Intégration déploiement

Semantic versionning

Qu’est-ce que le semantic versionning ?

L'objectif du Semantic Versioning est de donner un numéro unique à chaque nouvelle version d'un logiciel (au sens large) afin de pouvoir identifier ce dernier, tout en apportant un maximum d'informations à l'utilisateur sur ses évolutions.

Le SemVer réponds à des règles spécifiques afin d’être utilisé de la même manière par tous les développeurs.

Major

Le chiffre "MAJOR" est le plus critique, c'est le seul qui indique une évolution de l’API tellement importante que la rétrocompatibilité n'est plus assurée, indiquant donc que cette version ne sera pas forcément compatible avec les versions précédentes.

En français on parle de “majeure”

Minor

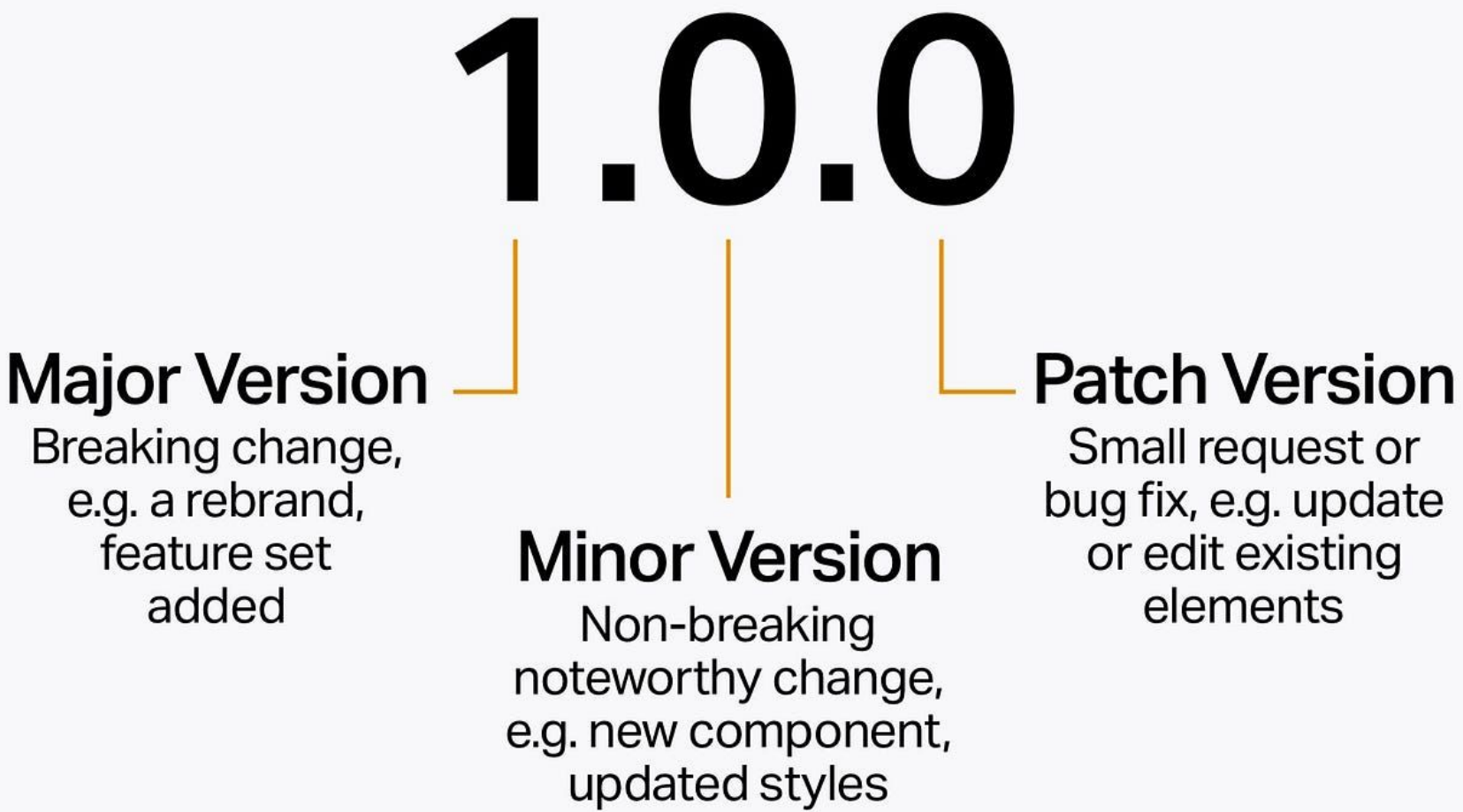
Ce chiffre permet d'indiquer que des fonctionnalités ont été ajoutées (ou que des fonctionnalités ont été dépréciées) mais que l’API reste compatible avec les anciennes versions.

