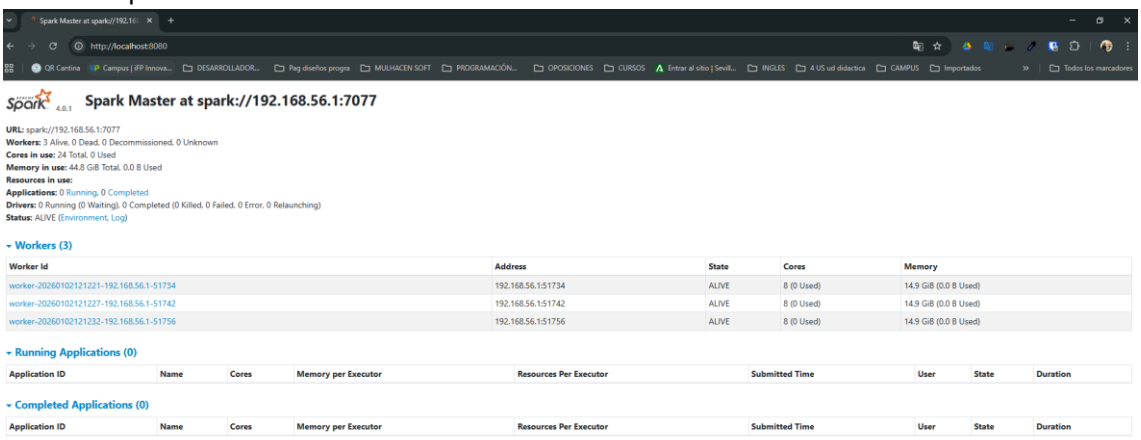
PRACTICA CON SPARK Y WORKERS

1. **Cambia el tamaño de la matriz N y compara los tiempos**
   Estos tiempos son con 3 worker activos.



Matriz N = 128

```
50  # --- Generar matrices aleatorias ---
51  N = 128
52  A = np.random.randint(0, 10, (N, N))
53  B = np.random.randint(0, 10, (N, N))
54
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE PORTS

```
PS C:\Users\MARIA> & C:/Users/MARIA/AppData/Local/Programs/Python/Python311/python.exe c:/Users/MARIA/Desktop/spark/strassen_spark.py
WARNING: Using incubator modules: jdk.incubator.vector
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
26/01/02 12:23:13 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
✅ Multiplicación completada.
⏱Tiempo total: 14.19 segundos
```

Matriz N = 256

```
50  # --- Generar matrices aleatorias ---
51  N = 256
52  A = np.random.randint(0, 10, (N, N))
53  B = np.random.randint(0, 10, (N, N))
54
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE PORTS

```
PS C:\Users\MARIA> & C:/Users/MARIA/AppData/Local/Programs/Python/Python311/python.exe c:/Users/MARIA/Desktop/spark/strassen_spark.py
WARNING: Using incubator modules: jdk.incubator.vector
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
26/01/02 12:24:18 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
✅ Multiplicación completada.
⏱Tiempo total: 23.95 segundos
```

Matriz N = 512

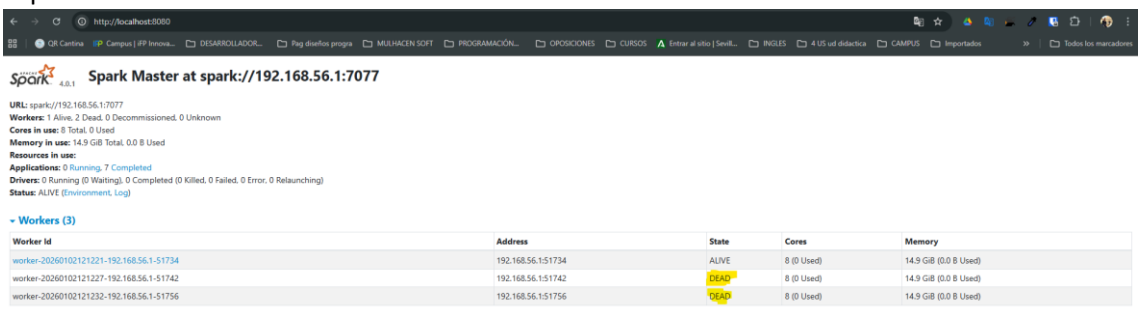
```
50   # --- Generar matrices aleatorias ---
51   N = 512
52   A = np.random.randint(0, 10, (N, N))
53   B = np.random.randint(0, 10, (N, N))
54
55   k = N // 2
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE PORTS

