

EMPLOYEE ANALYTICS DASHBOARD

Employee Count

1,470

Attrition Count

237

Attrition Rate

16%

Active Employee

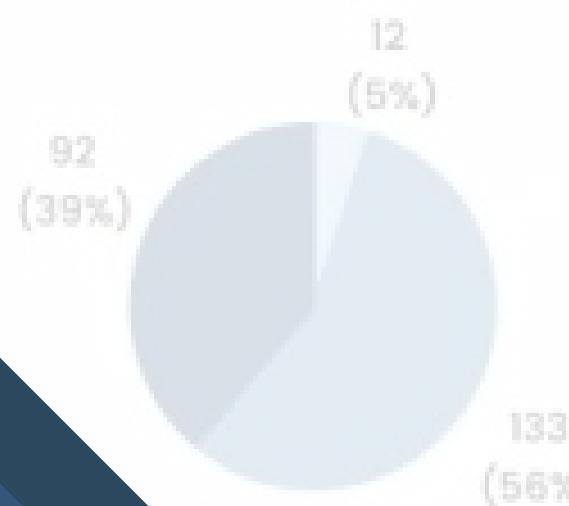
1,233

Average Age

37

SUPER-STORE APP

Department Wise Attrition



Total of Employee

Age range 18 - 60



Job Satisfaction Role

	1	2	3	4
Healthcare Rep.	26	19	14	10
Human Resourc.	10	16	14	10
Laboratory Tech.	56	48	78	88
Manager	21	21	27	33
Manufacturing D.	26	32	49	38
Research Direct.	15	16	27	22
Research Scienti.	54	53	90	95
Sales Executive	69	64	91	112
Sales Répräsent.	12	21	27	23
Grand Total	289	280	442	459

