Infrastructure as a Service), tel que Amazon Web Services ou Rackspace. Votre serveur Ubuntu doit exécuter un démon pour lancer d'autres applications, pour lesquelles nous recommandons le service Upstart. Vous pouvez trouver plus sur Upstart avec les liens suivants:

[Upstart - Pour commencer](#)
[Premiers pas avec les scripts de démarrage sur Ubuntu](#)
[UbuntuBootupHowTo](#)
[Intro, livre de recettes et meilleures pratiques](#)
[Exécuter NodeJS en tant que service sur Ubuntu Karmic](#)

Copier des fichiers sur votre serveur puis générer

Une approche privilégiée du déploiement sur un serveur consiste à utiliser Git ou GitHub. Cela implique essentiellement de se connecter à votre serveur, en allant dans le répertoire à partir duquel vous souhaitez exécuter votre application, puis en clonant vos fichiers directement à partir de GitHub. Vous créez ensuite votre application sur le serveur. Cette approche garantit que les fichiers spécifiques à la plate-forme sont correctement construits, mais nécessite que Meteor soit installé sur le serveur (plus de 500 Mo) et que des versions légèrement différentes risquent de se terminer en production si vos serveurs sont légèrement différents.

```
cd /var/www
sudo git clone http://github.com/myaccount/myapp.git
cd /var/www/myapp
meteor build --directory ../myapp-production
sudo service myapp restart
```

Bundle puis copier sur le serveur

Vous pouvez également créer votre application, puis la déployer.

```
cd myapp
meteor build --directory ../output
cd ..
scp output -r username@destination_host:/var/www/myapp-production
```

Écrire votre script de démarrage

Vous aurez besoin d'un script de démarrage dans votre `/etc/init/ directory`. Nommez-le avec le

nom de votre application, se terminant par `.conf` , tel que `/etc/init/myapp.conf` . Le script de démarrage de base ressemble à ceci:

```
## /etc/init/myapp.conf
description "myapp.mydomain.com"
author      "somebody@gmail.com"

# Automatically Run on Startup
start on started mountall
stop on shutdown

# Automatically Respawn:
respawn
respawn limit 99 5

script
  export HOME="/root"
  export MONGO_URL='mongodb://myapp.compose.io:27017/meteor'
  export ROOT_URL='http://myapp.mydomain.com'
  export PORT='80'

  exec /usr/local/bin/node /var/www/myapp/main.js >> /var/log/myapp.log 2>&1
end script
```

Script de démarrage pour les jeux de réplicas

Si vous exécutez un jeu de réplicas ou si vous avez besoin de partager votre base de données, vous aurez besoin d'un script qui ressemble à ceci:

```
# /etc/init/myapp.conf
description "myapp.mydomain.com"
author      "somebody@gmail.com"

# used to be: start on startup
# until we found some mounts weren't ready yet while booting:
start on started mountall
stop on shutdown

# Automatically Respawn:
respawn
respawn limit 99 5

script
  # upstart likes the $HOME variable to be specified
  export HOME="/root"

  # our example assumes you're using a replica set and/or oplog integration
  export MONGO_URL='mongodb://mongo-a,mongo-b,mongo-c:27017/?replicaSet=meteor'

  # root_url and port are the other two important environment variables to set
  export ROOT_URL='http://myapp.mydomain.com'
  export PORT='80'

  exec /usr/local/bin/node /var/www/production/main.js >> /var/log/node.log 2>&1
end script
```

Exécuter votre script de démarrage

Enfin, vous devez démarrer le démon Upstart et initialiser votre application en tant que service.

```
sudo service myapp start
```

Configuration d'un serveur pour héberger plusieurs applications Meteor

<https://www.phusionpassenger.com/>

<https://github.com/phusion/passenger>

<https://github.com/phusion/passenger/wiki/Phusion-Passenger:-Meteor-tutorial#wiki-installing>

Lire Déploiement avec Upstart en ligne: <https://riptutorial.com/fr/meteor/topic/3377/deploiement-avec-upstart>

Chapitre 11: Déploiement continu sur Galaxy de Codeship

Remarques

Ce sujet est fortement inspiré par la [migration des applications Meteor de Nate Strausers de Modulus vers Galaxy avec déploiement continu à partir de Codeship](#) .

Exemples

Installer

- Créez un `deployment_token.json` :

```
METEOR_SESSION_FILE=deployment_token.json meteor login
```

- Créez les variables d'environnement suivantes sur Codeship: (https://codeship.com/projects/PROJECT_NUMBER/configure_environment)
 - METEOR_TARGET: `votre.domaine.com`
 - METEOR_TOKEN: Copiez / Collez le contenu de `deployment_token.json`. Quelque chose comme: `{"sessions": {"www.meteor.com": {"session": "12345 ...`
 - METEOR_SETTING: Copiez / Collez le contenu de vos paramètres.json. Quelque chose comme: `{"private": {...`
- Créez un nouveau pipeline de déploiement ici https://codeship.com/projects/YOUR_PROJECT_NUMBER/deployment_branches/new
 - Nous déployons uniquement la branche principale. Donc set: Branch est exactement: `master`.
- Ajoutez un "script personnalisé" comme déploiement avec le contenu suivant:

```
echo $METEOR_TOKEN > deployment_token.json
echo $METEOR_SETTINGS > deployment_settings.json
meteor npm prune --production
DEPLOY_HOSTNAME=galaxy.meteor.com METEOR_SESSION_FILE=deployment_token.json meteor deploy
$METEOR_TARGET --settings deployment_settings.json
```

Lire [Déploiement continu sur Galaxy de Codeship en ligne](#):

<https://riptutorial.com/fr/meteor/topic/6743/dploiement-continu-sur-galaxy-de-codeship>

Chapitre 12: Détection de l'environnement

Exemples

Configurations d'environnement avancées

Pour les applications plus complexes, vous voudrez créer un objet `` settings.json `` en utilisant plusieurs variables d'environnement.

```
if(Meteor.isServer){
  Meteor.startup(function()){
    // this needs to be run on the server
    var environment, settings;

    environment = process.env.METEOR_ENV || "development";

    settings = {
      development: {
        public: {
          package: {
            name: "jquery-datatables",
            description: "Sort, page, and filter millions of records. Reactively.",
            owner: "LumaPictures",
            repo: "meteor-jquery-datatables"
          }
        },
        private: {}
      },
      staging: {
        public: {},
        private: {}
      },
      production: {
        public: {},
        private: {}
      }
    };

    if (!process.env.METEOR_SETTINGS) {
      console.log("No METEOR_SETTINGS passed in, using locally defined settings.");
      if (environment === "production") {
        Meteor.settings = settings.production;
      } else if (environment === "staging") {
        Meteor.settings = settings.staging;
      } else {
        Meteor.settings = settings.development;
      }
      console.log("Using [ " + environment + " ] Meteor.settings");
    }
  });
}
```

Spécification des paramètres de l'application avec METEOR_SETTINGS

La variable d'environnement `METEOR_SETTINGS` peut accepter des objets JSON et exposera cet objet dans l'objet `Meteor.settings`. Tout d'abord, ajoutez un `settings.json` à votre racine d'application avec des informations de configuration.

```
{
  "public":{
    "ga":{
      "account":"UA-XXXXXXX-1"
    }
  }
}
```

Ensuite, vous devrez lancer votre application en utilisant votre fichier de paramètres.

```
# run your app in local development mode with a settings file
meteor --settings settings.json

# or bundle and prepare it as if you're running in production
# and specify a settings file
meteor bundle --directory /path/to/output
cd /path/to/output
MONGO_URL="mongodb://127.0.0.1:27017" PORT=3000 METEOR_SETTINGS=$(cat /path/to/settings.json)
node main.js
```

Ces paramètres peuvent alors être accessibles à partir de `Meteor.settings` et utilisés dans votre application.

```
Meteor.startup(function(){
  if(Meteor.isClient){
    console.log('Google Analytics Account', Meteor.settings.public.ga.account);
  }
});
```

Détection d'environnement sur le serveur

Les variables d'environnement sont également disponibles pour le serveur via l'objet `process.env`.

```
if (Meteor.isServer) {
  Meteor.startup(function () {
    // detect environment by getting the root url of the application
    console.log(JSON.stringify(process.env.ROOT_URL));

    // or by getting the port
    console.log(JSON.stringify(process.env.PORT));

    // alternatively, we can inspect the entire process environment
    console.log(JSON.stringify(process.env));
  });
}
```

Détection de l'environnement client à l'aide des méthodes Meteor

Pour détecter l'environnement sur le serveur, nous devons créer une méthode d'assistance sur le

serveur, car le serveur déterminera dans quel environnement il se trouve, puis appellera la méthode d'assistance à partir du client. Fondamentalement, nous transmettons simplement les informations sur l'environnement du serveur au client.

```
//-----  
-----  
// server/server.js  
// we set up a getEnvironment method  
  
Meteor.methods({  
  getEnvironment: function(){  
    if(process.env.ROOT_URL == "http://localhost:3000"){  
      return "development";  
    }else{  
      return "staging";  
    }  
  }  
});  
  
//-----  
-----  
// client/main.js  
// and then call it from the client  
  
Meteor.call("getEnvironment", function (result) {  
  console.log("Your application is running in the " + result + "environment.");  
});
```

Détection de l'environnement client à l'aide de NODE_ENV

À partir de Meteor 1.3, Meteor expose désormais la variable `NODE_ENV` sur le client par défaut.

```
if (Meteor.isClient) {  
  Meteor.startup(function () {  
    if(process.env.NODE_ENV === "testing"){  
      console.log("In testing...");  
    }  
    if(process.env.NODE_ENV === "production"){  
      console.log("In production...");  
    }  
  });  
}
```

Lire Détection de l'environnement en ligne: <https://riptutorial.com/fr/meteor/topic/4198/detection-de-l-environnement>

Chapitre 13: Electrify - Compiler Meteor comme une application localement installable

Exemples

Installer Electrify pour une application Meteor

Les ports électroniques utilisent les applications Web HTML pour les applications natives pour une gamme de périphériques, y compris la création d'applications de bureau natives. C'est aussi très facile de commencer!

Pour commencer, il faut avoir `electron`, `nodejs`, `npm`, `git` et `meteor` installés. La connaissance de ces outils est essentielle pour travailler avec Meteor, alors assurez-vous de connaître ces choses en premier.

Électron

```
npm install -g electrify
```

- `electron` est ce que nous utilisons! Lire plus [ici](#) .
- `electrify` est un outil pour emballer les applications Meteor. Mode lecture [ici](#) .

Autres exigences pour l'installation et l'utilisation d'Electrify avec Meteor

Météore

```
curl https://install.meteor.com/ | sh
```

Il existe plusieurs façons d'installer Meteor, voir [ici](#) .

- `meteor` est le framework JavaScript que nous utiliserons pour construire notre application. Il nous fournit beaucoup de simplifications de codage pour certains problèmes relativement conceptuels dans les applications Web. sa simplicité a été notée comme utile pour des projets prototypiques. Lire plus [ici](#) .

NodeJS

```
apt-get install nodejs build-essentials
```

Il existe plusieurs manières d'installer, selon votre système d'exploitation. Découvrez de quelle manière vous avez besoin [ici](#) .

- `nodejs` est le package pour Node.js, qui est un environnement Javascript pour exécuter

JavaScript sur le côté serveur. Lire plus [ici](#) .

npm

npm doit être fourni avec l'installation `nodejs` . Vérifiez que c'est en lançant la commande `npm -v` après avoir installé `nodejs` .

- `npm` est le gestionnaire de paquetages de nœuds. C'est une énorme collection de modules open source que vous pouvez facilement ajouter à vos projets Node. Lire plus [ici](#) .

Utiliser l'électrification sur une application Meteor

Téléchargeons un exemple de projet Meteor Todos, en utilisant un script shell (ligne de commande) Linux, pour tester Electrifying un projet pour la première fois:

Conditions requises pour cette section:

Git

```
apt-get install git-all
```

Il y a plusieurs façons d'installer Git. Vérifiez-les [ici](#) .

- `git` est un système de contrôle de version pour les fichiers. Ils peuvent être stockés à distance (en ligne) dans des référentiels publics (GitHub étant plutôt célèbre) ou des référentiels privés (BitBucket fournit des référentiels privés gratuits limités, par exemple). Lire la suite [[ici](#)] [5].

```
#!/usr/bin/bash

# Change this parameter to choose where to clone the repository to.
TODOSPATH="/home/user/development/meteor-todos"

# Download the repository to the $TODOSPATH location.
git clone https://github.com/meteor/todos.git "$TODOSPATH"

# Change directory (`cd`) into the Todos project folder.
cd "$TODOSPATH"
```

Nous devrions maintenant avoir un dossier de projet nommé "meteor-todos", à l'emplacement spécifié dans le paramètre `TODOSPATH`. Nous avons également changé de répertoire (`cd`) dans le dossier du projet, ajoutons donc Electrify à ce projet!

```
# It's really this simple.
electrify
```

C'est vrai - une commande de mot unique, et notre projet est prêt. Les autorisations peuvent provoquer des erreurs lorsque vous tentez d'exécuter `electrify` une commande, dans le cas contraire, essayez `sudo electrify` pour remplacer les autorisations.

Cependant, essayez de résoudre ces problèmes de permission - ce n'est pas une bonne pratique d'inutile `sudo` (que je développerais, mais je pourrais écrire un tout autre sujet sur pourquoi c'est!)

Lire Electrify - Compiler Meteor comme une application localement installable en ligne:

<https://riptutorial.com/fr/meteor/topic/2526/electrify---compiler-meteor-comme-une-application-localement-installable>

Chapitre 14: Enregistrement

Exemples

Basic Server Side Logging

La première étape de la journalisation consiste simplement à exécuter Meteor à partir du shell, et vous obtiendrez les journaux du serveur dans la console de commande.

```
meteor
```

L'étape suivante consiste à diriger le contenu de `std_out` et `std_err` vers un fichier journal, comme ceci:

```
meteor > my_app_log.log 2> my_app_err.log
```

Outils de journalisation côté client

Une fois que votre serveur est connecté, il est temps de passer du côté client. Si vous n'avez pas encore exploré l'API de la console, préparez-vous à un traitement. Il y a en fait toutes sortes de choses que vous pouvez faire avec l'API intégrée de la console, native de chaque installation Chrome et Safari. À tel point que vous pourriez ne plus avoir besoin de Winston ou d'autres cadres de journalisation.

La première chose à faire est d'installer les outils de journalisation et de développement côté client. Chrome et Safari sont livrés avec eux, mais Firefox nécessite l'extension Firebug.

[Extension de Firebug](#)

Ensuite, vous voudrez consulter la documentation de l'API de la console. Les deux documents suivants sont des ressources inestimables pour l'apprentissage de la journalisation de la console.

[Outils de développement Chrome](#)

[Firebug \(Client\)](#)

Outils avancés de journalisation de serveur

Une fois que vous disposez à la fois de la journalisation côté serveur et de vos outils de développement côté client, vous pouvez commencer à examiner les extensions spécifiques à Meteor, telles que l'extension Meteor Chrome DevTools. Cela vous permet d'observer la connexion du serveur au client! Parce que la base de données est partout. Comme c'est la journalisation.

[Extension Chrome DevTools \(Serveur\)](#)

Erreur de journalisation sur le volet de la base de données

L'exemple suivant est compris entre 0,5 et 0,7 jours et montre comment enregistrer une erreur lorsque la base de données n'a pas encore renseigné le curseur côté client.

```
Template.landingPage.postsList = function(){
  try{
    return Posts.find();
  }catch(error){
    //color code the error (red)
    console.error(error);
  }
}
```

Informations de journalisation sur le contexte de données dans un assistant de modèle

Ce qui suit utilise l'API Chrome Logging. Si la syntaxe `.group()` est utilisée dans plusieurs modèles, elle organisera graphiquement les journaux de la console à partir de différents modèles en une arborescence hiérarchique.

Vous pouvez également voir comment inspecter le contexte de données actuel et comment corder des données.

```
Template.landingPage.getId = function(){
  // using a group block to illustrate function scoping
  console.group('coolFunction');

  // inspect the current data object that landingPage is using
  console.log(this);

  // inspect a specific field of the locally scoped data object
  console.log(JSON.stringify(this._id));

  // close the function scope
  console.groupEnd();
  return this._id;
}
```

Journalisation des événements et des interactions utilisateur

Exemple simple d'utilisation de l'API Chrome Logging.

```
Template.landingPage.events({
  'click .selectItemButton':function(){
    // color code and count the user interaction (blue)
    console.count('click .selectItemButton');
  }
});
```

Journalisation avec des variables de niveau journal

La journalisation peut souvent encombrer la console, il est donc courant de définir des niveaux de journalisation pour contrôler le détail des données à consigner. Un modèle courant consiste à spécifier des variables de niveau de journalisation.

```
var DEBUG = false;
var TRACE = false;
Template.landingPage.events({
  'click .selectItemButton':function(){
    TRACE && console.count('click .selectItemButton');

    Meteor.call('niftyAction', function(errorMessage, result){
      if(errorMessage){
        DEBUG && console.error(errorMessage);
      }
    });
  }
});
```

Désactiver la journalisation en production

Certaines équipes constatent qu'elles souhaitent laisser les instructions du journal de la console dans leur code, mais ne les affichent pas en production. Ils remplacent les fonctions de journalisation si une variable n'est pas définie (éventuellement une variable d'environnement). En outre, cela peut être considéré comme un élément de sécurité dans certaines situations.

```
if (!DEBUG_MODE_ON) {
  console = console || {};
  console.log = function(){};

  console.log = function(){};
  console.error = function(){};
  console.count = function(){};
  console.info = function(){};
}
```

Winston

Si vous avez besoin de quelque chose de plus puissant que les options de journalisation par défaut, vous voudrez peut-être examiner un outil comme Winston. Allez dans Atmosphère et recherchez simplement l'un des nombreux packages Winston disponibles.

<https://atmospherejs.com/?q=winston>

Soyez averti, cependant: Winston est un produit sophistiqué, et même s'il présente de nombreuses fonctionnalités, il ajoutera une complexité supplémentaire à votre application.

Niveau de la logle

Une mention spéciale doit être faite pour le package LogLevel développé par la communauté. Cela semble être un compromis entre la légèreté et la simplicité d'utilisation, tout en fonctionnant bien avec le pipeline de lots de Meteor et en préservant les numéros de lignes et les noms de

fichiers.

<https://atmospherejs.com/practicalmeteor/loglevel>

Lire Enregistrement en ligne: <https://riptutorial.com/fr/meteor/topic/3376/enregistrement>

Chapitre 15: Enveloppant les méthodes asynchrones dans une fibre pour une exécution synchrone.

Syntaxe

1. Meteor.wrapAsync (func, [contexte])

Paramètres

Paramètres	Détails
func: Fonction	Une fonction asynchrone / synchrone à encapsuler dans une fibre prenant un rappel avec paramètres (error, result) .
contexte: Any (facultatif)	Un contexte de données dans lequel la fonction est exécutée.

Remarques

Une fonction encapsulée de manière asynchrone peut toujours être exécutée de manière asynchrone si un rappel avec des paramètres (error, result) => {} est donné comme paramètre à la fonction encapsulée.

L'incorporation de Meteor.wrapAsync permet de Meteor.wrapAsync code Meteor.wrapAsync avec les rappels, car les callbacks peuvent maintenant être négligés en compensation du fait que le bloc d'appel est sa Fiber actuelle.

Pour comprendre comment fonctionnent les fibres, lisez ici: <https://www.npmjs.com/package/fibers>

Exemples

Exécution synchrone de méthodes NPM asynchrones avec callbacks.

Cet exemple enveloppe la méthode asynchrone oauth2.client.getToken(callback) du package NPM du package simple-oauth2 dans une fibre afin que la méthode puisse être appelée de manière synchrone.

```
const oauth2 = require('simple-oauth2')(credentials);  
  
const credentials = {
```

```
clientID: '#####',
clientSecret: '#####',
site: "API Endpoint Here."
};

Meteor.startup(() => {
  let token = Meteor.wrapAsync(oauth2.client.getToken)({});
  if (token) {
    let headers = {
      'Content-Type': "application/json",
      'Authorization': `Bearer ${token.access_token}`
    }

    // Make use of requested OAuth2 Token Here (Meteor HTTP.get).
  }
});
```

Lire Enveloppant les méthodes asynchrones dans une fibre pour une exécution synchrone. en ligne: <https://riptutorial.com/fr/meteor/topic/2530/enveloppant-les-methodes-asynchrones-dans-une-fibre-pour-une-execution-synchrone->

Chapitre 16: ES2015 modules (Import & Export)

Remarques

Documentation MDN pour les importations:

<https://developer.mozilla.org/en/docs/web/javascript/reference/statements/import> Documentation

MDN pour les exportations: <https://developer.mozilla.org/en/docs/web/javascript/reference/statements/exportation> ExplorationJS chapitre sur les modules:

http://exploringjs.com/es6/ch_modules.html

Exemples

Importation dans des modules d'application

Node modules

```
import url from 'url';
import moment from 'moment';
```

Forfaits Meteor

```
import { Meteor } from 'meteor/meteor';
import { SimpleSchema } from 'meteor/aldeed:simple-schema';
```

Importer dans les packages Meteor

Dans package.js:

```
Npm.depends({
  moment: "2.8.3"
});
```

Dans un fichier de package:

```
import moment from 'moment';
```

Exportation de variables à partir de modules d'application

```
// Default export
export default {};

// Named export
export const SomeVariable = {};
```

Exportation de symboles de packages Meteor

Dans votre fichier `mainModule` :

```
export const SomeVar = {};
```

Lire ES2015 modules (Import & Export) en ligne:

<https://riptutorial.com/fr/meteor/topic/3763/es2015-modules--import--amp--export->

Chapitre 17: ESLint

Exemples

Ajout d'eslint à votre projet Meteor

Nous utiliserons le populaire `eslint-config-airbnb` comme base ainsi que des règles spécifiques à Meteor en utilisant `eslint-import-resolver-meteor`.

Nous devons également installer `babel-parser` pour protéger les fonctionnalités ES7 telles que `async / waiting`.

```
cd my-project
npm install --save-dev eslint-config-airbnb eslint-plugin-import eslint-plugin-react eslint-plugin-jsx-ally eslint babel-eslint eslint-import-resolver-meteor
touch .eslintrc.json
```

Ensuite, utilisez simplement le `.eslintrc.json` de `.eslintrc.json` pour commencer, vous pouvez remplacer les règles comme vous le souhaitez.

```
{
  "parser": "babel-eslint",
  "settings": {
    "import/resolver": "meteor"
  },
  "extends": "airbnb",
  "rules": {}
}
```

Utiliser un script npm pour filtrer votre code

Modifiez votre `package.json` pour ajouter le script suivant:

```
{
  "scripts": {
    "lint": "eslint .;exit 0"
  }
}
```

Ensuite, lancez-le en utilisant `npm run lint`

Nous utilisons `exit 0` comme une astuce pour terminer le script en douceur lorsque le linting échoue, sinon `npm` utilisera le code retour `eslint` et le crash.

