Formation CQP Bloc 2 - U5 - S19

Traitement de données géospatiales avec Krawler

https://kalisio.com

Kalisio

Un cas d'usage concret

Créer une cartographie liée à la pandémie COVID-19

Objectifs du fil rouge

➜ Calculer le taux d'hospitalisation par département et le représenter sur une carte avec une symbologie adaptée

a. Recenser les données disponibles
   - nombre d'hospitalisations
   - limites administratives
   - population

b. Définir les traitements appropriés
   - extraction des données
   - transformation des données

c. Définir la représentation
   - symbologie
   - configuration de l'outil de visualisation

➜ Retrouvez le contenu des exercices sur https://github.com/kalisio/cqp-geom

Krawler: capabilities

https://kalisio.github.io/krawler

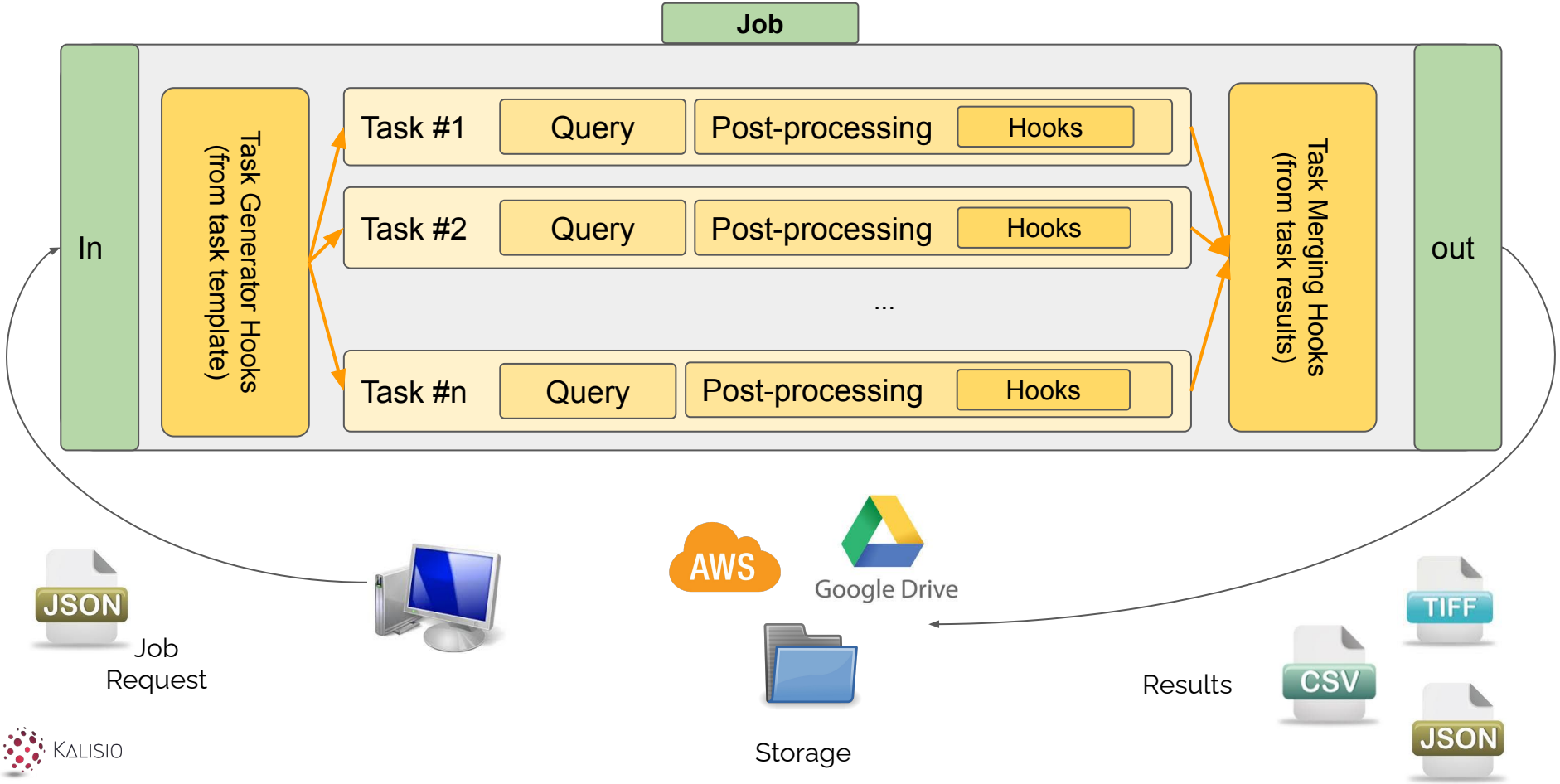
- Provides a service abstraction for
  - the **storage**: where the downloaded/exported data will reside
  - the **task**: what data to be downloaded
  - the **job**: what tasks to be run to fulfill a request (sequencing)