En français on parle de “mineur”

Patch

Ce dernier indique que des bugs ont été corrigés mais que la rétro-compatibilité est assurée.

En français, on parle de “Correctif”


1.0.0
Major Version
Breaking change, e.g. a rebrand, feature set added
Minor Version
Non-breaking noteworthy change, e.g. new component, updated styles
Patch Version
Small request or bug fix, e.g. update or edit existing elements

Quelques règles du SemVer

- Les chiffres MAJOR, MINOR et PATCH doivent être des entiers positifs supérieurs ou égaux à 0 et ne doivent pas contenir de 0 antéposé (comme 01 ou 046).
- Un chiffre MAJOR à 0 indique un développement initial, l'API peut donc changer à tout moment sans être reflétée dans le numéro de version. L'API ne doit pas être considérée comme étant stable.
- Si le chiffre MAJOR est incrémenté, MINOR et PATCH doivent être remis à zéro, il en va de même pour PATCH si MINOR est incrémenté.
- Une fois qu'un composant est publié, le contenu de sa version NE DOIT PAS être modifié. Toute modification DOIT être publiée dans une nouvelle version.

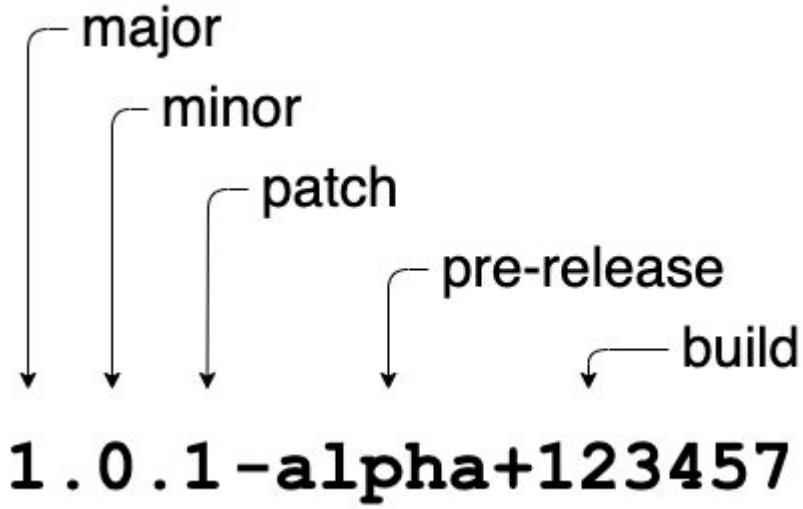
Identifieurs et métadonnées

Il est aussi possible d'ajouter des informations supplémentaires au code de la version, en ajoutant des identifieurs pour identifier la version actuelle comme étant une release ou un pre-release par exemple.

On écrira alors : 10.3.6-rc pour "release candidate", mais on pourra aussi trouver des identifieurs comme alpha ou beta, toujours séparé par un tiret (hyphen)

D'autres métadonnées peuvent être ajoutée, comme le numéro de build interne par exemple, mais elles devront toujours être précédées du signe "+", exemple : 10.3.6-rc+1452

Pour aller plus loin



npm version

npm version [<newversion> | major | minor | patch | premajor | preminor | prepatch | prerelease | from-git]

npm version patch -m "Upgrade to %s for reasons"

Cette commande va utiliser le message de commit comme un version commit en intégrant à la place de %s le résultat de la version upgrade

Utiliser dans un package.json

On peut définir une règle d'installation de version à travers le package.json de 3 manières :

- “1.0.4” // Prends cette version spécifique uniquement
- “^1.0.4” // Prends à partir de cette version et si une plus récente compatible est disponible (même majeure), l'installe (“2.3.0”)
- “~1.0.4” // Prends à partir de cette version, et si une similaire est disponible, dans le même numéro de version majeure, l'installe (“1.3.2”)

Publier notre application React sur npm

Première étape : créer un compte npm

Vous devez en créer un si vous n'avez pas encore de compte. Créez un compte npm sur https://www.npmjs.com/ gratuitement

packages privés et publics

Il est possible de publier des packages en tant que packages publics ou privés sur le registre npm.

