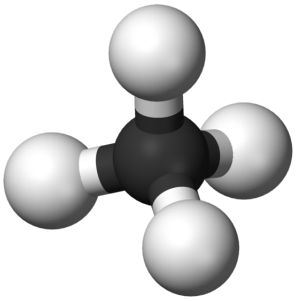
Université d'Angers



Faculté des sciences

Département Informatique

Année 2020/2021

Cahier des spécifications techniques QuChemPedIA

Chefs de projet :
Boudet Alexandre
Périchet Thomas
Rafiki Younes

Professeurs référents :
Da Mota Benoît
Genest David

Client :
Cauchy Thomas

30 Septembre 2020

Table des matières

1 Introduction

1.1 Contexte

Le projet QuChemPedIA est un projet de grande envergure regroupant plusieurs acteurs de différents axes qui, durant ces deux dernières années, ont eu pour but de répondre à une demande bien précise qui était de proposer une alternative plus performante, polyvalente et collaborative aux rares solutions existantes de consultation de calcul sur des molécules scientifiques.

L'objectif étant de répondre à des problématiques complexes de calcul et apporter des fonctionnalités intuitives pour faciliter la navigation aux utilisateurs non informaticiens, tout en préservant la qualité des informations contenues sur le site.

En effet, les plateformes de consultations en ligne de molécules existes, néanmoins, aucune ne correspondait entièrement aux besoins des chimistes, d'où l'idée de QuChemPedIA, une plateforme collaborative, permettant aux chimistes de contribuer, de rechercher et de comparer leurs travaux avec d'autres sur le site.

Le site web présente plusieurs fonctionnalités qui sont les suivantes :

- Importation de nouvelles molécules dans la base de données
- Recherche de molécules dans la base de données
- Calcul de molécules
- Visualisation détaillée d'une molécule

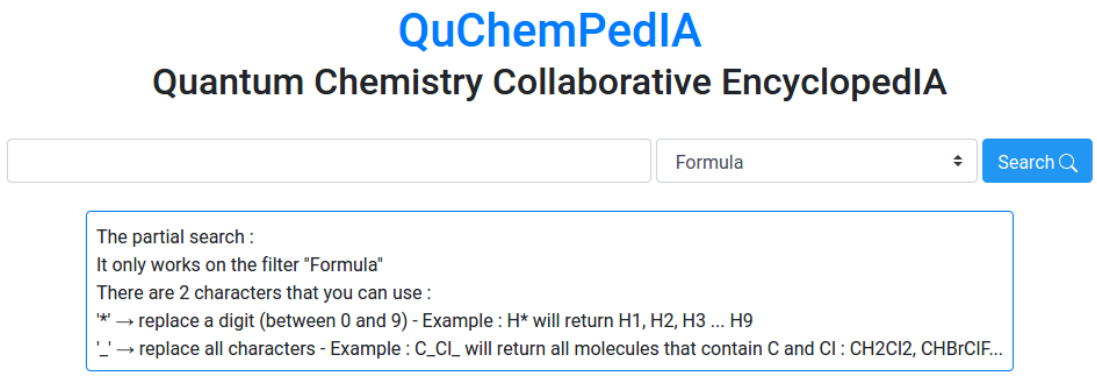


FIGURE 1 – La page d'accueil du site QuChemPedIA

L'année dernière, nous avions eu ce projet à réaliser en collaboration avec les Master 2 qui étaient nos chefs de projet. Mais durant le développement, il y a eu des problèmes comme :

- Un problème de complexité au niveau du *front-end* du site ;
- Une architecture de projet non conventionnelle qui porte préjudice à la maintenabilité du site web ;
- Des soucis de configuration d'environnement sur la machine virtuelle de test ;
- Un manque de communication avec les chefs de projet ;
- Un manque de compétences des chefs de projet sur les technologies utilisées

La réalisation de ce projet avait commencé il y a quelques années et chaque contributeur de QuChemPedIA rajoutaient des nouvelles technologies sans que le projet d'origine ne soit terminé. Cela a mené à des cohabitations de plusieurs frameworks qui ont rendues le projet très compliqué à reprendre pour de futurs étudiants.

