Proyecto Opcional

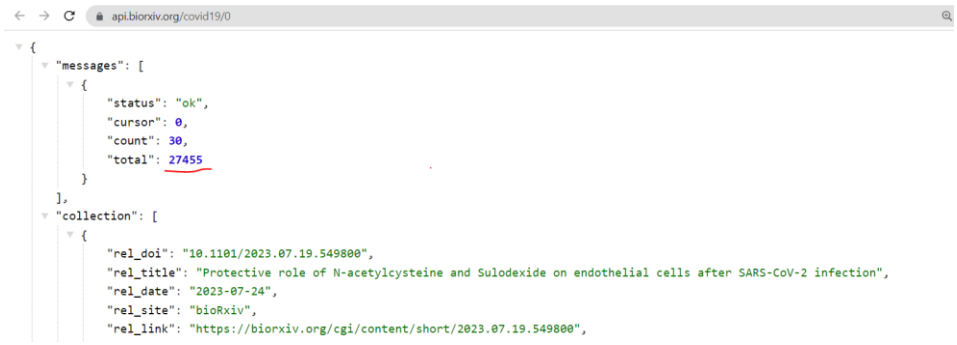
Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
Redes (IC 4302)
Segundo Semestre 2023

1. **Objetivo General**
   - ➔ Desarrollar habilidades en tecnologías y lenguajes de programación que serán utilizados durante el curso.
2. **Objetivos Específicos**
   - ➔ Implementar un REST API crawler utilizando Python.
   - ➔ Desarrollar scripts de procesamiento de datos mediante el uso de Spark SQL.
   - ➔ Implementar un sistema de búsqueda de artículos científicos de Covid19.
   - ➔ Automatizar una solución mediante el uso de Docker, Kubernetes y Helm Charts.
   - ➔ Desarrollar arquitecturas de microservicios en Kubernetes.
   - ➔ Implementar servicios básicos de extracción de entidades.
   - ➔ Instalar y configurar servicios de bases de datos NoSQL.
3. **Datos Generales**
   - ➔ El valor del proyecto: 15%
   - ➔ Nombre del proyecto: bioRxiv Search.
   - ➔ La tarea debe ser implementada en grupos de máximo 5 personas.
   - ➔ **El proyecto es opcional, el porcentaje obtenido será sumado a la nota final del curso**.
   - ➔ La **fecha de entrega** es 18 de agosto del 2023 antes de las 8:53 pm.
   - ➔ Cualquier indicio de copia será calificado con una nota de 0 y será procesado de acuerdo con el reglamento. La copia incluye código/configuraciones que se puede encontrar en Internet y que sea utilizado parcial o totalmente sin el debido reconocimiento al autor.
   - ➔ La revisión es realizada de forma virtual mediante citas en las cuáles todos los y las integrantes del grupo deben estar presentes, el proyecto debe estar completamente automatizado, el profesor decide en que computadora probar.
   - ➔ Se debe incluir documentación con instrucciones claras para ejecutar el proyecto.
   - ➔ Se deben incluir pruebas unitarias para verificar los componentes individuales de su proyecto, si estas no están presentes se obtendrá una nota de 0.
   - ➔ Se espera que todos y todas las integrantes del grupo entiendan la implementación suministrada.
   - ➔ Se deben seguir buenas prácticas de programación en el caso de que apliquen, entre ellas:
     - ◆ Documentación interna y externa.
     - ◆ Estándares de código.
     - ◆ Diagramas de arquitectura.
     - ◆ Diagramas de flujo
     - ◆ **Pruebas unitarias**.

- ➔ Toda documentación debe ser implementada en Markdown.
- ➔ Si el proyecto no se encuentra automatizado obtendrá una nota de 0. Si el proyecto no se entrega completo, la porción que sea entregada debe estar completamente automatizada, de lo contrario se obtendrá una nota de 0.
- ➔ En caso de no entregar documentación se obtendrá una nota de 0.
- ➔ En caso de no tener pruebas unitarias se obtendrá una nota de 0.
- ➔ El email de entrega debe contener una copia del proyecto en formato tar.gz y un enlace al repositorio donde se encuentra almacenado, debe seguir los lineamientos en el programa de curso.
- ➔ Los grupos de trabajo se deben autogestionar, la persona facilitadora del curso no es responsable si un miembro del equipo no realiza su trabajo.
- ➔ En consulta o en mensajes de correo electrónico o Telegram, como mínimo se debe haber leído del tema y haber tratado de entender sobre el mismo, las consultas deben ser concretas y se debe mostrar que se intentó resolver el problema en cuestión.