ANGELA MARIA RUIZ TOVAR

Attrition Rate by Age Group



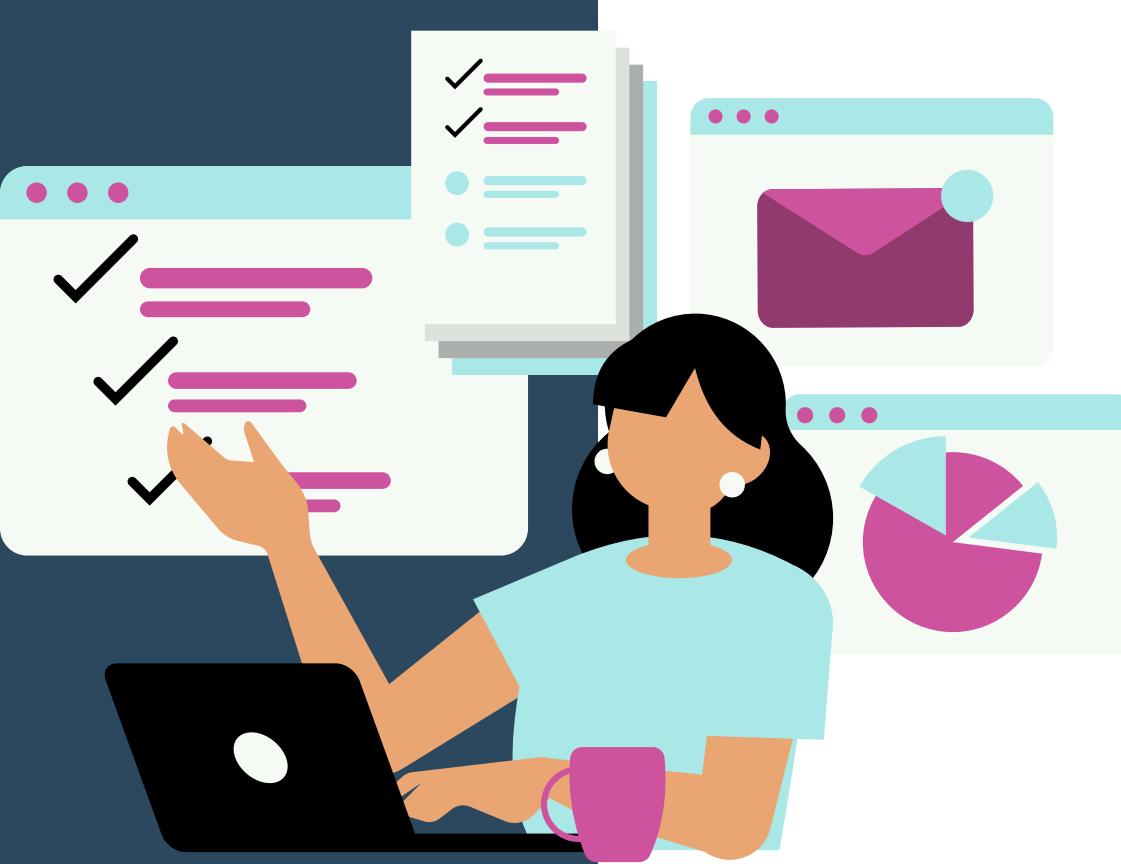
Problematica

En el contexto actual de transformación digital, los supermercados requieren soluciones tecnológicas que les permitan modernizar su infraestructura, responder a una alta demanda de usuarios y aprovechar el valor de los datos recolectados. Por eso nace Super-Store App, una aplicación basada en microservicios que permite al administrador gestionar productos, usuarios y visualizar la información más importante, todo a través de un solo clic.



Objetivo generales

- Desarrollar una aplicación de tienda en línea funcional utilizando una arquitectura de microservicios.
- Implementar cada microservicio de forma independiente, gestionando una funcionalidad específica del negocio.
- Empaquetar cada microservicio de la aplicación utilizando Docker Compose, facilitando su despliegue y gestión.
- Crear un entorno distribuido entre dos nodos para la aplicación web, empleando imágenes previamente publicadas en Docker Hub, asegurando su correcto funcionamiento.
- Desarrollar una aplicación con Apache Spark que permita analizar el dataset de la tienda y cargar los resultados en el clúster correspondiente.
- Generar un panel de visualización que permita mostrar de forma clara y efectiva la información recolectada por la aplicación.



Selección del Dataset

Sample - Superstore.csv (2.29 MB)

Detail Compact Column

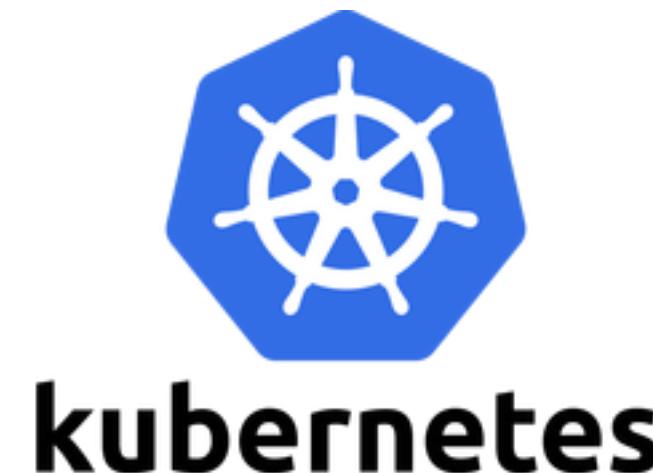
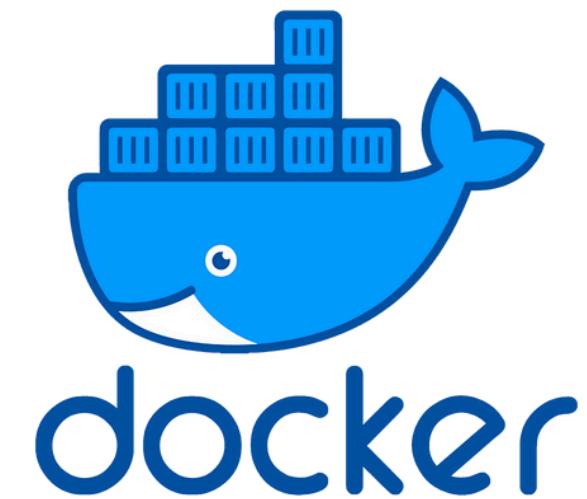
About this file

Dataset containing Information related to Sales, Profits and other interesting facts of a Superstore giant.