```
PS C:\Users\MARIA> & C:/Users/MARIA/AppData/Local/Programs/Python/Python311/python.exe c:/Users/MARIA/Desktop/spark/strassen_spark.py
WARNING: Using incubator modules: jdk.incubator.vector
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
26/01/02 12:16:38 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
✅ Multiplicación completada.
⏱️Tiempo total: 69.09 segundos
```

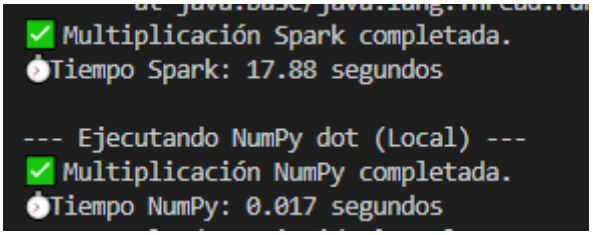
2. **Cambiar el número de workes locales**

   Aquí cambia a 1 activo



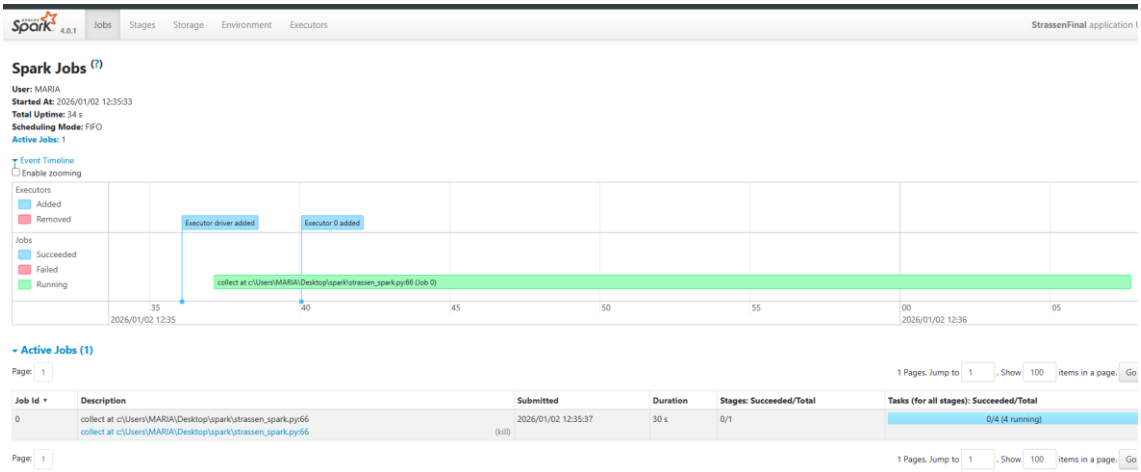
3. **Sustituye el algoritmo por np.dot(A,B)**

   Numpy es más rápido que Spark. Numpy ejecuta operaciones vectorizadas directamente en memoria RAM utilizando bibliotecas de bajo nivel altamente optimizada

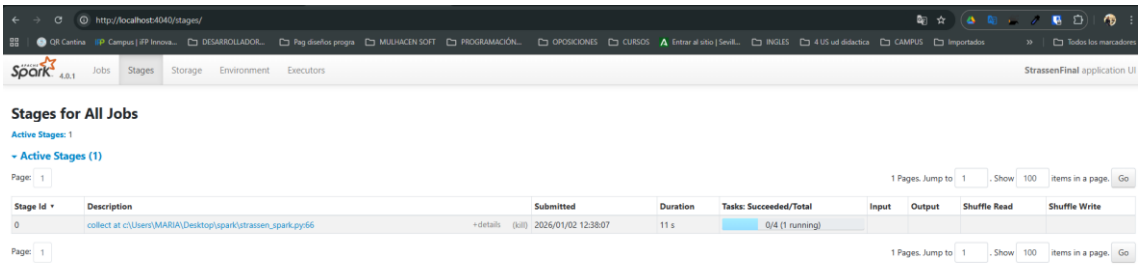


4. **Observa en el Spark Web UI el DAG y las etapas**

   Spark DAG



Spark stage

