Proyecto #2: Journal Search Platform (JSP)

> Integrantes:

- Max Richard Lee Chung - 2019185076
- Miguel Ku Liang - 2019061913

Guía de instalación

El programa necesita de la aplicación "Docker" y habilitar el servicio de "Kubernetes" para tener un clúster inicial y básico, llamado "docker-desktop". La aplicación para controlar y monitorear el comportamiento del clúster de Docker se denomina "Lens". Además, se utilizarán dos carpetas para los servicios automatizados del proyecto tales como helm_charts y Docker_images. Como herramientas de complemento, se utilizó la herramienta Postman para poder realizar pruebas de conexión de la API (interfaz de programación de aplicaciones).

Instalación de helm_charts

Para el desarrollo de este proyecto, se van a implementar dos helm charts principales (databases y monitoring) y una para las aplicaciones implementadas en Docker (application, el cual será mencionado posteriormente). Dentro del helm chart databases, se utilizó la base de datos MariaDB y Elasticsearch. Dentro del helm chart monitoring, se utilizó Grafana y Prometheus junto con el operador de Elasticsearch (eck-operator). Antes de instalar todos componentes, se debe de ubicar o crear una carpeta cualquiera para luego descargar las dependencias necesarias. Ya con la carpeta seleccionada, se ingresa a la consola de comando y realizar un "cd" a dicha carpeta y ejecutar los siguientes comandos:

```
helm create databases
helm create monitoring
helm create application            # Se comentará más adelante su uso
```

Luego de ejecutar los comandos, se debe de eliminar el contenido de la carpeta "templates" de todos los archivos generados para evitar algún problema a la hora de instalar los recursos. En este punto, se debe de gestionar las dependencias de los helm charts de databases y monitoring dentro del archivo Chart.yaml al final del códgio ya proporcionado, el cual se debe de conocer las versión, enlace del repositorio y nombre para poder extraer y descargar como un archivo comprimido los recursos de cada uno. Los repositorios se deben de buscar en Google buscando, como por ejemplo, el repositorio de Bitnami o Elasticsearch. Al conocer el enlace del repositorio, se añade a la libería de helm charts y buscar el nombre del recurso al que se quiera utilizar u actualizar como se muestra en el siguiente bloque de código.

```
helm add repo <enlace proporcionado>    # Agregar a la librería de helm charts
helm update repo <enlace proporcionado> # Actualizar el repositorio
helm search repo <recurso a buscar>     # Buscar la información del recurso
```

Al conocer los datos específicos de los recuros a utilizar, se agregan las dependencias como se observa en el siguiente apartado. Además, se agregaron variables para habilitar o deshabilitar el recurso en específico, el cual se comentará más adelante sobre cómo configurarlo.

```
# Chart.yaml de monitoring
dependencies:
- name: kube-prometheus
  version: "8.1.11"
  repository: "https://charts.bitnami.com/bitnami"
  condition: enablePrometheus
- name: grafana
  version: "8.2.11"
  repository: "https://charts.bitnami.com/bitnami"
  condition: enableGrafana
- name: eck-operator
  version: "2.4.0"
  repository: "https://helm.elastic.co"
  condition: enableOperator
```

```
# Chart.yaml de databases
dependencies:
- name: rabbitmq
  version: "11.0.3"
  repository: "https://charts.bitnami.com/bitnami"
  condition: enableRabbitMQ
- name: mariadb
  version: "11.3.3"
  repository: "https://charts.bitnami.com/bitnami"
  condition: enableMariaDB
```

Al tener listas las dependencias con los recursos deseados, se debe de hacer un "cd" a cada una de las carpetas generadas para poder ejecutar el siguiente comando, con el fin de "actualizar" las dependencias y que se descarguen los archivos comprimidos como se mencionó anteriormente e instalarlas.

```
cd databases                        # Ingresa a la carpeta
helm dependency update              # Actualiza las dependencias
cd ..                               # Se sale de la carpeta
helm install databases databases    # Se instala el helm chart
```

En caso contrario

```
cd monitoring
helm dependency update
cd ..
helm install monitoring monitoring
```

Sin embargo, si se instalan directamente, las bases de datos no tendrán las configuraciones que se desea, por lo que se modificó el archivo values.yaml para dicho aspecto. Para ello, se agregaron variables para habilitar o deshabilitar dicho recurso y habilitar las métricas del servicio de monitoreo para que Prometheus pueda observarlos y notificar en Grafana.

```
enableRabbitMQ: true
enableMariaDB: true
```