1.2 Exigence des clients

Le client, Thomas Cauchy, ainsi que Benoit Da Mota ont opté pour une refonte intégrale du projet et des technologies ainsi qu'une baisse des fonctionnalités (Plus de création/-gestion d'utilisateurs, des changements mineurs).

Notre objectif cette année sera donc d'avoir **un site testé et fonctionnel. Permettant ainsi aux futurs étudiants de pouvoir continuer à développer ce projet sans difficultés**, tout en utilisant des technologies faciles à appréhender et pérennes.
Un premier cahier des charges a donc été réalisé :

- La refonte du *front-end* : On doit garder la même charte graphique que l'ancien site avec comme contrainte le choix d'une technologie plus intuitive ;
- Créer un système de "recherche partielle" : Lorsqu'un utilisateur saisit une recherche ;
- Réaliser une API REST (pour*Application programming interface Representational state transfer* pour la consultation des données : Cette API doit être accessible pour tout le monde (et donc publique) ;
- Réaliser une API REST afin d'insérer ou supprimer des molécules dans la base de données : Cette API doit être accessible uniquement par certaines personnes (et donc privée). Cette fonction n'est pas obligatoire mais on a décidé de la développer malgré tout ;
- Mettre en place l'intégration continue : Il faut utiliser un outil fiable ;
- Et enfin documenter : Pour guider les personnes qui vont reprendre le projet QuChemPedIA ;

2 Choix du langage et des technologies

2.1 Framework

Le choix des technologies s'avère cruciale pour le projet. En effet, cela va déterminer l'aboutissement du projet. Tout d'abord, nous devons choisir un langage identique pour chaque grande partie du projet afin d'assurer une cohérence globale. Les années précédentes, le projet était réalisé en *Python*. Nous avons décidé de ré-utiliser ce langage car il est simple et pratique pour concevoir des API dynamique (au même titre que *Express JS*).

Pour l'API de consultation ainsi que l'API d'insertion et suppression (ou administration), nous avons choisi d'utiliser *Flask* . C'est un framework *Python* très simple à prendre en main et beaucoup plus minimaliste que son grand frère *Django*. Au départ, nous avions choisi d'utiliser *Django* en accord avec le projet de l'année dernière. Mais nous nous sommes rendus compte qu'il était beaucoup trop lourd et beaucoup trop gourmand pour réaliser des API pour ce projet.

Nous avons hésité à utiliser *Vue.js*, un framework *front-end* permettant de construire une interface utilisateur. Mais *Vue.js* est assez long à prendre en main et complexe. Nous avons donc décidé d'utiliser uniquement de *l'HTML*, du *JavaScript* ainsi que des frameworks *CSS* comme *Boostrap* etc.

Pour la base de données, nous sommes restés sur *ElasticSearch*, un logiciel permettant l'indexation et la recherche de données s'effectuant grâce à une API REST dans un format JSON. Utiliser une base de données relationnelle n'est pas une bonne idée au vu du volume de données que nous aurons à stocker.

2.2 Outils de test

2.2.1 Pytest

Pytest est la librairie standard consacrée aux tests unitaires sur des applications Python. La rédaction des tests est assez intuitive et facile à implémenter

2.3 Continuous Integration

Concernant l'intégration continue nous avons réfléchi de manière à ne pas ajouter trop de nouvelles technologie au projet. C'est pourquoi nous nous sommes renseigné pour homogénéiser un maximum certains traitement.

C'est pour cela que nous avons retenu l'intégration continue par le biais de GitHub, elle sera donc hébergée au même endroit que le projet QuChemPedIA.

