TRABAJO

Materia: Cómputo paralelo y distribuido.

Equipo:

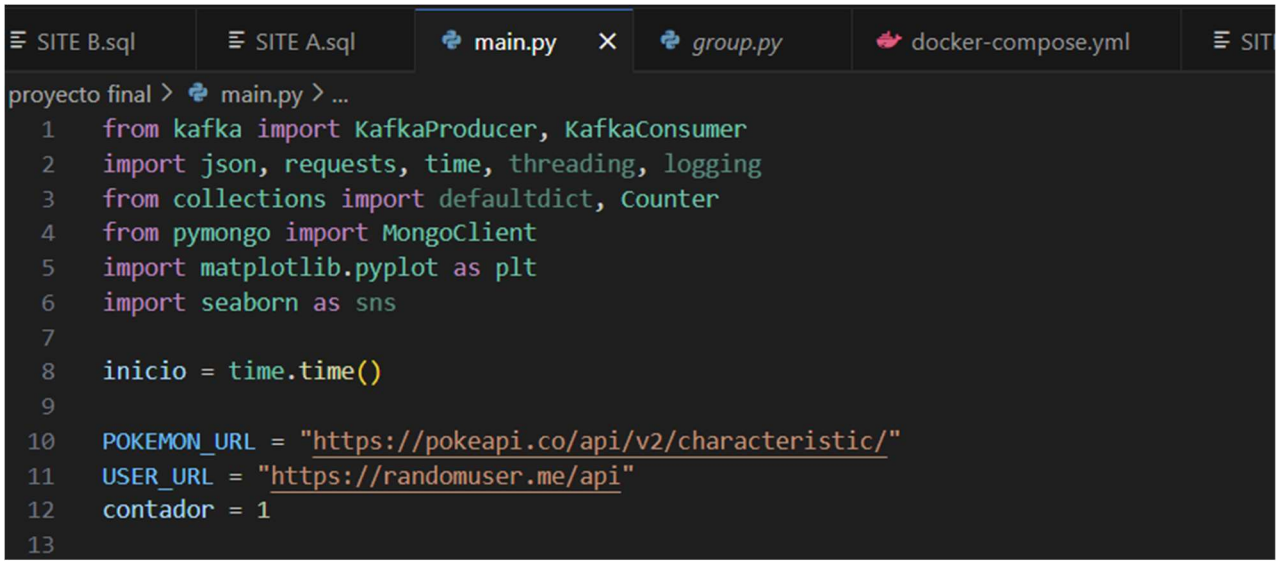
Axel Ricardo Moncloa Muro

Leonardo Trevizo Herrera

3.25 Proyecto final.

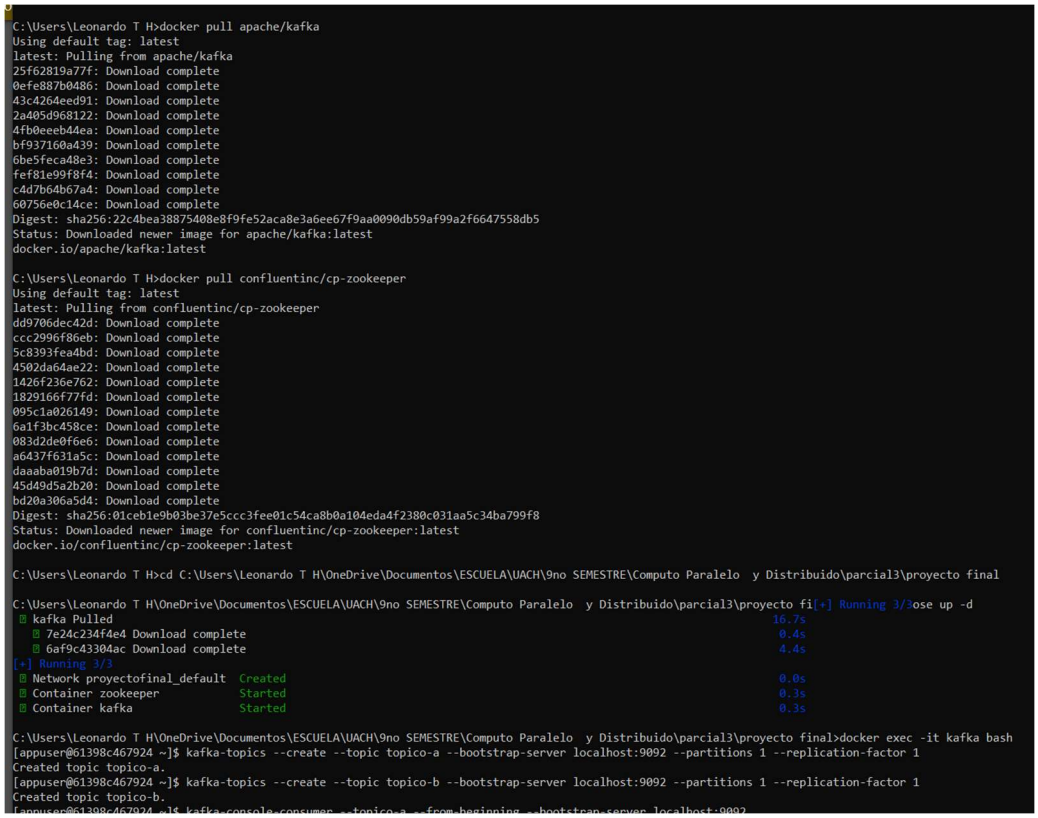
28 de Noviembre de 2024.

- Utilizar al menos dos API’s (Airlabs, Nasa, Openweather, etc.)