- Provides an abstraction to create a **processing pipeline** on top of the services
  - **hooks** are functions that are run before/after a service operation
    - e.g. a conversion after a download task or a task generator before a job run

- Can be used as a **web application**
  - the hooks are “fixed” (i.e. tied to a specific business process)
  - REST API on top of all services
  - optimal performances

- Can be used as a **command-line utility**
  - the hooks can be activated on-demand to cover multiple business use cases
  - less optimal because a new app is instantiated for each job
  - No REST API

Krawler: overview



Krawler: jobs & tasks definition

- ➜ The request is formalized using a JSON Job object/file
- ➜ The job object/file define a set of Tasks optionally based on a template

```
Job: {
     id: String,
     options: {
          workersLimit: Number
          ...
     },
     taskTemplate: {
          ...
     },
     tasks: [
          {
               id: String,
               type: String,
               url: String,
               options: Object
          }
     ]
```

Specific job parameters

Common options for all tasks (similar to task object definition)

The type of the task defines the way to access the data. At least, standard OGC protocols (WMS, WCS, WFS…) must be proposed

Specific request parameters

Krawler: jobs & tasks definition

- ➜ It includes a definition of the store or a reference on it
- ➜ It includes the configuration of the hooks for the command-line mode

```
Job: {
     ...
     store: 'job-store',
     store: {
          id: 'job-store',
          type: 'fs',
          path: './data'
     },
     hooks: {
          service: {
               before: {
                    hookName: hookOptions
                    ...
               },
               after: { … }
          }
     }
}
```

Storage to be created/used for output

Hooks to be applied for this job, it depends on the required business process

Target service (either tasks or jobs)

Set of applied hooks in stage with configuration options

Target stage (before/after)

Krawler: task types

- `{ type: 'store', options: { store: xxx } }` = Read from a store
- `{ type: 'http', options: { url: xxx } }` = HTTP request
- `{ type: 'wms', options: { url: xxx, param: yyy } }` = WMS request
- `{ type: 'wfs', options: { url: xxx, param: yyy } }` = WFS request
- `{ type: 'wcs', options: { url: xxx, param: yyy } }` = WCS request
- `{ type: 'overpass', options: { url: xxx, param: yyy } }` = OSM Overpass API request
- `{ type: 'noop' }` = NOOP, useful to start a pipeline

https://kalisio.github.io/krawler/reference/services.html#tasks

Krawler: main hooks

- Manage stores
  - `createStore(s), removeStore(s)`
- Manage DB
  - `connectMongo, disconnectMongo, etc.`
  - `connectPG, disconnectPG, etc.`
- Manage Docker
  - `connectDocker, disconnectDocker, etc.`

- Read JSON data from a source to a store
  - `readJSON, readCSV, readXML, readYML, readGeoTiff`

- Process or transform JSON data
  - `transformJSON, mergeJson, convertToGeoJson, reprojectGeoJson`
  - `apply, discardIf, runCommand`

- Write JSON data to a store
  - `writeJSON, writeCSV, writeXML, writeYML`

https://kalisio.github.io/krawler/reference/hooks.html

Exercices

➜ Créer un fichier de limites administratives incluant les informations de population

a. Créer les “store” de type mémoire en entrée et fichier en sortie

b. Créer une tâche pour lire les données de population en mémoire

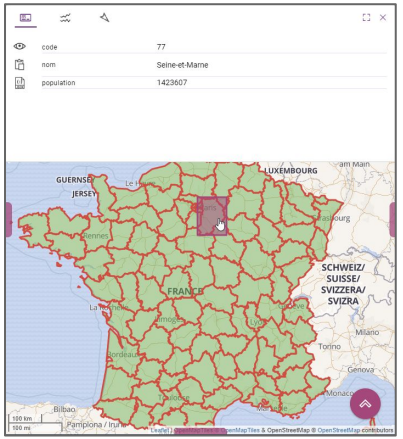
c. Créer une tâche pour télécharger les limites administratives en mémoire

d. Fusionner les données de population avec les limites en fonction du code de département