Sur GitHub la partie CI se situe dans le repository dans la section *Actions* et une instance de CI est appelée un *Workflows*.

Voici un le fichier CI (*.github/workflows/python-app.yml*) du projet QuChemPedIA :

```
name: Continuous Integration QuChemPedIA

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: Set up Python 3.8
      uses: actions/setup-python@v2
      with:
        python-version: 3.8
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install flake8 pytest
        if [ -f requirements.txt ]; then pip install -r requirements.txt;
        ↪ fi
    - name: Lint with flake8
      run: |
        # stop the build if there are Python syntax errors or undefined
        ↪ names
        flake8 . --count --select=E9,F63,F7,F82 --show-source
        ↪ --statistics
        # exit-zero treats all errors as warnings. The GitHub editor is
        ↪ 127 chars wide
        flake8 . --count --exit-zero --max-complexity=10
        ↪ --max-line-length=127 --statistics
    - name: Test with pytest
      run: |
        pytest
```

3 Architecture du projet

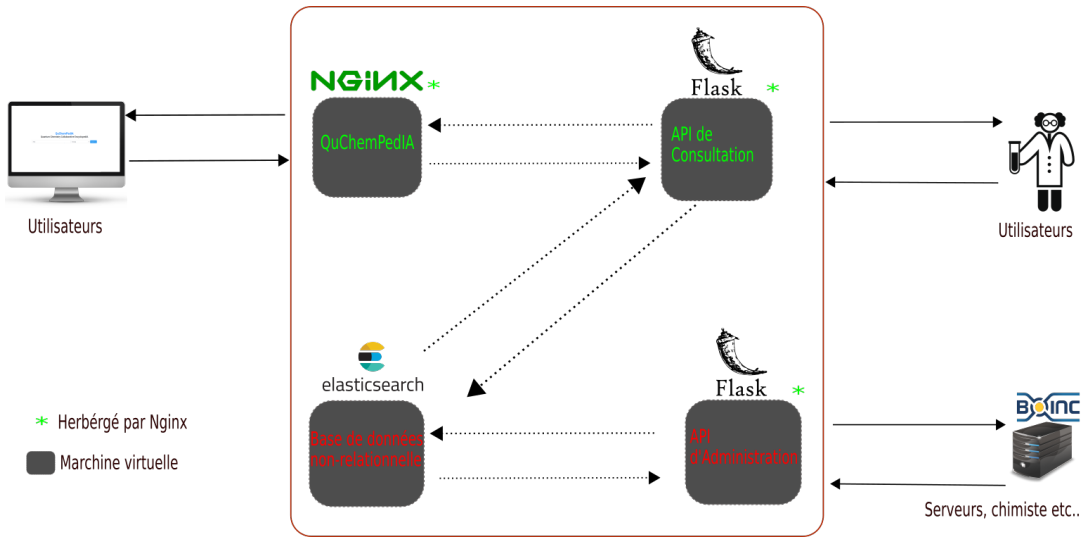


FIGURE 2 – La structure de QuChemPedIA après la refonte

Voici, ci-dessus, la structure globale du projet QuChemPedIA après la refonte. Nous pouvons observer quatre cadres gris dans le grand ensemble rouge. Les cadres gris représentent des machines virtuelles. En effet, nous avions découpé le projet en quatre parties bien distinctes :

- **La partie Front**
  C'est la machine virtuelle qui contient toutes les pages HTML de QuChemPedIA. C'est un serveur *Nginx* qui héberge le site web ;
- **La partie Base de données**
  C'est la machine virtuelle qui va herbergé une base de données de type *Non relationnelle Elasticsearch* ;
- **L'API de Consultation**
  C'est l'application qui va faire le lien entre la partie front et la base de données *Elasticsearch* ;
- **L'API d'Administration**
  C'est l'application qui va permettre d'ajouter, de consulter ou bien de supprimer des molécules dans la base de données *Elasticsearch* ;