Lire ESLint en ligne: <https://riptutorial.com/fr/meteor/topic/3772/eslint>

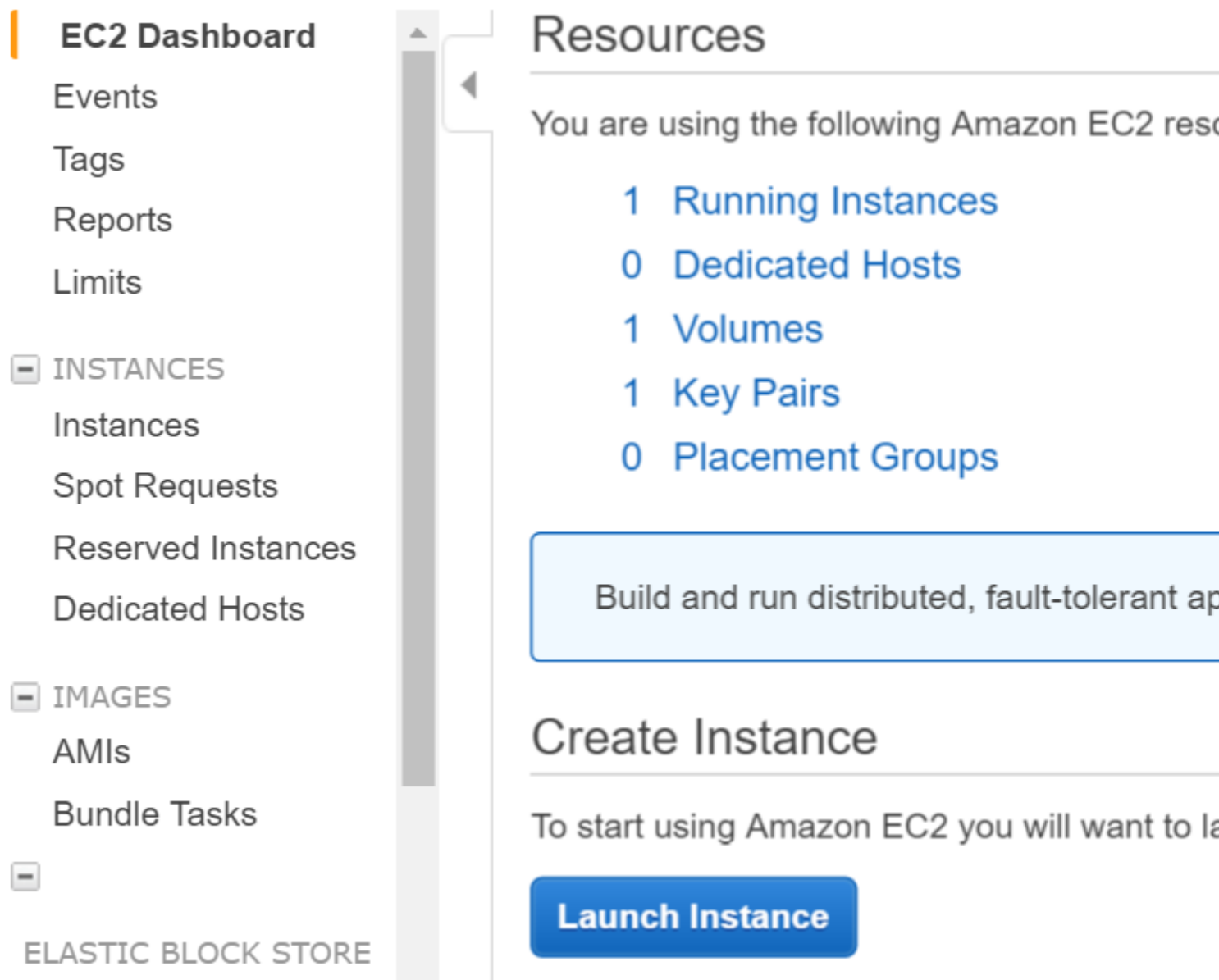
Chapitre 18: Guide d'initiation à l'installation de Meteor 1.4 sur AWS EC2

Exemples

Inscription au service AWS

Étant donné que beaucoup de débutants sont confus au sujet de l'hébergement en nuage, j'écris ce guide pour parcourir la mise en place de météores sur aws avec ubuntu os. Si votre instance est déjà en cours d'exécution, n'hésitez pas à passer cette étape et à passer directement à l'installation de météore sur aws.

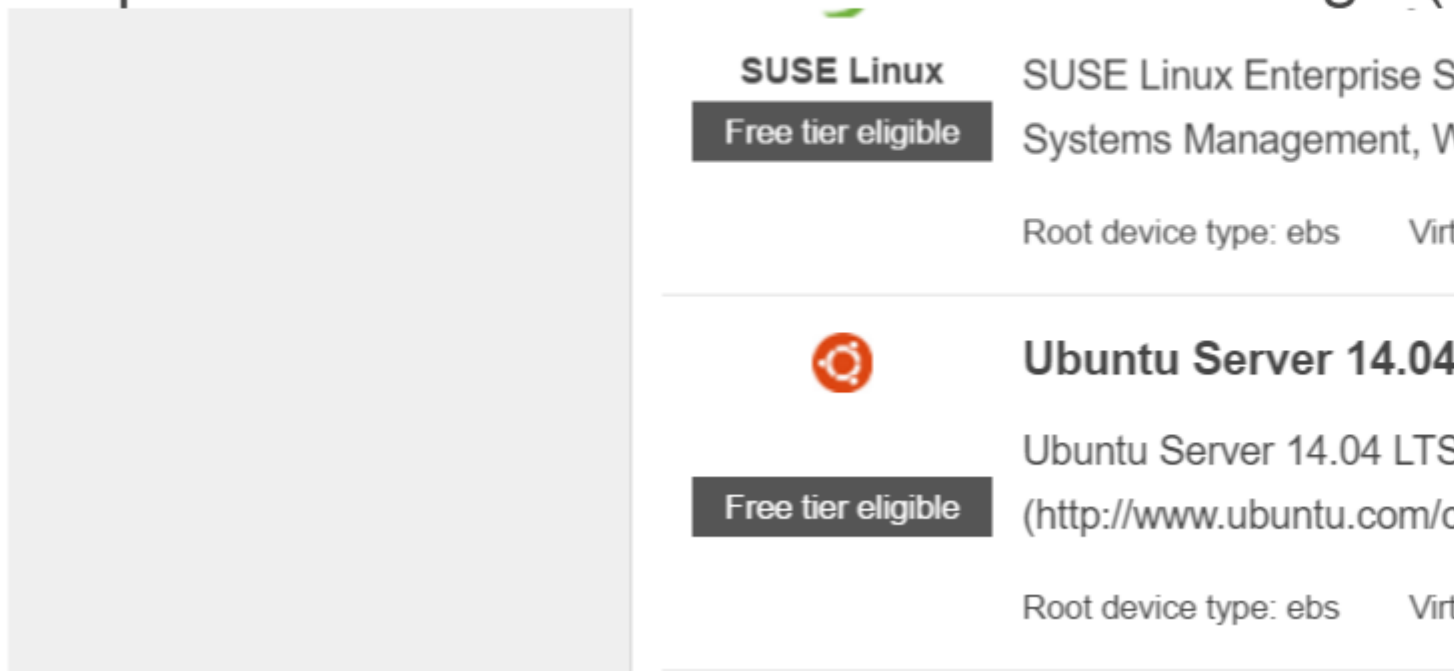
Connectez-vous à AWS Console. Sélectionnez EC2. Allez dans le tableau de bord EC2. Sous Create Instance, cliquez sur instance de lancement.



The screenshot shows the AWS Management Console interface. On the left is a navigation sidebar with the following items: EC2 Dashboard (highlighted with an orange bar), Events, Tags, Reports, Limits, INSTANCES (with a minus sign icon), Instances, Spot Requests, Reserved Instances, Dedicated Hosts, IMAGES (with a minus sign icon), AMIs, Bundle Tasks, and ELASTIC BLOCK STORE (with a minus sign icon). The main content area is titled 'Resources' and displays a list of EC2 resources: 1 Running Instances, 0 Dedicated Hosts, 1 Volumes, 1 Key Pairs, and 0 Placement Groups. Below this list is a light blue box with the text 'Build and run distributed, fault-tolerant ap...'. Underneath is a section titled 'Create Instance' with the text 'To start using Amazon EC2 you will want to la...' and a prominent blue button labeled 'Launch Instance'.

Sélectionnez l'instance Ubuntu à l'étape suivante

Step 1: Choose an Amazon Machine Image (AMI)



Créez une paire de clés et téléchargez une clé privée sur votre ordinateur local.

Connectez-vous via un shell à aws (en utilisant une clé privée, assurez-vous que la clé privée se trouve dans votre chemin ou exécutez la commande depuis le répertoire qui contient la clé privée)

```
ssh -i "myprivatekey.pem" ubuntu@ec2-xx-xx-xx-xx.ap-south-1.compute.amazonaws.com
```

ec2-xx-xx-xx-xx.ap-south-1.compute.amazonaws.com est le nom de l'instance DNS publique sur la console amazon. Ubuntu est un nom d'utilisateur. Vous pouvez également utiliser l'adresse IP publique.

ÉTAPES POUR INSTALLER METEOR SUR AWS INSTANCE (en utilisant mupx)

1. copier la clé privée de la machine locale dans le dossier ssh du serveur aws

exemple `/home/ubuntu/.ssh/myprivatekey.pem`

2. mettre à jour le packager avec la dernière version

```
sudo apt-get update
```

3. installer les propriétés du logiciel python

```
sudo apt-get install python-software-properties
```

4. installer npm et node (facultativement aussi installer nvm)

```
sudo apt-get install npm
```

Installez nvm

```
curl https://raw.githubusercontent.com/creationix/nvm/v0.11.1/install.sh | bash
```

Noeud d'installation

```
nvm install 4.4.7
```

```
nvm use 4.4.7
```

5. Installer aws cli

```
sudo apt-get install awscli
```

6. Installez meteor up

```
sudo npm install -g mupx
```

```
sudo npm install -g mupx-letsencrypt
```

(meteor 1.4 est actuellement disponible uniquement par mpux-letsencrypt)

7. Initialisez mupx en allant dans votre répertoire de projet ou créez un nouveau répertoire s'il n'existe pas

```
mupx-letsencrypt init
```

Si vous obtenez une erreur comme ci-dessous, alors le noeud hérité est là, vous devez créer un lien

```
/usr/bin/env: node: No such file or directory
```

```
sudo ln -s /usr/bin/nodejs /usr/bin/node
```

8. Installer le météore

```
curl https://install.meteor.com | /bin/sh
```

9. edit mup.json (Assurez-vous de remplir le nom d'utilisateur: ubuntu et l'emplacement correct de la clé privée à partir de l'étape 1)

utiliser l'éditeur de fichier nano (pour éditer des fichiers sur ubuntu, peut également utiliser vi)

Exemple mup.json

```

{
  // Server authentication info
  "servers": [
    {
      "host": "ec2-xx-xx-xx-xx.ap-south-1.compute.amazonaws.com",
      "username": "ubuntu",
      // "password": "password",
      // or pem file (ssh based authentication)
      "pem": "~/.ssh/myprivatekey.pem",
      // Also, for non-standard ssh port use this
      // "sshOptions": { "port" : 49154 },
      // server specific environment variables
      "env": {}
    }
  ],

  // Install MongoDB on the server. Does not destroy the local MongoDB on future setups
  "setupMongo": true,

  // WARNING: Node.js is required! Only skip if you already have Node.js installed on server.
  "setupNode": false,

  // WARNING: nodeVersion defaults to 0.10.36 if omitted. Do not use v, just the version
  // number.
  // "nodeVersion": "4.4.7",

  // Install PhantomJS on the server
  "setupPhantom": true,

  // Show a progress bar during the upload of the bundle to the server.
  // Might cause an error in some rare cases if set to true, for instance in Shippable CI
  "enableUploadProgressBar": true,

  // Application name (no spaces).
  "appName": "my-app",

  // Location of app (local directory). This can reference '~' as the users home directory.
  // i.e., "app": "/Users/ubuntu/my-app",
  // This is the same as the line below.
  "app": "/Users/ubuntu/my-app",

  // Configure environment
  // ROOT_URL must be set to https://YOURDOMAIN.com when using the spiderable package & force
  // SSL
  // your NGINX proxy or Cloudflare. When using just Meteor on SSL without spiderable this is
  // not necessary
  "env": {
    "PORT": 80,
    "ROOT_URL": "http://myapp.com",
    // only needed if mongodb is on separate server
    "MONGO_URL": "mongodb://url:port/MyApp",
    "MAIL_URL": "smtp://postmaster%40myapp.mailgun.org:adj87sjhd7s@smtp.mailgun.org:587/"
  },

  // Meteor Up checks if the app comes online just after the deployment.
  // Before mup checks that, it will wait for the number of seconds configured below.

```

```
"deployCheckWaitTime": 60
}
```

10. Installez Meteor, y compris mongo, en exécutant la commande suivante dans le répertoire du projet.

```
mupx-letsencrypt setup
```

11. déployer un projet en utilisant mupx

```
mupx-letsencrypt deploy
```

Quelques commandes utiles

Pour vérifier les journaux mupx

```
mupx logs -f
```

Pour vérifier Docker

```
docker -D info
```

Pour vérifier l'état du réseau

```
netstat -a
```

Pour vérifier le processus en cours d'exécution, y compris le processeur et l'utilisation de la mémoire

```
top
```

Installez le client mongo pour obtenir un accès mongo shell sur aws

```
sudo apt-get install mongodb-clients
```

Lancer des requêtes mongoddb

```
mongo projectName
```

Une fois à l'intérieur de la coquille mongo

```
db.version()
db.users.find()
```

Merci arunoda pour avoir fourni un excellent outil <https://github.com/arunoda/meteor-up>

Merci à l'équipe mupx-letsencrypt pour le bon travail. <https://www.npmjs.com/package/mupx->

letsencrypt

Lire Guide d'initiation à l'installation de Meteor 1.4 sur AWS EC2 en ligne:

<https://riptutorial.com/fr/meteor/topic/4773/guide-d-initiation-a-l-installation-de-meteor-1-4-sur-aws-ec2>

Chapitre 19: Installation complète - Mac OSX

Exemples

Installation du noeud et du NPM

Ce quickstart est écrit pour Mac OSX Mavericks et est un peu plus détaillé que les autres instructions d'installation. Il est à espérer que cela couvrira quelques cas marginaux, tels que la définition de votre chemin et la configuration de NPM, ce qui peut entraîner une mauvaise installation.

```
# install node
# as of OSX Mavericks, we need the GUI installer (!)
# when a good command line alternative is found, we'll post it
http://nodejs.org/download/

# install npm
curl -0 -L https://npmjs.org/install.sh | sh

# check node is installed correctly
node --version

# check npm is installed correctly
npm -version

# find your npm path
which npm

# make sure npm is in your path
sudo nano ~/.profile
export PATH=$PATH:/usr/local/bin
```

Procédure d'installation de Meteor

Ce quickstart est écrit pour Mac OSX Mavericks et est un peu plus détaillé que les autres instructions d'installation. Il est à espérer que cela couvrira quelques cas marginaux, tels que la définition de votre chemin et la configuration de NPM, ce qui peut entraîner une mauvaise installation.

```
# install meteor
curl https://install.meteor.com | sh

# check it's installed correctly
meteor --version

# install node
# as of OSX Mavericks, we need the GUI installer (!)
# when a good command line alternative is found, we'll post it
http://nodejs.org/download/

# install npm
curl -0 -L https://npmjs.org/install.sh | sh
```

```
# check node is installed correctly
node --version

# check npm is installed correctly
npm -version

# find your npm path
which npm

# make sure npm is in your path
sudo nano ~/.profile
export PATH=$PATH:/usr/local/bin
```

Installation Mongo

Meteor n'existe pas de manière isolée et il est courant d'installer un certain nombre d'outils supplémentaires pour le développement, tels que Mongo, Robomongo, Atom, Linters, etc.

```
# make sure mongo is in your local path
nano ~/.profile
export PATH=$PATH:/usr/local/mongodb/bin

# or install it to the global path
nano /etc/paths
/usr/local/mongodb/bin

# create mongo database directory
mkdir /data/
mkdir /data/db
chown -R username:admin /data

# run mongodb server
mongod
ctrl-c

# check that you can connect to your meteor app with stand-alone mongo
terminal-a$ meteor create helloworld
terminal-a$ cd helloworld
terminal-a$ meteor

terminal-b$ mongo -port 3001

# install robomongo database admin tool
http://robomongo.org/

# check you can connect to your mongo instance with robomongo
terminal-a$ meteor create helloworld
terminal-a$ cd helloworld
terminal-a$ meteor

Dock$ Robomongo > Create > localhost:3001
```

Autres outils de développement

```
# install node-inspector
terminal-a$ npm install -g node-inspector
```



```
# start meteor
terminal-a$ cd helloworld
terminal-a$ NODE_OPTIONS='--debug-brk --debug' mrt run

# alternatively, some people report this syntax being better
terminal-a$ sudo NODE_OPTIONS='--debug' ROOT_URL=http://helloworld.com meteor --port 80

# launch node-inspector along side your running app
terminal-b$ node-inspector

# go to the URL given by node-inspector and check it's running
http://localhost:8080/debug?port=5858

# install jshint
npm install -g jshint
```

Lire Installation complète - Mac OSX en ligne:

<https://riptutorial.com/fr/meteor/topic/3294/installation-complete---mac-osx>

Chapitre 20: Intégration continue et nuages de périphériques (avec Nightwatch)

Remarques

Nightwatch fournit des tests d'acceptation et de bout en bout pour les applications Meteor depuis la version v0.5, et a géré les migrations de PHP vers Spark vers Blaze et React; et toutes les principales plates-formes d'intégration continue. Pour une aide supplémentaire, veuillez consulter:

[Documentation de l'API Nightwatch](#)
[Groupe Google Nightwatch.js](#)

Exemples

Travis

Travis est le service d'intégration continue original qui est devenu populaire dans la communauté Meteor. Il est solide et fiable, a depuis longtemps un niveau d'hébergement open source et a réalisé des centaines de milliers de tests Nightwatch au fil des ans.

.travis.yml

Il suffit de mettre un fichier `.travis.yml` dans la racine de votre application, comme ceci:

```
# this travis.yml file is for the leaderboard-nightwatch example, when run standalone
language: node_js

node_js:
  - "0.10.38"

services:
  - mongodb

sudo: required

env:
  global:
    - TRAVIS=true
    - CONFIG_PREFIX=`npm config get prefix`
    - DISPLAY=:99.0
    - NODE_ENV=`travis`
  matrix:

cache:
  directories:
    - .meteor/local/build/programs/server/assets/packages
    - .meteor

before_install:
  # set up the node_modules dir, so we know where it is
  - "mkdir -p node_modules &"
```

```

# install nightwatch, selenium, , so we can launch nightwatch and selenium
- "meteor npm install nightwatch selenium-server-standalone-jar chromedriver"

# fire up xvfb on port :99.0
- "sh -e /etc/init.d/xvfb start"

# set the xvfb screen size to 1280x1024x16
- "/sbin/start-stop-daemon --start --quiet --pidfile /tmp/custom_xvfb_99.pid --make-pidfile
--background --exec /usr/bin/Xvfb -- :99 -ac -screen 0 1280x1024x16"

# install meteor
- "curl https://install.meteor.com | /bin/sh"

# give meteor a few seconds after installing
- "sleep 10"

# setup Meteor app
- "cd webapp"
- "meteor &"

# give Meteor some time to download packages, init data, and to start
- "sleep 60"

# then run nightwatch using the chromedriver
script: "nightwatch -c .meteor/nightwatch.json"

```

Cercle

Circle est le nouveau service d'intégration continue devenu populaire parmi les météorites. Il a toutes les dernières nouveautés en matière d'intégration continue. Le script suivant prend en charge de nombreuses nouvelles fonctionnalités, notamment:

- captures d'écran
- artefacts
- sous-modules git
- détection de l'environnement
- mise en cache des répertoires
- optimisation du parallélisme
- scripts npm
- déploiement continu
- Webhooks

.circle.yml

```

## Customize the test machine
machine:

# Timezone
timezone:
  America/Los_Angeles # Set the timezone

# Add some environment variables
environment:
  CIRCLE_ENV: test

```

```

CXX: g++-4.8
DISPLAY: :99.0
NPM_PREFIX: /home/ubuntu/nvm/v0.10.33
INITIALIZE: true
NODE_ENV: circle

## Customize checkout
checkout:
  post:
    #- git submodule sync
    #- git submodule update --init --recursive # use submodules

general:
  build_dir: webapp
  artifacts:
    - "./tests/nightwatch/screenshots" # relative to the build directory

## Customize dependencies
dependencies:
  cache_directories:
    - "~/.meteor" # relative to the user's home directory
    - ~/nvm/v0.10.33/lib/node_modules/starrynight
    - ~/nvm/v0.10.33/bin/starrynight

  pre:
    # Install Starrynight unless it is cached
    - if [ ! -e ~/nvm/v0.10.33/bin/starrynight ]; then npm install -g starrynight; else echo
"Starrynight seems to be cached"; fi;
    # Install Meteor
    - mkdir -p ${HOME}/.meteor
    # If Meteor is already cached, do not need to build it again.
    - if [ ! -e ${HOME}/.meteor/meteor ]; then curl https://install.meteor.com | /bin/sh; else
echo "Meteor seems to be cached"; fi;
    # Link the meteor executable into /usr/bin
    - sudo ln -s $HOME/.meteor/meteor /usr/bin/meteor
    # Check if the helloworld directory already exists, if it doesn't, create the helloworld
app
    # The following doesn't work, because it should be checking ${HOME}/active-
entry/helloworld
    # - if [ ! -e ${HOME}/helloworld ]; then meteor create --release METEOR@1.1.0.3
helloworld; else echo "helloworld app seems to be cached"; fi;

  override:
    #- meteor list

## Customize test commands
test:
  pre:
    #- starrynight fetch
    #- cd packages && rm -rf temp
    #- cd packages && ls -la
    #- starrynight autoconfig
    - meteor update --release METEOR@1.3.3
    - meteor npm install --save jquery bootstrap react react-dom react-router react-bootstrap
react-komposer
    - cat .meteor/nightwatch.json
    - meteor:
      background: true
    - sleep 60
  override:

```

```
- meteor npm run-script nightwatch

## Customize deployment commands
#deployment:
#  production:
#    branch: master
#    commands:
#      - printf "<Meteor username>\n<Meteor password>\n" | meteor deploy myapp.meteor.com

## Custom notifications
#notify:
#webhooks:
#  A list of hashes representing hooks. Only the url field is supported.
#- url: https://someurl.com/hooks/circle
```

SauceLabs

SauceLabs est une plate-forme de test automatisée pour l'entreprise. Il prend en charge à la fois l'intégration continue, les tests inter-navigateurs et un nuage de périphériques mobiles. Les coûts sont plus élevés qu'avec Travis, Circle ou BrowserStack.

```
{
  "selenium" : {
    "start_process" : false,
    "host" : "ondemand.saucelabs.com",
    "port" : 80,
  },
  "test_settings" : {
    "chrome_saucelabs": {
      "selenium_host": "ondemand.saucelabs.com",
      "selenium_port": 80,
      "username": "${SAUCE_USERNAME}",
      "access_key": "${SAUCE_ACCESS_KEY}",
      "use_ssl": false,
      "silent": true,
      "output": true,
      "screenshots": {
        "enabled": false,
        "on_failure": true,
        "path": ""
      },
    },
    "desiredCapabilities": {
      "name": "test-example",
      "browserName": "chrome"
    },
    "globals": {
      "myGlobal": "some_sauce_global"
    }
  },
}
```

BrowserStack

BrowserStack utilise un nuage de périphériques pour les tests entre navigateurs. L'intention est de

permettre le test des scripts Selenium sur chaque périphérique possible.

```
{
  "selenium" : {
    "start_process" : false,
    "host" : "hub.browserstack.com",
    "port" : 80,
  },

  "test_settings" : {
    "default" : {
      "launch_url" : "http://hub.browserstack.com",
      "selenium_port" : 80,
      "selenium_host" : "hub.browserstack.com",
      "silent": true,
      "screenshots" : {
        "enabled" : false,
        "path" : "",
      },
      "desiredCapabilities": {
        "browserName": "firefox",
        "javascriptEnabled": true,
        "acceptSslCerts": true,
        "browserstack.user": "USERNAME",
        "browserstack.key": "KEY"
      }
    }
  }
}
```

Lire Intégration continue et nuages de périphériques (avec Nightwatch) en ligne:

<https://riptutorial.com/fr/meteor/topic/6550/integration-continue-et-nuages---de-peripheriques--avec-nightwatch->

Chapitre 21: Intégration d'API tiers

Exemples

Appel HTTP basique

Conceptuellement, l'intégration d'API REST tierces peut être aussi simple que d'ajouter le package `http` et d'appeler le noeud final externe.

```
meteor add http
```

```
HTTP.get('http://foo.net/api/bar/');
```

Créer un package pour votre wrapper API

Les appels HTTP de base ne fournissent cependant pas de possibilité de réutilisation du code. Et ils peuvent être confondus avec toutes les autres fonctionnalités que vous essayez d'implémenter. Pour ces raisons, il est courant d'implémenter un wrapper API.

```
Foo = {
  identify: function(input){
    return Http.get('http://foo.net/api/identify/' + input);
  },
  record_action_on_item: function(firstInput, secondInput){
    return Http.put('http://foo.net/api/record_action_on_item/' + firstInput + '&' +
secondInput);
  }
}
```

Meteor prend en charge `Http.get()`, `Http.post()`, `Http.put()`, etc. http://docs.meteor.com/#http_get

Si l'API est bavarde et détaillée, vous pouvez recevoir plusieurs paquets; Dans ce cas, vous devrez les rassembler. C'est un gros problème. Si vous pensez que l'API renvoie plusieurs paquets, vous allez probablement vouloir utiliser le module npm «demande» sur le serveur. Vous voudrez utiliser un `Npm.require('request')`. <https://github.com/mikeal/request>

Créer un package d'atmosphère pour votre wrapper API

Après avoir créé un wrapper API, vous souhaitez probablement créer un package Atmosphère pour le redistribuer et le partager entre les applications. Les fichiers de votre paquet ressembleront probablement à ceci.

```
packages/foo-api-wrapper/package.js
packages/foo-api-wrapper/readme.md
packages/foo-api-wrapper/foo.api.wrapper.js
```

En particulier, votre fichier `foo-api-wrapper/package.js` vaudra ressembler à ceci:

```
Package.describe({
  summary: "Atmosphere package that impliments the Foo API.",
  name: "myaccount:foo",
  version: '0.0.1'
});

Package.on_use(function (api) {
  api.export('Foo');
  api.addFiles('foo.api.wrapper.js', ["client","server"]);
});
```

Et votre `foo-api-wrapper/foo.api.wrapper.js` devrait contenir l'objet wrapper `Foo` API.