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City			
1	9994	5009 unique values	2014-01-02 2017-12-29	2014-01-06 2018-01-04	Standard Class Second Class Other (2081)	60% 19% 21%	793 unique values	793 unique values	Consumer Corporate Other (1783)	52% 30% 18%		New York City 9% Los Angeles 7% Other (8332) 83%
1	CA-2016-152156	11/8/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson			
2	CA-2016-152156	11/8/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson			
3	CA-2016-138688	6/12/2016	6/16/2016	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles			
4	US-2015-108966	10/11/2015	10/18/2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale			
5	US-2015-108966	10/11/2015	10/18/2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale			
6	CA-2014-115812	6/9/2014	6/14/2014	Standard Class	BH-11710	Brosina Hoffman	Consumer	United States	Los Angeles			
7	CA-2014-115812	6/9/2014	6/14/2014	Standard Class	BH-11710	Brosina Hoffman	Consumer	United States	Los Angeles			
8	CA-2014-115812	6/9/2014	6/14/2014	Standard Class	BH-11710	Brosina Hoffman	Consumer	United States	Los Angeles			
9	CA-2014-115812	6/9/2014	6/14/2014	Standard Class	BH-11710	Brosina Hoffman	Consumer	United States	Los Angeles			
10	CA-2014-115812	6/9/2014	6/14/2014	Standard Class	BH-11710	Brosina Hoffman	Consumer	United States	Los Angeles			
11	CA-2014-115812	6/9/2014	6/14/2014	Standard Class	BH-11710	Brosina Hoffman	Consumer	United States	Los Angeles			
12	CA-2014-115812	6/9/2014	6/14/2014	Standard Class	BH-11710	Brosina Hoffman	Consumer	United States	Los Angeles			
13	CA-2017-114412	4/15/2017	4/20/2017	Standard Class	AA-10480	Andrew Allen	Consumer	United States	Concord			