Tout comme la partie Front, l'API de Consultation et d'Administration sont herbergées sur *Nginx*. Pour des raisons de sécurité, nous avons séparé en deux catégories les machines virtuelles :

- La machine qui contient le site web ainsi que l'API de Consultation sont accessibles en publique. En effet, n'importe qui peut accéder à ces deux applications. Un utilisateur pourrait récupérer les résultats d'une recherche via l'API de Consultation sans passer par le site web (afin qu'il puisse utiliser les données récoltés à sa façon) ;
- Afin d'empêcher l'accès à la base de données aux utilsateurs non désirés, nous devions rendre en privée l'API d'Administration ainsi que la base de données *Elastic-Search*.

3.1 API de Consultation

Voici la documentation de l'API de Consultation avec les différentes routes. La documentation ainsi que le nommage des routes seront en anglais (étant une demande du client *Thomas Cauchy*).

La route /api/search

GET /api/search - *Retourne une liste de molécules*			
Paramètres			
Paramètres	**Valeur**	**Description**	**Type**
q	ex : "h2o","h*o","h_"	Recherche de l'utilisateur de molécules	String
type	["formule","inchi",smile"]	Type de recherche	String
page	ex : 1,2,3...	Numérotation de la page	Int
showresult	ex : 5, 10, 15...	Nombre de résultat par page	Int

Code retour	
Code retour	**Description**
200	La route retourne bien une liste de molécules
400	Mauvaise requête, les paramètres ne sont pas bons
404	Aucune molécule n'a été trouvé
500	Erreur serveur

Recherche partielle

Une demande du client était d'implémenter une fonctionnalité permettant à l'utilisateur de réaliser des recherches partielles.

- '*' : Ce caractère correspond à au moins un chiffre. Par exemple, si l'utilisateur souhaite rechercher toutes les molécules comprenant au moins un atome d'hydrogène, il devra écrire 'H*'.
- '_' : Ce caractère correspond à n'importe quel résultat. Par exemple, si l'utilisateur souhaite rechercher toutes les molécules commençant par CH, il devra saisir 'CH_' qui retournera donc CHN, CHCO2, CH2Cl2 etc. L'utilisateur peut très bien saisir 'CH_Cl_' qui retournera l'ensemble des molécules commençant par CH et qui inclus Cl.

Exemple d'utilisation avec CURL

Dans les exemples ci-dessous, nous sommes en local et par conséquent l'adresse du lien sera 127.0.0.1 :5000.
On souhaite rechercher des molécules par la formule *CHN*, on veut simplement afficher la première page tout en précisant qu'on retourne que deux résultats par page.

- q = CHN ;
- type = formula ;
- page = 1 ;
- showresult = 1 ;

```
curl -X GET "http://127.0.0.1:5000/api/search?type=formula&q=chn&page=1&showresult=2"
```

Résultats :


```
{
    "data": [
        {
            "charge": 0,
            "formula": "CHN",
            "id": "C0tO7HUBOSF22BvMoYMo",
            "inchi": "InChI=1S/CHN/c1-2/h1H",
            "job_type": "[\"FREQ\", \"OPT\"]",
            "list_theory": "[\"DFT\"]",
            "multiplicity": 1,
            "nb_heavy_atoms": 2,
            "smi": "N#C",
            "solvent": "Gas"
        },
        {
            "charge": 0,
            "formula": "CHN",
            "id": "XvFO7HUBkjVcihM6oca9",
            "inchi": "InChI=1S/CHN/c1-2/h1H",
            "job_type": "[\"OPT_ES\"]",
            "list_theory": "[\"DFT\"]",
            "multiplicity": 1,
            "nb_heavy_atoms": 2,
            "smi": "[N][CH]",
            "solvent": "Gas"
        }
    ],
    "total": 16
}
```

Si aucune molécule correspond à la recherche :

```
{
  "Error": "Molecule does not exist"
}
```

Si les paramètres ont été mal saisit :

```
{
  "Error": "Something is missing please check your URL"
}
```

Cette route nous retourne donc une liste de résultats correspondant à la recherche effectuée. Il faut savoir que ce n'est pas l'intégralité du Json de la molécule mais seulement les principaux attributs. Si l'utilisateur souhaite avoir le détail d'une molécule, il lui suffit d'utiliser la route ci-dessous avec l'identifiant correspondant. On peut aussi avoir accès au nombre total de résultat trouvé via l'attribut *total*.

