Open EDX & Pandasuite

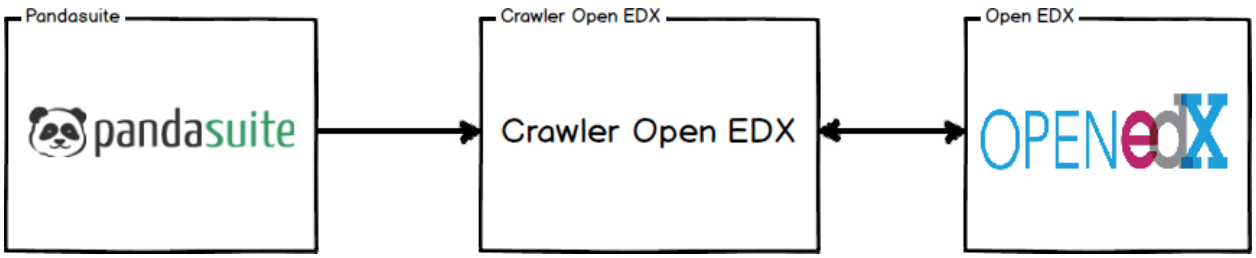
Générale

Le but est de rendre accessible, de façon transparente, le contenu de Open EDX dans la Pandasuite, permettant ainsi de générer simplement des cours hors ligne sur tablettes.

Comment ?

Grâce au crawler (cf: Open EDX Crawler), le contenu de Open EDX est rendu accessible à Pandasuite à travers des webservices qui permettent de rechercher le contenu par mots clefs.

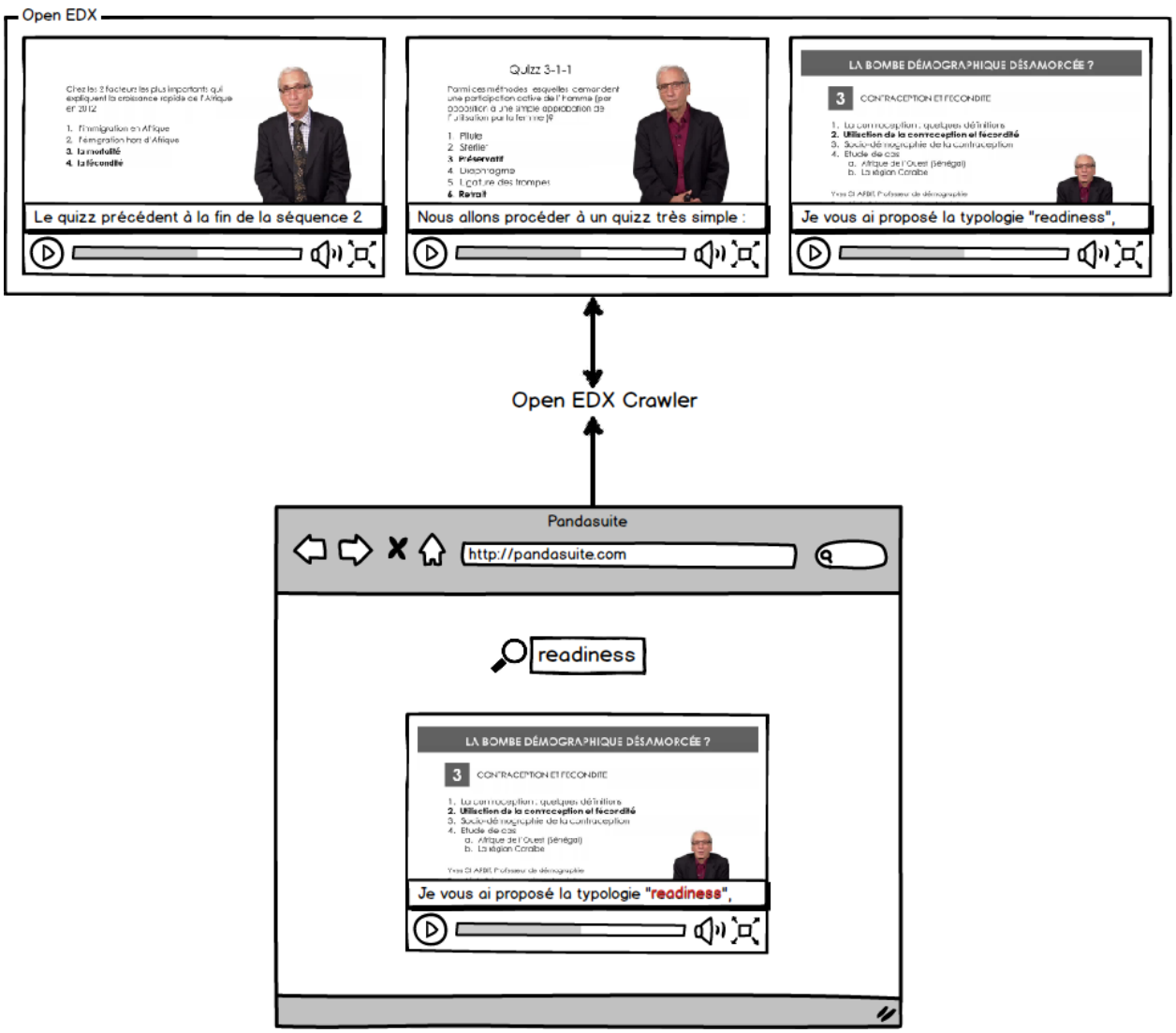
Ci-dessous un schéma explicatif des communications établies pour accéder au contenu :