Packages publics : ils sont destinés aux projets ou bibliothèques open source accessibles et utilisables par n'importe qui. Ces packages sont détectables et peuvent être installés par n'importe qui à l'aide de la commande npm install.

Packages privés : ils sont destinés au code propriétaire ou aux projets internes d'une organisation. Ces packages ne sont pas accessibles au public ni détectables sur le registre npm. Au lieu de cela, ils sont partagés avec des utilisateurs ou des équipes spécifiques qui ont reçu des autorisations d'accès. Ceux-ci nécessitent souvent une authentification pour y accéder et les utiliser.

Nommer son package

Je dois choisir un nom pour mon package, il ne doit pas s'agir de quelque chose de spécial, je veux dire que cela peut être n'importe quoi, mais gardez à l'esprit qu'il doit être unique.

La sélection d'un nom de package unique est cruciale. Pour confirmer l'unicité du nom de package que vous proposez sur npm, visitez le site Web de npm et recherchez le nom de package que vous proposez, et c'est tout !

Vous pouvez également confirmer l'unicité d'un nom de package via le terminal en exécutant cette commande npm search <your-package-name>

Exporter son composant principal

Dans votre fichier index.js, là où votre App est importé et utilisé, fait un

export default App;

en fin de fichier

C’est ce composant complet qu’on va publier sur npm

Créer un npmignore

Similaire à un gitignore ou dockerignore, l’idée sera de limiter au strict nécessaire la publication du package

Ajouter les dépendances nécessaires

npm install --save-dev @babel/cli @babel/core @babel/plugin-transform-react-jsx

Créer une nouvelle cible

mac / linux :

"build-npm-ci": "NODE_ENV=production && rm -rf dist && mkdir dist && npx babel src --out-dir dist --copy-files",

windows :

"build-npm": "SET NODE_ENV=production && rm -rf dist && mkdir dist && npx babel src --out-dir dist --copy-files",

Quelques commandes

On va tester alors nos commandes :

npm run build-npm

npm adduser

npm publish

Profile
Packages
Account
Billing Info
Access Tokens

Organizations

super-responsable

Access Tokens

Generate New Token | Delete Selected Tokens

Name	Type	Created	Last used	Expires	Delete
npm_pBj8.....GmMo	Publish	Oct 11, 2024	Oct 11, 2024	Never	×

Rows 1 to 1 of 1

Read the Documentation

New Access Token

Access tokens can be used instead of your password when using the npm CLI to download or publish packages.

Name

ci-cd-ynov

Select type

The type of access token defines its permissions. Read more about types of access tokens.

- ○ **Read-only**
  A read-only token can download public or private packages from the npm registry.
- ◉ **Automation**
  An automation token will **bypass** two-factor authentication (2FA) when publishing. If you have 2FA enabled, you will not be prompted when using an automation token, making it suitable for CI/CD workflows.
- ○ **Publish**
  A publish token can read **and** publish packages to the npm registry. If you have 2FA enabled, it **will** be required when using this token.

For improved security and control over token permissions, we recommend you create a granular access token instead.

Secrets | Variables

Environment secrets

This environment has no secrets.

Manage environment secrets

Repository secrets

New repository secret

Name	Last updated
CODECOV_TOKEN	8 months ago
FRONT_URL	3 months ago
NPM_TOKEN	now
REACT_APP_SERVER_URL	3 months ago

Ajouter l’étape sur le workflow

```
- name: build and publish package on NPM 📦
  run: |
      git config --global user.email "loise.fenoll@ynov.com"
          git config --global user.name "Loise Fenoll"
          npm run build-npm
          npm version patch
          npm publish
          git add .
          git commit -m “updated version”
          git push
  env:
    NODE_AUTH_TOKEN: ${{ secrets.NPM_TOKEN }}
```