GET /api/details - *Retourne le détail d'une molécule*			
Paramètres			
Paramètres	**Valeur**	**Description**	**Type**
id	"sDgM1HQBOSF22BvM3FOz"	Identifiant Elasticsearch d'une molécule	String

Code retour	
200	La route retourne bien la molécule
400	Mauvaise requête, le paramètre est mal renseigné
404	Aucune molécule n'a été trouvé
500	Erreur serveur

Exemple d'utilisation avec CURL

On souhaite rechercher une molécule par son identifiant ElasticSearch qui correspond à la molécule CHBrClF.

- id = A0tO7HUBOSF22BvMl4MP ;

```
curl -X GET "http://127.0.0.1:5000/api/details/A0tO7HUBOSF22BvMl4MP"
```

Résultats :

```
{
    "data": {
        "comp_details": {
            "excited_states": {},
            "freq": {
                "temperature": 298.15
            },
            "general": {
                "all_unique_theory": [
                    "DFT"
                ],
              ...
    },
    "id": "A0tO7HUBOSF22BvMl4MP"
}
```

Si aucune molécule correspond à la recherche :

```
{
  "Error": "Molecule with id = 'A0tO7HUBOSF2BvMl4MP' does not
    exists!"
}
```

Si les paramètres ont été mal saisit :

```
{
  "Error": "Something is missing please check your URL"
}
```

3.2 API d'Administration

Voici la documentation de l'API d'Administration avec les différentes routes. La documentation ainsi que le nommage des routes seront en anglais (étant une demande du client *Thomas Cauchy*).

GET /api/search - *Retourne une liste de molécules*			
Paramètres			
Paramètres	**Valeur**	**Description**	**Type**
q	ex : "h2o"	Recherche de l'utilisateur de molécules	String

Code retour	
200	La route retourne bien une liste de molécules
400	Mauvaise requête, les paramètres ne sont pas bons
404	Aucune molécule n'a été trouvé
500	Erreur serveur

GET /api/details - *Retourne le détail d'une molécule*			
Paramètres			
Paramètres	**Valeur**	**Description**	**Type**
id	"sDgM1HQBOSF22BvM3FOz"	Identifiant Elasticsearch d'une molécule	String

Code retour	
200	La route retourne bien la molécule
400	Mauvaise requête, le paramètre est mal renseigné
404	Aucune molécule n'a été trouvé
500	Erreur serveur

POST /api/add - *Ajoute une molécule dans la base de données ainsi que son fichier log*			
Paramètres			
Paramètres	**Valeur**	**Description**	**Type**
mol_json	ex : h2o.json	Insère dans la base de données la molécule	Json
mol_log	ex : freq.log	Place dans une arborescence le fichier log	File

Code retour	
200	La molécule et le fichier de log ont bien été ajoutés
400	Mauvaise requête, les paramètres ne sont pas bons
404	La molécule et le fichier log n'ont pas été inséré
500	Erreur serveur

DELETE /api/delete - *Supprime une molécule de la base de données et le fichier log*			
Paramètres			
Paramètres	**Valeur**	**Description**	**Type**
id	"sDgM1HQBOSF22BvM3FOz"	Identifiant Elasticsearch de la molécule à supprimer	String

Code retour	
200	La molécule et le fichier log ont bien été supprimés
400	Mauvaise requête, les paramètres ne sont pas bons
404	La molécule et le fichier log n'ont pas pu être supprimés
500	Erreur serveur

4 Déploiement

Le déploiement des trois applications (API Administration, API Consultation, Web) doit s'effectuer sur des machines virtuelles fournit par M. Da Mota.