Recherche par mots clefs

L'indexation faite par le crawler permet de rechercher le contenu par mots clef. l'indexation est faite sur les titres, chapitres, séquences, ainsi que sur les sous-titres.

Ci-dessous un schéma explicatif :


Open EDX
Le quizz précédent à la fin de la séquence 2
Quizz 3-1-1
Nous allons procéder à un quizz très simple :
LA BOMBE DÉMOGRAPHIQUE DÉSAMORCÉE ?
3 CONTRACEPTION ET FECONDITE
Je vous ai proposé la typologie "readiness",
Open EDX Crawler
Pandasuite
http://pandasuite.com
readiness
LA BOMBE DÉMOGRAPHIQUE DÉSAMORCÉE ?
3 CONTRACEPTION ET FECONDITE
Je vous ai proposé la typologie "readiness",

Open EDX crawler

Générale

Les technologies suivantes ont été utilisé pour récupérer et exposer le contenu de Open EDX :

- PhantomJS
- NodeJS
- Sidekiq
- Ruby On Rails
- Redis
- ElasticSearch
- MongoDB
- Nginx
- Docker
- Ubuntu server

Les conteneurs

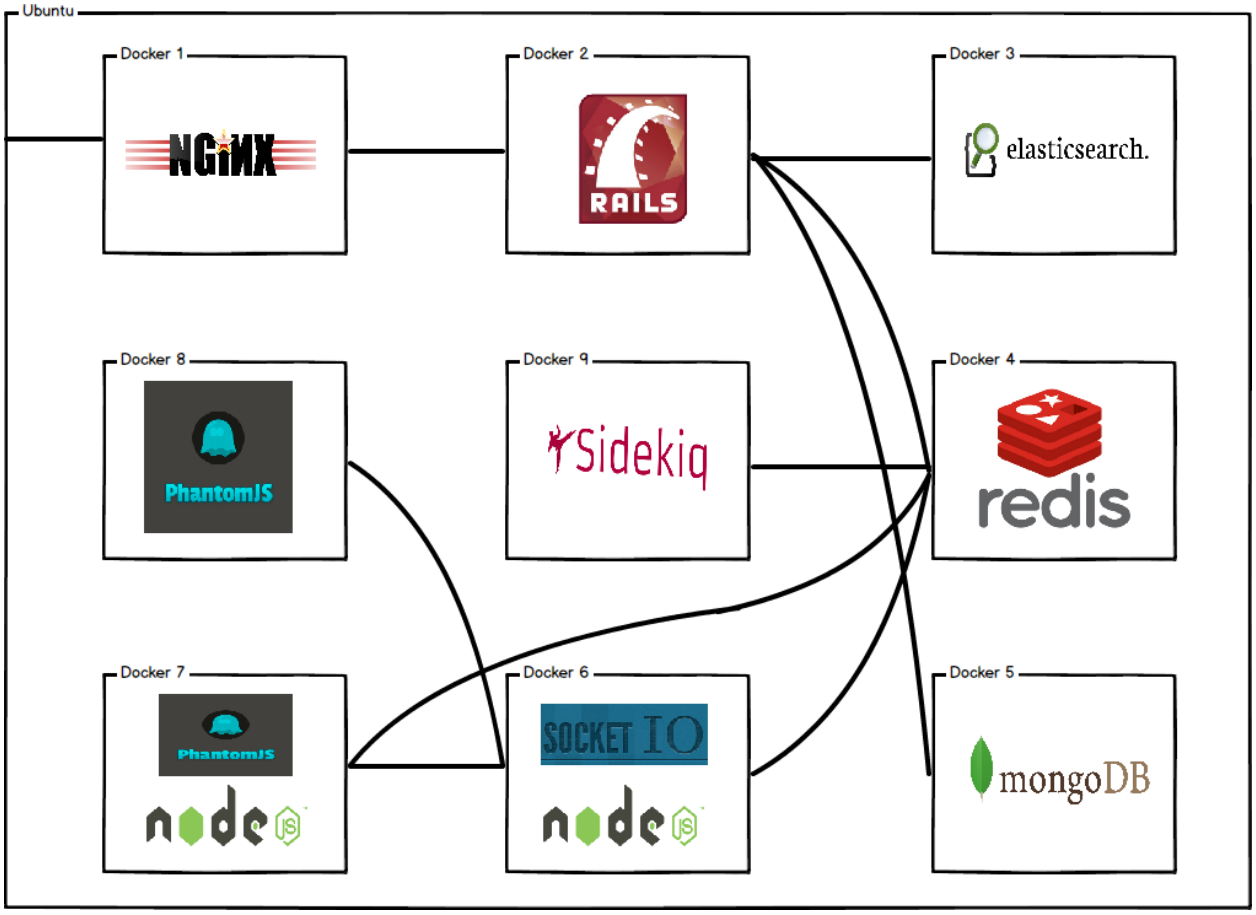
Les differentes parties du projet ont été isolées sous la forme de 9 conteneurs Docker, pour faciliter la scalabilité et la répartition de la charge et des flux, le tout, hébergé sur un machine de type Ubuntu server.