La primera configuración es la de RabbitMQ, en donde únicamente se le agregó la contraseña y habilitar el servicio de monitoreo.

```
rabbitmq:
  auth:
    # Se agrega un password fija para installar/desinstalar
    password: "rabbitmqpass"
  # Habilitar el servicio de monitoreo
  metrics:
    enabled: true
    serviceMonitor:
      enabled: true
```

La segunda configuración es la de MariaDB, en donde se tuvo que crear la instancia estática de una contraseña principal y un usuario con su respecta contraseña para evitar conflictos con los datos persistentes almacenados en Kubernetes. Además, se le habilitó el servicio de monitoreo.

```
mariadb:
  auth:
    # Se agrega un password fija para installar/desinstalar
    rootPassword: "mariadbpass"
    username: "user"
    password: "user"
  # Habilitar el servicio de monitoreo
  metrics:
    enabled: true
    serviceMonitor:
      enabled: true
```

Además, Elasticsearch necesita de forma básica un clúster y una instancia de Kibana para comunicar con la base de datos como se muestra a continuación, en el cual son aplicados como un modelo básico predefinido (template) dentro de la carpeta de databases en un archivo único.

```
## elasticsearch.yaml
# Deploy an Elasticsearch cluster
apiVersion: elasticsearch.k8s.elastic.co/v1
```

```
kind: Elasticsearch
metadata:
  name: quickstart
spec:
  version: 8.4.3
  nodeSets:
  - name: default
    count: 1
    config:
      node.store.allow_mmap: false
---
# Deploy a Kibana instance
apiVersion: kibana.k8s.elastic.co/v1
kind: Kibana
metadata:
  name: quickstart
spec:
  version: 8.4.3
  count: 1
  elasticsearchRef:
    name: quickstart
```

También, es necesario correr los siguientes comandos en la terminal para tener localmente las imágenes a utilizar del clúster y la instancia de Kibana.

```
docker pull docker.elastic.co/elasticsearch/elasticsearch:8.4.3
docker pull docker.elastic.co/kibana/kibana:8.4.3
```

Instalación de los docker images

Dentro de esta sección, es necesario crear una cuenta en Docker Hub para poder guardar las imágenes de las aplicaciones desarrolladas para el proyecto, el cual se usarán para las aplicaciones de "API", "Details_Downloader", "Downloader" y "Loader", utilizando lenguaje Python para otorgarles la lógca de recibir mensajes de la cola de RabbitMQ y procesarlas.

Luego de tener la cuenta habilitada, dentro de la carpeta general, se crearon cuatro carpetas para las aplicaciones respectivas, en donde cada uno tiene otra carpeta con los datos únicos de configuración de la aplicación de Python y un archivo de configuración "Docker file" para poder publicarlo al Docker Hub.

Con respecto al archivo de configuración, se buscó la versión de Python a utilizar dentro de las aplicaciones, el cual se selecionó una versión comprimida (slim) para la aplicación "API" y otra versión (buster) para las demás aplicaciones. Además, se siguieron las recomendaciones dadas por el profesor con la configuración necesaria para poder ejecutar la aplicación.

```
FROM python:3.10.7-slim-bullseye

WORKDIR /app
```

```
COPY app/. .

RUN pip install --no-cache-dir -r requirements.txt

CMD [ "python", "-u", "./app.py"]
```

```
FROM python:3.6-buster

RUN apt-get update && apt-get -yy install libmariadb-dev

WORKDIR /app

COPY app/. .

RUN pip install --upgrade pip
RUN pip3 install --no-cache-dir -r requirements.txt

CMD [ "python", "-u", "./app.py"]
```

Por consiguiente, dentro de la carpeta de configuración de Python, tiene un archivo de texto (requirements.txt) con las librerías necesarias para instalarlas de forma automática luego de publicar la aplicación a Docker Hub. Dicha librarías utilizadas son la de pika (para poder enviar o recibir mensajes del servidor a cliente y viceversa), flask, flask-ngrok, flask-cors y firebase-admin para el "API", mientras que las demás aplicaciones utilizan pika, elasticsearch, mariadb y requests.