Alternativas de solución

Despliegue y empaquetamiento

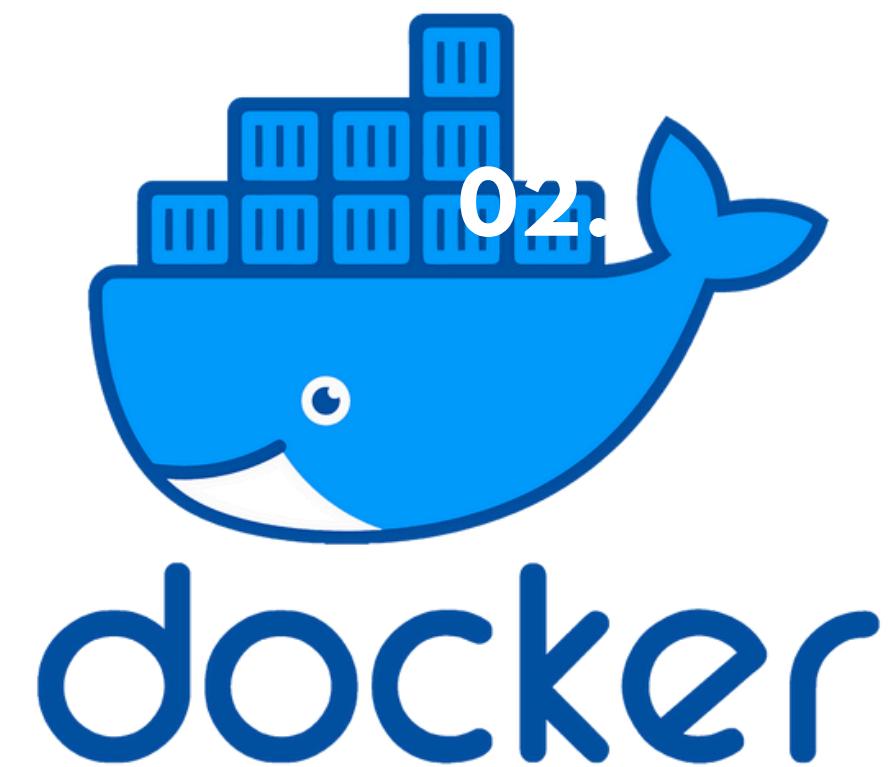


kubernetes

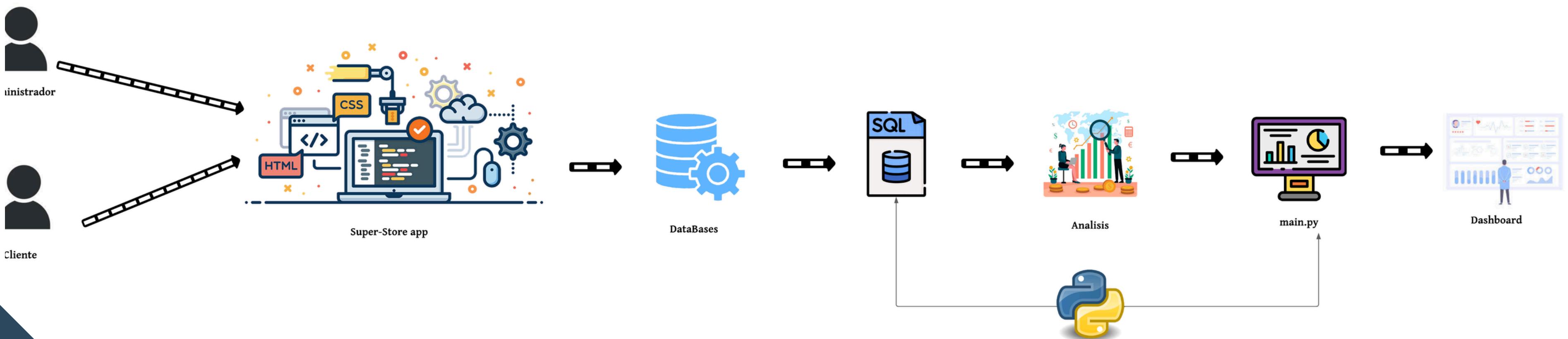
Procesamiento distribuido



Alternativas de solución escogidas



Arquitectura del Sistema



Microservicios



Usuarios



Productos



Carrito



Ordenes

APLICACIÓN WEB

http:192.168.100.2:8080 - 192.168.100.3:8080

The screenshot shows a web page with a light gray background. At the top, there's a yellow header bar. Below it, the title 'Super Store' is displayed in a large, bold, dark font. Underneath the title, the tagline 'Tu tienda en línea favorita' is written in a smaller, italicized dark font. In the center of the page, there's a paragraph of text: 'Explora una amplia variedad de productos de alta calidad, desde moda hasta tecnología. Compra fácil, rápido y seguro con Super Store.' Below this text is a blue button with the text 'Comenzar a Comprar' in white. On the far left edge of the page, there are several concentric pink circles.



