Publicodes chez Nos Gestes Climat

Comment gérer de grosses bases de règles avec @publicodes/tools ?

Emile Rolley

(Rappel) Nos Gestes Climat

C'est le simulateur d'empreinte carbone individuelle de l'ADEME.

Comment ça marche ?

1. Vous répondez à un questionnaire sur votre mode de vie
2. Vous obtenez votre empreinte carbone individuelle annuelle totale (et par grandes catégories)
3. Vous obtenez des actions personnalisées en fonction de vos réponses

Techniquement

Il y a deux grandes parties :

Le modèle de calcul

Il décrit dans un ensemble de fichiers (écrits en Publicodes), les règles de calcul de l'empreinte carbone.

Le simulateur

Une application React qui utilise le moteur de Publicodes pour évaluer les règles du modèle en fonction des réponses de l'utilisateur·ice.

Si vous ne l'avez pas encore fait, je vous invite à aller faire le test : https://nosgestesclimat.fr

(Rappel) Publicodes

*"Publicodes est un langage **déclaratif** pour modéliser des **domaines métiers complexes** en les décomposant en **règles élémentaires simples** .", https://publi.codes*

- Tout est décrit avec des règles
- Les calculs plus complexes sont une composition de règles plus simples
- Le langage est simple et lisible par des non-informaticien·nes

Par exemple, voici un extrait du modèle de calcul de l'empreinte carbone des animaux domestiques :

```yaml
divers . animaux domestiques . empreinte totale:
  formule:
    somme:
      - empreinte . chats
      - empreinte . chiens

divers . animaux domestiques . empreinte . chats:
  icônes: 🐱
  titre: Chat
  formule: nombre * empreinte

divers . animaux domestiques . empreinte . chiens:
  formule:
    somme:
      - petit chien
      - chien moyen
      - gros chien
```

```yaml
divers . animaux domestiques . empreinte . chats . nombre:
  question: Combien de chats possédez-vous dans votre foyer ?
  par défaut: 0

divers . animaux domestiques . empreinte . chats . empreinte:
  titre: Empreinte d'un chat de taille moyenne
  description: |
    On considère un chat de taille moyenne (4kg).
  formule:
    somme:
      - alimentation
      - soins vétérinaires par animal
      - litière
  unité: kgCO2e
```

A mon arrivé

Il y avait un seul modèle de calcul :

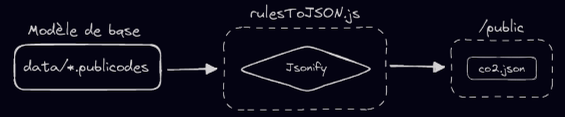
Stats

Nb. règles	Nb. fichiers	Nb. lignes	Poids
898	31	6 527	273 Ko

Perfs

	Temps de parsing	Temps d'évaluation
local (Bun)	~340 *ms*	~55 *ms*

Workflow



*Toutes les règles du modèles doivent être **compilées** dans un seul fichier JSON afin de pouvoir être **évaluées par le moteur dans le navigateur**.*

Aujourd'hui

Il y a **36** modèles compilés (17 *régions* et 2 *langues*) :

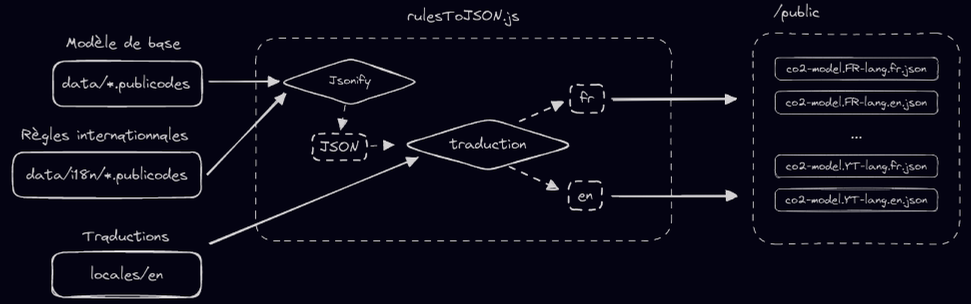
Stats (modèle de *base*)

Nb. règles	Nb. fichiers	Nb. lignes	Poids
1720	85	16 129	~800 Ko

Perfs (modèle de *base*)

	Temps de parsing	Temps d'évaluation
local (Node)	~660 *ms*	~200 *ms*
local (Bun)	~700 *ms*	~125 *ms*
preview Vercel	~1200 *ms*	~120 *ms*

Workflow



Problématiques

Dans un contexte où le nombre de modèles de calcul augmente, plusieurs questions s'imposent.