Inclure le package API dans votre application

À ce stade, vous construisez toujours votre package, vous devrez donc ajouter le package à votre application:

```
meteor add myaccount:foo
```

Et éventuellement le publier sur Atmosphere:

```
meteor publish myaccount:foo
```

Utilisation de l'objet wrapper API dans votre application

Maintenant que nous avons rassemblé tous ces éléments, vous devriez maintenant pouvoir passer des appels comme suit à partir de votre application:

```
Foo.identify('John');
Foo.record_action_on_item('view', "HackerNews");
```

De toute évidence, vous voudrez ajuster les noms de fonction, les arguments, les URL, etc., pour créer la syntaxe appropriée pour l'API.

Lire [Intégration d'API tiers en ligne](https://riptutorial.com/fr/meteor/topic/3118/integration-d-api-tiers): <https://riptutorial.com/fr/meteor/topic/3118/integration-d-api-tiers>

Chapitre 22: Jeux de répliques et de fragmentation

Remarques

Pour ceux qui ne sont pas familiers, un jeu de réplicas est défini comme une configuration redondante de trois serveurs. Une base de données fragmentée est définie en tant que base de données à découpage horizontal, où chaque fragment est défini comme un jeu de réplicas. Par conséquent, un cluster Mongo fragmenté implique un minimum de 11 serveurs pour un cluster à 2 partitions et augmente de trois serveurs pour chaque fragment supplémentaire. Ainsi, un cluster fragmenté a toujours 11, 14, 17, 20, 23, etc. instances de serveur. Autrement dit, il y a 2 fragments de 3 serveurs chacun, 3 contrôleurs de configuration supplémentaires et 2 routeurs. 11 serveurs au total pour un cluster à 2 partitions.

Exemples

Jeu de répliques Quickstart

Construisez vous-même **trois** serveurs en utilisant le matériel physique ou virtuel de votre choix. (Ce tutoriel suppose que vous utilisez Ubuntu comme système d'exploitation.) Répétez ensuite les instructions suivantes trois fois ... une fois pour chaque serveur.

```
# add the names of each server to the host file of each server
sudo nano /etc/hosts
    10.123.10.101 mongo-a
    10.123.10.102 mongo-b
    10.123.10.103 mongo-c

# install mongodb on the server
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen' | sudo tee
/etc/apt/sources.list.d/mongodb.list
sudo apt-get update
sudo apt-get install mongodb-10gen

# create the /data/ directories
sudo mkdir /data
sudo mkdir /data/logs
sudo mkdir /data/db

# make sure the mongodb user and group have access to our custom directories
sudo chown -R mongodb:mongodb /data

# edit the mongo upstart file in /etc/init/mongodb.conf
sudo nano /etc/init/mongodb.conf
    start on started mountall
    stop on shutdown
    respawn
    respawn limit 99 5
```

```

setuid mongod
setgid mongod
script
  exec /usr/bin/mongod --config /etc/mongod.conf >> /data/logs/mongo-a.log 2>&1
end script

# edit mongod configuration file
sudo nano /etc/mongod.conf
  dbpath=/data/db
  logpath=/data/logs/mongod.log
  logappend=true
  port=27017
  noauth=true
  replSet=meteor
  fork=true

# add a mongo log-rotation file
sudo nano /etc/logrotate.d/mongod
/data/logs/*.log {
  daily
  rotate 30
  compress
  dateext
  missingok
  notifempty
  sharedscripts
  copytruncate
  postrotate
    /bin/kill -SIGUSR1 `cat /data/db/mongod.lock 2> /dev/null` 2> /dev/null || true
  endscript
}

# make sure mongod service is started and running
sudo service mongod start
sudo reboot

```

Configuration du jeu de réplicas

Ensuite, allez dans le shell mongo et lancez le jeu de réplicas, comme ceci:

```

meteor mongo

> rs.initiate()
PRIMARY> rs.add("mongo-a")
PRIMARY> rs.add("mongo-b")
PRIMARY> rs.add("mongo-c")
PRIMARY> rs.setReadPref('secondaryPreferred')

```

Lire Jeux de réplicas et de fragmentation en ligne:

<https://riptutorial.com/fr/meteor/topic/4332/jeux-de-replicas-et-de-fragmentation>

Chapitre 23: L'optimisation des performances

Remarques

Il convient de noter que Meteor est simplement Javascript et Node.js. Oui, il s'agit d'une implémentation très spécifique de ces deux technologies, dotée d'un écosystème unique et exploitant les API isomorphes et un magasin de données JSON pour obtenir des résultats vraiment étonnants. Mais en fin de compte, Meteor est une technologie Web, et elle est écrite en Javascript. Ainsi, toutes vos techniques de performance javascript classiques s'appliquent toujours. Commencez par là

[25 techniques de performance Javascript](#)

[Optimisation des performances pour le Javascript haute vitesse](#)

[Optimisation du code JavaScript](#)

[Astuces de performance pour JavaScript dans V8](#)

[10 conseils d'amélioration de la performance Javascript](#)

[JavaScript est efficace](#)

[Améliorer la performance de vos projets Meteor JS](#)

Exemples

Conception et déploiement de logiciels prêts pour la production

N'oubliez pas que toutes les meilleures pratiques de l'architecture Web typique s'appliquent toujours. Pour un excellent aperçu du sujet, veuillez vous reporter à l'excellent livre de Michael Nygard, [Release It! Concevoir et déployer un logiciel prêt pour la production](#) . Ecrire votre application dans Meteor ne vous dispense pas d'auditer des bibliothèques tierces, d'écrire des disjoncteurs, d'envelopper des appels dans des délais, de surveiller vos pools de ressources et tout le reste. Si vous voulez que votre application fonctionne correctement, vous devez vous assurer que vous utilisez des modèles de stabilité et que vous évitez les anti-patterns.

Modèles de stabilité

- Des délais d'attente
- Disjoncteurs
- Cloisons
- Poignée de main
- Stabilité Anti-Patterns

Points d'intégration

- Bibliothèques tierces
- Effets d'échelle
- Capacités déséquilibrées
- Capacité Anti-Patterns

Contention du pool de ressources

- AJAX Overkill
- Séances de dépassement
- Espace blanc excessif
- Eutropification des données

Si ces concepts ne vous sont pas familiers et ne vous semblent pas de seconde nature, cela signifie que vous n'avez pas mis en ligne de systèmes de production plus importants. Achetez une copie du livre. Ce sera du temps et de l'argent bien dépensés.

Lire L'optimisation des performances en ligne: <https://riptutorial.com/fr/meteor/topic/3363/l-optimisation-des-performances>

Chapitre 24: Le débogage

Exemples

Débogueurs de navigateur

Chrome et Safari ont tous deux des débogueurs intégrés. Avec Chrome, il vous suffit de cliquer avec le bouton droit de la souris sur une page Web et d'inspecter l'élément. Avec Safari, vous devez aller dans Préférences > Avancé et cliquer sur "Afficher le menu Développement dans la barre de menus".

Avec Firefox, vous devez installer [Firebug](#)

Ajouter des points d'arrêt du débogueur à votre application

Vous devrez ajouter des instructions de `debugger` à votre code:

```
Meteor.methods({
  doSomethingUseful: function(){
    debugger;
    niftyFunction();
  }
});
```

Débogage côté serveur avec l'inspecteur de noeud

Pour le débogage côté serveur, vous devez utiliser un outil tel que Node Inspector. Avant de commencer, consultez certains de ces tutoriels utiles.

[HowToNode - Débogage avec l'inspecteur de noeud](#)

[Strongloop - Applications de débogage](#)

[Déboguer facilement Meteor.js Walkthrough avec des captures d'écran de l'utilisation de Node Inspector avec Meteor](#)

tl; dr - il existe un certain nombre d'utilitaires dans l'écosystème Meteor conçus pour être exécutés en même temps que votre application Meteor. Ils ne fonctionnent que si votre application Meteor est opérationnelle et qu'ils peuvent se connecter à un site Web en cours d'exécution. meteor mongo, Robomongo, Nightwatch ... ce sont tous des utilitaires qui nécessitent que votre application soit déjà en cours d'exécution. NodeInspector est l'un de ces utilitaires.

```
# install node-inspector
terminal-a$ npm install -g node-inspector

# start meteor
terminal-a$ NODE_OPTIONS='--debug-brk --debug' mrt run

# alternatively, some people report this syntax being better
terminal-a$ sudo NODE_OPTIONS='--debug' ROOT_URL=http://myapp.com meteor --port 80
```

```
# launch node-inspector along side your running app
terminal-b$ node-inspector

# go to the URL given by node-inspector
http://localhost:8080/debug?port=5858
```

Débogage côté serveur avec débogage npm

Outre Node Inspector, certaines personnes ont signalé un succès avec un utilitaire npm appelé `debug`.

[MeteorHacks - Déboguer Meteor avec debug npm](#)

Meteor Shell

À partir de Meteor 1.0.2, il existe un nouveau shell de commandes que vous pouvez utiliser pour effectuer un débogage interactif et gérer votre application du côté serveur, comme vous le faites avec la console Chrome du côté client! Vérifiez-le:

```
meteor shell
```

Autres utilitaires de débogage

[Décharge de météores](#)

[Jouets Meteor](#)

[Constellation](#)

[Meteor DevTools](#)

Lire [Le débogage en ligne](#): <https://riptutorial.com/fr/meteor/topic/3378/le-debogage>

Chapitre 25: Le routage

Exemples

Routage avec routeur de fer

Installer le routeur de fer

Depuis le terminal:

```
meteor add iron:router
```

Configuration de base

```
Router.configure({
  //Any template in your routes will render to the {{> yield}} you put inside your layout
  template
    layoutTemplate: 'layout',
    loadingTemplate: 'loading'
});
```

Rendre sans données

```
//this is equal to home page
Router.route('/', function () {
  this.render('home')
});

Router.route('/some-route', function () {
  this.render('template-name');
});
```

Rendu avec données et paramètres

```
Router.route('/items/:_id', function () {
  this.render('itemPage', {
    data: function() {
      return Items.findOne({_id: this.params._id})
    }
  });
});
```

Rend à un rendement secondaire

```
Router.route('/one-route/route', function() {
  //template 'oneTemplate' has {{> yield 'secondary'}} in HTML
  this.render('oneTemplate');

  //this yields to the secondary place
  this.render('anotherTemplate', {
    to: 'secondary'
  });
});
```

```
});

//note that you can write a route for '/one-route'
//then another for '/one-route/route' which will function exactly like above.
});
```

S'abonner et attendre les données avant le rendu du modèle

```
Router.route('/waiting-first', {
  waitOn: function() {
    //subscribes to a publication
    //shows loading template until subscription is ready
    return Meteor.subscribe('somePublication')
  },

  action: function() {
    //render like above examples
  }
});
```

S'abonner à plusieurs publications et attendre les données avant de générer le modèle

```
Router.route('/waiting-first', {
  waitOn: function() {
    //subscribes to a publication
    //shows loading template until subscription is ready
    return [Meteor.subscribe('somePublication1'),Meteor.subscribe('somePublication2')];
  },

  action: function() {
    //render like above examples
  }
});
```

Guide pour routeur de fer: <http://iron-meteor.github.io/iron-router/>

Avec FlowRouter

[FlowRouter](#) est plus modulaire que Iron Router.

Installez FlowRouter

```
meteor add kadora:flow-router
```

Rendu d'un modèle

En particulier, vous devez ajouter manuellement un package de rendu de mise en page avec votre moteur de rendu:

- [Blaze Layout](#) pour Blaze: `meteor add kadora:blaze-layout`

- Réagir à la mise en page pour React: `meteor add kadira:react-layout`

Ensuite, vous pouvez effectuer un rendu par modélisation dynamique (dans le cas de Blaze):

```
<template name="mainLayout">
  {{> Template.dynamic template=area}}
</template>
```

```
FlowRouter.route('/blog/:postId', {
  action: function (params) {
    BlazeLayout.render("mainLayout", {
      area: "blog"
    });
  }
});
```

Rendu d'un modèle avec des paramètres et / ou une requête

Les paramètres sont spécifiés sur la route, comme avec Iron Router:

```
FlowRouter.route("/blog/:catId/:postId", {
  name: "blogPostRoute",
  action: function (params) {
    //...
  }
});
```

Mais les paramètres ne sont pas transmis en tant que contexte de données au modèle enfant. Au lieu de cela, le modèle enfant doit les lire:

```
// url: /blog/travel/france?showcomments=yes
var catId = FlowRouter.getParam("catId"); // returns "travel"
var postId = FlowRouter.getParam("postId"); // returns "france"

var color = FlowRouter.getQueryParam("showcomments"); // returns "yes"
```

Lire Le routage en ligne: <https://riptutorial.com/fr/meteor/topic/5119/le-routage>

Chapitre 26: Les atouts

Exemples

Accès aux ressources sur le serveur

Les ressources du serveur statique doivent être placées dans le répertoire `private`.

Fichiers texte

Les fichiers texte sont accessibles à l'aide de la `Assets.getText(assetPath, [asyncCallback])`. Par exemple, le fichier JSON suivant s'appelle `my_text_asset.json` et se trouve dans le répertoire `private`:

```
{
  "title": "Meteor Assets",
  "type": "object",
  "users": [
    {
      "firstName": "John",
      "lastName": "Doe"
    },
    {
      "firstName": "Jane",
      "lastName": "Doe"
    },
    {
      "firstName": "Matthias",
      "lastName": "Eckhart"
    }
  ]
}
```

Vous pouvez accéder à ce fichier sur le serveur en utilisant le code suivant:

```
var myTextAsset = Assets.getText('my_text_asset.json');
var myJSON = JSON.parse(myTextAsset);
console.log(myJSON.title); // prints 'Meteor Assets' in the server's console
```

Fichiers binaires

Si vous souhaitez accéder aux actifs du serveur en tant que fichier binaire EJSON, utilisez la `Assets.getBinary(assetPath, [asyncCallback])`. Voici un exemple de code pour accéder à une image nommée `my_image.png` qui se trouve dans le répertoire `private/img`:

```
var myBinaryAsset = Assets.getBinary('img/my_image.png');
```

Lire Les atouts en ligne: <https://riptutorial.com/fr/meteor/topic/3379/les-atouts>

Chapitre 27: Meteor + React + ReactRouter

Introduction

Ce document montre comment utiliser ReactRouter avec Meteor et React. De zéro à une application qui fonctionne, y compris les rôles et l'authentification.

Je vais montrer chaque étape avec un exemple

1- Créer le projet

2- Ajouter React + ReactRouter

3- Ajouter des comptes

4- Paquets de rôles

Exemples

Créer le projet

1- Tout d'abord, installez <https://www.meteor.com/install>

2- Créer un projet. (`--bare` est de créer un projet vide)

```
meteor create --bare MyAwesomeProject
```

3- Créez la structure de fichier minimale (`-p` pour créer des répertoires intermédiaires):

```
cd MyAwesomeProject
```

```
mkdir -p client server imports/api imports/ui/{components,layouts,pages}  
imports/startup/{client,server}
```

4- Maintenant, créez un fichier HTML dans client / main.html

```
<head>  
  <meta charset="utf-8">  
  <title>My Awesome Meteor_React_ReactRouter_Roles App</title>  
</head>  
  
<body>  
  Welcome to my Meteor_React_ReactRouter_Roles app  
</body>
```

5- Assurez-vous que cela fonctionne: (3000 est le port par défaut, vous pouvez donc ignorer le `'-p 3000'`)

```
meteor run -p 3000
```

et ouvrez votre navigateur sur 'localhost: 3000'

Remarque:

- Je saute d'autres fichiers que vous devrez créer pour raccourcir les choses. Plus précisément, vous devrez créer des fichiers `index.js` dans les répertoires `client`, `imports / startup / {client, serveur}` et `serveur`.
- Vous pouvez voir un exemple complet dans https://github.com/rafa-lft/Meteor_React_Base. Rechercher le tag `Step1_CreateProject`

Ajouter React + ReactRouter

Si nécessaire, `cd MyAwesomeProject` répertoire de votre projet `cd MyAwesomeProject`

1- Ajouter réagir et réagir au routeur

```
meteor npm install --save react-router@3.0.0 react@15.5.4 react-dom@15.5.4
```

2- Modifiez client / main.html et remplacez le contenu par:

```
<body>
  <div id="react-root"></div>
</body>
```

Quoi que le `reactRouter` décide d'afficher, il le montrera dans l'élément `'# react-root'`

3- Créer le fichier Layouts dans imports / ui / layouts / App.jsx

```
import React, { Component } from 'react';
import PropTypes from 'prop-types';

class App extends Component {
  constructor(props) {
    super(props);
  }

  render() {
    return (
      <div>
        {this.props.children}
      </div>
    );
  }
}

App.propTypes = {
  children: PropTypes.node
};

export default App;
```

4- Créer le fichier Routes dans imports / startup / client / Routes.jsx

```

import ReactDOM from 'react-dom';
import React, { Component } from 'react';
import { Router, Route, IndexRoute, browserHistory } from 'react-router';

import App from '../..//ui/layouts/App.jsx';

import NotFound from '../..//ui/pages/NotFound.jsx';
import Index from '../..//ui/pages/Index.jsx';

class Routes extends Component {
  constructor(props) {
    super(props);
  }

  render() {
    return (
      <Router history={ browserHistory }>
        <Route path="/" component={ App }>
          <IndexRoute name="index" component={ Index }/>
          <Route path="*" component={ NotFound }/>
        </Route>
      </Router>
    );
  }
}

Routes.propTypes = {};

Meteor.startup(() =>{
  ReactDOM.render(
    <Routes/>,
    document.getElementById('react-root')
  );
});

```

Remarque:

- Je saute d'autres fichiers que vous devrez créer pour raccourcir les choses. Plus précisément, vérifiez les importations / ui / pages {Index.jsx, NotFound.jsx}.
- Vous pouvez voir un exemple complet dans https://github.com/rafa-lft/Meteor_React_Base. Recherchez le tag *Step2_ReactRouter*

Étape 3 - Ajouter des comptes

Si nécessaire, `cd MyAwesomeProject` répertoire de votre projet `cd MyAwesomeProject`

1- Ajouter des packages de comptes: `meteor add accounts-base accounts-password react-meteor-data`

2- Ajoutez les routes pour vous *connecter* et vous *abonner aux* pages de imports / startup / Routes.jsx La méthode *render ()* sera la suivante:

```

render() {
  return (
    <Router history={ browserHistory }>
      <Route path="/" component={ App }>
        <IndexRoute name="index" component={ Index }/>
        <Route name="login" path="/login" component={ Login }/>
        <Route name="signup" path="/signup" component={ Signup }/>
        <Route name="users" path="/users" component={ Users }/>
        <Route name="editUser" path="/users/:userId" component={ EditUser }/>
        <Route path="*" component={ NotFound }/>
      </Route>
    </Router>
  );
}

```

Remarque:

- Je saute d'autres fichiers dont vous aurez besoin pour raccourcir les choses. Spécifiquement, vérifiez les importations / startup / server / index.js imports / ui / layouts / {App, NavBar} .jsx et import / ui / pages / {Connexion, Inscription, Utilisateurs, EditUser} .jsx
- Vous pouvez voir un exemple complet dans https://github.com/rafa-lft/Meteor_React_Base . Rechercher le tag *Step3_Accounts*

Ajouter des rôles

1- Ajouter un package de rôles (<https://github.com/alanning/meteor-roles>)

```
meteor add alanning:roles
```

2- Créez des constantes de rôles. Dans le fichier imports / api / accounts / roles.js

```

const ROLES = {
  ROLE1: 'ROLE1',
  ROLE2: 'ROLE2',
  ADMIN: 'ADMIN'
};

export default ROLES;

```

3- Je ne montrerai pas comment ajouter / mettre à jour des rôles sur un utilisateur, mais simplement sur le côté serveur, vous pouvez définir des rôles d'utilisateur par

Roles.setUserRoles(user.id, roles); Vérifiez plus d'informations dans <https://github.com/alanning/meteor-roles> et <http://alanning.github.io/meteor-roles/classes/Roles.html>

4- En supposant que vous avez déjà configuré tous les fichiers de comptes et de rôles (voir exemple complet dans https://github.com/rafa-lft/Meteor_React_Base . Recherchez le tag *Step4_roles*), nous pouvons maintenant créer une méthode qui sera chargée d'autoriser (ou non) accès aux différentes routes. Dans imports / startup / client / Routes.jsx

```

class Routes extends Component {
  constructor(props) {
    super(props);
  }

  authenticate(roles, nextState, replace) {
    if (!Meteor.loggingIn() && !Meteor.userId()) {
      replace({
        pathname: '/login',
        state: {nextPathname: nextState.location.pathname}
      });
      return;
    }
    if ('*' === roles) { // allow any logged user
      return;
    }
    let rolesArr = roles;
    if (!_isArray(roles)) {
      rolesArr = [roles];
    }
    // rolesArr = _.union(rolesArr, [ROLES.ADMIN]); // so ADMIN has access to everything
    if (!Roles.userIsInRole(Meteor.userId(), rolesArr)) {
      replace({
        pathname: '/forbidden',
        state: {nextPathname: nextState.location.pathname}
      });
    }
  }

  render() {
    return (
      <Router history={ browserHistory }>
        <Route path="/" component={ App }>
          <IndexRoute name="index" component={ Index }/>
          <Route name="login" path="/login" component={ Login }/>
          <Route name="signup" path="/signup" component={ Signup }/>

          <Route name="users" path="/users" component={ Users }/>

          <Route name="editUser" path="/users/:userId" component={ EditUser }
            onEnter={_.partial(this.authenticate, ROLES.ADMIN)} />

          { /* *****
            Below links are there to show Roles authentication usage.
            Note that you can NOT hide them by
            { Meteor.user() && Roles.userIsInRole(Meteor.user(), ROLES.ROLE1) &&
            <Route name=.....
            }
            as doing so will change the Router component on render(), and ReactRouter will
complain with:
            Warning: [react-router] You cannot change <Router routes>; it will be ignored

            Instead, you can/should hide them on the NavBar.jsx component... don't worry: if
someone tries to access
            them, they will receive the Forbidden.jsx component
            *****/ }
          <Route name="forAnyOne" path="/for_any_one" component={ ForAnyone }/>

          <Route name="forLoggedOnes" path="/for_logged_ones" component={ ForLoggedOnes }

```

```

        onEnter={_.partial(this.authenticate, '*')} />

<Route name="forAnyRole" path="/for_any_role" component={ ForAnyRole }
  onEnter={_.partial(this.authenticate, _.keys(ROLES))}/>

<Route name="forRole1or2" path="/for_role_1_or_2" component={ ForRole1or2 }
  onEnter={_.partial(this.authenticate, [ROLES.ROLE1, ROLES.ROLE2])} />

<Route name="forRole1" path="/for_role1" component={ ForRole1 }
  onEnter={_.partial(this.authenticate, ROLES.ROLE1)} />

<Route name="forRole2" path="/for_role2" component={ ForRole2 }
  onEnter={_.partial(this.authenticate, ROLES.ROLE2)} />

<Route name="forbidden" path="/forbidden" component={ Forbidden } />

<Route path="*" component={ NotFound } />
</Route>
</Router>
);
}
}

```

Nous avons ajouté un déclencheur *onEnter* à certaines routes. Pour ces routes, nous passons également les rôles autorisés à entrer. Notez que le rappel *onEnter* reçoit 2 paramètres à l'origine. Nous utilisons des traits de soulignement partiels (<http://underscorejs.org/#partial>), pour en ajouter un autre (rôles). La méthode *authenticate* (appelée par *onEnter*) reçoit les rôles et:

- Vérifiez si l'utilisateur est connecté. Sinon, redirige vers '/ login'.
- Si rôles === '*', nous supposons que tout utilisateur connecté peut entrer, donc nous l'autorisons
- Sinon, nous vérifions si l'utilisateur est autorisé (Roles.userIsInRole) et, sinon, nous redirigeons vers interdit.
- Si vous le souhaitez, vous pouvez décommenter une ligne, ADMIN a donc accès à tout.