Les differents conteneurs ont été créés de la façon suivantes :

1. Nginx, un serveur http et reverse proxy haute performance, est utilisé pour lié l'API Rails avec l'internet.
2. Ruby On Rails, un puissant framework web ruby, est utilisé pour parcourir le contenu récupéré depuis Open EDX, ainsi que pour la supervision des jobs, lors de l'extraction du contenu depuis Open EDX, à travers une API communiquant en JSON.
3. ElasticSearch, un systeme distribué d'indexation de documents, est utilisé pour indexer le contenu récupéré depuis open EDX par mots clefs.
4. Redis, un serveur de structure de données clef-valeur avancé, est utilisé pour créer/sauvegarder l'état des jobs, ainsi que les données brut récupérées depuis Open EDX avant leur migration et indexation dans MongoDB et ElasticSearch.
5. MongoDB, un base de données de document JSON de type NOSQL, est utilisé pour stocker les données contenues dans Redis pour les rendre plus accessible.
6. NodeJS et Socket.io, un serveur web rapide de type asynchrone associé à une technologie websocket, est utilisé pour permettre aux instances de phantomJS de communiquer avec Redis.

7. NodeJS et PhantomJS, un serveur web rapide de type asynchrone associé à un navigateur Webkit, de type headless, accessible depuis une interface javascript, est utilisé pour instancier les jobs asynchrones qui vont récupérer le contenu depuis Open EDX.
8. PhantomJS, un navigateur Webkit, de type headless, accessible depuis une interface javascript, est utilisé pour récupérer la liste de tous les cours, qui seront ensuite parcourus par les jobs du conteneur précédant (7), en se faisant passer pour un navigateur standard.
9. Sidekiq, un systeme de processing en arrière plan permettant d'instancier des jobs asynchrone, est utiliser pour récuperer toute les vidéos ainsi que générer les thumbnails de ces dernières de façon asynchrone. Accessible depuis l'API Rails du conteneur 2.

Ci-dessous, un schéma représentant les differents modules, la répartitions des technologies, ainsi que les connexions entre eux.