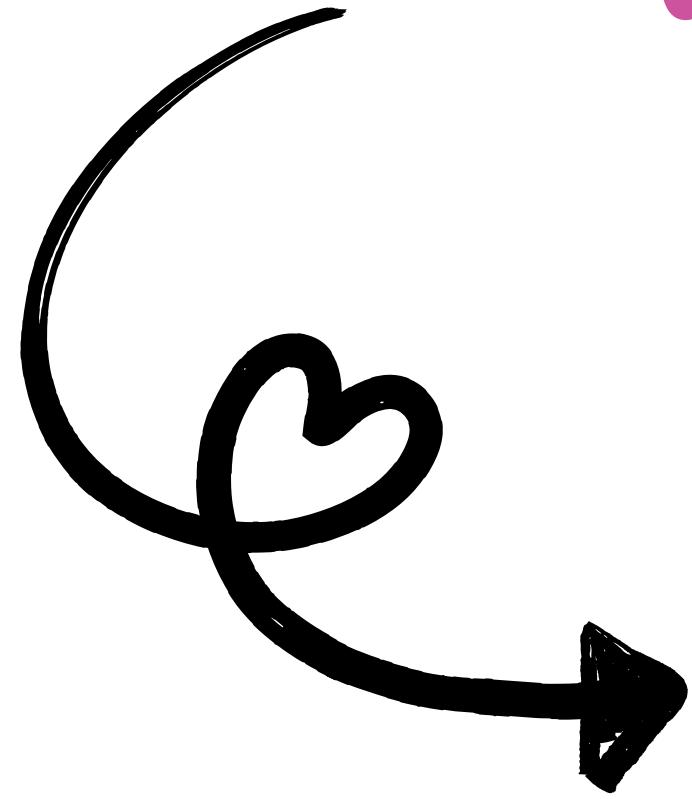
HTML



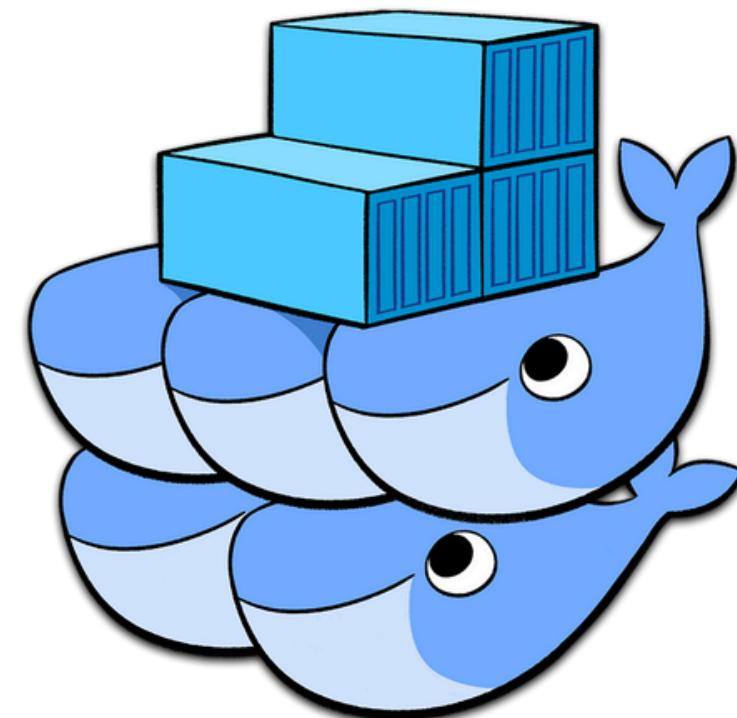
CSS



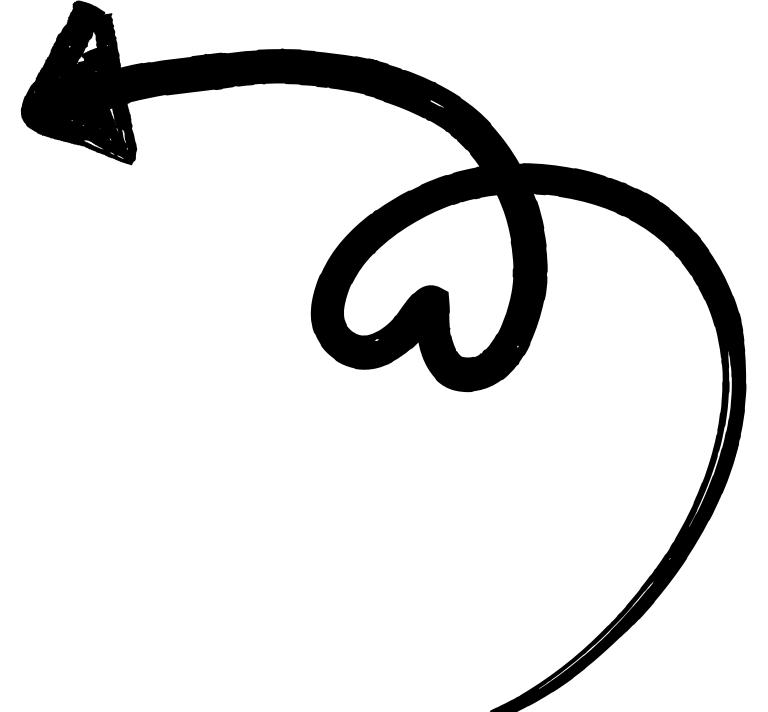
EMPQUETADO



docker



docker swarm



BALANCEO DE CARGA

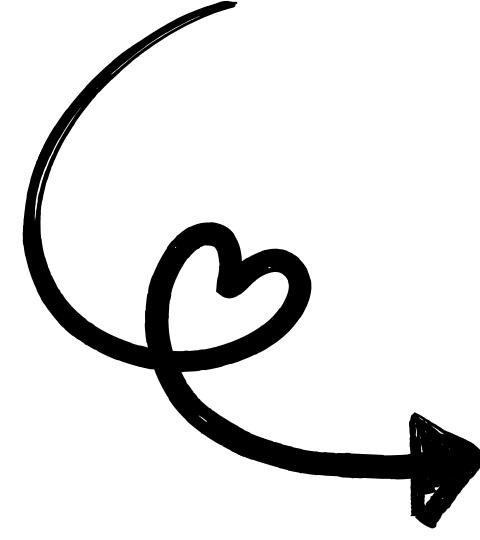
DOCKER-COMPOSE.YML

```
1 version: '3.8'
2 services:
3   usuarios:
4     image: angelaruiz2006/super-store-usuarios:v1
5     ports:
6       - "3009:3009"
7     networks:
8       - app-network
9   deploy:
10    replicas: 2
11 environment:
12   MYSQL_HOST: mysql
13   MYSQL_USER: root
14   MYSQL_PASSWORD: password
15   MYSQL_DATABASE: usuariosbd
16
17 productos:
18   image: angelaruiz2006/super-store-productos:v1
19   ports:
20     - "3001:3001"
21   networks:
22     - app-network
23   deploy:
24     replicas: 3
25
26 ordenes:
27   image: angelaruiz2006/super-store-ordenes:v1
28   ports:
29     - "3010:3010"
30   networks:
31     - app-network
32   environment:
33     PRODUCTS_URL: "http://productos:3001"
34     CART_URL: "http://carrito:3308"
35   deploy:
36     replicas: 3
```

```
1 carrito:
2   image: angelaruiz2006/super-store-carrito:v1
3   ports:
4     - "3308:3308"
5   networks:
6     - app-network
7   environment:
8     USUARIOS_URL: "http://usuarios:3009"
9     PRODUCTOS_URL: "http://productos:3001"
10  deploy:
11    replicas: 3
12
13 frontend:
14   image: angelaruiz2006/super-store-front:v1
15   ports:
16     - "8080:80"
17   networks:
18     - app-network
19   deploy:
20     replicas: 2
21
22 mysql:
23   image: angelaruiz2006/super-store-mysql:v1
24   environment:
25     MYSQL_ROOT_PASSWORD: password
26   volumes:
27     - mysql-data:/var/lib/mysql
28   ports:
29     - "3307:3306"
30   networks:
31     - app-network
32   deploy:
33     replicas: 1
34     resources:
35       limits:
36         cpus: '0.5'
37         memory: 512M
38       restart_policy:
39         condition: on-failure
40       max_attempts: 3
41
```

```
1 servidor:
2   image: angelaruiz2006/super-store-servidor:v1
3   ports:
4     - "3020:3020"
5   networks:
6     - app-network
7   deploy:
8     replicas: 2
9   environment:
10    FLASK_ENV: production
11
12 networks:
13   app-network:
14     driver: overlay
15
16 volumes:
17   mysql-data:
18     driver: local
```