e. Stocker les données spatialisées dans un fichier GeoJSON

f. Supprimer les données intermédiaires et le stores

Tester votre résultat sur https://kano.test.kalisio.xyz/ ou https://geojson.io/



Exercices

➜ Solution

```
module.exports = {
  id: 'population-departements',
  store: 'memory', // Default job store
  tasks: [{
    id: 'departements.json',
    type: 'http', // Download task
    options: {
      url: `https://raw.githubusercontent.com/gregoiredavid/france-geojson/master/departements-version-simplifiee.geojson`
    }
  }, {
    id: 'population.csv',
    type: 'store', // Read task
    options: { store: 'fs' }
  }],
  hooks: {
    tasks: {
      after: {
        readJson: {
          match: { id: 'departements.json' },
          features: true
        },
        readCSV: {
          match: { id: 'population.csv' },
          headers: true, delimiter: ';' // Default delimiter is ,
        }
      }
    },
    jobs: {
      before: {
        createStores: [
          { id: 'memory' }, // Input store
          { id: 'fs', options: { path: __dirname } } // Output store
        ]
      },
      after: {
        mergeJson: {
          deep: true, // Add hospitalisations to departement features
          by: (item) => item.Code || item.properties.code,
          transform: { mapping: { Ensemble: 'population' }, unitMapping: { population: { asNumber: true } }, pick: ['population'] }
        },
        convertToGeoJson: {},
        writeJson: { store: 'fs' },
        clearOutputs: {}, // Cleanup
        removeStores: ['memory', 'fs']
      }
    }
  }
}
```

Exercices

➜ Traiter les données liées aux hospitalisations

a. Créer un “store” de type mémoire en entrée

b. Créer un “store” de type fichier en sortie

c. Créer une tâche pour télécharger les données Santé Publique France en mémoire

d. Extraire (i.e. filtrer) et renommer (i.e. hospitalisations) les méta-données pertinentes

e. Stocker les données dans un fichier JSON

f. Supprimer les données intermédiaires et le stores

hypothèse simplificatrice => se fixer à une date donnée, e.g. 30/05/2020

Exercices

➜ Solution

```
module.exports = {
  id: 'spf-donnees-hospitalieres',
  store: 'memory', // Default job store
  tasks: [{
    id: 'spf-donnees-hospitalieres-2020-05-30',
    type: 'http', // Download task
    options: {
      url: `https://www.data.gouv.fr/fr/datasets/r/63352e38-d353-4b54-bfd1-f1b3ee1cabd7`
    }
  }],
  hooks: {
    tasks: {
      after: {
        readCSV: {
          headers: true,
          delimiter: ';' // Default delimiter is ,
        },
        transformJson: {
          filter: { jour: '2020-05-30', sexe: '0' }, // Select total not men/women data
          mapping: {
            dep: 'code',
            hosp: 'hospitalisations'
          },
          unitMapping: {
            hospitalisations: { asNumber: true, empty: 0 } // Convert from string
          },
          pick: ['code', 'hospitalisations']
        },
        writeJson: {
          store: 'fs'
        }
      }
    },
    jobs: {
      before: {
        createStores: [{ // Input store
          id: 'memory'
        }, { // Output store
          id: 'fs',
          options: { path: __dirname }
        }]
      },
      after: { // Cleanup
        clearOutputs: {},
        removeStores: ['memory', 'fs']
      }
    }
  }
}
```

Exercices

➜ Spatialiser les données liées aux hospitalisations

a. Créer les “store” de type mémoire en entrée et fichier en sortie

b. Créer une tâche pour lire les données préalablement traitées en mémoire

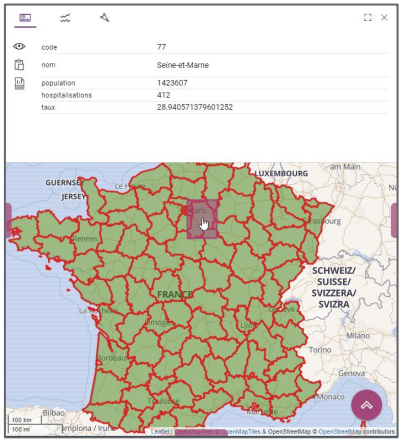
c. Créer une tâche pour lire les limites administratives avec population en mémoire