Le code contient plusieurs exemples de différents itinéraires autorisés pour quiconque (pas de rappel *onEnter*), pour tout utilisateur enregistré, pour tout utilisateur connecté avec au moins un rôle et pour des rôles spécifiques.

Notez également que *ReactDOM* (au moins sur la version 3) ne permet pas de modifier les routes sur *Render*. Donc, vous ne pouvez pas cacher les routes dans les *Routes.jsx*. Pour cette raison, nous redirigeons vers / interdit dans la méthode d'authentification.

5- Un bug commun avec *ReactDOM* et *Meteor*, concerne les mises à jour de statut d'utilisateur non affichées. Par exemple, l'utilisateur s'est déconnecté, mais nous affichons toujours son nom sur la barre de navigation. Cela se produit parce que *Meteor.user ()* a changé, mais nous ne sommes pas en train de faire un nouveau rendu.

Ce bogue peut être résolu en appelant *Meteor.user ()* dans le *createContainer*. En voici un exemple, utilisé dans *imports / ui / layouts / NavBar.jsx*:


```
export default createContainer(({/* {params}*/) =>{
  Meteor.user(); // so we render again in logout or if any change on our User (ie: new roles)
  const loading = !subscription.ready();
  return {subscriptions: [subscription], loading};
}, NavBar);
```

Remarque:

- Je saute d'autres fichiers dont vous aurez besoin pour raccourcir les choses.
Spécifiquement, vérifiez les importations / startup / server / index.js imports / ui / layouts / {App, NavBar} .jsx et import / ui / pages / {Connexion, Inscription, Utilisateurs, EditUser} .jsx
- Vous pouvez voir un exemple complet dans https://github.com/rafa-lft/Meteor_React_Base .
Recherchez le tag *Step4_roles*

Lire Meteor + React + ReactRouter en ligne: <https://riptutorial.com/fr/meteor/topic/10114/meteor-plus-react-plus-reactrouter>

Chapitre 28: Meteor + Réagir

Remarques

[React](#) est une bibliothèque JavaScript pour créer des interfaces utilisateur. C'est [open source](#), développé et maintenu par Facebook. Meteor prend en charge la production de React.

Ressources:

- [Réagir au tutoriel](#)
- [Didacticiel Meteor + React](#)

Exemples

Configuration et "Hello World"

Ajouter Réagissez à votre projet:

```
meteor npm install --save react react-dom react-mounter
```

Créez le fichier `client/helloworld.jsx` pour afficher un composant React simple:

```
import React, { Component } from 'react';
import { mount } from 'react-mounter';

// This component only renders a paragraph containing "Hello World!"
class HelloWorld extends Component {
  render() {
    return <p>Hello World!</p>;
  }
}

// When the client application starts, display the component by mounting it to the DOM.
Meteor.startup(() => {
  mount(HelloWorld);
});
```

Créer un conteneur réactif à l'aide de createContainer

Disons qu'il y a une collection appelée `Todos` et que le paquet de `autopublish` est ajouté. Voici le composant de base.

```
import { createContainer } from 'meteor/react-meteor-data';
import React, { Component, PropTypes } from 'react';
import Todos from '/imports/collections/Todos';

export class List extends Component {
  render() {
    const { data } = this.props;
```

```

return (
  <ul className="list">
    {data.map(entry => <li {...entry} />)}
  </ul>
)
}
}

List.propTypes = {
  data: PropTypes.array.isRequired
};

```

En bas, vous pouvez ajouter un conteneur pour alimenter les données réactives dans le composant. Ça ressemblerait à ça.

```

export default createContainer(() => {
  return {
    data: Todos.find().fetch()
  };
}, List);

```

Affichage d'une collection MongoDB

Cet exemple montre comment une collection MongoDB peut être affichée dans un composant React. La collection est continuellement synchronisée entre le serveur et le client, et la page est instantanément mise à jour lorsque le contenu de la base de données change.

Pour connecter des composants React et des collections Meteor, vous aurez besoin du paquet `react-meteor-data`.

```

$ meteor add react-meteor-data
$ meteor npm install react-addons-pure-render-mixin

```

Une collection simple est déclarée dans `both/collections.js`. Chaque fichier source des `both` répertoires est à la fois un code côté client et un code côté serveur:

```

import { Mongo } from 'meteor/mongo';

// This collection will contain a list of random numbers
export const Numbers = new Mongo.Collection("numbers");

```

La collection doit être publiée sur le serveur. Créez une publication simple dans `server/publications.js`:

```

import { Meteor } from 'meteor/meteor';
import { Numbers } from '/both/collections.js';

// This publication synchronizes the entire 'numbers' collection with every subscriber
Meteor.publish("numbers/all", function() {
  return Numbers.find();
});

```

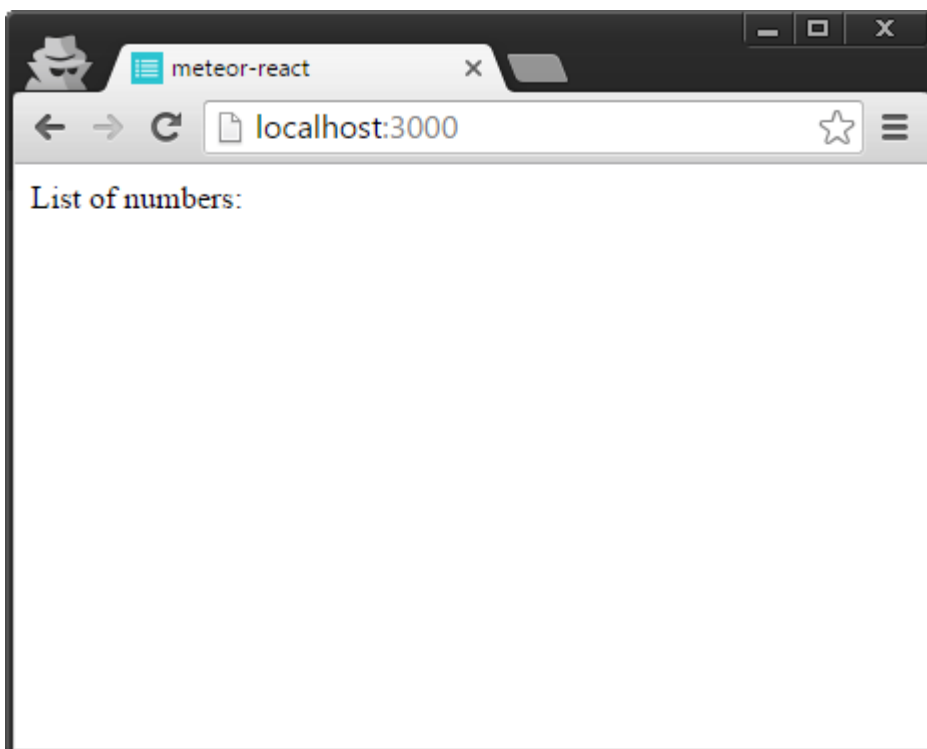
En utilisant la fonction `createComponent` , nous pouvons transmettre des valeurs réactives (comme la collection `Numbers`) à un composant React. `client/shownumbers.jsx` :

```
import React from 'react';
import { createContainer } from 'meteor/react-meteor-data';
import { Numbers } from '/both/collections.js';

// This stateless React component renders its 'numbers' props as a list
function _ShowNumbers({numbers}) {
  return <div>List of numbers:
    <ul>
      // note, that every react element created in this mapping requires
      // a unique key - we're using the _id auto-generated by mongodb here
      {numbers.map(x => <li key={x._id}>{x.number}</li>)}
    </ul>
  </div>;
}

// Creates the 'ShowNumbers' React component. Subscribes to 'numbers/all' publication,
// and passes the contents of 'Numbers' as a React property.
export const ShowNumbers = createContainer(() => {
  Meteor.subscribe('numbers/all');
  return {
    numbers: Numbers.find().fetch(),
  };
}, _ShowNumbers);
```

Initialement, la base de données est probablement vide.

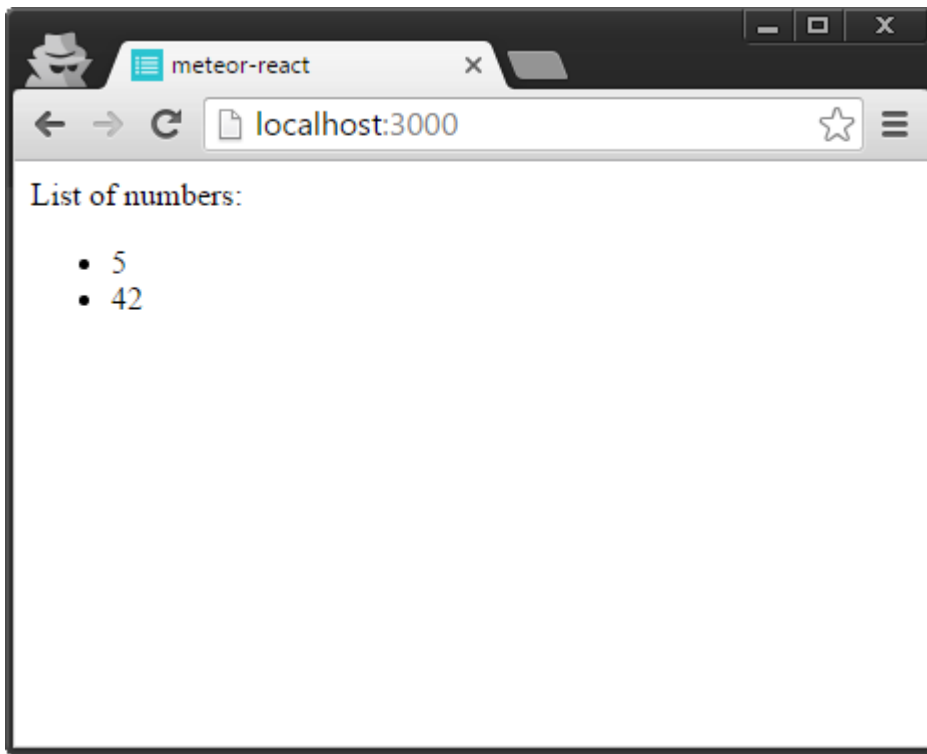


Ajoutez des entrées à MongoDB et observez la mise à jour automatique de la page.

```
$ meteor mongo
MongoDB shell version: 3.2.6
connecting to: 127.0.0.1:3001/meteor
```

```
meteor:PRIMARY> db.numbers.insert({number: 5});
WriteResult({ "nInserted" : 1 })

meteor:PRIMARY> db.numbers.insert({number: 42});
WriteResult({ "nInserted" : 1 })
```



Lire Meteor + Réagir en ligne: <https://riptutorial.com/fr/meteor/topic/3121/meteor-plus-reagir>

Chapitre 29: Migrations du schéma Mongo

Remarques

Il est souvent nécessaire d'exécuter des scripts de maintenance sur votre base de données. Les champs sont renommés; les structures de données sont modifiées; les fonctionnalités que vous avez utilisées pour prendre en charge sont supprimées; les services sont migrés. La liste des raisons pour lesquelles vous pourriez vouloir changer votre schéma est sans limites. Donc, le «pourquoi» est assez explicite.

Le «comment» est un peu moins familier. Pour les personnes habituées aux fonctions SQL, les scripts de base de données ci-dessus seront étranges. Mais remarquez comment ils sont tous en JavaScript, et comment ils utilisent la même API que nous utilisons dans Meteor, à la fois sur le serveur et sur le client. Nous avons une API cohérente via notre base de données, notre serveur et notre client.

Exécutez les commandes de migration du schéma à partir du shell mongo meteor:

```
# run meteor
meteor

# access the database shell in a second terminal window
meteor mongo
```

Exemples

Ajouter un champ de version à tous les enregistrements d'une collection

```
db.posts.find().forEach(function(doc) {
  db.posts.update({_id: doc._id}, {$set: {'version': 'v1.0'}}, false, true);
});
```

Supprimer un tableau de tous les enregistrements d'une collection

```
db.posts.find().forEach(function(doc) {
  if(doc.arrayOfObjects) {
    // the false, true at the end refers to $upsert, and $multi, respectively
    db.accounts.update({_id: doc._id}, {$unset: {'arrayOfObjects': ""}}, false, true);
  }
});
```

Renommer la collection

```
db.originalName.renameCollection("newName" );
```

Rechercher un champ contenant une chaîne spécifique

Avec le pouvoir des regex, il y a une grande responsabilité

```
db.posts.find({'text': /.foo.*|.bar.*/i})
```

Créer un nouveau champ à partir de l'ancien

```
db.posts.find().forEach(function(doc) {
  if(doc.oldField){
    db.posts.update({'_id': doc._id}, {$set: {'newField': doc.oldField}}, false, true);
  }
});
```

Sortez les objets d'un tableau et placez-les dans un nouveau champ

```
db.posts.find().forEach(function(doc) {
  if(doc.commenters){
    var firstCommenter = db.users.findOne({'_id': doc.commenters[0]._id });
    db.clients.update({'_id': doc._id}, {$set: {'firstPost': firstCommenter }}, false, true);

    var firstCommenter = db.users.findOne({'_id': doc.commenters[doc.commenters.length -
1]._id });
    db.clients.update({'_id': doc._id}, {$set: {'lastPost': object._id }}, false, true);
  }
});
```

Blob Record d'une collection à une autre collection (par exemple, supprimer Join & Flatten)

```
db.posts.find().forEach(function(doc) {
  if(doc.commentsBlobId){
    var commentsBlob = db.comments.findOne({'_id': commentsBlobId });
    db.posts.update({'_id': doc._id}, {$set: {'comments': commentsBlob }}, false, true);
  }
});
```

Assurez-vous que le champ existe

```
db.posts.find().forEach(function(doc) {
  if(!doc.foo){
    db.posts.update({'_id': doc._id}, {$set: {'foo': ''}}, false, true);
  }
});
```

Assurez-vous que le champ a une valeur spécifique

```
db.posts.find().forEach(function(doc) {
  if(!doc.foo){
    db.posts.update({'_id': doc._id}, {$set: {'foo': 'bar'}}, false, true);
  }
});
```

Supprimer un enregistrement si un champ spécifique est une valeur spécifique

```
db.posts.find().forEach(function(doc) {
  if(doc.foo === 'bar'){
    db.posts.remove({_id: doc._id});
  }
});
```

Changer la valeur spécifique du champ en nouvelle valeur

```
db.posts.find().forEach(function(doc) {
  if(doc.foo === 'bar'){
    db.posts.update({_id: doc._id}, {$set: {'foo': 'squee'}}, false, true);
  }
});
```

Champ spécifique non défini sur Null

```
db.posts.find().forEach(function(doc) {
  if(doc.olderfield){
    // the false, true at the end refers to $upsert, and $multi, respectively
    db.accounts.update({_id: doc._id}, {$unset: {'olderfield': ""}}, false, true);
  }
});
```

Convertir ObjectId en chaîne

```
db.posts.find().forEach(function(doc) {
  db.accounts.update({_id: doc._id}, {$set: {'_id': doc._id.str}}, false, true);
});
```

Convertir les valeurs de champ des nombres en chaînes

```
var newvalue = "";
db.posts.find().forEach(function(doc) {
  if(doc.foo){
    newvalue = '' + doc.foo + '';
    db.accounts.update({_id: doc._id}, {$set: {'doc.foo': newvalue}});
  }
});
```

Convertir les valeurs de champs de chaînes en nombres

```
var newvalue = null;
db.posts.find().forEach(function(doc) {
  if(doc.foo){
    newvalue = '' + doc.foo + '';
    db.accounts.update({_id: doc._id}, {$set: {'doc.foo': newvalue}});
  }
});
```



```
    }  
  });
```

Créer un horodatage à partir d'un ObjectID dans le champ `_id`

```
db.posts.find().forEach(function(doc) {  
  if(doc._id) {  
    db.posts.update({_id: doc._id}, {$set:{ timestamp: new  
Date(parseInt(doc._id.str.slice(0,8), 16) *1000) }}, false, true);  
  }  
});
```

Créer un ObjectID à partir d'un objet de date

```
var timestamp = Math.floor(new Date(1974, 6, 25).getTime() / 1000);  
var hex      = ('00000000' + timestamp.toString(16)).substr(-8); // zero padding  
var objectId = new ObjectId(hex + new ObjectId().str.substring(8));
```

Rechercher tous les enregistrements contenant des éléments dans un tableau

Ce que nous faisons ici fait référence à l'index du tableau en utilisant la notation par points

```
db.posts.find({"tags.0": {$exists: true }})
```

Lire Migrations du schéma Mongo en ligne: <https://riptutorial.com/fr/meteor/topic/3708/migrations-du-schema-mongo>

Chapitre 30: Mise à l'échelle horizontale

Exemples

Déploiement d'une application avec une base de données séparée (MONGO_URL)

Vous devez séparer votre couche d'application de la couche de base de données, ce qui signifie que vous devez spécifier MONGO_URL. Cela signifie que vous devez exécuter votre application via la commande bundle, la décompresser, définir des variables d'environnement, puis lancer le projet en tant qu'application de noeud. Voici comment...

```
#make sure you're running the node v0.10.21 or later
npm cache clean -f
npm install -g n
sudo n 0.10.21

# bundle the app
mkdir myapp
cd myapp
git clone http://github.com/myaccount/myapp
meteor bundle --directory ../deployPath
cd ../deployPath

# make sure fibers is installed, as per the README
export MONGO_URL='mongodb://127.0.0.1:27017/mydatabase'
export PORT='3000'
export ROOT_URL='http://myapp.com'

# run the site
node main.js
```

Configuration du jeu de réplicas

Ensuite, allez dans le shell mongo et lancez le jeu de répliques, comme ceci:

```
mongo

> rs.initiate()
PRIMARY> rs.add("mongo-a")
PRIMARY> rs.add("mongo-b")
PRIMARY> rs.add("mongo-c")
PRIMARY> rs.setReadPref('secondaryPreferred')
```

Configuration d'un jeu de réplicas pour utiliser l'oscillation

Le jeu de réplicas nécessitera un utilisateur oplog pour accéder à la base de données.

```
mongo
```

```
PRIMARY> use admin
PRIMARY>
db.addUser({user:"oplogger",pwd:"YOUR_PASSWORD",roles:[],otherDBRoles:{local:["read"]}});
PRIMARY> show users
```

Script de démarrage Oplog

Votre script de démarrage devra être modifié pour utiliser plusieurs adresses IP du jeu de réplicas.

```
start on started mountall
stop on shutdown

respawn
respawn limit 99 5

script
  # our example assumes you're using a replica set and/or oplog integration
  export MONGO_URL='mongodb://mongo-a:27017,mongo-b:27017,mongo-c:27017/meteor'

  # here we configure our OPLOG URL
  export MONGO_OPLOG_URL='mongodb://oplogger:YOUR_PASSWORD@mongo-a:27017,mongo-
b:27017,mongo-c:27017/local?authSource=admin'

  # root_url and port are the other two important environment variables to set
  export ROOT_URL='http://myapp.mydomain.com'
  export PORT='80'

  exec /usr/local/bin/node /var/www/production/main.js >> /var/log/node.log 2>&1
end script
```

Éclatement

Oplog Tailing sur Sharded Mongo

Lire Mise à l'échelle horizontale en ligne: <https://riptutorial.com/fr/meteor/topic/3706/mise-a-l-echelle-horizontale>

Chapitre 31: Mongo Database Management

Remarques

Si vous n'êtes pas opposé à l'utilisation d'un utilitaire Ruby, Genghis est un classique:

<http://genghisapp.com/>

Mais pour une utilisation évolutive de la production, rendez-vous à MongoHQ.

<http://www.mongohq.com/>

Aussi, le Mongo Monitoring Service, de 10Gen, les fabricants de Mongo:

<https://mms.mongodb.com/>

[MongoClient](#) est écrit dans Meteor, Complètement Gratuit, Open Source et Cross-Platform.

Outil de gestion MongoDB natif multi-plateforme [RoboMongo](#)

Exemples

Analyser une base de données héritée

Il existe deux excellents utilitaires pour l'analyse des bases de données en boîte noire. Le premier est la variété.js, qui vous donnera un aperçu de haut niveau. Le second est schema.js, qui vous permettra de creuser dans les collections pour plus de détails sur les champs individuels. Lorsque vous héritez d'une base de données Mongo de production, ces deux utilitaires peuvent vous aider à comprendre ce qui se passe et comment les collections et les documents sont structurés.

[variété.js](#)

```
mongo test --eval "var collection = 'users'" variety.js
```

[schema.js](#)

```
mongo --shell schema.js
```

Connectez-vous à une base de données sur * .meteorapp.com

Le drapeau --url peut être difficile à utiliser. Il y a une fenêtre de 60 secondes pour s'authentifier, puis le nom d'utilisateur / mot de passe est réinitialisé au hasard. Assurez-vous donc que robomongo est ouvert et prêt à configurer une nouvelle connexion lorsque vous exécutez la commande.

```
# get the MONGO_URL string for your app
meteor mongo --url $METEOR_APP_URL
```

Téléchargez une base de données à partir de * .meteor.com

Même chose qu'auparavant, mais vous devez copier les informations dans la commande mongodump. Vous devez exécuter les commandes suivantes rapidement, et cela nécessite une coordination manuelle / visuelle. Être averti! Ceci est un piratage rediculeusement! Mais amusant! Pensez-y comme un jeu vidéo! :RÉ

```
# get the MONGO_URL string for your app
meteor mongo --url $METEOR_APP_URL

# then quickly copy all the info into the following command
mongodump -u username -p password --port 27017 --db meteor_app_url_com --host production-db-
b1.meteor.io
```

Exporter des données depuis une instance de développement Meteor locale?