Luego de crear el archivo de texto, es necesario definir el código fuente de Python para poder conectarlo con la cola de RabbitMQ por medio de variables de entorno. También, definir el comportamiento lógico para procesar la información. A continuación se muestra un ejemplo de las variables de entorno del trabajador "loader".

```
IP = os.getenv('POD_IP')
BIO = os.getenv('BIORXIV')
HOSTMARIA = os.getenv('MARIADB')
MARIAPASS = os.getenv('MARIAPASS')
RABBIT = os.getenv('RABBITMQ')
RABBITPASS = os.getenv('RABBITPASS')
INQUEUE = os.getenv('INQUEUE')
OUTQUEUE = os.getenv('OUTQUEUE')
```

Además, como se mencionó en la anterior sección sobre la carpeta de "application", es necesario crear modelos predefinidos (al igual que el archivo quickstart de Elasticsearch) de las aplicaciones con sus respectivas variables de entorno y sus contraseñas (adquiridas por referencias a los secrets). Las variables de entorno es información guardada en variables que son visibles en todo momento y almacenadas en el sistema.

```

```

```
# Loader.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: loader
  labels:
    app: loader
spec:
  replicas: 1
  selector:
    matchLabels:
      app: loader
  template:
    metadata:
      labels:
        app: loader
    spec:
      containers:
      - name: loader
        image: basesdedatos2/loader
        env:
          - name: POD_IP
            valueFrom:
              fieldRef:
                fieldPath: status.podIP
          - name: BIORXIV
            value: "https://api.biorxiv.org/covid19/0"
          - name: RABBITMQ
            value: "databases-rabbitmq"
          - name: RABBITPASS
            valueFrom:
              secretKeyRef:
                name: databases-rabbitmq
                key: rabbitmq-password
                optional: false
          - name: MARIADB
            value: "databases-mariadb"
          - name: MARIAPASS
            valueFrom:
              secretKeyRef:
                name: databases-mariadb
                key: mariadb-password
                optional: false
          - name: INQUEUE
            value: "EntryQueue"
          - name: OUTQUEUE
            value: "OutputQueue"
          - name: ELASTIC
            value: quickstart-es-default
          - name: ELASTICPASS
            valueFrom:
              secretKeyRef:
                name: quickstart-es-elastic-user
```

```
                key: elastic
                optional: false
          - name: ELASTICINDEX
            value: jobs
          - name: ELASTICINDEX
            value: groups
```

```
# Downloader.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: downloader
  labels:
    app: downloader
spec:
  replicas: 1
  selector:
    matchLabels:
      app: downloader
  template:
    metadata:
      labels:
        app: downloader
    spec:
      containers:
      - name: downloader
        image: basesdedatos2/downloader
        env:
          - name: POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: BIORXIV
            value: "https://api.biorxiv.org/covid19/"
          - name: RABBITMQ
            value: "databases-rabbitmq"
          - name: RABBITPASS
            valueFrom:
              secretKeyRef:
                name: databases-rabbitmq
                key: rabbitmq-password
                optional: false
          - name: MARIADB
            value: "databases-mariadb"
          - name: MARIADBPASS
            valueFrom:
              secretKeyRef:
                name: databases-mariadb
                key: mariadb-password
                optional: false
          - name: INQUEUE
```

```
            value: "OutputQueue"
          - name: OUTQUEUE
            value: "DetDownQueue"
          - name: ESENDPOINT
            value: quickstart-es-default
          - name: ESPASSWORD
            valueFrom:
              secretKeyRef:
                name: quickstart-es-elastic-user
                key: elastic
                optional: false
          - name: ESINDEXGROUPS
            value: groups
```

```
# detailsDownloader.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: details
  labels:
    app: details
spec:
  replicas: 1
  selector:
    matchLabels:
      app: details
  template:
    metadata:
      labels:
        app: details
    spec:
      containers:
      - name: details
        image: basesdedatos2/details
        env:
          - name: POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: BIORXIV_DETAILS
            value: "https://api.biorxiv.org/details/"
          - name: RABBITMQ
            value: "databases-rabbitmq"
          - name: RABBITPASS
            valueFrom:
              secretKeyRef:
                name: databases-rabbitmq
                key: rabbitmq-password
                optional: false
          - name: MARIADB
            value: "databases-mariadb"
```

```
          - name: MARIADBPASS
            valueFrom:
              secretKeyRef:
                name: databases-mariadb
                key: mariadb-password
                optional: false
          - name: INQUEUE
            value: "DetDownQueue"
          - name: OUTQUEUE
            value: "JatsQueue"
          - name: ESENDPOINT
            value: quickstart-es-default
          - name: ESPASSWORD
            valueFrom:
              secretKeyRef:
                name: quickstart-es-elastic-user
                key: elastic
                optional: false
          - name: ESINDEXGROUPS
            value: groups
```

```
# API.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api
  labels:
    app: api
spec:
  replicas: 1
  selector:
    matchLabels:
      app: api
  template:
    metadata:
      labels:
        app: api
    spec:
      containers:
      - name: api
        image: basesdedatos2/api
        env:
          - name: ELASTICURL
            value: quickstart-es-default
          - name: ELASTICPASS
            valueFrom:
              secretKeyRef:
                name: quickstart-es-elastic-user
                key: elastic
                optional: false
```

```
            - name: ESINDEXGROUPS
                value: groups
```