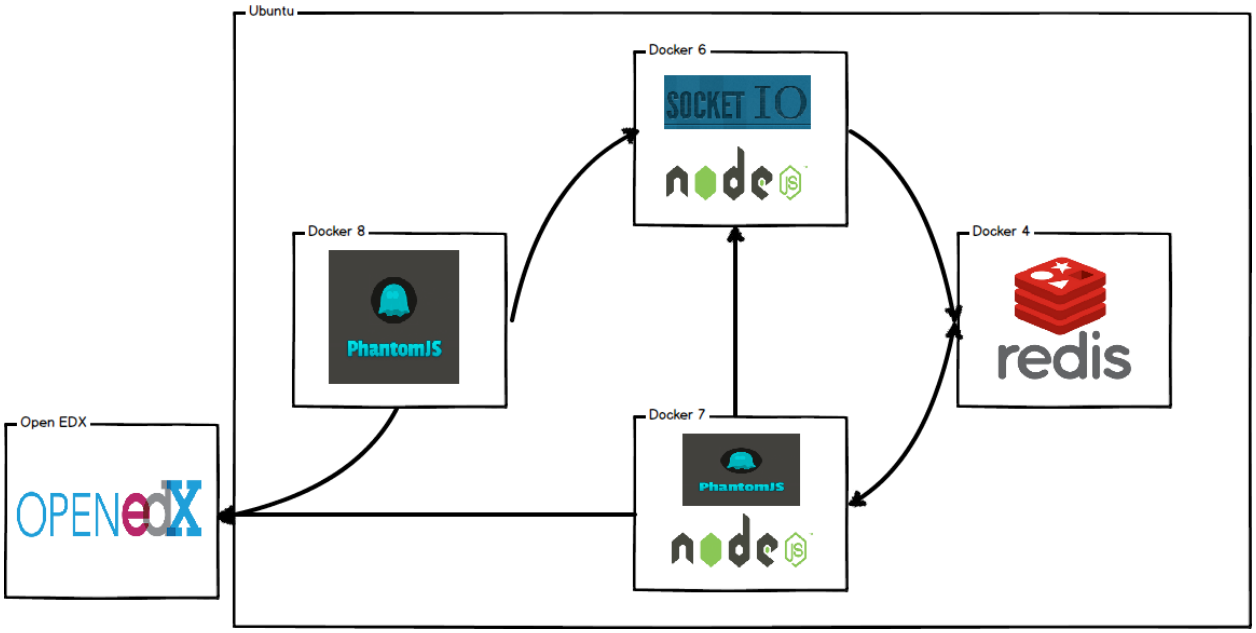
Les processes

1. Extraction du contenu

1. Le conteneur 8, va se connecter à Open EDX, en extraire la liste des cours, puis, en utilisant SocketIO à travers le conteneur 6 va les stocker à l'intérieur de redis.
2. Le conteneur 7, sera averti de cette nouvelle entrée et va instancier les jobs permettant de récuperer la liste des chapitres de chaque cours.
3. Chacun de ces jobs vont, à leur tours, communiquer, en utilisant SocketIO à travers le conteneur 6, avec Redis pour stocker la liste des chapitres.
4. Encore une fois, Le conteneur 7 sera averti de ces nouvelles entrées, et va instancier les jobs nécessaires pour récuperer le contenu de chaque chapitre.
5. Ces dernier jobs vont ensuite stocker le contenu des chapitres, en utilisant SocketIO à travers le conteneur 6, à l'intérieur de Redis.

Cette premiere étape génère plus de 2000 jobs, et dure plusieurs heures.
Elle impose une surveillance permanente de son état à cause de l'instabilité de PhantomJS, provoqué par son utilisation intensive.

Ci-dessous, un schéma représentant les differents modules, la répartitions des technologies, ainsi que les connexions entre eux.