1. Une question de performance

Comment réduire le temps d'instanciation du moteur dans le navigateur, afin de pouvoir servir une page interactive le plus rapidement possible ?

2. Une question de maintenabilité

Comment modulariser les jeux de règles afin de pouvoir facilement les réutiliser dans d'autres modèles et charger uniquement les règles nécessaires ?

Optimisation des modèles

L'idée

On souhaite continuer à avoir la même granularité du modèle pour des raisons de transparence. En revanche, pour le test en lui-même, on a **uniquement besoin des questions** (des règles dont la valeur dépends de l'utilisateur·ice).

On alors dit que l'on pourrait calculer à la compilation toutes les règles qui ne dépendent pas de l'utilisateur·ice (des **constantes**).

> Ainsi on pourrait utiliser se modèle optimisé pour le test et le modèle complet pour la documentation.

Exemple

Par exemple, reprenons les règles de calcul de l'empreinte des animaux domestiques :

```yaml
chats:
  icônes: 🐱
  titre: Chat
  formule: nombre * empreinte

chats . nombre:
  question: Combien de chats possédez-vous ?
  par défaut: 0

chats . empreinte:
  titre: Empreinte d'un chat de taille moyenne
  description: |
    On considère un chat de taille moyenne (4kg).
  formule:
    somme:
      - alimentation
      - soins vétérinaires par animal
      - litière
  unité: kgCO2e
```

```yaml
chats . alimentation:
  formule: |
    besoin journalier nourriture
    * commun . jours par an
    * empreinte nourriture
  unité: kgCO2e

chats . alimentation . empreinte nourriture:
  titre: Empreinte nourriture pour chats
  formule: 2.5
  unité: kgCO2e/kg
```

La règle empreinte nourriture est une ***constante***, on peut donc la remplacer par sa valeur dans la règle alimentation :

```yaml
chats . alimentation:
  formule: |
    besoin journalier nourriture
    * commun . jours par an
    * 2.5
  unité: kgCO2e
```

Exemple

De même, on peut remplacer la règle commun . jours par an par sa valeur :

```
chats . alimentation:
  formule: |
    besoin journalier nourriture
    * 365
    * 2.5
  unité: kgCO2e
```

On ne peut pas aller plus loin, car la règle besoin journalier nourriture dépend du niveau d'activité du chat.

En revanche, peut faire la même chose pour la règle litière :

```
chats . litière:
  titre: Empreinte de la litière pour un chat
  formule: quantité annuelle * empreinte
  unité: kgCO2e
  note: L'impact de la litière semble résider ...

chats . litière . quantité annuelle:
  titre: Quantité litière par an pour un chat
  formule: 33
  unité: kg
  note: https://www.planetoscope.com/Animaux/1211-.html

chats . litière . empreinte:
  titre: Empreinte d'un kilogramme de litière pour chat
  formule: 0.0506
  unité: kgCO2e/kg
  note: Nous faisons l'hypothèse ici qu'une litière ...
```

Qui devient :

```
chats . litière:
  titre: Empreinte de la litière pour un chat
  formule: 33 * 0.0506
  unité: kgCO2e
  note: L'impact de la litière semble résider ...
```

Exemple

Au final, pour la règle `chats . empreinte` on obtient :

```yaml
chats . empreinte:
  titre: Empreinte d'un chat de taille moyenne
  description: |
      On considère un chat de taille moyenne (4kg).
  formule:
    somme:
      - alimentation
      - 3.966386554621849
      - 1.6698
  unité: kgCO2e
```

Avec :

```yaml
chats . alimentation:
  formule: besoin journalier nourriture * 365 * 2.5
  unité: kgCO2e
```

Au lieu de :

```yaml
chats . empreinte:
  titre: Empreinte d'un chat de taille moyenne
  description: |
    On considère un chat de taille moyenne (4kg).
  formule:
    somme:
      - alimentation
      - soins vétérinaires par animal
      - litière
  unité: kgCO2e
```

Pour évaluer la règle `chats . empreinte`, on utilise plus que **6 règles au lieu 16**.