Por último, luego de tener los archivos listos para poder publicarlos a la nube, se siguen los siguientes comandos para poder realizarlo desde la terminal, donde el user es el nombre de usuario de Docker. Además, si la ejecución de comandos no sirve en la aplicación Lens, se puede utilizar el Command Prompt para ejecutarlos.

```
docker login                              # Inicio de sesión al repositorio de
Docker Hub
docker build -t <user>/<aplicación> .     # Construye la imagen
docker images                             # Permite revisar la lista de imágenes
docker push <user>/<aplicación>           # Publica la imagen al repositorio en la
nube
helm install application application
```

Cuerpo lógico de los trabajadores

Como se mencionó anteriormente, es necesario crear el comportamiento para ejecutar lo que se desea. Como primera instancia, se debe de establecer las conexiones tanto de la cola de mensajería de RabbitMQ, la base de datos de MariaDB y Elasticsearch como se muestra a continuación.

```
credentials = pika.PlainCredentials('user', RABBITPASS)
parameters = pika.ConnectionParameters(host = RABBIT, credentials = credentials,
heartbeat = 600)
connection = pika.BlockingConnection(parameters)
channel = connection.channel()
channel.queue_declare(queue = INQUEUE)
```

```
# Conexion al servicio de la base de datos Mariadb
mariaDatabase = mariadb.connect(
  host=HOSTMARIA,
  port=3306,
  user="user",
  password=MARIAPASS,
  database="my_database"
)
```

```
# Conexion a Elasticsearch
client = Elasticsearch("https://" + ESENDPOINT + ":9200", basic_auth = ("elastic",
ESPASSWORD), verify_certs = False)
if client.indices.exists(index = ESINDEXGROUPS):
    client.indices.delete(index = ESINDEXGROUPS)
    print("Indice groups eliminado")
```

```
client.indices.create(index = ESINDEXGROUPS)
print("Indice groups creado")
```

Luego, se agregaron variables para manejo de variables o funciones para poder analizar y procesar los datos. Como por ejemplo, la función "callback" aplicados al trabajador downloader y details downloader, es utilizado para recibir mensajes de RabbitMQ y empieza a procesarlos. Dentro del trabajador API, se deben de crear diferentes funciones para poder realizar peticiones desde la aplicación Thunkable.

Con respecto a las tablas para guardar datos en MariaDB, se crearon las tablas desde el trabajador Loader con los siguientes comandos con su conexión con el fin de permitir la automatización de esta.

```
# Crear las tablas si no existen
connection.execute("DROP TABLE IF EXISTS history")
connection.execute("DROP TABLE IF EXISTS groups")
connection.execute("DROP TABLE IF EXISTS jobs")
connection.execute("CREATE TABLE jobs (\
                      id INT NOT NULL AUTO_INCREMENT,\
                      created DATETIME NOT NULL,\
                      status VARCHAR(45) NOT NULL,\
                      end DATETIME NOT NULL,\
                      loader VARCHAR(45) NOT NULL,\
                      grp_size INT NOT NULL,\
                      PRIMARY KEY (id)\
                   )")
connection.execute("CREATE TABLE groups (\
                      id INT NOT NULL AUTO_INCREMENT,\
                      id_job INT NOT NULL,\
                      created DATETIME NOT NULL,\
                      end DATETIME,\
                      stage VARCHAR(45) NOT NULL,\
                      grp_number INT NOT NULL,\
                      status VARCHAR(45),\
                      offsetData INT NOT NULL,\
                      PRIMARY KEY (id),\
                      FOREIGN KEY (id_job) REFERENCES jobs(id)\
                   )")
connection.execute("CREATE TABLE history (\
                     id INT NOT NULL AUTO_INCREMENT,\
                     component VARCHAR(45) NOT NULL,\
                     status VARCHAR(45) NOT NULL,\
                     created DATETIME NOT NULL,\
                     end DATETIME,\
                     message TEXT,\
                     grp_id INT NOT NULL,\
                     stage VARCHAR(45) NOT NULL,\
                     PRIMARY KEY (id),\
                     FOREIGN KEY (grp_id) REFERENCES groups(id)\
                  )")
```