Cette commande crée un répertoire / dump et stocke chaque collection dans un fichier blob BSON distinct. C'est le meilleur moyen de sauvegarder ou de transférer des bases de données entre systèmes.

```
mongodump --db meteor
```

Restaurer des données à partir d'un fichier de vidage

L'analogie de la `meteordump` commande est `meteorrestore`. Vous pouvez effectuer une importation partielle en sélectionnant la collection spécifique à importer. Particulièrement utile après avoir exécuté une commande drop.

```
# make sure your app is running
meteor

# then import your data
mongorestore --port 3001 --db meteor /path/to/dump

# a partial import after running > db.comments.drop()
mongorestore --port 3001 --db meteor /path/to/dump -c comments.bson
```

Exporter une collection vers JSON

Exécutez meteor, ouvrez une autre fenêtre de terminal et exécutez la commande suivante.

```
mongoexport --db meteor --collection foo --port 3001 --out foo.json
```

Importer un fichier JSON dans Meteor

Importer dans une instance Meteor par défaut est assez facile. Notez que vous pouvez ajouter une option `--jsonArray` si votre fichier json est exporté en tant que tableau depuis un autre système.

```
mongoimport --db meteor --port 3001 --collection foo --file foo.json
```

Copie de données entre bases de données intermédiaires et locales

Mongo prend en charge la copie de base de données à base de données, ce qui est utile si vous souhaitez copier des bases de données volumineuses sur une base de données intermédiaire dans une instance de développement local.

```
// run mongod so we can create a staging database
// note that this is a separate instance from the meteor mongo and minimongo instances
mongod

// import the json data into a staging database
// jsonArray is a useful command, particularly if you're migrating from SQL
mongoimport -d staging -c assets < data.json --jsonArray

// navigate to your application
cd myappdir

// run meteor and initiate it's database
meteor

// connect to the meteor mongodb
meteor mongo --port 3002

// copy collections from staging database into meteor database
db.copyDatabase('staging', 'meteor', 'localhost');
```

Compacter une base de données Mongo sur une boîte Ubuntu

Préallocation Mongo met de l'espace disque dans des conteneurs vides, alors quand vient le temps d'écrire quelque chose sur le disque, il n'a pas à mélanger les bits en premier. Il le fait par un algorithme de doublage, doublant toujours la quantité d'espace disque pré-alloué jusqu'à ce qu'il atteigne 2 Go; et ensuite chaque fichier pré-appel est de 2 Go. Une fois les données pré-allouées, elles ne sont plus allouées à moins que vous ne le leur disiez spécifiquement. Donc, l'utilisation de l'espace MongoDB, qui est observable, a tendance à augmenter automatiquement, mais pas à la baisse.

Quelques recherches sur la préallocation Mongo ...

[réduire-mongodb-taille-fichier-base de données](#)

[mongo-préallocation de fichiers](#)

```
// compact the database from within the Mongo shell
db.runCommand( { compact : 'mycollectionname' } )

// repair the database from the command line
mongod --config /usr/local/etc/mongod.conf --repair --repairpath /Volumes/X/mongo_repair --
nojurnal

// or dump and re-import from the command line
mongodump -d databasename
echo 'db.dropDatabase()' | mongo databasename
mongorestore dump/databasename
```

Réinitialiser un jeu de répliques

Supprimez les fichiers de la base de données locale. Quittez simplement le shell Mongo, accédez au / dbpath (où que vous le configuriez) et supprimez les fichiers dans ce répertoire.

Connectez-vous à distance à une instance Mongo sur * .meteor.com

Avez-vous connu le drapeau `--url` ? Très utile.

```
meteor mongo --url YOURSITE.meteor.com
```

Accès aux fichiers journaux Mongo sur une instance Meteor locale

Ils ne sont pas facilement accessibles. Si vous exécutez la commande 'meteor bundle', vous pouvez générer un fichier tar.gz, puis exécuter votre application manuellement. En faisant cela, vous devriez pouvoir accéder aux journaux mongo ... probablement dans le répertoire `.meteor/db`. Si vous avez vraiment besoin d'accéder aux fichiers journaux mongod, configurez une instance mongod standard, puis connectez Meteor à une instance mongo externe, en définissant la variable d'environnement `MONGO_URL`:

```
MONGO_URL='mongodb://user:password@host:port/databasename'
```

Une fois cela fait, vous devriez pouvoir accéder aux journaux aux endroits habituels ...

```
/var/log/mongodb/server1.log
```

Faire pivoter les fichiers journaux sur une boîte Ubuntu

Vous devez faire pivoter ces fichiers journaux ou ils finiront par consommer tout votre espace disque. Commencez par des recherches ...

[mongodb-log-file-growth](#)

[tourner-log-files](#)

Les fichiers journaux peuvent être visualisés avec la commande suivante ...

```
ls /var/log/mongodb/
```

Mais pour configurer la rotation des fichiers journaux, vous devez procéder comme suit:

```
// put the following in the /etc/logrotate.d/mongod file
/var/log/mongo/*.log {
    daily
    rotate 30
    compress
    dateext
    missingok
    notifempty
    sharedscripts
```

```
copytruncate
postrotate
    /bin/kill -SIGUSR1 `cat /var/lib/mongo/mongod.lock 2> /dev/null` 2> /dev/null || true
endscript
}

// to manually initiate a log file rotation, run from the Mongo shell
use admin
db.runCommand( { logRotate : 1 } )
```

Lire Mongo Database Management en ligne: <https://riptutorial.com/fr/meteor/topic/3707/mongo-database-management>

Chapitre 32: MongoDB

Introduction

MongoDB est un programme gratuit et open-source de base de données multi-plates-formes. Contrairement aux bases de données SQL classiques, MongoDB utilise BSON (comme JSON) pour stocker des données. Meteor a été conçu pour utiliser MongoDB pour le stockage de base de données et cette rubrique explique comment implémenter le stockage MongoDB dans les applications Meteor.

Exemples

Exportation d'une base de données Mongo distante, importation dans une base de données Meteor Mongo locale

Utile lorsque vous souhaitez récupérer une copie d'une base de données de production pour jouer avec localement.

1. `mongodump --host some-mongo-host.com:1234 -d DATABASE_NAME -u DATABASE_USER -p DATABASE_PASSWORD` Ceci créera un répertoire de `dump` local; Dans ce répertoire, vous verrez un répertoire avec votre `DATABASE_NAME`.
2. Avec votre application de météorologie locale en cours d'exécution, à partir du répertoire `dump`, exécutez: `mongorestore --db meteor --drop -h localhost --port 3001 DATABASE_NAME`

Obtenez l'URL Mongo de votre base de données locale Meteor Mongo

Pendant que votre application Meteor s'exécute localement:

```
meteor mongo --url
```

Connectez votre application Meteor locale à une autre base de données Mongo

Définissez la variable d'environnement `MONGO_URL` avant de démarrer votre application Meteor locale.

Exemple Linux / MacOS:

```
MONGO_URL="mongodb://some-mongo-host.com:1234/mydatabase" meteor
```

ou

```
export MONGO_URL="mongodb://some-mongo-host.com:1234/mydatabase"  
meteor
```

Exemple Windows

Note: ne pas utiliser "

```
set MONGO_URL=mongodb://some-mongo-host.com:1234/mydatabase
meteor
```

MNP

```
//package.json

"scripts": {
  "start": "MONGO_URL=mongodb://some-mongo-host.com:1234/mydatabase meteor"
}

$ npm start
```

Running Meteor sans MongoDB

Définissez `MONGO_URL` sur toute valeur arbitraire, à l'exception d'une URL de base de données, et assurez-vous qu'aucune collection n'est définie dans votre projet Meteor (y compris les collections définies par les packages Meteor) pour exécuter Meteor sans MongoDB.

Notez que sans MongoDB, les méthodes serveur / client associées aux paquets liés au système de compte utilisateur de Meteor ne seront pas définies. Ex: `Meteor.userId()`

Linux / Mac:

```
MONGO_URL="none" meteor
```

OU

```
export MONGO_URL="none"
meteor
```

Les fenêtres:

```
set MONGO_URL=none
meteor
```

Commencer

Vous pouvez démarrer le shell `mongo` en exécutant la commande suivante dans votre projet Meteor:

```
meteor mongo
```

Remarque: le démarrage de la console de base de données côté serveur ne fonctionne que lorsque Meteor exécute l'application localement.

Après cela, vous pouvez lister toutes les collections en exécutant la commande suivante via le shell `mongo` :

```
show collections
```

Vous pouvez également exécuter des opérations de base MongoDB, telles que l'interrogation, l'insertion, la mise à jour et la suppression de documents.

Documents de requête

Les documents peuvent être interrogés en utilisant la méthode `find()` , par exemple:

```
db.collection.find({name: 'Matthias Eckhart'});
```

Cela listera tous les documents dont l'attribut `name` défini sur `Matthias Eckhart` .

Insérer des documents

Si vous souhaitez insérer des documents dans une collection, exécutez:

```
db.collection.insert({name: 'Matthias Eckhart'});
```

Mise à jour des documents

Si vous souhaitez mettre à jour des documents, utilisez la méthode `update()` , par exemple:

```
db.collection.update({name: 'Matthias Eckhart'}, {$set: {name: 'John Doe'}});
```

L'exécution de cette commande mettra à jour un **seul** document en définissant la valeur `John Doe` pour le `name` du champ (initialement, la valeur était `Matthias Eckhart`).

Si vous souhaitez mettre à jour **tous les** documents correspondant à un critère spécifique, définissez le paramètre `multi` sur `true` , par exemple:

```
db.collection.update({name: 'Matthias Eckhart'}, {$set: {name: 'John Doe'}}, {multi: true});
```

Désormais, tous les documents de la collection dont l'attribut `name` était initialement défini sur `Matthias Eckhart` ont été mis à jour avec `John Doe` .

Suppression de documents

Les documents peuvent être facilement supprimés à l'aide de la méthode `remove()` , par exemple:

```
db.collection.remove({name: 'Matthias Eckhart'});
```

Cela supprimera tous les documents qui correspondent à la valeur spécifiée dans le champ du `name` .

Lire MongoDB en ligne: <https://riptutorial.com/fr/meteor/topic/1874/mongodb>

Chapitre 33: Nightwatch - Configuration et configuration

Remarques

Nightwatch fournit des tests d'acceptation et de bout en bout pour les applications Meteor depuis la version v0.5, et a géré les migrations de PHP vers Spark vers Blaze et React; et toutes les principales plates-formes d'intégration continue. Pour une aide supplémentaire, veuillez consulter:

[Documentation de l'API Nightwatch](#)
[Groupe Google Nightwatch.js](#)

Exemples

Configuration

La raison principale pour laquelle Nightwatch est si puissant est son excellent fichier de configuration. Contrairement à la plupart des autres environnements de test, Nightwatch est entièrement configurable et personnalisable pour différents environnements et piles technologiques.

.meteor / nightwatch.json

Le fichier de configuration suivant concerne Meteor v1.3 et versions ultérieures et prend en charge deux environnements: un environnement `default` qui lance le navigateur chromedriver et un environnement `phantom` qui exécute les tests dans un environnement sans tête.

```
{
  "nightwatch": {
    "version": "0.9.8"
  },
  "src_folders": [
    "./tests/nightwatch/walkthroughs"
  ],
  "custom_commands_path": [
    "./tests/nightwatch/commands"
  ],
  "custom_assertions_path": [
    "./tests/nightwatch/assertions"
  ],
  "output_folder": "./tests/nightwatch/reports",
  "page_objects_path": "./tests/nightwatch/pages",
  "globals_path": "./tests/nightwatch/globals.json",
  "selenium": {
    "start_process": true,
    "server_path": "./node_modules/starrynight/node_modules/selenium-server-standalone-jar/jar/selenium-server-standalone-2.45.0.jar",
    "log_path": "tests/nightwatch/logs",
    "host": "127.0.0.1",
```

```

    "port": 4444,
    "cli_args": {
      "webdriver.chrome.driver":
"./node_modules/starrynight/node_modules/chromedriver/bin/chromedriver"
    }
  },
  "test_settings": {
    "default": {
      "launch_url": "http://localhost:5000",
      "selenium_host": "127.0.0.1",
      "selenium_port": 4444,
      "pathname": "/wd/hub",
      "silent": true,
      "disable_colors": false,
      "firefox_profile": false,
      "ie_driver": "",
      "screenshots": {
        "enabled": false,
        "path": "./tests/nightwatch/screenshots"
      },
      "desiredCapabilities": {
        "browserName": "chrome",
        "javascriptEnabled": true,
        "acceptSslCerts": true,
        "loggingPrefs": {
          "browser": "ALL"
        }
      },
      "exclude": "./tests/nightwatch/unittests/*",
      "persist_globals": true,
      "detailed_output": false
    },
    "phantom": {
      "desiredCapabilities": {
        "browserName": "phantomjs",
        "javascriptEnabled": true,
        "databaseEnabled": false,
        "locationContextEnabled": false,
        "applicationCacheEnabled": false,
        "browserConnectionEnabled": false,
        "webStorageEnabled": false,
        "acceptSslCerts": true,
        "rotatable": false,
        "nativeEvents": false,
        "phantomjs.binary.path": "./node_modules/starrynight/node_modules/phantomjs-
prebuilt/bin/phantomjs"
      }
    },
    "unittests": {
      "selenium": {
        "start_process": false,
        "start_session": false
      },
      "filter": "./tests/nightwatch/unittests/*",
      "exclude": ""
    }
  }
}

```

Installation et utilisation

Pour que Nightwatch fonctionne, vous aurez besoin d'une copie locale de **sélénium**, un serveur de commande et de contrôle qui gère les instances de navigateur automatisées. Vous aurez également besoin d'un navigateur Web que le sélénium peut contrôler, tel que **chromedriver** ou **phantomjs** .

Ajoutez les dépendances devDep suivantes à votre `package.json` :

```
{
  "devDependencies": {
    "nightwatch": "0.9.8",
    "selenium-server-standalone-jar": "2.45.0",
    "chromedriver": "2.19.0",
    "phantomjs-prebuilt": "2.1.12"
  }
}
```

Installez ensuite toutes les dépendances.

```
cd myapp
meteor npm install
```

Vous devriez alors pouvoir exécuter Nightwatch avec les commandes suivantes:

```
nightwatch -c .meteor/nightwatch.json
nightwatch -c .meteor/nightwatch.json --env phantom
```

Si vous n'avez pas encore écrit de test ou configuré votre structure de dossiers, vous risquez de rencontrer des erreurs.

Configuration des scripts de lancement

La racine de votre application doit être un fichier `package.json` , dans lequel vous pouvez définir des scripts et des `devDependencies`.

```
{
  "name": "myapp",
  "version": "1.0.0",
  "scripts": {
    "start": "meteor --settings settings-development.json",
    "nightwatch": "nightwatch -c .meteor/nightwatch.json",
    "phantom": "nightwatch -c .meteor/nightwatch.json --env phantom",
  }
}
```

Vous pourrez alors lancer la surveillance nocturne avec les commandes suivantes:

```
meteor npm run-script nightwatch
meteor npm run-script phantom
```

Dans cet exemple, il serait presque plus facile de simplement exécuter `nightwatch -c .meteor/nightwatch.json` . Cependant, avec des commandes plus complexes, avec des variables,

des options et des paramètres d'environnement complexes, cela devient un moyen très utile pour spécifier des scripts devops pour une équipe.

Structure des dossiers

Une installation de base Nightwatch pour Meteor aura les répertoires et fichiers suivants installés.

```
/myapp
/myapp/.meteor/nightwatch.json
/client/main.html
/client/main.js
/client/main.css
/tests
/tests/nightwatch
/tests/nightwatch/assertions
/tests/nightwatch/commands
/tests/nightwatch/data
/tests/nightwatch/logs
/tests/nightwatch/pages
/tests/nightwatch/reports
/tests/nightwatch/screenshots
/tests/nightwatch/walkthroughs
/tests/nightwatch/walkthroughs/critical_path.js
/tests/nightwatch/globals.json
```

Tests basés sur les données

Nightwatch accepte un second fichier de configuration `globals.json` qui injecte des données dans le programme de test lui-même, très similaire à la manière dont `Meteor.settings` rend les données de la ligne de commande disponibles dans toute l'application.

globals.json

```
{
  "default" : {
    "url" : "http://localhost:3000",
    "user": {
      "name": "Jane Doe",
      "username" : "janedoe",
      "password" : "janedoe123",
      "email" : "janedoe@test.org",
      "userId": null
    }
  },
  "circle" : {
    "url" : "http://localhost:3000",
    "user": {
      "name": "Jane Doe",
      "username" : "janedoe",
      "password" : "janedoe123",
      "email" : "janedoe@test.org",
      "userId": null
    }
  },
  "galaxy" : {
    "url" : "http://myapp.meteorapp.com",
```



```
"user": {
  "name": "Jane Doe",
  "username" : "janedoe",
  "password" : "janedoe123",
  "email" : "janedoe@test.org"
  "userId": null
}
}
```

Vous pouvez ensuite écrire vos tests qui ne sont pas codés en dur avec des utilisateurs spécifiques, des mots de passe, des entrées de recherche, etc.

```
module.exports = {
  "Login App" : function (client) {
    client
      .url(client.globals.url)
      .login(client.globals.user.email, client.globals.user.password)
      .end();
  }
};
```

Lire [Nightwatch - Configuration et configuration en ligne](https://riptutorial.com/fr/meteor/topic/5901/nightwatch---configuration-et-configuration):

<https://riptutorial.com/fr/meteor/topic/5901/nightwatch---configuration-et-configuration>

Chapitre 34: Node / NPM

Exemples

Version de nœud testée / prise en charge de Meteor

Pour déterminer la dernière version testée / prise en charge de Node pouvant être utilisée avec votre version de Meteor installée, vider la version de nœud directement à partir de l'instance de nœud intégrée à l'outil de génération.

```
meteor node -v
```

Lire Node / NPM en ligne: <https://riptutorial.com/fr/meteor/topic/4599/node---npm>

Chapitre 35: Outils de développement

Exemples

Environnements de développement intégrés

Le développement commence généralement par un éditeur ou un environnement de développement intégré. Les IDE suivants sont connus pour prendre en charge Meteor dans une certaine mesure:

- [Atom](#) - Javascript IDE qui exploite pleinement l'infrastructure javascript isomorphe de Meteor. Si vous voulez pouvoir pirater votre éditeur lui-même, c'est à vous de choisir.
- [Cloud9](#) - La toute dernière offre de développement Cloud qui prend en charge Meteor, avec un tutoriel.
- [MeteorDevTools](#) - Extension Chrome pour Blaze, DDP et Minimongo.
- [Sublime](#) - Éditeur de texte léger et populaire.
- [WebStorm](#) - L'IDE le plus complet actuellement disponible pour Meteor.

Outils de base de données

Une fois que vous avez dépassé votre application «Hello World», vous devez commencer à prêter attention à vos schémas de collection et de document et vous aurez besoin d'outils pour gérer votre base de données.

- [Robomongo](#) - Un favori de longue date de la communauté pour gérer Mongo. Hautement recommandé.
- [JSON Generator](#) ([Générateur JSON](#)) : utilitaire inestimable pour générer des jeux de données échantillons.
- [Page MacOSX Mongo Preference](#) - Préférences GUI pour MacOSX.
- [MongoHub](#) - Une autre interface graphique Mongo, similaire à RoboMongo. MacOSX uniquement.
- [Mongo3](#) - L'un des rares outils de gestion de cluster disponibles. Capable de visualiser les ensembles de réplication. Le seul inconvénient est qu'il est construit en Ruby.
- [Mongo Monitoring Service](#) - Une fois que vous êtes prêt à mettre quelque chose en production, MMS est inestimable. Maintenant connu sous le nom de MongoDB Atlas.
- [Mongo Express](#) - Interface d'administration basée sur le Web MongoDB, écrite avec Node.js et express

Utilitaires de collaboration à distance pour les développeurs distribués

Développer des applications Meteor signifie généralement développer une réactivité multi-clients, ce qui nécessite des outils de collaboration. Les outils suivants se sont avérés populaires au sein de la communauté Meteor.

- [Google Hangouts](#) - Conférence vidéo et chat.

- [Zenhub.io](#) - [Tableaux Kanban](#) pour GitHub.
- [InVision](#) - [Schéma de câblage](#) et prototypage collaboratifs.
- [Meeting Hero](#) - Planification de réunions collaboratives.
- [Hackpad](#) - [Édition collaborative](#) de documents.
- [Slack](#) - Flux de suivi de projet collaboratif.
- [MadEye](#) - Editeur Web collaboratif.
- [Screenhero](#) - Partage d'écran collaboratif.
- [Proto.io](#) - [Fil de fer](#) et prototypage.
- [HuBoard](#) - [Tableaux Kanban](#) pour GitHub.
- [Zapier](#) - Les meilleures applications. Ensemble.
- [Teamwork.com](#) - Gestion de projet traditionnelle et diagrammes de gestion.
- [Sprint.ly](#) - Plus de planches kanban et de planification du sprint qui fonctionnent avec GitHub.
- [LucidChart](#) - Alternative Visio en ligne.
- [Waffle.io](#) - [Alternative Trello / ZenHub](#) qui s'intègre à GitHub.

Clients REST

Si vous souhaitez intégrer Meteor à une API externe, il est probable que cela devienne une interface REST. Nous avons tendance à utiliser les applications Chrome suivantes pour tester les API REST.

- [Facteur](#)
- [DHC Rest Client](#)

Outils en ligne:

- [Hurl.it](#)
- [RequestBin](#)

Débogueurs

La plupart des opérations de débogage ont lieu dans le terminal ou dans les outils de développement Chrome ou Safari, qui sont suffisamment sophistiqués pour 99% de vos besoins. Cependant, si vous souhaitez déboguer sur Firefox ou utiliser des fonctionnalités supplémentaires de débogage du serveur, vous pouvez utiliser quelques utilitaires supplémentaires.

- [Firefox - Firebug](#)
- [Node-Inspector](#)
- [Meteor Toys](#) ou ajouter directement `meteor add meteortoys:allthings`

Mobile Coding sur iOS

[Texttastic Code Editor](#) - [Éditeur de code](#) avec mise en évidence de la syntaxe pour les appareils iOS.

[Working Copy](#) - Clonez les référentiels Github sur votre iPad et codez-les lors de vos déplacements.

[CodeHub](#) - Parcourez et [gérez](#) vos référentiels GitHub. Outil de gestion.

[iOctocat](#) - Utilitaire social pour suivre des projets Github.

[iMockups pour iPad](#) - Wireframes et maquettes. Prend en charge les wireframes pour ordinateurs de bureau et mobiles.

[Blueprint](#) - iOS wireframing et maquettes. Principalement pour le développement iOS, mais quelque peu utilisable pour les applications Web.

[JSON Designer](#) - Conception de l'architecture de données et du schéma de données.

Lire Outils de développement en ligne: <https://riptutorial.com/fr/meteor/topic/4200/outils-de-developpement>

Chapitre 36: Publication d'une piste de publication

Remarques

Publier une version Release Track est en fait assez simple si vous comprenez a) que la commande `publish-release` requiert un fichier `.json` comme paramètre et b) à quoi ressemble ce fichier. C'est sans aucun doute l'obstacle le plus important pour démarrer, car il n'est pratiquement pas documenté.

Gardez à l'esprit que chaque paquet dans la version doit être publié et sur Atmosphere. Le fichier `.meteor / versions` d'une application est particulièrement adapté pour trouver tous les packages et versions nécessaires à la publication.

Après cela, il s'agit de déterminer ce que vous êtes prêt à prendre en charge, ce que vous voulez inclure, etc. Voici un diagramme de Venn partiel sur la manière dont fonctionne actuellement la version clinique. et devrait vous donner une idée générale de la façon dont nous procédons au processus de prise de décision de ce qui est inclus.