Nous avons décidé d'utiliser Ansible pour réaliser un déploiement sous forme de recettes automatisées. Une fois la recette réalisée il suffit de lancer Ansible pour créer l'environnement de production.

4.1 Machine virtuelle

Si un redéploiement sur une nouvelle VM vierge est nécessaire il y a une pré-installation à effectuer :

1. Connectez-vous en root à la VM en SSH
2. Copier - Coller la recette Ansible dans l'espace utilisateur root
3. Avant tout pour que l'installation d'Ansible se déroule correctement il faut installer :

   apt install gnupg2

4. Ensuite installer Ansible via le lien suivant :

   `https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html#installing-ansible-on-debian`

5. La VM est désormais opérationnelle et n'attend plus que la lancement de la configuration Ansible

4.2 Accès au log des API

Pour les accès aux logs des API :

API Administration

```
journalctl -u api_administration.service
```

API Consultation

```
journalctl -u api_consultation.service
```

Voici les machines virtuelles mise à notre disposition :

Machine virtuelle pour le Web (Statique)	
Local hostname	quchempedia-pub
Local IP	192.168.0.204
Connexion SSH	ssh -p 2225 root@quchempedia.univ-angers.fr
Port SSH en local	ssh root@192.168.0.204
Chiffrement HTTPS sur port	80 et 443
Règles rproxy (provisoires)	https ://quchempedia.univ-angers.fr/reboot/
URL de production	https ://quchempedia.univ-angers.fr/
Montage disque réseau	/mnt/data_dir/ pour logs des molécules

Machine virtuelle pour l'API Administration	
Local hostname	quchempedia-pri
Local IP	192.168.0.205
Connexion SSH	ssh -p 2226 root@quchempedia.univ-angers.fr
Port SSH en local	ssh root@192.168.0.205
Chiffrement HTTPS sur port	8080
Règles rproxy (provisoires)	https ://quchempedia.univ-angers.fr :8080/private_api/
Montage disque réseau	/mnt/data_dir/ pour logs des molécules

Machine virtuelle pour l'API Consultation	
Local hostname	quchempedia-api
Local IP	192.168.0.206
Connexion SSH	ssh -p 2228 root@quchempedia.univ-angers.fr
Port SSH en local	ssh root@192.168.0.206
Règles rproxy (provisoires)	https ://quchempedia.univ-angers.fr/api/

Machine virtuelle pour Elastic Search	
Local hostname	elasticsearch
Local IP	192.168.0.102
Connexion SSH	ssh -p 2227 root@quchempedia.univ-angers.fr
Port SSH en local	ssh root@192.168.0.102
Montage disque réseau	/mnt/data_elasticsearch/ pour les données

4.3 Ansible

Pour la réalisation d'une recette avec Ansible nous nous sommes appuyés sur ce tutoriel en français de Grafikart qui explique la base de la configuration.

`https://www.youtube.com/watch?v=DwNapBHypE8`

En outre, voici la documentation d'Ansible :

`https://docs.ansible.com/ansible/latest/index.html`

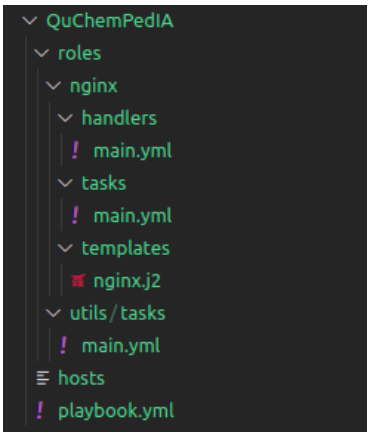


FIGURE 3 – Exemple architecture pour la recette Ansible Web (Statique)

Pour ce qui est de la documentation de la recette elle reste en soit assez intuitive étant donné que c'est juste de la configuration.