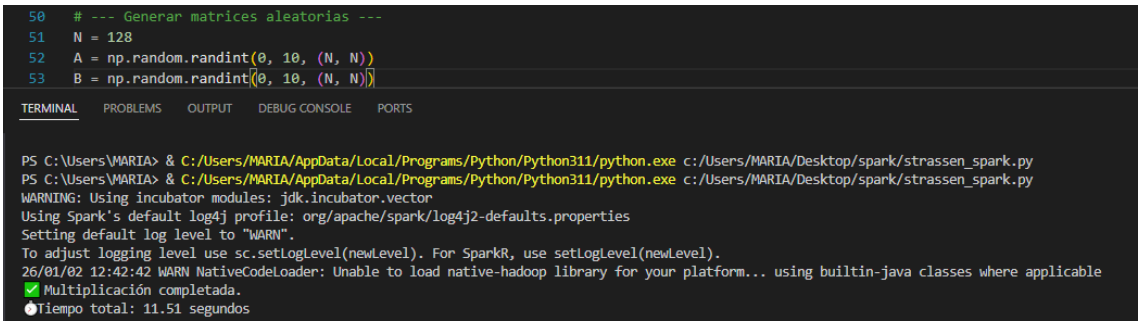


5. **Mide el speedup. Tiempo con 1 worker / tiempo con 3 worker**

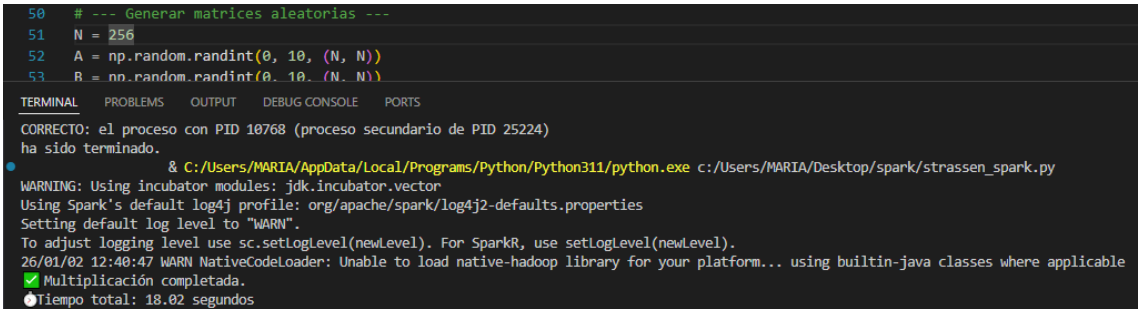
   En el ejercicio 1 esta el tiempo trabajando con 3 worker

   Realizo la prueba para los tiempo trabajando con 1 worked

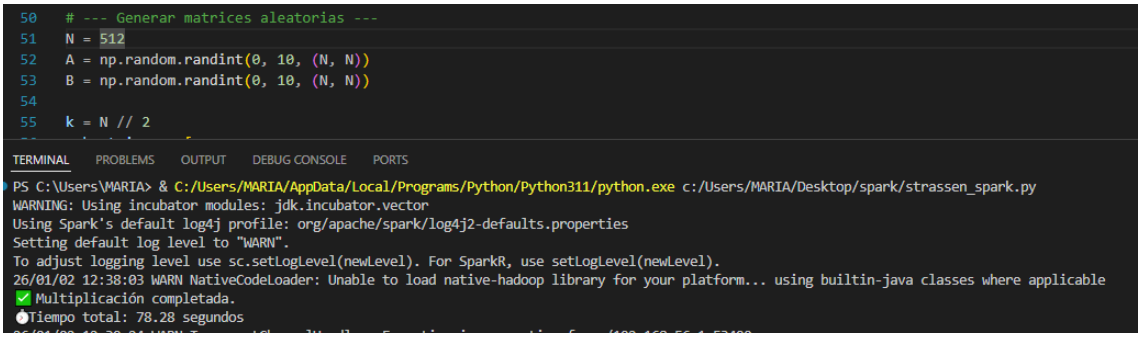
Matriz N = 128



Matriz N = 256



Matriz N = 512



En este ejercicio me di cuenta de que el verdadero cuello de botella no fue la potencia de los ordenadores, sino el tiempo que se pierde organizando todo. Al usar tres workers, el sistema tarda más tiempo preparando las piezas de la matriz y mandándolas por la red que lo que tarda realmente en hacer la multiplicación. Básicamente, para una matriz de este tamaño, sale más a cuenta que un solo worker haga todo el trabajo a que Spark pierda tiempo repartiendo tareas y esperando a que los demás terminen.