Pour plus de discussion, voir le sujet sur les forums Meteor:

<https://forums.meteor.com/t/custom-meteor-release/13736/6>

Exemples

Utilisation de base

L'idée est qu'un responsable de la distribution souhaite exécuter quelque chose comme la commande suivante:

```
meteor publish-release clinical.meteor.rc6.json
```

Ce qui permettra ensuite aux utilisateurs de la distribution d'exécuter ceci:

```
meteor run --release clinical:METEOR@1.1.3-rc6
```

Manifeste manifeste

Un manifeste de version est similaire à un fichier `package.json` NPM, dans la mesure où sa principale préoccupation est de spécifier une liste de packages Atmosphère et de fournir un peu de métadonnées sur cette liste de packages. Le format de base ressemble à ceci:

```
{
  "track": "distraname:METEOR",
  "version": "x.y.z",
}
```

```
"recommended": false,
"tool": "distraname:meteor-tool@x.y.z",
"description": "Description of the Distro",
"packages": {
  "accounts-base": "1.2.0",
  "accounts-password": "1.1.1",
  ...
}
}
```

Personnalisation de l'outil Meteor

Si vous devez étendre l'outil meteor ou la ligne de commande, vous devrez créer et publier votre propre package meteor-tool. La documentation de Ronen est la meilleure pour ce processus:

<http://practicalmeteor.com/using-meteor-publish-release-to-extend-the-meteor-command-line-tool/1>

Il est facile de faire fonctionner une commande meteor helloworld, mais après cela, j'ai senti qu'il était plus facile de créer une application de nœud séparée pour tester les commandes. Quelle est la manière dont StarryNight a été créée. Il s'agit en quelque sorte d'un terrain de jeu et d'un bloc-notes pour les commandes avant d'essayer de les intégrer dans une version de l'outil météorite.

Extraire un manifeste de version de .meteor / versions

StarryNight contient un petit utilitaire qui analyse le fichier `.meteor/versions` une application et le convertit en un manifeste de version.

```
npm install -g starrynight
cd myapp
starrynight generate-release-json
```

Si vous ne souhaitez pas utiliser StarryNight, copiez simplement le contenu de votre fichier `.meteor/versions` dans le champ des `packages` de votre fichier manifeste. Assurez-vous de convertir en syntaxe JSON et d'ajouter des deux-points et des guillemets.

Affichage du manifeste de version pour une version spécifique

```
meteor show --ejson METEOR@1.2.1
```

Publication d'un communiqué de presse

```
meteor publish-release --from-checkout
```

Récupération des derniers commits pour chaque package dans une version

Lors de la construction d'une piste de publication personnalisée, il est courant de conserver les paquets dans le répertoire `/packages` tant que sous-modules git. La commande suivante vous

permet de récupérer tous les derniers commits pour les sous-modules de votre répertoire /packages en même temps.

```
git submodule foreach git pull origin master
```

Lire [Publication d'une piste de publication en ligne](https://riptutorial.com/fr/meteor/topic/4201/publication-d-une-piste-de-publication):

<https://riptutorial.com/fr/meteor/topic/4201/publication-d-une-piste-de-publication>

Chapitre 37: Publication de données

Remarques

Dans le sous-système de données de Meteor, une publication de serveur et ses abonnements client correspondants sont les principaux mécanismes de transport de données réactif et en direct, où les données sous-jacentes sont constamment synchronisées entre le serveur et le client.

Exemples

Abonnement de base et publication

Tout d'abord, supprimez la `autopublish` publication automatique publie automatiquement la base de données entière sur le côté client, de sorte que les effets des publications et des abonnements ne sont pas visibles.

Pour supprimer la `autopublish` :

```
$ meteor remove autopublish
```

Ensuite, vous pouvez créer des publications. Voici un exemple complet.

```
import { Mongo } from 'meteor/mongo';
import { Meteor } from 'meteor/meteor';

const Todos = new Mongo.Collection('todos');

const TODOS = [
  { title: 'Create documentation' },
  { title: 'Submit to Stack Overflow' }
];

if (Meteor.isServer) {
  Meteor.startup(function () {
    TODOS.forEach(todo => {
      Todos.upsert(
        { title: todo.title },
        { $setOnInsert: todo }
      );
    });
  });

  // first parameter is a name.
  Meteor.publish('todos', function () {
    return Todos.find();
  });
}

if (Meteor.isClient) {
  // subscribe by name to the publication.
  Meteor.startup(function () {
```

```
Meteor.subscribe('todos');
})
}
```

Publications mondiales

Une publication globale ne possède pas de nom et ne nécessite pas d'abonnement du client connecté et est donc disponible pour le client connecté dès que le client se connecte au serveur.

Pour ce faire, on nomme simplement la publication comme `null` comme ça

```
Meteor.publish(null, function() {
  return SomeCollection.find();
})
```

Publications nommées

Une publication nommée est une publication qui possède un nom et doit être explicitement abonnée au client.

Considérez ce code côté serveur:

```
Meteor.publish('somePublication', function() {
  return SomeCollection.find()
})
```

Le client doit le demander par:

```
Meteor.subscribe('somePublication')
```

Abonnements à des modèles

Le système de template par défaut de Meteor Spacebars et son sous-système de rendu sous-jacent Blaze s'intègrent parfaitement aux méthodes du cycle de vie de la publication.

Pour tirer parti de cela, il faut s'inscrire sur l'instance du modèle, plutôt que sur le symbole `Meteor` comme suit:

Commencez par configurer le modèle

```
<template name="myTemplate">
  We will use some data from a publication here
</template>
```

Puis appuyez sur le rappel de cycle de vie correspondant

```
Template.myTemplate.onCreated(function() {
  const templateInstance = this;
  templateInstance.subscribe('somePublication')
```

```
)
```

Maintenant, lorsque ce modèle est détruit, la publication sera également automatiquement arrêtée.

Remarque: Les données auxquelles vous êtes abonné seront disponibles pour tous les modèles.

Publier dans une collection nommée éphémère côté client.

Car si vous devez affiner ce qui est publié.

```
import { Mongo } from 'meteor/mongo';
import { Meteor } from 'meteor/meteor';
import { Random } from 'meteor/random';

if (Meteor.isClient) {
  // established this collection on the client only.
  // a name is required (first parameter) and this is not persisted on the server.
  const Messages = new Mongo.Collection('messages');
  Meteor.startup(function () {
    Meteor.subscribe('messages');
    Messages.find().observe({
      added: function (message) {
        console.log('Received a new message at ' + message.timestamp);
      }
    });
  })
}

if (Meteor.isServer) {
  // this will add a new message every 5 seconds.
  Meteor.publish('messages', function () {
    const interval = Meteor.setInterval(() => {
      this.added('messages', Random.id(), {
        message: '5 seconds have passed',
        timestamp: new Date()
      })
    }, 5000);
    this.added('messages', Random.id(), {
      message: 'First message',
      timestamp: new Date()
    });
    this.onStop(() => Meteor.clearInterval(interval));
  });
}
```

Créer et répondre à une erreur sur une publication.

Sur le serveur, vous pouvez créer une publication comme celle-ci. `this.userId` est l'ID de l'utilisateur actuellement connecté. Si aucun utilisateur n'est connecté, vous pouvez générer une erreur et y répondre.

```
import Secrets from '/imports/collections/Secrets';

Meteor.publish('protected_data', function () {
```

```

if (!this.userId) {
  this.error(new Meteor.Error(403, "Not Logged In.));
  this.ready();
} else {
  return Secrets.find();
}
});

```

Sur le client, vous pouvez répondre avec les éléments suivants.

```

Meteor.subscribe('protected_data', {
  onError(err) {
    if (err.error === 403) {
      alert("Looks like you're not logged in");
    }
  },
});

```

Fichier / imports / collections / Secrets crée une référence à la collection de secrets comme ci-dessous:

```

const Secrets = new Mongo.Collection('secrets');

```

Réinscription réactive à une publication

Un modèle autorun peut être utilisé pour (ré) souscrire à une publication. Il établit un contexte réactif qui est ré-exécuté chaque fois que *des données réactives dépendent de changements*. De plus, une exécution automatique s'exécute toujours une fois (la première fois qu'elle est exécutée).

Les autoruns de modèle sont normalement placés dans une méthode `onCreated`.

```

Template.myTemplate.onCreated(function() {
  this.parameter = new ReactiveVar();
  this.autorun(() => {
    this.subscribe('myPublication', this.parameter.get());
  });
});

```

Cela fonctionnera une fois (la première fois) et mettre en place un abonnement. Il sera alors ré-exécuté chaque fois que la variable réactive du `parameter` change.

Attendez dans la vue Blaze pendant l'extraction des données publiées

Code JS du modèle

```

Template.templateName.onCreated(function() {
  this.subscribe('subscription1');
  this.subscribe('subscription2');
});

```

Code HTML du modèle

```
<template name="templateName">
  {{#if Template.subscriptionsReady }}
    //your actual view with data. it can be plain HTML or another template
  {{else}}
    //you can use any loader or a simple header
    <h2> Please wait ... </h2>
  {{/if}}
</template>
```

Validation du compte d'utilisateur sur Publier

Il est parfois utile de sécuriser davantage vos publications en exigeant une connexion utilisateur. Voici comment vous y parvenez via Meteor.

```
import { Recipes } from '../imports/api/recipes.js';
import { Meteor } from 'meteor/meteor';

Meteor.publish('recipes', function() {
  if(this.userId) {
    return Recipe.find({});
  } else {
    this.ready(); // or: return [];
  }
});
```

Publier plusieurs curseurs

Plusieurs curseurs de base de données peuvent être publiés à partir de la même méthode de publication en renvoyant un tableau de curseurs.

Les curseurs "enfants" seront traités comme des jointures et ne seront pas réactifs.

```
Meteor.publish('USER_THREAD', function(postId) {
  let userId = this.userId;

  let comments = Comments.find({ userId, postId });
  let replies = Replies.find({ userId, postId });

  return [comments, replies];
});
```

Simuler le retard dans les publications

Dans le monde réel, des délais de connexion et de serveur pourraient survenir, pour simuler des retards dans l'environnement de développement `Meteor._sleepForMs(ms)` ; peut être utilisé

```
Meteor.publish('USER_DATA', function() {
  Meteor._sleepForMs(3000); // Simulate 3 seconds delay
  return Meteor.users.find({});
});
```

Fusion de publications

Les publications peuvent être fusionnées sur le client, ce qui permet de créer des documents de forme différente dans un même curseur. L'exemple suivant montre comment un annuaire d'utilisateurs peut publier une quantité minimale de données publiques pour les utilisateurs d'une application et fournir un profil plus détaillé pour l'utilisateur connecté.

```
// client/subscriptions.js
Meteor.subscribe('usersDirectory');
Meteor.subscribe('userProfile', Meteor.userId());

// server/publications.js
// Publish users directory and user profile

Meteor.publish("usersDirectory", function (userId) {
  return Meteor.users.find({}, {fields: {
    '_id': true,
    'username': true,
    'emails': true,
    'emails[0].address': true,

    // available to everybody
    'profile': true,
    'profile.name': true,
    'profile.avatar': true,
    'profile.role': true
  }});
});

Meteor.publish('userProfile', function (userId) {
  return Meteor.users.find({_id: this.userId}, {fields: {
    '_id': true,
    'username': true,
    'emails': true,
    'emails[0].address': true,

    'profile': true,
    'profile.name': true,
    'profile.avatar': true,
    'profile.role': true,

    // privately accessible items, only available to the user logged in
    'profile.visibility': true,
    'profile.socialsecurity': true,
    'profile.age': true,
    'profile.dateofbirth': true,
    'profile.zip': true,
    'profile.workphone': true,
    'profile.homephone': true,
    'profile.mobilephone': true,
    'profile.applicantType': true
  }});
});
```

Lire Publication de données en ligne: <https://riptutorial.com/fr/meteor/topic/1323/publication-de-donnees>

Chapitre 38: Réactif (Vars et Dictionnaires)

Exemples

Requête Réactive

Exemple de code:

Dans main.html

```
<template name="test">
  <input type="checkbox" id="checkbox1" name="name" value="data">Check Me
  {{showData}}
</template>
```

Dans Main.js

```
var check_status='';
//Reactive Var Initialization
Template.main.onCreate(function (){
  check_status=new ReactiveVar({});

});

Template.main.helpers({
  showData : function(){
    return Collection.find(check_status.get());
  }
});

Template.main.events({
  "change #checkbox1" : function(){
    check_status.set({field: 'data'});
  }
});
```

Explication:

Lorsque nous initialisons le var réactif `check_status` nous définissons la valeur égale à `{}`. Dans l'assistant, au moment du rendu, les mêmes données sont transmises à la requête `Collection.find(check_status.get())` qui *affiche toutes les données*.

Dès que vous cliquez sur la case à cocher, l'événement décrit dans `Template.main.events` est déclenché, ce qui définit la valeur de `check_status` sur `{field: data}`. Comme il s'agit d'une `showData` *réactive*, le modèle `showData` est réexécuté et cette fois, la requête est `Collection.find({field: data})`, alors seuls les champs, où 'data' correspondant aux `field` sont renvoyés.

Vous devrez ajouter le package `reactive var` (`meteor add reactive-var`) avant d'utiliser ces commandes.

Lire Réactif (Vars et Dictionnaires) en ligne: <https://riptutorial.com/fr/meteor/topic/6535/reactif--vars-et-dictionnaires->

Chapitre 39: Recettes de l'interface utilisateur Blaze (Bootstrap; No jQuery)

Remarques

Les exemples de Blaze ci-dessus sont hautement compatibles avec la bibliothèque <http://bootsnipp.com/> , qui fournit uniquement le HTML et le CSS pour les composants et laisse le javascript au développeur. Cela permet aux composants de partager les mêmes méthodes de tri, de filtrage, de requête et de curseur sous-jacentes.

Exemples

Menu déroulant

L'exemple suivant crée un menu déroulant Bootstrap, utilisant uniquement Blaze et aucune JQuery.

Modèle d'objet de document

```
<nav class="nav navbar-nav">
  <li class="dropdown">
    <a href="#" class="dropdown-toggle" data-toggle="dropdown">{{getSelectedValue}} <span
class="glyphicon glyphicon-user pull-right"></span></a>
    <ul class="fullwidth dropdown-menu">
      <li id="firstOption" class="fullwidth"><a href="#">15 Minutes <span class="glyphicon
glyphicon-cog pull-right"></span></a></li>
      <li class="divider"></li>
      <li id="secondOption"><a href="#">30 Minutes <span class="glyphicon glyphicon-stats
pull-right"></span></a></li>
      <li class="divider"></li>
      <li id="thirdOption"><a href="#">1 Hour <span class="badge pull-right"> 42
</span></a></li>
      <li class="divider"></li>
      <li id="fourthOption"><a href="#">4 Hour <span class="glyphicon glyphicon-heart
pull-right"></span></a></li>
      <li class="divider"></li>
      <li id="fifthOption"><a href="#">8 Hours <span class="glyphicon glyphicon-log-out
pull-right"></span></a></li>
    </ul>
  </li>
</nav>
```

Javascript

```
Template.examplePage.helpers({
  getSelectedValue: function() {
    return Session.get('selectedValue');
  }
})
```

```

});
Template.dropDownWidgetName.events({
  'click #firstOption':function(){
    Session.set('selectedValue', 1);
  },
  'click #secondOption':function(){
    Session.set('selectedValue', "blue");
  },
  'click #thirdOption':function(){
    Session.set('selectedValue', $('#thirdOption').innerText);
  },
  'click #fourthOption':function(){
    Session.set('selectedValue', Session.get('otherValue'));
  },
  'click #fifthOption':function(){
    Session.set('selectedValue', Posts.findOne(Session.get('selectedPostId')).title);
  },
});

```

Navbars

Une tâche très courante consiste à créer des barres de navigation réactives et à créer des barres d'action / pied de page qui ont des contrôles différents en fonction de la page sur laquelle se trouve un utilisateur ou du rôle auquel un utilisateur appartient. Revenons sur comment faire ces contrôles.

Routeur

```

Router.configure({
  layoutTemplate: 'appLayout',
});
Router.route('checklistPage', {
  path: '/lists/:_id',
  onBeforeAction: function() {
    Session.set('selectedListId', this.params._id);
    this.next();
  },
  yieldTemplates: {
    'navbarFooter': {
      to: 'footer'
    }
  }
});

```

Créer un modèle de barre de navigation

```

<template name="navbarFooter">
  <nav id="navbarFooterNav" class="navbar navbar-default navbar-fixed-bottom"
  role="navigation">
    <ul class="nav navbar-nav">
      <li><a id="addPostLink"><u>A</u></a></li>
      <li><a id="editPostLink"><u>E</u></a></li>
      <li><a id="deletePostLink"><u>D</u></a></li>
    </ul>
    <ul class="nav navbar-nav navbar-right">
      <li><a id="helpLink"><u>H</u></a></li>

```

```
</ul>
</nav>
</template>
```

Définir les rendements dans la mise en page

```
<template name="appLayout">
  <div id="appLayout">
    <header id="navbarHeader">
      {{> yield 'header'}}
    </header>

    <div id="mainPanel">
      {{> yield}}
    </div>

    <footer id="navbarFooter" class="{{getTheme}}">
      {{> yield 'footerActionElements' }}
    </footer>
  </div>
</template>
```

Modals

Cette approche est une approche pure-blaze pour basculer des éléments d'interface utilisateur dans et hors de l'existence. Considérez ceci comme un remplacement des dialogues modaux. En fait, il existe plusieurs manières d'implémenter des boîtes de dialogue modales en utilisant cette méthode (ajoutez simplement des masques et des animations d'arrière-plan).

Modèle d'objet de document

```
<template name="topicsPage">
  <div id="topicsPage" class="container">
    <div class="panel">
      <div class="panel-heading">
        Nifty Panel
      </div>
      <!-- .... -->
      <div class="panel-footer">
        <!-- step 1. we click on the button object -->
        <div id="createTopicButton" class="btn {{ getPreferredButtonTheme }}">Create
Topic</div>
      </div>
    </div>

    <!-- step 5 - the handlebars gets activated by the javascript controller -->
    <!-- and toggle the creation of new objects in our model -->
    {{#if creatingNewTopic }}
    <div>
      <label for="topicTextInput"></label>
      <input id="topicTextInput" value="enter some text..."></input>
      <button class="btn btn-warning">Cancel</button>
      <button class="btn btn-success">OK</button>
    </div>
    {{/if}}
  </div>
```

```
</template>
```

Javascript

```
// step 2 - the button object triggers an event in the controller
// which toggles our reactive session variable
Template.topicsPage.events({
  'click #createTopicButton':function(){
    if(Session.get('is_creating_new_topic'){
      Session.set('is_creating_new_topic', false);
    }else{
      Session.set('is_creating_new_topic', true);
    }
  }
});

// step 0 - the reactive session variable is set false
Session.setDefault('is_creating_new_topic', false);

// step 4 - the reactive session variable invalidates
// causing the creatNewTopic function to be rerun
Template.topicsPage.creatingNewTopic = function(){
  if(Session.get('is_creating_new_topic')){
    return true;
  }else{
    return false;
  }
}
```

Le marquage

La couche de base de données Tout d'abord, nous voulons configurer le protocole de distribution de données pour nous assurer que nous pouvons conserver les données dans la base de données et les transmettre au client. Trois fichiers doivent être créés ... un sur le serveur, un sur le client et un partagé entre les deux.

```
// client/subscriptions.js
Meteor.subscribe('posts');

//lib/model.js
Posts = new Meteor.Collection("posts");
Posts.allow({
  insert: function(){
    return true;
  },
  update: function () {
    return true;
  },
  remove: function(){
    return true;
  }
});

// server.publications.js
Meteor.publish('posts', function () {
  return Posts.find();
});
```

```
});
```

Cet exemple suppose le schéma de document suivant pour le modèle de marquage:

```
{
  _id: "3xHCsDexdPHN6vt7P",
  title: "Sample Title",
  text: "Lorem ipsum, solar et...",
  tags: ["foo", "bar", "zkrk", "squee"]
}
```

Modèle d'objet de document

Deuxièmement, nous voulons créer notre modèle d'objet dans la couche application. Voici comment vous utiliseriez un panneau Bootstrap pour rendre une publication avec titre, texte et balises. Notez que `selectedPost`, `tagObjects` et `tag` sont tous des fonctions d'assistance du modèle `blogPost`. `title` et le `text` sont des champs de notre document.

```
<template name="blogPost">
  {{#with selectedPost }}
    <div class="blogPost panel panel-default">
      <div class="panel-heading">
        {{ title }}
      </div>
      {{ text }}
      <div class="panel-footer">
        <ul class="horizontal-tags">
          {{#each tagObjects }}
            <li class="tag removable_tag">
              <div class="name">{{tag}}<i class="fa fa-times"></i></div>
            </li>
          {{/each}}
          <li class="tag edittag">
            <input type="text" id="edittag-input" value="" /><i class="fa fa-plus"></i>
          </li>
        </ul>
      </div>
    </div>
  {{/with}}
</template>
```

Javascript

Ensuite, nous voulons configurer certains contrôleurs pour renvoyer des données, implémenter des entrées de données, etc.

```
// you will need to set the selectedPostId session variable
// somewhere else in your application
Template.blogPost.selectedPost = function(){
  return Posts.findOne({_id: Session.get('selectedPostId')});
}

// next, we use the _.map() function to read the array from our record
// and convert it into an array of objects that Handlebars/Spacebars can parse
Template.blogPost.tagObjects = function () {
  var post_id = this._id;
  return _.map(this.tags || [], function (tag) {
```

```

        return {post_id: post_id, tag: tag};
    });
};

// then we wire up click events
Template.blogPost.events({
  'click .fa-plus': function (evt, tmpl) {
    Posts.update(this._id, {$addToSet: {tags: value}});
  },
  'click .fa-times': function (evt) {
    Posts.update({_id: this._id}, {$pull: {tags: this.tag}});
  }
});

```

Coiffant

Enfin, nous souhaitons définir des vues différentes pour les téléphones, les tablettes et les ordinateurs de bureau. et certains styles d'interface utilisateur de base en fonction de la saisie de l'utilisateur. Cet exemple utilise le précompilateur Less, bien que la syntaxe soit à peu près la même pour Sass et Stylus.

```

// default desktop view
.fas-plus: hover {
  cursor: pointer;
}
.fas-times: hover {
  cursor: pointer;
}
// landscape orientation view for tablets
@media only screen and (min-width: 768px) {
  .blogPost {
    padding: 20px;
  }
}
// portrait orientation view for tablets
@media only screen and (max-width: 768px) {
  .blogPost {
    padding: 0px;
    border: 0px;
  }
}
// phone view
@media only screen and (max-width: 480px) {
  blogPost {
    .panel-footer {
      display: none;
    }
  }
}
}

```

Alertes et erreurs

Les alertes et les erreurs sont presque les plus simples de tous les modèles de composants Meteor. En fait, ils sont si simples qu'ils sont à peine enregistrés en tant que modèles d'eux-mêmes. Au lieu d'ajouter des modules ou des modèles FlashAlert, il vous suffit de donner un style approprié au modèle de guidon, d'ajouter une aide et de le connecter à une variable de session réactive.