Configuración de herramientas no automatizadas

Grafana

Configurar Grafana primero es necesario haber instalado el helm chart monitoring para tener un Pod funcional de grafana, en donde se selecciona para poder acceder al enlace con el puerto de Grafana. Al estar en la página, se le solicita un usuario, en donde el usuario predeterminado es "admin" y la contraseña se adquiere dentro del Secret "monitoring-grafana-admin", el cual es generado automáticamente.

Una vez dentro de la aplicación, es necesario agregar a Prometheus para monitorear y alertar eventos. Dentro de la esquina inferior izquierda, se encuentra el botón de configuracíón con forma de engranaje, al darle click, se despliega la página y pestaña de "Data sources", en donde se da la opción de crear uno nuevo. La nueva ventana despliega todas las opciones de "Data Source", el cual se le da a la opción de Prometheus. Ya en la opción de ingresar los datos del nuevo recurso, es necesario conocer el enlace del Prometheus creado por el helm chart cuyo enlace es el nombre del "Service" (servicio) "monitoring-kube-prometheus-prometheus" seguido de :9090 (puerto predeterminado) y se agrega.

```
http://monitoring-kube-prometheus-prometheus:9090
```

Por último, es necesario buscar un tablero "dashboard" para poder ver todas las métricas que Prometheus monitorea en Grafana, cuyo tablero se busca uno apropiado con los datos requeridos. Dicho tablero es preferible de la misma página de Grafana para poder importarlo con un ID en la sección de "Dashboards".

Kibana

El acceso al servicio de Kibana se realiza a partir del puerto expuesto dado por el pod "quickstart-kb-...". Ya dentro de la página redirigida, el sistema solicita una cuenta, en donde el usuario predefinido es "elastic" y la contraseña se obtiene mediante la búsqueda del secreto (secret) "quickstart-es-elastic-user".

MariaDB

MariaDB no tiene una forma de accesar fácilmente a diferencia de otros servicios, por lo que hay que habilitar o exponer el puerto para poder acceder a la base de datos por medio de MySQL Workbench, de esta manera, el siguiente comando habilita dicho espacio.

```
kubectl port-forward databases-mariadb-0 3305:3306
```

Dentro del MySQL Workbench, se ingresan los datos del servidor, tales como nombre del servidor, nombre de host (127.0.0.1), puerto (3305), nombre de usuario (root) y contraseña (mariadbpass). Además, se debe de deshabilitar el uso de SSL por medio de la pestaña "Advanced/Others:" y se le agrega useSSL=1. Luego se puede probar la conexión e ingresar a la base de datos.

Preparación para ejecutar las pruebas

Para realizar las pruebas primero debemos conectarnos con MariaDB y Elasticsearch para poder visualizar los datos que se estan insertando o cambiando. Para MariaDB, se ejecuta el commando mencionado en el apartado anterior.

Para Elasticsearch, iremos al "Lens" en el apartado de "Pods" donde buscaremos el nombre de "quickstart-kb-...". Dentro de esa opción, nos dirigimos a la parte de "Containers" en donde le daremos al botón de "Forward". Esto nos llevará a una página donde pondremos de username "elastic" y el password lo encontraremos en Lens en la parte de "Secrets" en la opción llamada "quickstart-es-elastic-user". Ahí podremos copiar la contraseña que nos permitirá ingresar a Elasticsearch. En la página nos podemos dirigir a "Dev Tools" en el cual podremos ejecutar consultas a Elasticsearch.

Una vez hecho esto, ejecutamos el comando para ejecutar el proyecto con sus respectivas aplicaciones.

```
helm install application application
```

Pruebas de helm charts

El pod loader crea un job inicial para crear los grupos de documentos en los que se van a trabajar en el pod downloader y detail downloader. Cada nuevo grupo, genera un mensaje que es publicado a la cola de salida del loader. Este mensaje contiene el id del job y el número de grupo.