d. Fusionner les données avec les limites en fonction du code de département

e. Stocker les données spatialisées dans un fichier GeoJSON

f. Supprimer les données intermédiaires et le stores

Tester votre résultat sur https://kano.test.kalisio.xyz/ ou https://geojson.io/



Exercices

➜ Solution

```
module.exports = {
  id: 'hospitalisations-departements-2020-05-30',
  store: 'memory', // Default job store
  tasks: [{
    id: 'population-departements.json',
    type: 'store', // Read task
    options: {
      store: 'fs'
    }
  }, {
    id: 'spf-donnees-hospitalieres-2020-05-30.json',
    type: 'store', // Read task
    options: {
      store: 'fs'
    }
  }],
  hooks: {
    tasks: {
      after: {
        readJson: { features: true }
      }
    },
    jobs: {
      before: {
        createStores: [{ // Input store
          id: 'memory'
        }, { // Output store
          id: 'fs',
          options: { path: __dirname }
        }]
      },
      after: {
        mergeJson: {
          deep: true, // Add hospitalisations to departement features
          by: (item) => item.code || item.properties.code,
          transform: { pick: ['hospitalisations'] }
        },
        convertToGeoJson: {},
        writeJson: { store: 'fs' },
        clearOutputs: {}, // Cleanup
        removeStores: ['memory', 'fs']
      }
    }
  }
}
```

Exercices

➜ Modifier le traitement précédent pour calculer un taux d'hospitalisation

a. importer la librairie d'analyse et de traitement https://turfjs.org
b. appliquer une fonction calculant le taux sur chaque feature GeoJson en l'utilisant pour manipuler les features

➜ Créer un "job" permettant d'exécuter toutes les étapes

a. utiliser une tâche de type `'noop'` pour lancer le processus
b. utiliser un hook pour exécuter le "job" traitant les données SPF
   ⚠ chaque krawler utilisant le port `3030` par défaut il faudra éviter les conflits
c. procéder de même pour exécuter le "job" spatialisant les données d'hospitalisation
   ⚠ les tâches étant parallélisées par défaut il faudra s'assurer du séquencement

Tester vos résultat sur https://kano.test.kalisio.xyz/ ou https://geojson.io/

Exercices

➜ Solution taux d'hospitalisation

```
const turf = require('@turf/turf')

…
hooks: {
  jobs: {
    after: {
      …
      convertToGeoJson: {},
      apply: {
        dataPath: 'result.data',
        function: (geojson) => {
          turf.featureEach(geojson, (feature) => {
            const properties = feature.properties
            properties.taux = (100000 * properties.hospitalisations) / properties.population
          })
        }
      }
    }
  }
}
```

```
$env:NODE_PATH="D:\Development\kalisio\krawler\node_modules\"
```

Exercices

➜ Solution “job” avec toutes les étapes

```
module.exports = {
  id: 'job',
  options: { workersLimit: 1 },
  tasks: [{
    id: 'task',
    type: 'noop'
  }],
  hooks: {
    tasks: {
      after: {
        spf: {
          hook: 'runCommand',
          stdout: true, stderr: true,
          command: 'krawler spf-donnees-hospitalieres-jobfile.js --port 3031'
        },
        departements: {
          hook: 'runCommand',
          stdout: true, stderr: true,
          command: 'krawler taux-departements-jobfile.js --port 3031'
        }
      }
    },
    jobs: {
      before: {
      },
      after: {
      }
    }
  }
}
```

Exercices

➜ Rendre les différents “jobs” dynamiques au niveau de la date des données à traiter

a. avant l’exécution récupérer la date comme paramètre de la ligne de commande
☛ utiliser au besoin https://github.com/tj/commander.js pour manipuler les paramètres
b. paramétrer les hooks en fonction de cette date
☛ utiliser au besoin https://momentjs.com pour manipuler la date (e.g. formattage)

⚠ Définir la variable d’environnement `NODE_PATH` pointant sur le dossier `node_modules` de Krawler pour ce faire

➜ Automatiser le déploiement et l’exécution du traitement

a. créer une image contenant vos “jobs” à partir de l’image Krawler
b. exécuter le “job” principal en mode CRON dans un container

Tester vos résultat sur https://kano.test.kalisio.xyz/ ou https://geojson.io/

Exercices

➜ Solution “jobs” dynamiques

```
const moment = require('moment')
const cli = require('commander')

cli.option('-d, --date <date>', 'The date of the data to be generated', moment()).parse(process.argv)
const date = moment(cli.date)
if (!date.isValid()) {
  console.error('Invalid date, exiting')
  process.exit(1)
} else {
  console.log(`Processing data for ${date}`)
}

module.exports = {
  id: 'spf-donnees-hospitalieres',
  store: 'memory', // Default job store
  tasks: [{
    id: `spf-donnees-hospitalieres-${date.format('YYYY-MM-DD')}`,
    type: 'http', // Download task
    options: {
      url: `https://www.data.gouv.fr/fr/datasets/r/63352e38-d353-4b54-bfd1-f1b3ee1cabd7`
    }
  }],
  hooks: {
    tasks: {
      after: {
        readCSV: {
          headers: true,
          delimiter: ';' // Default delimiter is ,
        },
        transformJson: {
          filter: { jour: `${date.format('YYYY-MM-DD')}`, sexe: '0' }, // Select total not men/women data
          ...
```