DESARROLLO EN PYSPARK



EJECUCIÓN EN CLÚSTER SPARK

ARCHIVO MAIN.PY

```

1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import sum, col, desc, to_date, year, trim, month, when, mean
3 from pyspark.sql.types import DoubleType
4 import os
5 import json
6
7 # Rutas
8 INPUT_PATH = "/root/labSpark/dataset/sampleSuperstore.csv"
9 OUTPUT_PATH = "/root/resultados.txt"
10 JSON_OUTPUT_PATH = "/root/server/resuestas.json"
11
12 def initialize_spark(app_name):
13     return SparkSession.builder \
14         .appName(app_name) \
15         .config("spark.driver.memory", "4g") \
16         .getOrCreate()
17
18 def load_and_prepare_data(spark, input_path):
19     df = spark.read \
20         .option("header", "true") \
21         .option("inferSchema", "true") \
22         .option("encoding", "ISO-8859-1") \
23         .csv(input_path)
24
25     # Conversión y limpieza de columnas
26     df = df.withColumn("Sales", trim(col("Sales")).cast("double"))
27     df = df.withColumn("Profit", trim(col("Profit")).cast("double"))
28     df = df.withColumn("Discount", trim(col("Discount")).cast("double"))
29     df = df.withColumn("Quantity", trim(col("Quantity")).cast("int"))
30
31     # Formateo de fecha
32     df = df.withColumn("Order Date", to_date(col("Order Date"), "M/d/yyyy"))
33     df = df.withColumn("Year", year(col("Order Date")))
34
35     # Cálculo de precio con descuento
36     df = df.withColumn("Total Discounted Sales", col("Sales") * (1 - col("Discount")))
37
38     # Cálculo de temporada
39     df = df.withColumn("Temporada", when(col("Order Date").between('12-01', '02-28'), 'Invierno') \
40                         .when(col("Order Date").between('06-01', '08-31'), 'Verano') \
41                         .otherwise('Otoño/Primavera'))
42
43     df.cache()
44     return df
45
46 def analyze_ventas_1(df, output_file):
47     resultados = {}
48     with open(output_file, "w", encoding="utf-8") as f:
49         # 1. Producto más vendido por valor
50         ventas_producto = df.groupBy("Product Name").agg(sum("Sales").alias("TotalVentas"))
51         producto_top_ventas = ventas_producto.orderBy(desc("TotalVentas")).first()
52         if producto_top_ventas:
53             f.write("\n❶ [1] Producto más vendido (valor): {producto_top_ventas['Product Name']}\\n")
54             f.write("❷ Ventas totales: ${producto_top_ventas['TotalVentas']:.2f}\\n")
55             resultados["producto_mas_vendido_valor"] = {
56                 "producto": producto_top_ventas['Product Name'],
57                 "ventas_totales": round(producto_top_ventas['TotalVentas'], 2)
58             }
59
60         # 2. Producto con más unidades vendidas
61         cantidad_producto = df.groupBy("Product Name").agg(sum("Quantity").alias("TotalUnidades"))
62         producto_top_unidades = cantidad_producto.orderBy(desc("TotalUnidades")).first()
63         if producto_top_unidades:
64             f.write("\n❶ [2] Producto más vendido (unidades): {producto_top_unidades['Product Name']}\\n")
65             f.write("❷ Unidades vendidas: {producto_top_unidades['TotalUnidades']}\\n")
66             resultados["producto_mas_vendido_unidades"] = {
67                 "producto": producto_top_unidades['Product Name'],
68                 "unidades_vendidas": int(producto_top_unidades['TotalUnidades'])
69             }
70
71         # 3. Producto más rentable
72         ganancia_producto = df.groupBy("Product Name").agg(sum("Profit").alias("TotalGanancia"))
73         producto_top_ganancia = ganancia_producto.orderBy(desc("TotalGanancia")).first()
74         if producto_top_ganancia:
75             f.write("\n❶ [3] Producto más rentable: {producto_top_ganancia['Product Name']}\\n")
76             f.write("❷ Ganancia total: ${producto_top_ganancia['TotalGanancia']:.2f}\\n")
77             resultados["producto_mas_rentable"] = {
78                 "producto": producto_top_ganancia['Product Name'],
79                 "ganancia_total": round(producto_top_ganancia['TotalGanancia'], 2)
80             }
81
82     # 4. Ciudad con más ventas
83     ventas_ciudad = df.groupBy("City").agg(sum("Sales").alias("TotalVentas"))
84     ciudad_top = ventas_ciudad.orderBy(desc("TotalVentas")).first()
85     if ciudad_top:
86         f.write("\n❶ [4] Ciudad con más ventas: {ciudad_top['City']}\\n")
87         f.write("❷ Ventas totales: ${ciudad_top['TotalVentas']:.2f}\\n")
88         resultados["ciudad_mas_ventas"] = {
89             "ciudad": ciudad_top['City'],
90             "ventas_totales": round(ciudad_top['TotalVentas'], 2)
91         }
92
93         # 5 y 6. Año con menos y más ventas
94         ventas_ano = df.groupBy("Year").agg(sum("Sales").alias("TotalVentas"))
95         ano_min = ventas_ano.orderBy(desc("TotalVentas")).first()
96         ano_max = ventas_ano.orderBy(desc("TotalVentas")).first()
97         if ano_max:
98             f.write("\n❶ [5] Año con menos ventas: {ano_min['Year']}\\n")
99             f.write("❷ Ventas totales: ${ano_min['TotalVentas']:.2f}\\n")
100            resultados["ano_menos_ventas"] = {
101                "ano": int(ano_min['Year']),
102                "ventas_totales": round(ano_min['TotalVentas'], 2)
103            }
104
105         if ano_max:
106             f.write("\n❶ [6] Año con más ventas: {ano_max['Year']}\\n")