Conditions préalables

Le code suivant nécessite respectivement le précompilateur LESS et Bootstrap-3. Vous devrez exécuter les commandes suivantes à l'invite de commande pour les faire fonctionner.

```
meteor add less
meteor add ian:bootstrap-3
```

Modèle d'objet de document: Définir un objet d'alerte Commencez par ajouter des éléments à votre modèle d'objet de document. Dans ce cas, nous voulons créer un élément div pour notre alerte, qui est connecté à deux aides au guidon.

```
<template name="postsPage">
  <div id="postsPage" class="page">
    <div id="postsPageAlert" class="{{alertColor}}">{{alertMessage}}</div>
    <div class="postsList">
      <!-- other code you can ignore in this example -->
    </div>
    <div id="triggerAlertButton" class="btn btn-default">
  </div>
</template>
```

Javascript: Définir les aides de modèle Ensuite, nous voulons câbler certains contrôleurs qui rempliront le modèle d'objet avec des données. Nous le faisons avec deux variables de session réactives et deux aides au guidon.

```
Session.setDefault('alertLevel', false);
Session.setDefault('alertMessage', "");

Template.postsPage.alertColor = function(){
  if(Session.get('alertLevel') == "Success"){
    return "alert alert-success";
  }else if(Session.get('alertLevel') == "Info"){
    return "alert alert-info";
  }else if(Session.get('alertLevel') == "Warning"){
    return "alert alert-warning";
  }else if(Session.get('alertLevel') == "Danger"){
    return "alert alert-danger";
  }else{
    return "alert alert-hidden"
  }
}

Template.postsPage.alertMessage = function(){
  return Session.get('alertMessage');
}
```

Styling: Définir la visibilité du DOM Ensuite, nous voulons revenir à notre CSS et définir deux vues de l'élément postsPage. Dans la première vue, nous affichons tout le contenu de notre modèle d'objet. Dans la seconde vue, seule une partie du contenu de notre modèle d'objet est affichée.

```
#postsPage{
  .alert{
    display: block;
```

```
}
.alert-hidden{
  display: none;
}
}
```

Javascript: déclencher l'alerte

Enfin, nous revenons à nos contrôleurs, et nous définissons un contrôleur d'événements, qui déclenchera notre alerte lorsque vous cliquerez dessus.

```
Template.postsPage.events({
  'click #triggerAlertButton':function(){
    Session.set('alertLevel', 'Success');
    Session.set('alertMessage', 'You successfully read this important alert message.');
```

Et c'est tout ce qu'il y a à faire! Super simple, non? Vous pouvez maintenant définir les variables de session `alertLevel` et `alertMessage` n'importe où dans votre base de code et votre application affichera de manière réactive les alertes et les messages d'erreur! :)

Flux de travail par onglets

Modèle d'objet de document

Commencez par créer vos onglets et volets dans votre modèle d'objet ...

```
<template name="samplePage">
  <div id="samplePage" class="page">
    <ul class="nav nav-tabs">
      <li id="firstPanelTab"><a href="#firstPanel">First</a></li>
      <li id="secondPanelTab"><a href="#secondPanel">Second</a></li>
    </ul>

    <div id="firstPanel" class="{{firstPanelVisibility}}">
      {{> firstPanel }}
    </div>
    <div id="secondPanel" class="{{secondPanelVisibility}}">
      {{> secondPanel }}
    </div>
  </div>
</template>
```

Javascript

```
// this variable controls which tab is displayed and associated application state
Session.setDefault('selectedPanel', 1);

Template.name.helpers({
  firstPanelVisibility: function (){
    if(Session.get('selectedPanel') === 1){
      return "visible";
    }else{
      return "hidden";
    }
  }
});
```



```

},
secondPanelVisibility: function (){
  if(Session.get('selectedPanel') === 2){
    return "visible";
  }else{
    return "hidden";
  }
},
thirdPanelVisibility: function (){
  if(Session.get('selectedPanel') === 3){
    return "visible";
  }else{
    return "hidden";
  }
},
firstPanelActive: function (){
  if(Session.get('selectedPanel') === 1){
    return "active panel-tab";
  }else{
    return "panel-tab";
  }
},
secondPanelActive: function (){
  if(Session.get('selectedPanel') === 2){
    return "active panel-tab";
  }else{
    return "panel-tab";
  }
},
thirdPanelActive: function (){
  if(Session.get('selectedPanel') === 3){
    return "active panel-tab";
  }else{
    return "panel-tab";
  }
}
});

```

Coiffant

```

.visible {
  display: block;
  visibility: visible;
}
.hidden {
  display: none;
  visibility: hidden;
}

```

Onglet actif Pour plus d'effet, vous pouvez étendre ce modèle en injectant des classes pour indiquer l'onglet actif.

```

<li id="firstPanelTab" class="{{firstPanelActive}}"><a href="#firstPanel">First</a></li>
<li id="secondPanelTab" class="{{secondPanelActive}}"><a href="#secondPanel">Second</a></li>

```

```

Template.firstPanel.helpers({
  firstPanelActive: function (){
    if(Session.get('selectedPanel') === 1){

```

```
        return "active";
    }else{
        return "";
    }
},
secondPanelActive: function (){
    if(Session.get('selectedPanel') === 2){
        return "active";
    }else{
        return "";
    }
},
});
```

Lire Recettes de l'interface utilisateur Blaze (Bootstrap; No jQuery) en ligne:

<https://riptutorial.com/fr/meteor/topic/4202/recettes-de-l-interface-utilisateur-blaze--bootstrap--no-jquery->

Chapitre 40: Récupération des données d'un Meteor.call

Exemples

Les bases de Meteor.call

```
Meteor.call(name, [arg1, arg2...], [asyncCallback])
```

- (1) nom chaîne
- (2) Nom de la méthode à appeler
- (3) arg1, arg2 ... objet pouvant être EJSON [facultatif]
- (4) Fonction asyncCallback [Facultatif]

D'une part, vous pouvez faire: (via la **variable Session** , ou via **ReactiveVar**)

```
var syncCall = Meteor.call("mymethod") // Sync call
```

Cela signifie que si vous faites quelque chose comme ça, côté serveur, vous ferez:

```
Meteor.methods({
  mymethod: function() {
    let asyncToSync = Meteor.wrapAsync(asynchronousCall);
    // do something with the result;
    return asyncToSync;
  }
});
```

Par contre, vous voudrez parfois le conserver via le résultat du rappel?

Côté client :

```
Meteor.call("mymethod", argumentObjectorString, function (error, result) {
  if (error) Session.set("result", error);
  else Session.set("result", result);
})
Session.get("result") -> will contain the result or the error;

//Session variable come with a tracker that trigger whenever a new value is set to the session
variable. \ same behavior using ReactiveVar
```

Du côté serveur

```
Meteor.methods({
  mymethod: function(ObjectorString) {
    if (true) {
      return true;
    } else {
```

```
        throw new Meteor.Error("TitleOfError", "ReasonAndMessageOfError"); // This will
and up in the error parameter of the Meteor.call
    }
}
});
```

Le but ici est de montrer que Meteor propose différentes manières de communiquer entre le client et le serveur.

Utilisation de la variable de session

Du côté serveur

```
Meteor.methods({
  getData() {
    return 'Hello, world!';
  }
});
```

Côté client

```
<template name="someData">
  {{#if someData}}
  <p>{{someData}}</p>
  {{else}}
  <p>Loading...</p>
  {{/if}}
</template>
```

```
Template.someData.onCreated(function() {
  Meteor.call('getData', function(err, res) {
    Session.set('someData', res);
  });
});

Template.someData.helpers({
  someData: function() {
    return Session.get('someData');
  }
});
```

Utiliser ReactiveVar

Du côté serveur

```
Meteor.methods({
  getData() {
    return 'Hello, world!';
  }
});
```

Côté client

```
<template name="someData">
  {{#if someData}}
    <p>{{someData}}</p>
  {{else}}
    <p>Loading...</p>
  {{/if}}
</template>
```

```
Template.someData.onCreated(function() {

  this.someData = new ReactiveVar();

  Meteor.call('getData', (err, res) => {
    this.someData.set(res);
  });
});

Template.someData.helpers({
  someData: function() {
    return Template.instance().someData.get();
  }
});
```

package `reactive-var` requis. Pour l'ajouter, lancez `meteor add reactive-var`.

Lire Récupération des données d'un Meteor.call en ligne:

<https://riptutorial.com/fr/meteor/topic/3068/recuperation-des-donnees-d-un-meteor-call>

Chapitre 41: Structure du répertoire

Introduction

Avant la publication de Meteor 1.3, les développeurs de Meteor étaient frustrés par la gestion des dépendances de fichiers et des variables globales par Meteor.js. En réponse, Meteor a établi de nouvelles normes pour les structures de projet afin de simplifier le système de dépendance du projet. Ce sujet explique la structure de projet standardisée et les principes sous-jacents.

Remarques

client

Tout le code du répertoire client est exécuté uniquement du côté client ou du navigateur Web.

client / compatibilité

Le répertoire de compatibilité contient du code hérité ou tiers, tel que les bibliothèques jQuery, etc.

lib

Le répertoire lib est chargé avant les autres répertoires de votre projet Meteor et est chargé à la fois sur le serveur et sur le client. C'est l'endroit préféré pour définir des modèles de données, des bibliothèques isomorphes et une logique métier.

importations

Le répertoire imports est un répertoire disponible sur le serveur, à la fois pour le serveur et pour le client, mais uniquement avant l'expédition de l'ensemble client au client.

paquets

Le répertoire des packages est l'endroit où les packages personnalisés sont stockés lors du développement local. Lorsque vous utilisez la commande standard `meteor add package:name` pour ajouter un paquet, Meteor va d'abord regarder dans ce répertoire si un paquet local a le nom de description correspondant dans son fichier `package.js`. Si ce n'est pas le cas, il interrogera l'atmosphère comme d'habitude.

privé

Le répertoire privé contient des fichiers statiques qui ne doivent être disponibles que sur le serveur Web.

Publique

Le répertoire public contient des fichiers statiques uniquement disponibles sur le client d'application. Cela peut inclure des actifs de marque, etc.

serveur

Le répertoire du serveur contient des ressources côté serveur. Cela peut inclure la logique d'authentification, les méthodes et d'autres codes pouvant nécessiter des considérations de sécurité.

des tests

Le répertoire de tests est omis par défaut lorsque votre application est regroupée et déployée.

Comme suggéré [par Richard Silverton](#), il est judicieux de placer non seulement le répertoire du projet meteor sous contrôle de version, mais aussi son répertoire parent.

De cette façon, vous pouvez garder les fichiers sous contrôle de version sans avoir la météorite pour les gérer.

Exemples

Structures de répertoire classiques

La première chose que vous devez savoir lors de la structuration de vos applications est que l'outil Meteor contient des répertoires codés en dur avec une logique spécifique. À un niveau très basique, les répertoires suivants sont "intégrés" dans le bundle Meteor.

```
client/                # client application code
client/compatibility/ # legacy 3rd party javascript libraries
imports/              # for lazy loading feature
lib/                  # any common code for client/server.
packages/             # place for all your atmosphere packages
private/              # static files that only the server knows about
public/               # static files that are available to the client
server/               # server code
tests/                # unit test files (won't be loaded on client or
server)
```

Page de référence: [Meteor Guide](#)> [Répertoires spéciaux](#)

Structure de répertoire uniquement pour les packages

De nombreuses personnes se retrouvent finalement à prendre en charge plusieurs applications et souhaitent partager du code entre les applications. Cela conduit au concept d'architecture de microservices et d'applications tout-en-un. Essentiellement, le code de l'ensemble de la structure de répertoire classique est transformé en packages.

Même s'il n'y a pas de logique codée en dur pour les répertoires dans les packages, nous trouvons qu'il est recommandé d'utiliser la structure de répertoires classique lors de la création de packages. Cela crée un chemin de refactorisation naturel car les entités sont prototypées dans l'application, puis extraites dans des packages à publier et à partager. Les noms de répertoire sont partagés, il y a donc moins de confusion parmi les membres de l'équipe.

```
client/                # client application code
packages/              # place for all your atmosphere packages
packages/foo/client    # client application code
packages/foo/lib       # any common code for client/server
packages/foo/server    # server code
packages/foo/tests     # tests
server/                # server code
```

Structure du répertoire des importations / modules

Les versions les plus récentes de Meteor sont compatibles avec `ecmascript`, alias ES6 ou ES2015. Au lieu de paquets, Javascript prend désormais en charge `import` instructions et les modules d' `import`, ce qui remplace le besoin d'applications uniquement par package. La structure de répertoire la plus récente est similaire à celle du package, mais utilise le répertoire `/imports` au lieu de `/packages`.

```
imports                                     #
imports/api                               # isomorphic methods
imports/lib                               # any common code for client/server
imports/client                            # client application code
imports/server                             # server code
```

Structure de répertoire en mode mixte

Et, bien sûr, vous pouvez combiner ces approches et utiliser les packages et les importations parallèlement au code spécifique à votre application. Une structure en mode de mixage est la plus courante dans trois situations: une application franken, qui consiste simplement à tirer un peu de l'extérieur sans stratégie globale; une application activement restructurée, des structures Classic ou Package-Only à la structure Imports / Modules.

```
client/                                   # client application code
client/compatibility/                    # legacy 3rd party javascript libraries
imports                                  #
imports/api                              # isomorphic methods
imports/lib                              # any common code for client/server
imports/client                           # client application code
imports/server                            # server code
lib/                                     # any common code for client/server.
packages/                                # place for all your atmosphere packages
packages/foo/client                       # client application code
packages/foo/lib                           # any common code for client/server
packages/foo/server                       # server code
packages/foo/tests                        # tests
private/                                  # static files that only the server knows about
public/                                   # static files that are available to the client
server/                                   # server code
tests/                                    # unit test files (won't be loaded on client or
server)
```

Ordre de chargement du répertoire

Les fichiers de modèles HTML sont toujours chargés avant tout le reste

Fichiers commençant par *main*. sont chargés en dernier

Les fichiers à l'intérieur d'un répertoire `lib /` sont chargés ensuite

Les fichiers avec des chemins plus profonds sont chargés ensuite

Les fichiers sont ensuite chargés par ordre alphabétique du chemin complet

[Lien de référence](#)

Page de référence: [Meteor Guide](#)> [Structure de l'application](#)> [Ordre de chargement des fichiers par défaut](#)

Lire [Structure du répertoire en ligne](#): <https://riptutorial.com/fr/meteor/topic/3072/structure-du-repertoire>

Chapitre 42: Tâches d'arrière-plan

Remarques

Le package **cron-tick** est un package très simple pour les tâches en arrière-plan, mais il ne prend pas en charge plusieurs processus. Si vous exécutez votre application dans plusieurs processus (ou conteneurs), utilisez plutôt **percolate: synced-cron**.

Exemples

Cron simple

Utilisez le paquet **percolate: synced-cron**

Définir un travail:

```
SyncedCron.add({
  name: 'Find new matches for a saved user filter and send alerts',
  schedule: function(parser) {
    // parser is a later.parse object
    return parser.text('every 10 minutes');
  },
  job: function() {
    user.alerts.map(a => a.findMatchesAndAlert());
  }
});
```

Démarrer vos travaux définis:

```
SyncedCron.start();
```

Il prend en charge la synchronisation des tâches entre plusieurs processus, comme Galaxy avec plus d'un conteneur.

Lire Tâches d'arrière-plan en ligne: <https://riptutorial.com/fr/meteor/topic/4772/taches-d-arriere-plan>

Chapitre 43: Téléchargement de fichier

Remarques

Le package CollectionFS a été mis en attente et abandonné par son auteur; cependant, comme il n'y a pas d'autre package dans Atmosphere ou l'écosystème Meteor pour utiliser la fonctionnalité GridFS de Mongo, et que le code fonctionne toujours parfaitement bien; Nous vous recommandons de ne pas supprimer l'exemple de la documentation StackOverflow tant qu'une autre solution GridFS ne peut pas être documentée pour son remplacement.

Recherche supplémentaire

[Téléchargements de Filepicker.io et conversion d'image](#)

[Fichier de sauvegarde de Dario](#)

[Modèle de téléchargement de fichiers de Micha Roon](#)

[Package de téléchargement de fichier EventedMind](#)

Exemples

Serveur / Client

Le téléchargement de fichiers peut être facile ou très compliqué, selon ce que vous voulez faire. En général, transférer un fichier n'est pas si difficile. Mais il y a beaucoup de cas limites autour des pièces jointes, des fichiers binaires, etc. Et le véritable point d'ancrage est la mise à l'échelle horizontale et la création d'une solution qui fonctionne lorsque le serveur est cloné une seconde, troisième et nième fois.

Commençons par un modèle de téléchargement serveur / client de base. Nous commençons par ajouter un élément d'entrée de fichier au modèle d'objet du document.

```
<template name="example">
  <input type=file />
</template>
```

Ensuite, attachez un événement à l'élément d'entrée de votre contrôleur et appelez une méthode Meteor locale «startFileTransfer» pour lancer le transfert.

```
// client/example.js
Template.example.events({
  'change input': function(ev) {
    _.each(ev.srcElement.files, function(file) {
      Meteor.startFileTransfer(file, file.name);
    });
  }
});

// client/save.js
/**
 * @blob (https://developer.mozilla.org/en-US/docs/DOM/Blob)
```

```

* @name the file's name
* @type the file's type: binary, text (https://developer.mozilla.org/en-US/docs/DOM/FileReader#Methods)
*
* TODO Support other encodings: https://developer.mozilla.org/en-US/docs/DOM/FileReader#Methods
* ArrayBuffer / DataURL (base64)
*/
Meteor.startFileTransfer = function(blob, name, path, type, callback) {
  var fileReader = new FileReader(),
      method, encoding = 'binary', type = type || 'binary';
  switch (type) {
    case 'text':
      // TODO Is this needed? If we're uploading content from file, yes, but if it's from an
      // input/textarea I think not...
      method = 'readAsText';
      encoding = 'utf8';
      break;
    case 'binary':
      method = 'readAsBinaryString';
      encoding = 'binary';
      break;
    default:
      method = 'readAsBinaryString';
      encoding = 'binary';
      break;
  }
  fileReader.onload = function(file) {
    Meteor.call('saveFileToDisk', file.srcElement.result, name, path, encoding, callback);
  }
  fileReader[method](blob);
}

```

Le client appelle alors la méthode du serveur `saveFileToDisk`, qui effectue le transfert proprement dit et met tout sur le disque.

```

//
/**
* TODO support other encodings:
* http://stackoverflow.com/questions/7329128/how-to-write-binary-data-to-a-file-using-node-js
*/
Meteor.methods({
  saveFileToDisk: function(blob, name, path, encoding) {
    var path = cleanPath(path), fs = __meteor_bootstrap__.require('fs'),
        name = cleanName(name || 'file'), encoding = encoding || 'binary',
        chroot = Meteor.chroot || 'public';
    // Clean up the path. Remove any initial and final '/' -we prefix them-,
    // any sort of attempt to go to the parent directory '..' and any empty directories in
    // between '/////' - which may happen after removing '..'
    path = chroot + (path ? '/' + path + '/' : '/');

    // TODO Add file existence checks, etc...
    fs.writeFile(path + name, blob, encoding, function(err) {
      if (err) {
        throw (new Meteor.Error(500, 'Failed to save file.', err));
      } else {
        console.log('The file ' + name + ' (' + encoding + ') was saved to ' + path);
      }
    });
  }
});

```

```

function cleanPath(str) {
  if (str) {
    return str.replace(/\.\/g, '').replace(/\/+/g, '').
      replace(/^\/+/, '').replace(/\/+$/, '');
  }
}
function cleanName(str) {
  return str.replace(/\.\/g, '').replace(/\/g, '');
}
});

```

C'est en quelque sorte l'approche la plus simple et cela laisse beaucoup à désirer. C'est peut-être bien pour télécharger un fichier CSV ou quelque chose, mais c'est à peu près tout.

Dropzone (avec fer: routeur)

Si nous voulons quelque chose d'un peu plus raffiné, avec une interface utilisateur Dropzone intégrée et un point de terminaison REST, nous devons commencer à ajouter des itinéraires et des packages REST personnalisés avec des assistants d'interface utilisateur.

Commençons par importer Iron Router et Dropzone.

```

meteor add iron:router
meteor add awatson1978:dropzone

```

Et configurez la route URL de téléchargement qui est spécifiée dans l'aide de dropzone.

```

Router.map(function () {
  this.route('uploads', {
    where: 'server',
    action: function () {
      var fs = Npm.require('fs');
      var path = Npm.require('path');
      var self = this;

      ROOT_APP_PATH = fs.realpathSync('.');

      // dropzone.js stores the uploaded file in the /tmp directory, which we access
      fs.readFile(self.request.files.file.path, function (err, data) {

        // and then write the file to the uploads directory
        fs.writeFile(ROOT_APP_PATH + "/assets/app/uploads/" +self.request.files.file.name,
          data, 'binary', function (error, result) {
            if(error){
              console.error(error);
            }
            if(result){
              console.log('Success! ', result);
            }
          });
        });
    }
  });
});

```

Cool! Nous avons un téléchargeur de fichiers avec une interface utilisateur snazzy et un point de terminaison REST programmable. Malheureusement, cela ne s'adapte pas particulièrement bien.

Filepicker.io

Pour faire évoluer les choses, nous devons cesser d'utiliser le stockage local sur notre serveur et commencer à utiliser un service de stockage de fichiers dédié ou à mettre en œuvre une couche de stockage horizontale. La manière la plus simple de commencer avec le stockage de fichiers évolutif consiste à utiliser une solution telle que Filepicker.io, qui prend en charge S3, Azure, Rackspace et Dropbox. loadpicker a été un unipackage populaire de Filepicker pendant un certain temps.

```
meteor add mrt:filepicker
```

Le modèle Filepicker est assez différent des autres solutions, car il s'agit en réalité d'une intégration tierce. Commencez par ajouter une entrée de filtre de fichiers, que vous verrez fortement en fonction des attributs data-*, ce qui est un modèle assez rare dans les applications Meteor.

```
<input type="filepicker"
  id="filepickerAttachment"
  data-fp-button-class="btn filepickerAttachment"
  data-fp-button-text="Add image"
  data-fp-mimetypes="image/*"
  data-fp-container="modal"
  data-fp-maxsize="5000000"
  data-fp-services="COMPUTER, IMAGE_SEARCH, URL, DROPBOX, GITHUB, GOOGLE_DRIVE, GMAIL">
```

Vous devrez également définir une clé API, créer le widget Filepicker, le déclencher et observer ses sorties.

```
if(Meteor.isClient){
  Meteor.startup(function() {
    filepicker.setKey("YourFilepickerApiKey");
  });
  Template.yourTemplate.rendered = function(){
    filepicker.constructWidget($("#filepickerAttachment"));
  }
  Template.yourTemplate.events({
    'change #filepickerAttachment': function (evt) {
      console.log("Event: ", evt, evt.fpfile, "Generated image url:", evt.fpfile.url);
    });
  });
};
```

CollectionFS

Cependant, si vous êtes vraiment sérieux en matière de stockage et que vous souhaitez stocker des millions d'images, vous devez exploiter l'infrastructure GridFS de Mongo et créer vous-même une couche de stockage. Pour cela, vous aurez besoin de l'excellent sous-système CollectionFS.

Commencez par ajouter les paquets nécessaires.

```
meteor add cfs:standard-packages
meteor add cfs:filesystem
```

Et ajouter un élément de téléchargement de fichier à votre modèle d'objet.

```
<template name="yourTemplate">
  <input class="your-upload-class" type="file">
</template>
```

Ajoutez ensuite un contrôleur d'événements sur le client.

```
Template.yourTemplate.events({
  'change .your-upload-class': function(event, template) {
    FS.Utility.eachFile(event, function(file) {
      var yourFile = new FS.File(file);
      yourFile.creatorId = Meteor.userId(); // add custom data
      YourFileCollection.insert(yourFile, function (err, fileObj) {
        if (!err) {
          // do callback stuff
        }
      });
    });
  });
});
```

Et définissez vos collections sur votre serveur:

```
YourFileCollection = new FS.Collection("yourFileCollection", {
  stores: [new FS.Store.FileSystem("yourFileCollection", {path: "~/meteor_uploads"})]
});
YourFileCollection.allow({
  insert: function (userId, doc) {
    return !!userId;
  },
  update: function (userId, doc) {
    return doc.creatorId == userId
  },
  download: function (userId, doc) {
    return doc.creatorId == userId
  }
});
```

Merci à Raz pour cet excellent exemple. Vous voudrez consulter la documentation complète de CollectionFS pour plus de détails sur ce que peut faire CollectionFS.