Exercices

➜ Solution automatisation du traitement

Dockerfile

```
FROM kalisio/krawler:latest

WORKDIR /opt/krawler

# Install the jobs
COPY *jobfile* ./

# Run the job every night
CMD krawler --cron "0 0 0 * * *" --run all-jobfile.js
```

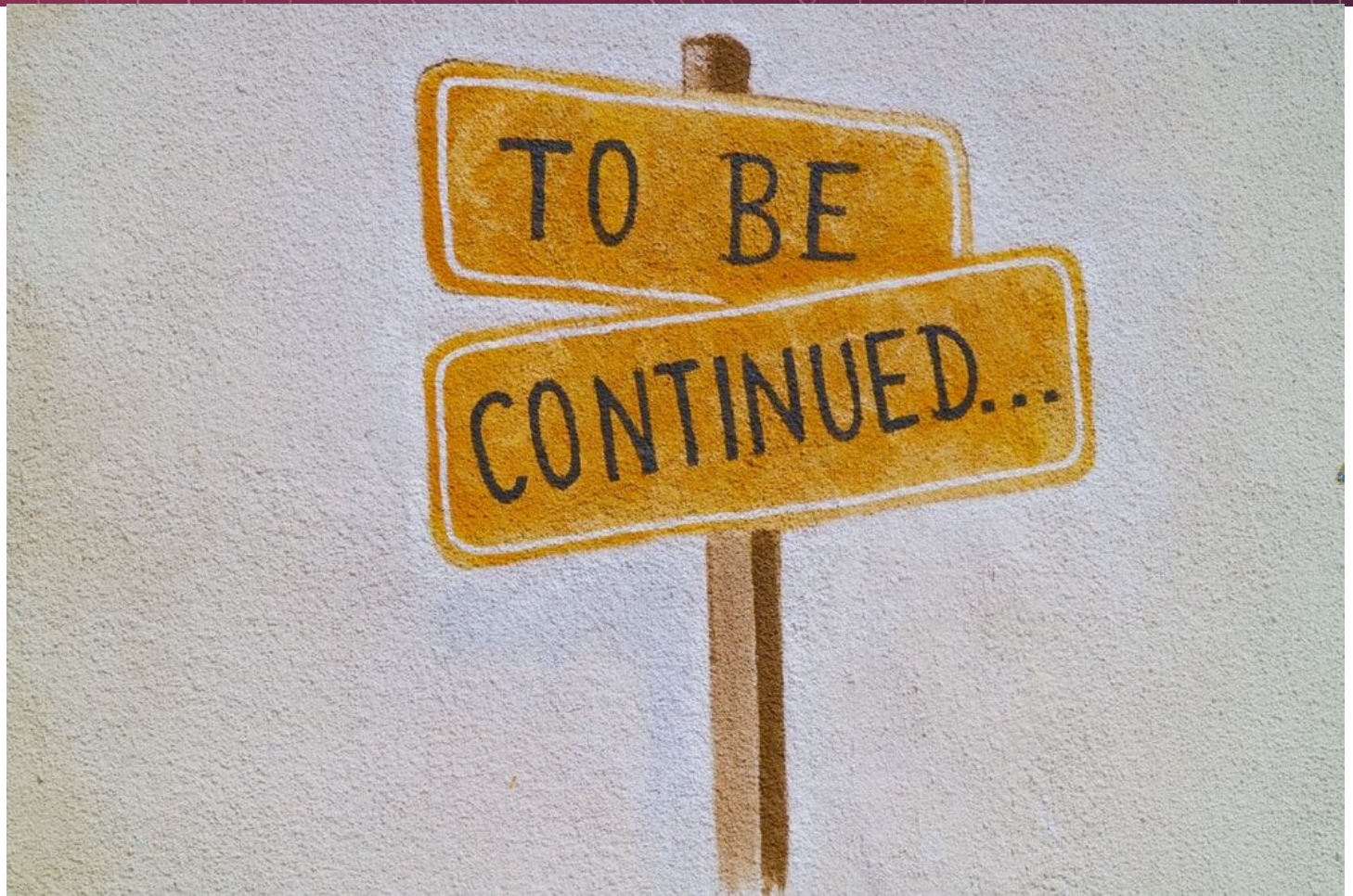
Build image

```
docker build -t covid19 .
```

Run container

```
docker run --name covid19 -d --rm -v D:\output:/opt/krawler covid19
```

Fil rouge



Spatial information that powers up your business

KALISIO