107             f.write("❷ Ventas totales: ${ano_max['TotalVentas']:.2f}\\n")
108             resultados["ano_mas_ventas"] = {
109                 "ano": int(ano_max['Year']),
110                 "ventas_totales": round(ano_max['TotalVentas'], 2)
111             }
112
113     return resultados
114
115 def analyze_ventas_2(df, output_file):
116     resultados = {}
117     with open(output_file, "a", encoding="utf-8") as f:
118         f.write("\n\n")
119         f.write("❸ ANÁLISIS DE VENTAS ==\\n")
120
121         # Filtrar filas con descuento válido
122         df_valid_descuento = df.filter(
123             (col("Discount").cast(DoubleType()).isNotNull()) &
124             (col("Discount") > 0) &
125             (col("Sales").cast(DoubleType()).isNotNull())
126         )
127
128         # 7. Producto con mayor descuento
129         if df_valid_descuento.count() > 0:
130             max_descuento = df_valid_descuento.orderBy(desc("Discount")).first()
131             f.write("\n❹ [7] Producto con mayor descuento: {max_descuento['Product Name']}\\n")
132             f.write("❺ Precio original: ${float(max_descuento['Sales']):.2f}\\n")
133             f.write("❻ Descuento aplicado: ${float(max_descuento['Discount']) * 100:.0f}%\\n")
134             f.write("❻ Precio con descuento: ${float(max_descuento['Total Discounted Sales']):.2f}\\n")
135             resultados["producto_mayor_descuento"] = {
136                 "producto": max_descuento['Product Name'],
137                 "precio_original": round(float(max_descuento['Sales']), 2),
138                 "descuento_porcentaje": round(float(max_descuento['Discount']) * 100, 0),
139                 "precio_con_descuento": round(float(max_descuento['Total Discounted Sales']), 2)
140             }
141
142         else:
143             f.write("\n❹ [7] No se encontraron productos con descuento aplicable\\n")
144             resultados["producto_mayor_descuento"] = {
145                 "mensaje": "No se encontraron productos con descuento aplicable"
146             }
147
148         # 8. Producto con menor descuento
149         if df_valid_descuento.count() > 0:
150             min_descuento = df_valid_descuento.orderBy("Discount").first()
151             f.write("\n❹ [8] Producto con menor descuento: {min_descuento['Product Name']}\\n")
152             f.write("❺ Unidades vendidas: {min_descuento['Quantity']}\\n")
153             f.write("❻ Descuento aplicado: ${float(min_descuento['Discount']) * 100:.0f}%\\n")
154             resultados["producto_menor_descuento"] = {
155                 "producto": min_descuento['Product Name'],
156                 "unidades_vendidas": int(min_descuento['Quantity']),
157                 "descuento_porcentaje": round(float(min_descuento['Discount']) * 100, 0)
158             }
159
160         else:
161             f.write("\n❹ [8] No se encontraron productos con descuento mayor a cero\\n")
162             resultados["producto_menor_descuento"] = {
163                 "mensaje": "No se encontraron productos con descuento mayor a cero"
164             }
165
166         # 9. Top 5 productos por cantidad vendida
167         top5_cantidad = df_valid_descuento.groupBy("Product Name") \
168             .agg(mean("Discount").alias("DescuentoPromedio"),
169                  sum("Quantity").alias("TotalUnidades"),
170                  sum("Sales").alias("TotalVentas")) \
171             .orderBy(desc("TotalUnidades")) \
172             .limit(5)
173
174         f.write("\n❹ [9] Top 5 productos por cantidad vendida:\\n")
175         top5_list = []
176         for row in top5_cantidad.collect():
177             f.write("❺ Producto: {row['Product Name']}, Unidades: {row['TotalUnidades']}\\n")
178             f.write("❻ Ventas: ${float(row['TotalVentas']):.2f}, \"")
179             f.write("❻ Descuento Promedio: ${float(row['DescuentoPromedio']) * 100:.2f}%\\n")
180             top5_list.append({
181                 "producto": row['Product Name'],
182                 "unidades": int(row['TotalUnidades']),
183                 "ventas": round(float(row['TotalVentas']), 2),
184                 "descuento_promedio_porcentaje": round(float(row['DescuentoPromedio']) * 100, 2)
185             })
186
187         resultados["top5_productos_cantidad"] = top5_list
188
189     return resultados
190
191 def main():
192     # Crear la carpeta /root/server/ si no existe
193     os.makedirs(os.path.dirname(JSON_OUTPUT_PATH), exist_ok=True)
194
195     # Eliminar archivos de salida si existen
196     if os.path.exists(JSON_OUTPUT_PATH):
197         os.remove(JSON_OUTPUT_PATH)
198
199     # Ejecutar análisis y combinar resultados
200     resultados = []
201     resultados.update(analyze_ventas_1(df, OUTPUT_PATH))
202     resultados.update(analyze_ventas_2(df, OUTPUT_PATH))
203     resultados.update(analyze_ventas_3(df, OUTPUT_PATH))
204
205     # Guardar resultados en JSON
206     with open(JSON_OUTPUT_PATH, "w", encoding="utf-8") as json_file:
207         json.dump(resultados, json_file, ensure_ascii=False, indent=4)
208
209     print("Analisis completado. Resultados guardados en: (OUTPUT_PATH)")
210
211     finally:
212         df.unpersist()
213
214 if __name__ == "__main__":
215     main()