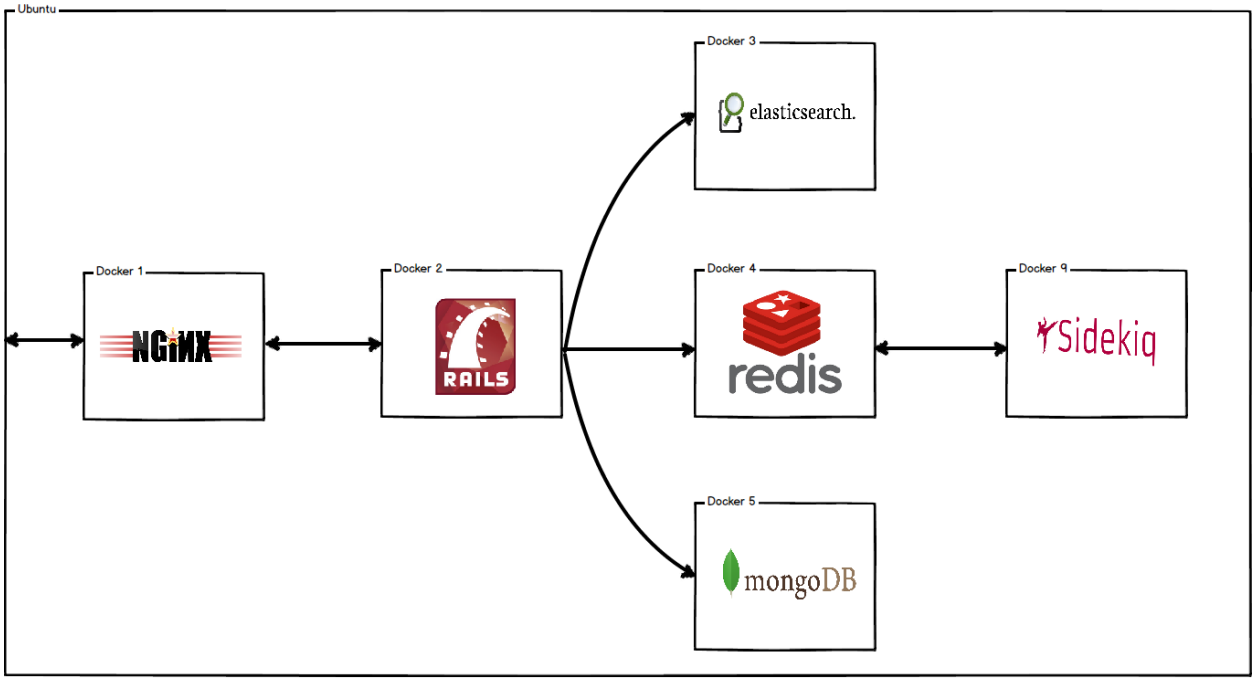


2. Migration du contenu

La migration du contenu se fait de Redis vers MongoDB et ElasticSearch.
La migration s'accompli en appelant une succession de web service créer dans le conteneur 2.

1. Un premier web service doit être appelé pour migrer toutes les données, générées par l'extraction du contenu, de Redis, vers MongoDB et ElasticSearch. Ce web service va, durant la sauvegarde des données dans MongoDB, indexer le contenu texte dans ElasticSearch, ainsi que récuperer les sous-titres des videos (si disponible).
2. Une fois terminé, un deuxieme web service doit être appelé pour instancier les jobs asynchrone (Sidekiq) qui vont télécharger les vidéos correspondant à chacune des entrées. Cette étape prend plusieurs heures et télécharge plus de 800 videos HD.
3. Une fois toutes les videos téléchargés, un dernier web service va instancier les jobs (Sidekiq) qui permettent générer les thumbnails de toutes les videos téléchargés. Encore une fois, cette étape prend plusieurs heures.

Ci-dessous, un schéma représentant les differents modules, la répartitions des technologies, ainsi que les connexions entre eux.



Les Données

Lorsque les données ont été migré depuis Redis dans MongoDB et ElasticSearch, elles sont accessible sous la forme suivante :


```
{
  "nb_courses": 1125,
  "nb_videos": 1125,
  "courses": [
    {
      "_id": {"$oid": "53be8dfe3962360176000000"},
      "chapter": "Tutoriel - Charte d'Ottawa",
      "course": "Université Montpellier 2: Ville durable : être acteur du changement",
      "id": "53be8dfe3962360176000000",
      "part": "{\"html\":[],\"title\":\"Tutoriel - Charte
d'Ottawa\",\"videos\":{\"HD\":\"https://…\",\"smartphone\":\"https://…\",\"standard\":\
"https://…\"},\"videosub\":{\"json\":\"\",\"text\":\"\",\"urljson\":\"/c4x/….srt.sjson\
"}}",
      "session": "Semaine 3 : La santé et l'environnement dans la ville",
      "subtitle": "",
      "video_thumbnail_url": null,
      "video_url": null,
      "video_url_ori": "..."
    },
    {
      "_id": {"$oid": "53be8dff3962360176010000"},
      "chapter": "Video - Le lien entre environnement et santé dans la ville",
      "course": "Université Montpellier 2: Ville durable : être acteur du changement",
      "id": "53be8dff3962360176010000",
      "part": "{\"html\":[\"\\n <section class=\\\"xmodule_display xmodule_HtmlModule\\\"
data-type=\\\"HTMLModule\\\">\\n <a href=\\\"/c4x/….pdf\\\"
target=\\\"_blank\\\">Diaporama</a> en pdf (ouverture dans une nouvelle
fenêtre).\\n</section>\\n\\n  \"],\"title\":\"Video - Le lien entre environnement et
santé dans la
ville\",\"videos\":{\"HD\":\"https://…\",\"smartphone\":\"https://…\",\"standard\":\"…\
"},\"videosub\":{\"json\":\"\",\"text\":\"\",\"urljson\":\"/c4x/….srt.sjson\"}}",
      "session": "Semaine 3 : La santé et l'environnement dans la ville",
      "subtitle": "",
      "video_thumbnail_url": null,
      "video_url": null,
      "video_url_ori": "https://..."
    },
    {...}
  ]
}
```