1. Optimisation des modèles

Les résultats

Réduction du nombre de règles

	Nb. règles	Poids
Règles de base	1720	793 Ko
Règles optimisées	**951 (-44%)**	476 Ko (-40%)

> Diminution de ~40% du nombre de règles et du poids du modèle

Les gains de performance

perf-base.js

```
import base from './public/co2-model.FR-lang.fr.json'
import Engine from 'publicodes'
new Engine(base)
```

perf-optim.js

```
import optim from './public/co2-model.FR-lang.fr-opti.json'
import Engine from 'publicodes'
new Engine(optim)
```

Commande	Moyenne [ms]	Min [ms]	Max [ms]	Ratio
bun run perf-base.js	765.6 ± 28.2	731.2	832.4	1.64 ± 0.07
bun run perf-optim.js	**466.5** ± 7.5	459.1	477.2	1.00

> Gain de ~40% sur le temps d'instanciation du moteur

2. Système d'import

Le besoin

Pouvoir réutiliser des règles dans plusieurs modèles de calcul.

En particulier, on souhaitait rajouter les règles permettant de calculer l'empreinte des trajets en ferry et de la piscine. Or, **le calcul avait déjà été implémenté** dans le modèle de *https://futur.eco*.

La solution

On a donc créé un système d'import de règles, qui permet de charger des règles depuis un autre modèle.

Pour cela, on a besoin de deux choses :

1. Publier le modèle sur NPM
2. Ajouter une macro (importer!) qui permet de récupérer les règles de puis un paquet NPM

1. Publier le modèle sur NPM

Il suffit de compiler le modèle en un fichier JSON <nom-du-modele>.model.json et de l'ajouter dans le package.json.

```
{
  "name": "myModel",
  "files": [ "myModel.model.json" ],
}
```

Template GitHub

Le workflow a été automatisé dans un template GitHub : *https://github.com/publicodes/model-template*

Il permet de pouvoir très rapidement publier un modèle sur NPM avec :

- un fichier index.js qui exporte le modèle compilé
- un fichier index.d.ts qui exporte les types du modèle (l'ensemble des règles disponibles)
- une documentation générée avec @publicodes/react-ui est déployée avec GitHub Pages
- *(bientôt)* le code d'un serveur exposant l'API REST du modèle avec @publicodes/rest-api

---

Exemple de paquet utilisant le template :

- @incubateur-ademe/publicodes-commun
- @incubateur-ademe/publicodes-negaoctet
- @incubateur-ademe/publicodes-impact-livraison

2. Utiliser la macro `importer!`

La macro `importer!` permet de charger des règles depuis un paquet NPM ou un fichier local.

Exemple

Import des règles communes depuis `publicodes-commun` dans le namespace `commun`

```yaml
importer!:
  depuis:
    nom: '@incubateur-ademe/publicodes-commun'
    url: https://github.com/incubateur-ademe/publicodes-commun
  dans: commun
  les règles:
    - intensité électricité
    - mois par an
    - semaines par an
    - jours par semaine
    - jours par an
```

Utilisation de la règle `commun . jours par an`

```yaml
alimentation . petit déjeuner annuel:
  titre: Petit déjeuner
  icônes: 🥐
  formule: petit déjeuner . par jour * commun . jours par an
```

2. Système d'import

🟦 Techniquement

Pour chacune des règles importées :

1. On récupère le modèle depuis le paquet NPM
2. On récupère toutes les dépendances de cette règle
3. On créer tous les espaces de noms nécessaires

🟩 Exemple

Lorsque l'on importe la règle `transport . ferry . empreinte par km volume` depuis le modèle de Futur.eco :

```yaml
importer!:
  depuis:
    nom: futureco-data
    url: https://github.com/laem/futureco-data
  les règles:
    - transport . ferry . empreinte par km volume
```

Exemple

On obtient les règles suivantes dans le modèle compilé :