Téléchargement de serveur

Les scripts suivants permettent de télécharger un fichier du système de fichiers du serveur sur le serveur. Principalement pour les fichiers de configuration et les gestionnaires de fichiers.

```
//https://forums.meteor.com/t/read-file-from-the-public-folder/4910/5

// Asynchronous Method.
Meteor.startup(function () {
```

```

console.log('starting up');

var fs = Npm.require('fs');
// file originally saved as public/data/taxa.csv
fs.readFile(process.cwd() + '/../web.browser/app/data/taxa.csv', 'utf8', function (err,
data) {
  if (err) {
    console.log('Error: ' + err);
    return;
  }

  data = JSON.parse(data);
  console.log(data);
});

// Synchronous Method.
Meteor.startup(function () {
  var fs = Npm.require('fs');
  // file originally saved as public/data/taxa.csv
  var data = fs.readFileSync(process.cwd() + '/../web.browser/app/data/taxa.csv', 'utf8');

  if (Icd10.find().count() === 0) {
    Icd10.insert({
      date: new Date(),
      data: JSON.parse(data)
    });
  }
});

Meteor.methods({
  parseCsvFile:function (){
    console.log('parseCsvFile');

    var fs = Npm.require('fs');
    // file originally saved as public/data/taxa.csv
    var data = fs.readFileSync(process.cwd() + '/../web.browser/app/data/taxa.csv', 'utf8');
    console.log('data', data);
  }
});

```

Lire Téléchargement de fichier en ligne:

<https://riptutorial.com/fr/meteor/topic/3119/telechargement-de-fichier>

Chapitre 44: Test d'acceptation (avec Nightwatch)

Remarques

Nightwatch fournit des tests d'acceptation et de bout en bout pour les applications Meteor depuis la version v0.5, et a géré les migrations de PHP vers Spark vers Blaze et React; et toutes les principales plates-formes d'intégration continue. Pour une aide supplémentaire, veuillez consulter:

[Documentation de l'API Nightwatch](#)
[Groupe Google Nightwatch.js](#)

Exemples

Surface de l'application

Au niveau le plus élémentaire, les tests d'acceptation sont essentiellement des tests en boîte noire, qui concernent essentiellement le test des entrées et des sorties d'un système fermé. En tant que tel, il y a trois caractéristiques essentielles aux tests d'acceptation: localiser une ressource, lire des données et écrire des données. En ce qui concerne les navigateurs et les applications Web, ces trois caractéristiques se résument essentiellement aux suivantes:

1. Charger une page Web ou une vue d'application
2. Inspecter les éléments de l'interface utilisateur (c.-à-d. DOM)
3. Déclencher un événement / simuler une interaction utilisateur

Nous appelons cela la surface de l'application. La surface est tout ce qu'un utilisateur voit ou vit. C'est l'extérieur d'un système de boîte noire. Et comme les utilisateurs interagissent avec les applications Web modernes sur les écrans vidéo à l'aide de navigateurs Web, notre couverture de surface est définie par des URL et des fenêtres de visualisation universelles. Et donc notre toute première visite commence comme suit:

```
module.exports = {
  "Hello World" : function (client) {
    client
      // the location of our Meteor app
      .url("http://localhost:3000")

      // the size of the viewport
      .resizeWindow(1024, 768)

      // test app output
      .verify.elementPresent('h1')
      .verify.containsText('h1', "Welcome to Meteor!")
      .verify.containsText('p', "You've pressed the button 0 times")
      .verify.elementPresent('button')
```

```

// simulate user input
.click('button').pause(500)

// test app output again, to make sure input worked
.verify.containsText('p', "button 1 times")

// saving a copy of our viewport pixel grid
.saveScreenshot('tests/nightwatch/screenshots/homepage.png')
.end();
}
};

```

Commandes personnalisées

Nightwatch prend en charge la création de commandes personnalisées pouvant simuler des frappes de touches, des clics de souris et d'autres entrées. Une commande personnalisée peut être chaînée avec d'autres commandes Nightwatch, comme ceci:

```

module.exports = {
  "Login App" : function (client) {
    client
      .url("http://localhost:3000")
      .login("janedoe@somewhere.com", "janedoe123")
      .end();
  }
};

```

Pour activer cela, définissez une commande dans `./tests/nightwatch/commands/login` ainsi:

```

exports.command = function(username, password) {

  this
    .verify.elementPresent('#login')

    // we clear the input in case there's any data remaining from previous visits
    .clearValue("#emailInput")
    .clearValue("#passwordInput")

    // we simulate key presses
    .setValue("#emailInput", username)
    .setValue("#passwordInput", password)

    // and we simulate a mouse click
    .click("#signInToAppButton").pause(1000)

  return this; // allows the command to be chained.
};

```

Pour que tout fonctionne, vous devrez ajouter des attributs d' `id` à votre page de connexion. À un certain niveau, il devra ressembler en gros à ce qui suit:

```

<template name="login">
  <div id="login">
    <input id="emailInput" name="email" type="email" />
    <input id="passwordInput" name="password" type="password" />
  </div>
</template>

```

```
<button id="#signInToAppButton">Sign In</button>
</div>
</template>
```

Inspection des objets Meteor sur le client

Comme Nightwatch a accès à la console du navigateur, il est possible d'inspecter les objets côté client à l'aide de l'API `.execute()`. Dans l'exemple suivant, nous vérifions l'objet `Session` pour une variable de session particulière. Premièrement, nous commençons par créer le fichier

`./tests/nightwatch/api/meteor/checkSession`, où nous conserverons la commande suivante:

```
// synchronous version; only works for checking javascript objects on client
exports.command = function(sessionVarName, expectedValue) {
  var client = this;
  this
    .execute(function(data) {
      return Session.get(data);
    }, [sessionVarName], function(result) {
      client.assert.ok(result.value);
      if(expectedValue) {
        client.assert.equal(result.value, expectedValue);
      }
    })
  return this;
};
```

On peut alors l'enchaîner comme ça:

```
module.exports = {
  "Check Client Session" : function (client) {
    client
      .url("http://localhost:3000")
      .checkSession("currentUser", "Jane Doe")
      .end();
  }
};
```

Formulaires et types d'entrées

Pour télécharger un fichier, vous devez d'abord créer un répertoire / data et ajouter le fichier à télécharger.

```
tests/nightwatch/data/IM-0001-1001.dcm
```

Votre formulaire aura besoin d'une entrée avec le type de fichier. (Certaines personnes n'aiment pas les options de style que cette entrée fournit; un modèle courant consiste à masquer cette entrée et à cliquer sur un autre bouton de la part de l'utilisateur.)

```
<form id="myform">
  <input type="file" id="fileUpload">
  <input type="text" name="first_name">
  <input type="text" name="last_name">
```

```

<input type="date" name="dob_month">
<input type="date" name="dob_day">
<input type="date" name="dob_year">

<input type="radio" name="gender" value="M">
<input type="radio" name="gender" value="F">
<input type="radio" name="gender" value="O">

<input type="select" name="hs_graduation_year">
<input type="text" name="city">
<input type="select" name="state">

<input type="submit" name="submit" value="Submit">
</form>

```

Vos tests devront alors utiliser `setValue ()` et résoudre le chemin d'accès à la ressource de fichier local.

```

module.exports = {
  "Upload Study" : function (client) {
    console.log(require('path').resolve(__dirname + '/../data' ));

    var stringArray = "Chicago";

    client
      .url(client.globals.url)
      .verify.elementPresent("form#myform")

      // input[type="file"]
      .verify.elementPresent("input#fileUpload")
      .setValue('input#fileUpload', require('path').resolve(__dirname + '/../data/IM-0001-1001.dcm'))

      // input[type="text"]
      .setValue('input[name="first_name"]', 'First')
      .setValue('input[name="last_name"]', 'Last')

      // input[type="date"]
      .click('select[name="dob_month"] option[value="3"]')
      .click('select[name="dob_day"] option[value="18"]')
      .click('select[name="dob_year"] option[value="1987"]')

      // input[type="radio"]
      .click('input[name="gender"][value="M"]')

      // input[type="number"]
      .click('select[name="hs_graduation_year"] option[value="2002"]')

      // input[type="text"]
      // sometimes Nightwatch will send text faster than the browser can handle
      // which will cause skipping of letters. In such cases, we need to slow
      // Nightwatch down; which we do by splitting our input into an array
      // and adding short 50ms pauses between each letter
      for(var i=0; i < stringArray.length; i++) {
        client.setValue('input[name="city"]', stringArray[i]).pause(50)
      }

      // input[type="select"]
      // after an array input above, we need to resume our method chain...

```

```

    client.click('select[name="state"] option[value="CA"]')

    // input[type="number"]
    .setValue('input[name="zip"]', '01234')

    //input [ type="submit" ]
    .click('button[type="submit"]')
    .end();
  }
};

```

Nous remercions [Daniel Rinehart](#) pour avoir inspiré cet exemple.

Composants et objets de page

Les objets de page sont similaires aux commandes personnalisées. sauf qu'ils sont des collections de commandes personnalisées associées à un composant d'interface utilisateur spécifique. Cela fonctionne extrêmement bien avec la conception à base de composants moderne, comme dans React.

```

module.exports = {
  url: 'http://localhost:3000/login',
  commands: [{
    login: function(email, password) {
      return this
        .clearValue('input[name="emailAddress"]')
        .clearValue('input[name="password"]')

        .setValue('input[name="emailAddress"]', email)
        .setValue('input[name="password"]', password)

        .verify.elementPresent('#loginButton')
        .click("#loginButton");
    },
    clear: function() {
      return this
        .waitForElementVisible('@emailInput')
        .clearValue('@emailInput')
        .clearValue('@passInput')
        .waitForElementVisible('@loginButton')
        .click('@loginButton')
    },
    checkElementsRendered: function(){
      return this
        .verify.elementPresent("#loginPage")
        .verify.elementPresent('input[name="emailAddress"]')
        .verify.elementPresent('input[name="password"]')
    },
    pause: function(time, client) {
      client.pause(time);
      return this;
    },
    saveScreenshot: function(path, client){
      client.saveScreenshot(path);
      return this;
    }
  }],
  elements: {

```

```

emailInput: {
  selector: 'input[name=email]'
},
passInput: {
  selector: 'input[name=password]'
},
loginButton: {
  selector: 'button[type=submit]'
}
}
};

```

La seule mise en garde concernant l'utilisation du modèle PageObject dans les composants de test est que l'implémentation rompt le flux de chaînage de méthode fourni par `verify.elementPresent` natif. Au lieu de cela, vous devrez affecter l'objet page à une variable et instancier une nouvelle chaîne de méthode pour chaque page. Un prix raisonnable à payer pour un modèle cohérent et fiable pour tester la réutilisation du code.

```

module.exports = {
  tags: ['accounts', 'passwords', 'users', 'entry'],
  'User can sign up.': function (client) {

    const signupPage = client.page.signupPage();
    const indexPage = client.page.indexPage();

    client.page.signupPage()
      .navigate()
      .checkElementsRendered()
      .signup('Alice', 'Doe', 'alice@test.org', 'alicedoe')
      .pause(1500, client);

    indexPage.expect.element('#indexPage').to.be.present;
    indexPage.expect.element('#authenticatedUsername').text.to.contain('Alice Doe');
  },
}

```

Lire Test d'acceptation (avec Nightwatch) en ligne: <https://riptutorial.com/fr/meteor/topic/6454/test-d-acceptation--avec-nightwatch->

Chapitre 45: Utilisation de polymère avec Meteor

Exemples

En utilisant le différentiel: vulcaniser

À la racine de votre projet, assurez-vous que Bower est installé (`npm install -g bower`) et exécutez `bower init` . Cela créera un fichier `bower.json` dans le répertoire de votre projet.

Créez un nouveau fichier appelé `.bowerrc` dans votre répertoire racine. Il devrait contenir les éléments suivants:

```
{
  "directory": "public/bower_components"
}
```

Cela permet à Bower de savoir qu'il doit enregistrer les composants dans le dossier `bower_components` répertoire public de votre application.

Ajoutez maintenant les composants Polymer que vous souhaitez utiliser avec votre application.

Dans le répertoire racine de votre application, `bower-install` chaque composant que vous souhaitez utiliser.

```
bower install --save PolymerElements/paper-button#^1.0.0 PolymerElements/paper-checkbox#^1.0.0
```

Ajouter [Vulcanize](#) à votre projet

```
Meteor add differential:vulcanize
```

Créez un nouveau fichier appelé `config.vulcanize` à la racine de votre projet. Il doit contenir les éléments suivants:

```
{
  "polyfill": "/bower_components/webcomponentsjs/webcomponents.min.js",
  "useShadowDom": true, // optional, defaults to shady dom (polymer default)
  "imports": [
    "/bower_components/paper-button/paper-button.html",
    "/bower_components/paper-checkbox/paper-checkbox.html"
  ]
}
```

"imports" devrait lister chaque composant que vous utiliserez dans votre application.

Vous pouvez maintenant utiliser les composants que vous avez importés dans vos modèles Blaze, comme vous le feriez avec tout autre élément:

```
<template name="example">
  <div>
    this is a material design button: <paper-button></paper-button>
    this is a material design checkbox: <paper-checkbox></paper-checkbox>
  </div>
</template>
```

Lire Utilisation de polymère avec Meteor en ligne:

<https://riptutorial.com/fr/meteor/topic/4598/utilisation-de-polymere-avec-meteor>

Chapitre 46: Utiliser des paquets de météorites privés sur Codeship

Remarques

Notez que nous n'avons pas discuté de l'utilisation et du développement de vos packages locaux. Il y a plusieurs manières, je suggère d'utiliser la variable d'environnement `PACKAGE_DIRS` décrite par [David Weldon sur son site Web](#) .

Exemples

Installer MGP

Nous utilisons le progiciel Dispatches [Great Meteor Github Packages \(mgp\)](#) :

```
npm install --save mgp
```

Ajoutez ensuite la commande suivante à vos scripts `package.json` :

```
"mgp": "mgp"
```

Créez un fichier nommé `git-packages.json` dans la racine de votre projet. Ajoutez une configuration pour chaque package Meteor Github (privé) dont dépend votre projet:

```
{
  "my:yet-another-private-package": {
    "git": "git@github.com:my/private-packages.git",
    "branch": "dev"
  }
}
```

Vous trouverez plus d'informations sur la configuration de vos paquets privés sur le [dépôt Github des projets](#).

Configurer Codeship pour installer des packages Github privés

Ajoutez la commande suivante aux commandes d'installation de Codeship:

```
meteor npm run mgp
```

Maintenant, nous devons donner à Codeship l'accès à ces référentiels privés. Il existe un [article de documentation de Codeship](#) décrivant ce processus en détail, mais voici les étapes à suivre pour Github:

- Créez un nouveau compte Github. Un soi-disant [utilisateur de la machine](#) .

- Supprimez la clé de déploiement de votre dépôt en cours de test. Ici: https://github.com/YOUR_USERNAME/REPO_UNDER_TEST/settings/keys
- Prenez la clé publique SSH à partir de vos paramètres de projets de codeship. Quelque part ici: https://codeship.com/projects/PROJECT_NUMBER/configure
- Ajoutez cette clé publique SSH aux clés SSH de votre ordinateur: <https://github.com/settings/keys>
- Donner à cet utilisateur la machine d'accès à tous vos référentiels référencés

Il devrait être similaire pour BitBucket et les autres.

Lire [Utiliser des paquets de météorites privés sur Codeship en ligne:](#)

<https://riptutorial.com/fr/meteor/topic/6742/utiliser-des-paquets-de-meteorites-privés-sur-codeship>

Chapitre 47: Utiliser Meteor avec un serveur proxy

Exemples

En utilisant la variable d'envoi HTTP [S]_PROXY`

Cette page décrit l'utilisation de l'outil de ligne de commande Meteor (par exemple, lors du téléchargement de packages, du déploiement de votre application, etc.) derrière un serveur proxy.

Comme beaucoup d'autres logiciels de ligne de commande, l'outil Meteor lit la configuration du proxy à partir des variables d'environnement `HTTP_PROXY` et `HTTPS_PROXY` (les variantes en minuscule fonctionnent aussi). Exemples de fonctionnement de Meteor derrière un proxy:

- sous Linux ou Mac OS X

```
export HTTP_PROXY=http://user:password@1.2.3.4:5678
export HTTPS_PROXY=http://user:password@1.2.3.4:5678
meteor update
```

- sur Windows

```
SET HTTP_PROXY=http://user:password@1.2.3.4:5678
SET HTTPS_PROXY=http://user:password@1.2.3.4:5678
meteor update
```

Configuration d'un niveau de proxy

- [Déployer l'application Meteor sur Ubuntu avec le proxy Nginx](#)
- [Comment créer un certificat SSL sur Nginx pour Ubuntu 14](#)
- [Comment déployer une application Meteor JS sur Ubuntu avec Nginx](#)
- [Comment installer un certificat SSL à partir d'une autorité de certification commerciale](#)
- [Certificats SSL NameCheap](#)

Lire [Utiliser Meteor avec un serveur proxy en ligne:](#)

<https://riptutorial.com/fr/meteor/topic/517/utiliser-meteor-avec-un-serveur-proxy>

Chapitre 48: Variables d'environnement

Paramètres

Paramètre	Détails
PORT	Port sur lequel l'application Meteor sera disponible
MONGO_URL	URL pour se connecter à l'instance Mongo.
ROOT_URL	...
OPLOG_URL	...
MONGO_OPLOG_URL	...
METEOR_ENV	...
NODE_ENV	...
NODE_OPTIONS	...
DISABLE_WEBSOCKETS	...
MAIL_URL	...
DDP_DEFAULT_CONNECTION_URL	...
HTTP_PROXY	...
HTTPS_PROXY	...
METEOR_OFFLINE_CATALOG	...
METEOR_PROFILE	...
METEOR_DEBUG_BUILD	...
TINYTEST_FILTER	...
MOBILE_ROOT_URL	...
NODE_DEBUG	...
BIND_IP	...
PACKAGE_DIRS	...

Paramètre	Détails
DÉBOGUER	...
METEOR_PRINT_CONSTRAINT_SOLVER_INPUT	...
METEOR_CATALOG_COMPRESS_RPCS	...
METEOR_MINIFY_LEGACY	...
METEOR_DEBUG_SQL	...
METEOR_WAREHOUSE_DIR	...
AUTOUPDATE_VERSION	...
USE_GLOBAL_ADK	...
METEOR_AVD	...
DEFAULT_AVD_NAME	...
METEOR_BUILD_FARM_URL	...
METEOR_PACKAGE_SERVER_URL	...
METEOR_PACKAGE_STATS_SERVER_URL	...
DEPLOY_HOSTNAME	...
METEOR_SESSION_FILE	...
METEOR_PROGRESS_DEBUG	...
METEOR_PRETTY_OUTPUT	...
APP_ID	...
AUTOUPDATE_VERSION	...
CONSTRAINT_SOLVER_BENCHMARK	...
DDP_DEFAULT_CONNECTION_URL	...
SERVER_WEBSOCKET_COMPRESSION	...
USE_JSESSIONID	...
METEOR_PKG_SPIDERABLE_PHANTOMJS_ARGS	...
WRITE_RUNNER_JS	...

Paramètre	Détails
TINYTEST_FILTER	...
METEOR_PARENT_PID	...
METEOR_TOOL_PATH	...
RUN_ONCE_OUTCOME	...
TREE_HASH_DEBUG	...
METEOR_DEBUG_SPRINGBOARD	...
METEOR_TEST_FAIL_RELEASE_DOWNLOAD	...
METEOR_CATALOG_COMPRESS_RPC	...
METEOR_TEST_LATEST_RELEASE	...
METEOR_WATCH_POLLING_INTERVAL_MS	...
EMACS	...
METEOR_PACKAGE_STATS_TEST_OUTPUT	...
METEOR_Test_TMP	...

Exemples

Utilisation de variables d'environnement avec Meteor

Les variables d'environnement peuvent être définies avant l'appel de météore, comme ceci:

```
PORT=4000 meteor
NODE_ENV="staging" meteor
```

Paramétrage du serveur SMTP Meteor

Exemple Gmail

```
MAIL_URL=smtp://username%40gmail.com:password@smtp.gmail.com:465/
```

Remarque: Cette configuration permet uniquement d'envoyer 2000 emails par jour. Veuillez consulter <https://support.google.com/a/answer/176600?hl=fr> pour [connaître les autres configurations](#).

Lire Variables d'environnement en ligne: <https://riptutorial.com/fr/meteor/topic/3154/variables-d-environnement>

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec meteor	Ankit , Christian Fritz , Community , Gal Dreiman , ghybs , grahan , hwillson , João Rodrigues , levon , Matthias Eckhart , mav , mertyildiran , Ray , reoh , robfallows , Tom Coleman , Zoltan Olah
2	Accéder aux machines de construction Meteor à partir de Windows	Tom Coleman
3	Agrégation MongoDB	AbigailW , levon
4	Application mobile	AbigailW , Anis D , Antti Haapala , ghybs
5	Applications hors ligne	AbigailW
6	Blaze Templating	Dan Cramer , distalx , ghybs , jordanwillis , khem poudel , RamenChef , robfallows , Thomas Gerot
7	Collections Mongo	AbigailW
8	Comptes utilisateurs Meteor	Barry Michael Doyle , KrisVos130
9	Configuration de base de codeship pour le test automatisé	schmidsi
10	Déploiement avec Upstart	AbigailW , ghybs
11	Déploiement continu sur Galaxy de Codeship	schmidsi
12	Détection de l'environnement	AbigailW , ghybs
13	Electrify - Compiler Meteor comme une	AbigailW , JuanGesino , Nick Bull , RamenChef

	application localement installable	
14	Enregistrement	AbigailW
15	Enveloppant les méthodes asynchrones dans une fibre pour une exécution synchrone.	Dranithix
16	ES2015 modules (Import & Export)	reoh
17	ESLint	saimeunt
18	Guide d'initiation à l'installation de Meteor 1.4 sur AWS EC2	AGdev
19	Installation complète - Mac OSX	AbigailW , RamenChef
20	Intégration continue et nuages de périphériques (avec Nightwatch)	4444 , AbigailW
21	Intégration d'API tiers	AbigailW
22	Jeux de répliques et de fragmentation	AbigailW , Anis D
23	L'optimisation des performances	AbigailW , RamenChef , reoh
24	Le débogage	AbigailW , distalx
25	Le routage	Ankit , ghybs , Luna , Michael Balmes
26	Les atouts	Matthias Eckhart
27	Meteor + React + ReactRouter	rafahoro
28	Meteor + Réagir	AbigailW , aedm , corvid , ghybs , RamenChef , Teagan Atwater , zliw

29	Migrations du schéma Mongo	AbigailW
30	Mise à l'échelle horizontale	AbigailW
31	Mongo Database Management	AbigailW , distalx , RamenChef , TechplexEngineer
32	MongoDB	distalx , Dranithix , hwillson , Matthias Eckhart , robfallows , Thomas Gerot
33	Nightwatch - Configuration et configuration	AbigailW
34	Node / NPM	hwillson
35	Outils de développement	AbigailW , Ankit , Ankit Balyan , Fermuch , Ilya Lyamkin
36	Publication d'une piste de publication	AbigailW
37	Publication de données	Abdelrahman Elkady , AbigailW , Chris Pena , corvid , Dair , dangsonbk , Eliezer Steinbock , Faysal Ahmed , ghybs , j6m8 , Maciek , RamenChef , Ramil Muratov , robfallows , Serkan Durusoy
38	Réactif (Vars et Dictionnaires)	Ankit
39	Recettes de l'interface utilisateur Blaze (Bootstrap; No jQuery)	AbigailW , Anis D
40	Récupération des données d'un Meteor.call	Ramil Muratov , Rolljee , Sacha
41	Structure du répertoire	AbigailW , anomepani , ghybs , Michael Balmes , Nick Carson , Phe0nix , reoh , Thomas Gerot
42	Tâches d'arrière-plan	Filipe Névola
43	Téléchargement de fichier	AbigailW
44	Test d'acceptation	AbigailW

(avec Nightwatch)		
45	Utilisation de polymère avec Meteor	Thaum Rystra
46	Utiliser des paquets de météorites privés sur Codeship	schmidsi
47	Utiliser Meteor avec un serveur proxy	AbigailW , Serkan Durusoy , Tom Coleman
48	Variables d'environnement	AbigailW , hcvst