4. **Descripción**

El presente proyecto pretende implementar un “motor de búsqueda” de artículos científicos de Covid19 utilizando la base de datos NoSQL llamada Elasticsearch, para este propósito utilizaremos un repositorio de descripciones de artículos científicos llamada bioRxiv, específicamente su API, el cual al día 24/07/2023 contaba con 27455 artículos:



El funcionamiento será el siguiente:

- Un usuario ingresa al Kibana Service, en el índice llamado *jobs*, creara un documento de la siguiente forma:

```
{
    "jobId": "{alfanumérico}",
    "pageSize": "{número}",
    "sleep": "{número en milisegundos para dormir entre cada request}"
}
```

- El componente *Controller*, continuamente está revisando el índice, en el momento que detecta este documento, hace un llamado al API bioRxiv, capturando el valor llamado *messages.total*, con este generara varios *splits*, los mismos son una porción de los artículos que se deben descargar, para calcular la cantidad de *splits* simplemente se divide *messages.total/pageSize*, por cada *split* se publica un mensaje en una cola de RabbitMQ con el siguiente formato:

```
{
    "jobId": "{alfanumérico}",
    "pageSize": "{número}",
    "sleep": "{número en milisegundos para dormir entre cada request}",
    "splitNumber": "{número de split}"
}
```

- Estos mensajes son leídos por el API Crawler, el cual descarga los archivos del API bioRxiv y los almacenan en un índice de Elasticsearch llamado *raw*, el ID del documento estará conformado por jobId y splitNumber, al finalizar publica el siguiente mensaje en una cola RabbitMQ.

```
{
    "jobId": "{alfanumérico}",
    "pageSize": "{número}",
    "sleep": "{número en milisegundos para dormir entre cada request}",
    "splitNumber": "{número de split}",
    "status": "DOWNLOADED"
}
```

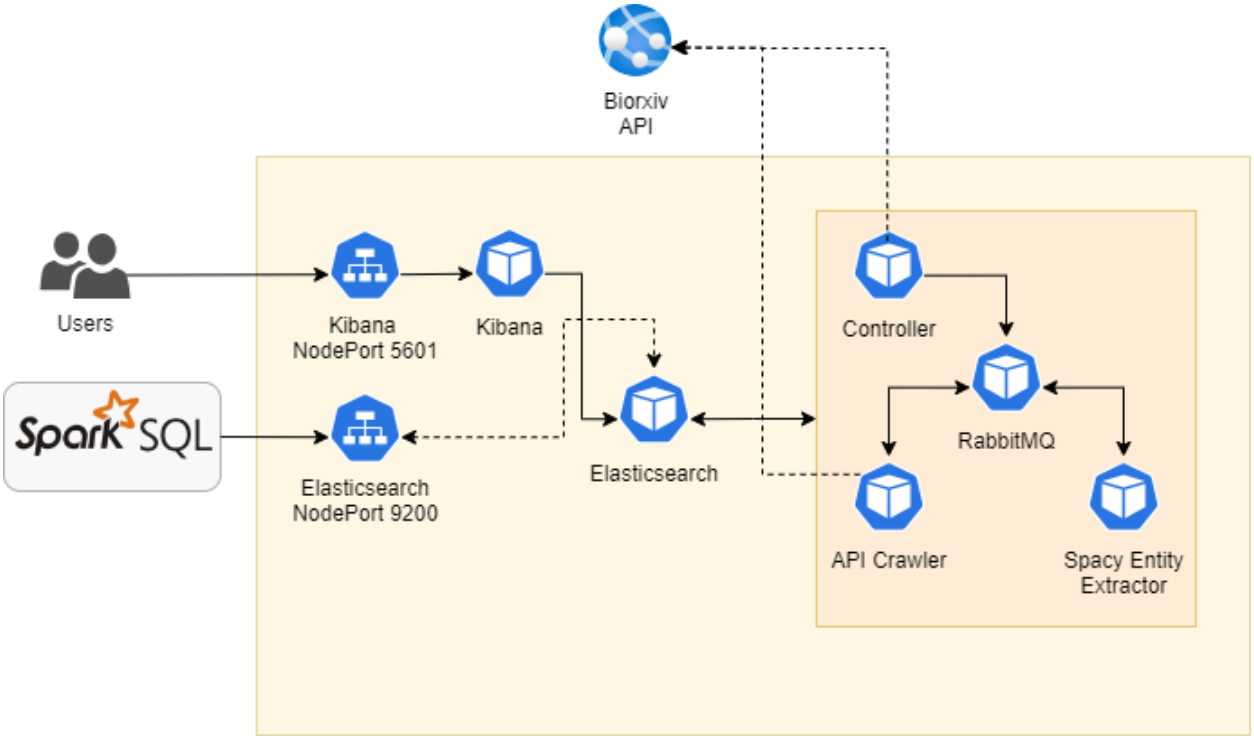
- Los mensajes son leídos por el componente Spacy Entity Extractor, el cual utilizara la funcionalidad por defecto de Spacy para realizar Named Entity Recognition, estas entidades se almacenarán en un nuevo campo de tipo array llamado *entities*, en un índice llamado *augmented*:
- El índice llamado *augmented*, será leído por una aplicación SparkSQL en Scala (de forma manual se ejecuta) y aplicará las siguientes transformaciones:
    - El *author_name*, deberá convertirse en formato Apellido, Nombre.
    - El *author_inst* deberá separarse en sus componentes, por ejemplo:

```
"rel_authors": [
    {
        "author_name": "Jinyong Kim",
        "author_inst": "Department of Internal Medicine, Seoul National University College of Medicine, Seoul National University Hospital, Seoul, Korea"
    },
    {
        "author_name": "Euijin Chang",
        "author_inst": "Department of Internal Medicine, Seoul National University College of Medicine, Seoul National University Hospital, Seoul, Korea"
    },
```