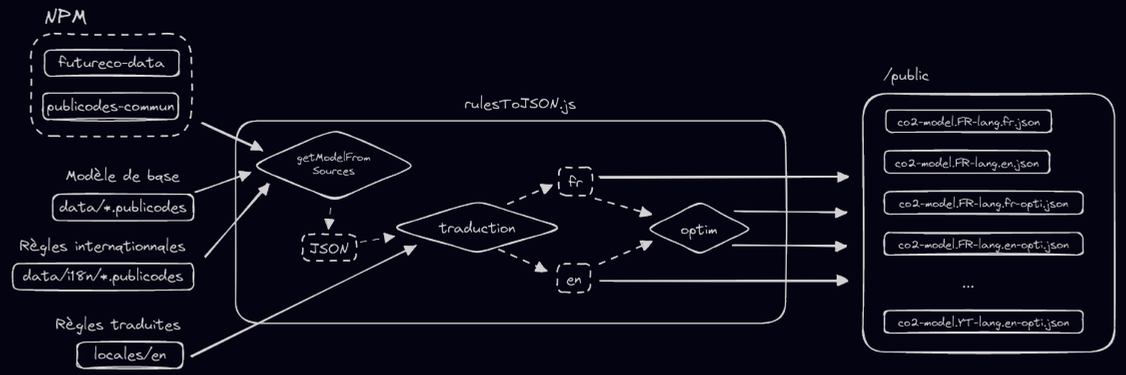
```
// La règle importée
"futureco-data . transport . ferry . empreinte par km volume": {
    "titre": "Empreinte par km et passager",
    "formule": "empreinte du bateau * part du volume",
    "unité": "kgCO2e/km"
},

// Dépendances de la règle `empreinte par km volume`
"futureco-data . transport . ferry . empreinte du bateau": {
    "titre": "Empreinte du bateau témoin par km",
    "formule": "empreinte totale . par km",
    "unité": "kgCO2e/km"
},
"futureco-data . transport . ferry . part du volume": {
    "titre": "Part du volume utile du bateau attribuée au passager",
    "formule": "volume passager / volume utile",
},
"futureco-data . transport . ferry . empreinte totale . par km": {
    "valeur": "empreinte totale / km par mille nautique",
},
...

// Espaces de noms créés
"futureco-data": null,
"futureco-data . transport . ferry": null,
"futureco-data . transport": null,
```

Conclusion

Le réel workflow de NGC



Au total, aujourd'hui en production :

- c'est 72 modèles JSON compilés
- dont 36 modèles optimisés avec @publicodes/tools/optims
- avec des règles importées depuis 2 paquets NPM différents (publicodes-commun et futureco-data)

Conclusion

Pour Nos Gestes Climat

- Gain de performance sur le temps d'instanciation du moteur (~40%)
- Réutilisation de règles du modèle futureco-data pour l'ajout de l'empreinte des trajets en ferry et de la piscine
- Possibilité de factoriser des règles communes dans un paquet NPM (publicodes-commun)
- Publication des modèles utilisés par ImpactCO2 (publicodes-impact-livraison et publicodes-negaoctet)

Pour Publicodes

@publicodes/tools

Compilation

Uniformisation du processus de compilation des modèles avec la fonction getModelFromSource

Ajout de la macro importer! pour charger des règles depuis un paquet NPM ou un fichier local (disponible avec getModelFromSource)

Optimisation

Ajout d'une passe d'optimisation pour réduire le nombre de règles et le poids d'un modèle avec la fonction constantFolding

Template GitHub

Proposition d'un workflow commun afin de faciliter la création et l'utilisation des modèles Publicodes avec le template GitHub.

Pour aller plus loin

La suite

- Il y a encore de la place pour améliorer l'optimisation des modèles
- Reprendre le travail sur l'implémentation du LSP pour VSCode (@publicodes/language-server)
- *(Un jour peut-être)* compiler directement les modèles en JS/WebAssembly pour ne pas avoir à charger le moteur dans le navigateur

Les ressources

- La documentation de Publicodes : *https://publi.codes/docs*
- La documentation de l'API : *https://publicodes.github.io/tools/index.html*
- Le dépôt du template GitHub : *https://github.com/publicodes/model-template*
- Des exemples d'utilisation de @publicodes/tools
  - nosgestesclimat : *https://github.com/incubateur-ademe/nosgestesclimat*
  - publicodes-commun : *https://github.com/incubateur-ademe/publicodes-commun*
  - publicodes-negaoctet : *https://github.com/incubateur-ademe/publicodes-negaoctet*
  - publicodes-impact-livraison : *https://github.com/incubateur-ademe/publicodes-negaoctet*
- Slides : *quelquepart*

Merci de votre attention :)