Il suffit de retranscrire une installation basique. Par exemple pour les API sous Flask nous avons suivi le tuto (Recherche Google) :

How to serve Flask applications with gunicorn and nginx on Ubuntu - Digital Ocean

Pour la VM d'Elastic Search il n'y a pas de recette Ansible étant donné que c'est juste de l'installation :

1. Installer ES en suivant l'URL suivante :

   `https://www.elastic.co/guide/en/elasticsearch/reference/current/deb.html`

2. Ensuite il faut modifier le fichier de configuration d'ES situé '/etc/elasticsearch/elasticsearch.yml', ensuite modifier les variables suivantes :

   network.host = {{ mettre l'IP de la machine }}

   http.port : = {{ PORT }}

   discovery.type : single-node

hosts

Le fichier hosts contient l'adresse des VM sur lesquelles on va déployer. Soit on indique une IP ou alors ou peut exécuter la recette en local en indiquant 'localhost'.

playbook.yml

Le playbook est le fichier principal c'est ici que l'on définit les rôles que nous allons utiliser pour procéder à l'installation. Dans l'exemple ci-dessus nous avons deux roles qui sont 'nginx' et 'utils'.

roles

Le dossier 'rôles' contient chaque grande étape du déploiement. Créer des rôles permet de clarifier l'architecture du projet Ansible.

Chaque rôle doit contenir un dossier 'tasks' avec son fichier 'main.yml' et c'est dans ce dernier que l'on défini les task du rôle en question.

(Facultatif) On peut définir un dossier 'handlers' qui contient des tasks réutilisable en appelant ce dernier dans le 'main.yml' à l'aide de 'notify'.

(Facultatif) Enfin on peut apercevoir le dossier templates qui contient des fichiers que l'on va pouvoir copier coller depuis la recette vers la VM. On peut utiliser des variables grâce au moteur de template Jinja intégré à Ansible.

Lancer la recette Ansible

Pour lancer Ansible il faut faire la commande suivante :

```
ansible-playbook -i hosts playbook.yml
```

Si tout se passe bien la recette s'exécutera sans problème.

A noter que la recette mettra à jour le projet en clonant à chaque fois la dernière version de la branche indiquée dans la configuration. Ici c'est la branche 'feat-deploy'.