    - El *category* debe convertir cada primera letra de palabra en mayúscula y remover espacios a los lados.
    - El *rel_date* debe convertirse al formato dd/mm/yyyy.

  Una vez transformado, se publicará a un índice llamado *documents*.
- Una vez que los datos se encuentran en MySQL o Elasticsearch, el usuario podrá realizar consultas mediante Kibana.

El siguiente diagrama resume el funcionamiento del proyecto:



- Los componentes API Crawler, Controller y Spacy Entity Extractor serán implementados como pods, los cuales deberán estar configurados dentro de un Deployment.
- Para instalar Elasticsearch existen múltiples implementaciones de Kubernetes/Helm para instalar Elasticsearch y su interfaz gráfica Kibana, para mencionar algunas:
    - Elastic Cloud on Kubernetes
    - Elasticsearch Helm Charts
    - Bitnami Elasticsearch Stack
- Al igual que Elasticsearch, RabbitMQ tiene varias opciones para instalar utilizando Helm, la más popular es Bitnami RabbitMQ.
- Elasticsearch y Kibana deben ser expuestos a la maquina host mediante servicios de tipo NodePort.
- La solución debe estar automatizado con Helm Charts.

Documentación

Se deberá entregar una documentación que al menos debe incluir:

- Instrucciones claras de como ejecutar su proyecto.
- Pruebas realizadas, con pasos para reproducirlas.

- Resultados de las pruebas unitarias.
- Recomendaciones y conclusiones (al menos 10 de cada una).
- La documentación debe cubrir todos los componentes implementados o instalados/configurados, en caso de que algún componente no se encuentre implementado, no se podrá documentar y tendrá un impacto en la completitud de la documentación.
- Referencias bibliográficas donde aplique.

Imaginen que la documentación está siendo creada para una persona con conocimientos básicos en el área de computación y ustedes esperan que esta persona pueda ejecutar su proyecto y probarlo sin ningún problema, el uso de imágenes, diagramas, code snippets, videos, etc. son recursos que pueden ser útiles.
La documentación debe estar implementada en Markdown y compilada en un PDF.

5. **Recomendaciones**
   - Utilizar telnet/curl/postman para interactuar con los componentes y verificar que los mismos están funcionando correctamente.
   - Utilizar Docker Desktop para instalar Kubernetes en Windows.
   - Utilizar Minikube para ejecutar Kubernetes en Linux
   - Realicen peer-review/checkpoints semanales y no esperen hasta el último momento para integrar su aplicación.
   - Crear un repositorio de GitHub e invitar al profesor, pueden usar el repositorio de alguno de los integrantes del grupo.
   - Realizar commits y push regulares.

6. **Entregables**

   - Documentación.
   - Docker files, Docker Compose files y Helm Charts junto con todos los archivos/scripts requeridos para ejecutar su proyecto.

7. **Evaluación**

Funcionalidad / Requerimiento	Porcentaje
Documentación	10%
Implementación (*): ● Controller (10%) ● API Crawler (20%) ● Spacy Entity Extractor (20%) ● Scala Spark Processor (30%) ● Elasticsearch/Kibana (5%)	90%

• RabbitMQ (5%)	
	100%

(*) Todo tiene que estar debidamente automatizado, de lo contrario se calificará con una nota de 0.