Este mensaje es recibido por la cola de entrada del downloader. Este luego descarga los documentos del endpoint "https://api.biorxiv.org/covid19/" hasta los documentos necesarios de cada grupo. Une vez descargado los documentos, los publica al índice "groups" de Elasticsearch y notifica al details_downloader por medio del mensaje inicial publicado en su cola de salida. Por medio de la consulta:

```
GET groups/_search
{
    "query": {
              "match_all": {}
    }
}
```

podemos revisar estos documentos.

El details_downloader una vez obtenido el mensaje, obtiene los documentos del grupo correspondiente en Elasticsearch para que por medio de los campos "rel_doi" y "rel_site" obtenga los detalles de cada documento. Este luego publica estos documentos al índice "groups de Elasticsearch y podremos consultarlos utilizando el mismo comando mencionado anteriormente. Además, podremos buscar la llave "details" para visualizar estos detalles que fueron agregados.

Pruebas de API

Como se mencionó anteriormente, se utilizó la herramienta Postman para probar de forma local el Flask con Ngrok de Python para verificar las entradas y salidas del API, de esta forma, se puede asegurar el correcto flujo como, por ejemplo, utilizar la URL de 127.0.0.1/5000 (puerto de salida de la API de Python) o el autogenerado de Ngrok con una extensión /elastic (llama a la base de datos de Elasticsearch para realizar la búsqueda con el texto deseado), /elasticMore (lo mismo que el anterior pero busca más artículos), /addLike/data (ingresar en FireBase el artículo con like) y /addGrp/data (ingresa un nuevo job a MariabDB). A continuación se mostrarán los casos de prueba de cada enlace con su respectivo cuerpo de entrada.

```
- http://127.0.0.1:5000/

Salida: "Journal Search Platform (JSP)'s API"

- http://127.0.0.1:5000/elastic

Cuerpo: {
  "data": <texto a buscar>
}
Salida: 10 artículos relacionados al texto ingresado

- http://127.0.0.1:5000/elasticMore

Cuerpo: {
  "data": <texto a buscar>
  "offset": <último título registrado>
}
Salida: 10 artículos extra relacionados al texto ingresado

- http://127.0.0.1:5000/addLike/userId

Salida: Resultado de los artículos del usuario

- http://127.0.0.1:5000/addGrp/int

Salida: String del entero del tamaño del grupo deseado del nuevo trabajo
```

Conclusión

La migración de datos entre plataformas es un sistema muy versátil de mensajería que puede ser aplicado a diferentes campos para obtener resultados temporizados, transición de datos y manipulación de la misma. Además, se pueden observar el rendimiento que conlleva cada procesamiento de los datos en la cola por medio de las aplicaciones de monitoreo.

Referencias

- Repositorio
- Stefano (2020). Answer. Recuperado de docker build failed at 'Downloading mariadb'
- Anweb (2018). Answer (2). Recuperado de How to connect to database using scripting language inside kubernetes cluster
- Vishal (2021). Recuperado de Python MySQL Execute Parameterized Query using Prepared Statement
- Donohue, T. (2022). Recuperado de How to get the IP address of a Pod in Kubernetes
- ScrapeOps (2022). Recuperado de Saving Scraped Data To MySQL Database With Scrapy Pipelines
- Reitz, K. (s.f.). Recuperado de Source code for requests.exceptions
- Alderman, D. (2020). Recuperado de How to Use the Data Viewer List in Thunkable X - 2020 To Do App
- Thunkable (2022). Recuperado de Thunkable Docs
- Bitnami - MariaDB (2022) Github. Recuperdo de MariaDB
- Bitnami - RabbitMQ (2022) Github. Recuperado de Elasticsearch
- Docker (2022). Docker Hub. Recuperado de Docker Hub

- Elastic (2022). Elastic Cloud on Kubernetes [2.4]. Recuperado de Elasticsearch-operator
- Elastic - elasticsearch-docker (2017). Can't pull official images. Recuperado de elasticsearch-docker
- Elasticsearch (2022). Python Elasticsearch Client. Recuperado de Python Elasticsearch Client
- RabbitMQ (2022). RabbitMQ Tutorials. Recuperado de RabbitMQ Tutorials
- Regolith (2021). How do I disable the SSL requirement in MySQL Workbench? Answer. Recuperado de Regolith
- Pika (2022). Introduction to Pika. Recuperado de Pika
- Elasticsearch (2022). Python Elasticsearch Client. Recuperado de Python Elasticsearch Client