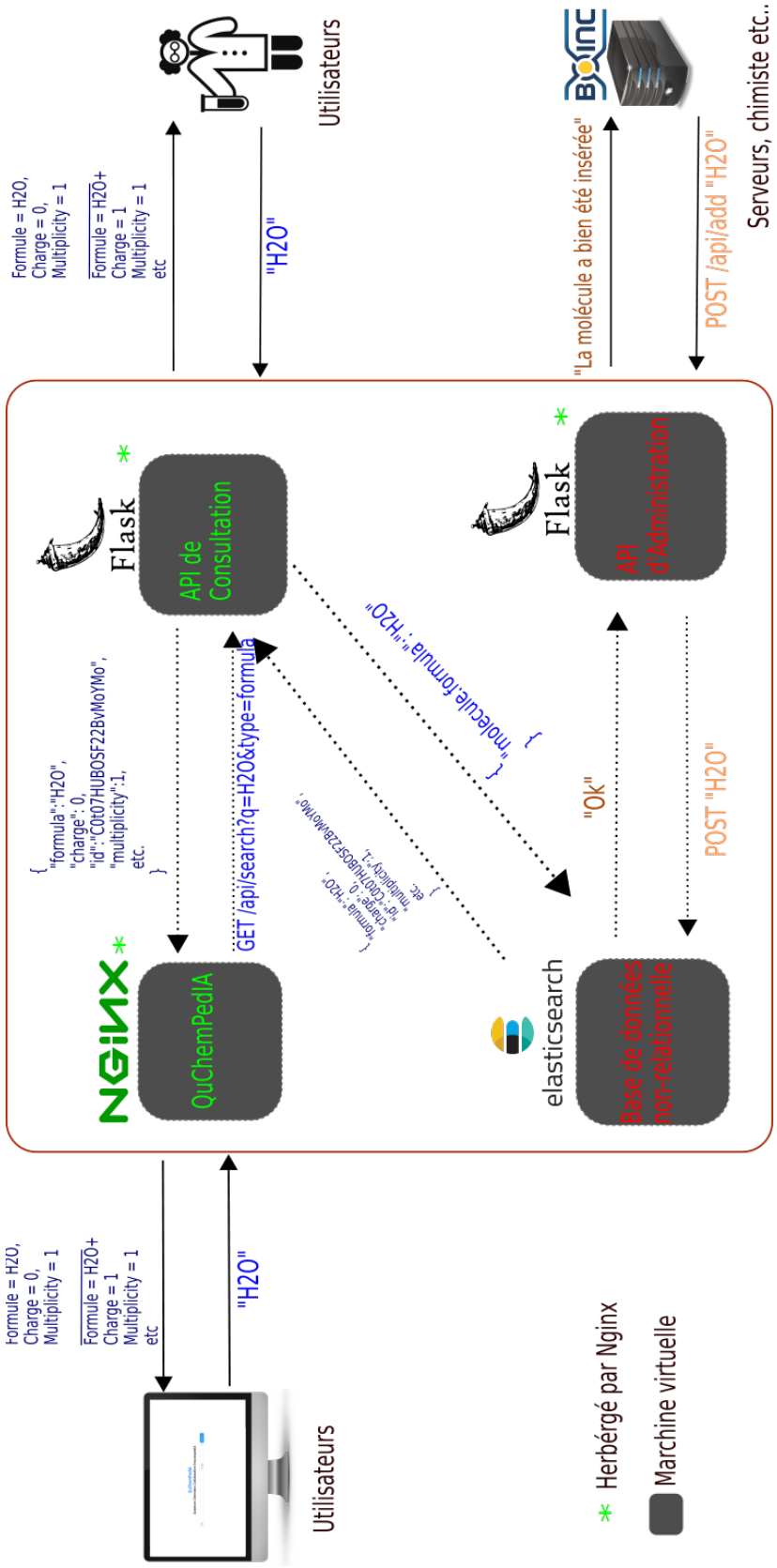
```
- name: Clonage du site {{ domain }} depuis {{ repo }}
  become: yes
  git: dest=/home/{{ user }}/{{ domain }} key_file="~/.ssh/id_rsa_api" accept_hostkey=yes clone=yes repo={{ repo }} version=feat-deploy force=yes
```

FIGURE 4 – Configuration Git dans le fichier 'roles/nginx/tasks/main.yml'

Ansible intègre une fonction de détection de changement, si on relance une recette qui à déjà été lancée alors il ne va pas ré-exécuter les tâches qui n'ont pas bougées. C'est pour cela qu'il ne faut pas hésiter à relancer Ansible à chaque modification de configuration ou alors si le projet GitHub a été modifié.

Ne pas oublier de restart le service sur la VM correspondante après une maj selon l'API : **systemctl restart api_{administration/consultation}.service**

Annexe


Utilisateurs
Formule = H2O,
Charge = 0,
Multiplicity = 1
Formule = H2O+
Charge = 1
Multiplicity = 1
etc
"H2O"
Serveurs, chimiste etc..
"La molécule a bien été insérée"
POST /api/add "H2O"
Flask
API de Consultation
Flask
API d'Administration
NGINX
QuChemPedIA
elasticsearch
Base de données non-relationnelle
GET /api/search?q=H2O&type=formula
"Ok"
POST "H2O"
Herbérgé par Nginx
Marchine virtuelle