- Crear en Python un productor de mensajes de las API’s

- Mediante Apache Kafka administrar los mensajes (crear un topic por cada API-al menos dos-)



```python
def obtener_data_api_pokemon(url:str, params:None):
    print(POKEMON_URL+str(params)+"/")
    response = requests.get(POKEMON_URL+str(params))
    if response.status_code == 200:
        return response.json()  # Devuelve datos en formato JSON
    else:
        print(f"Error: {response.status_code}")
        return None

def obtener_data_api_user(url:str):
    response = requests.get(USER_URL)
    if response.status_code == 200:
        return response.json().get('results')[0]  # Devuelve datos en formato JSON
    else:
        print(f"Error: {response.status_code}")
        return None

while True:
    actual = time.time()
    pokemon = obtener_data_api_pokemon(POKEMON_URL,contador)
    #user = requests.get(user_url)
    if pokemon:
        productor.send('topico-a', value=pokemon)  # Envía datos al topic 'topic-a'
        #print(f"Enviado a topico-a:\n {pokemon}")
        print(f"Enviado a topico-a:\n")
    time.sleep(3.5)
    user = obtener_data_api_user(USER_URL)
    if user:
        productor.send('topico-b', value=user)
        #print(f"Enviado a topico-b:\n {user}")
        print(f"Enviado a topico-b:\n")
    time.sleep(5.5)
    pokemon=None
    user=None
    contador +=1
    if (actual-inicio) > 40:
        break
```

SITE B.sql | SITE A.sql | main.py | group.py | docker-compose.yml | SITE - REPLICA.sql

proyecto final > docker-compose.yml

```yaml
version: '3.8'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:latest
    container_name: zookeeper
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000

  kafka:
    image: confluentinc/cp-kafka:latest
    container_name: kafka
    ports:
      - "9092:9092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9092
      KAFKA_LISTENERS: PLAINTEXT://0.0.0.0:9092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    depends_on:
      - zookeeper

```

- Crear en Python un consumidor que tome los datos del topic de Apache Kafka

```
23
24  consumer = KafkaConsumer(
25      'topico-a', 'topico-b',
26      bootstrap_servers=['localhost:9092'],
27      auto_offset_reset='earliest',
28      enable_auto_commit=True,
29      value_deserializer=lambda x: json.loads(x.decode('utf-8'))
30  )
31
32  consumerA = KafkaConsumer(
33      'topico-a',
34      bootstrap_servers=['localhost:9092'],
35      auto_offset_reset='earliest',
36      enable_auto_commit=True,
37      value_deserializer=lambda x: json.loads(x.decode('utf-8'))
38  )
39
40  consumerB = KafkaConsumer(
41      'topico-b',
42      bootstrap_servers=['localhost:9092'],
43      auto_offset_reset='earliest',
44      enable_auto_commit=True,
45      value_deserializer=lambda x: json.loads(x.decode('utf-8'))
46  )
47
48
```

- Almacenar los datos en MongoDB

```
88
89  counter=0
90  for mensaje in consumerA:
91
92      #print(f"Insertando en MongoDB: {datos}")
93      if counter<=20:
94          datos = mensaje.value  # Datos recibidos del topic
95          print(f"Insertando en MongoDB  A")
96          coleccionA.insert_one(datos)  # Inserta los datos como un documento en la colección
97      else:
98          break
99      counter+=1
100
101 counter=0
102 for mensaje in consumerB:
103     if counter<=20:
104         datos = mensaje.value  # Datos recibidos del topic
105         #print(f"Insertando en MongoDB: {datos}")
106         print(f"Insertando en MongoDB  B")
107         coleccionB.insert_one(datos)  # Inserta los datos como un documento en la colección
108     else:
109         break
110     counter+=1
111
112 datos_pokemon = list(coleccionA.find())
113 datos_usuario = list(coleccionB.find())
114
```

- Tomar los datos de MongoDB y graficar los resultados

```
111
112  datos_pokemon = list(coleccionA.find())
113  datos_usuario = list(coleccionB.find())
114
115  valores_pokemon = [dato['highest_stat']['name'] for dato in datos_pokemon if 'highest_stat' in dato]
116  #print(valores_pokemon)
117
118  valores_usuarios = [dato['results'][0]['nat'] for dato in datos_usuario if 'results' in dato]
119  #print(valores_usuarios)
120
121  # Contar las nacionalidades
122  nationality_counts = Counter(valores_usuarios)
123  nationality_labels = list(nationality_counts.keys())
124  nationality_values = list(nationality_counts.values())
125
126  # Contar valores de highest_stat
127  stat_counts = Counter(valores_pokemon)
128  stat_labels = list(stat_counts.keys())
129  stat_values = list(stat_counts.values())
130
131  # Crear la figura con subplots
132  fig, axes = plt.subplots(1, 2, figsize=(14, 6))  # 1 fila, 2 columnas
133
134  # Gráfica de nacionalidades
135  axes[0].bar(nationality_labels, nationality_values, color='skyblue')
136  axes[0].set_title("Distribución de Nacionalidades", fontsize=16)
137  axes[0].set_xlabel("Nacionalidad", fontsize=12)
138  axes[0].set_ylabel("Frecuencia", fontsize=12)
139  axes[0].grid(axis='y', linestyle='--', alpha=0.7)
140
141  # Gráfica de highest_stat
142  axes[1].bar(stat_labels, stat_values, color='orange')
143  axes[1].set_title("Distribución de Highest Stat", fontsize=16)
144  axes[1].set_xlabel("Highest Stat", fontsize=12)
145  axes[1].set_ylabel("Frecuencia", fontsize=12)
146  axes[1].grid(axis='y', linestyle='--', alpha=0.7)
147
148  # Ajustar el diseño
149  plt.tight_layout()
150
151  # Mostrar ambas gráficas
152  plt.show()
153
```

Resultados