```

```

1 # 4. Ciudad con más ventas
2 ventas_ciudad = df.groupBy("City").agg(sum("Sales").alias("TotalVentas"))
3 ciudad_top = ventas_ciudad.orderBy(desc("TotalVentas")).first()
4 if ciudad_top:
5     f.write("\n❶ [4] Ciudad con más ventas: {ciudad_top['City']}\\n")
6     f.write("❷ Ventas totales: ${ciudad_top['TotalVentas']:.2f}\\n")
7     resultados["ciudad_mas_ventas"] = {
8         "ciudad": ciudad_top['City'],
9         "ventas_totales": round(ciudad_top['TotalVentas'], 2)
10    }
11
12 # 5 y 6. Año con menos y más ventas
13 ventas_ano = df.groupBy("Year").agg(sum("Sales").alias("TotalVentas"))
14 ano_min = ventas_ano.orderBy(desc("TotalVentas")).first()
15 ano_max = ventas_ano.orderBy(desc("TotalVentas")).first()
16 if ano_max:
17     f.write("\n❶ [5] Año con menos ventas: {ano_min['Year']}\\n")
18     f.write("❷ Ventas totales: ${ano_min['TotalVentas']:.2f}\\n")
19     resultados["ano_menos_ventas"] = {
20         "ano": int(ano_min['Year']),
21         "ventas_totales": round(ano_min['TotalVentas'], 2)
22    }
23
24 if ano_max:
25     f.write("\n❶ [6] Año con más ventas: {ano_max['Year']}\\n")
26     f.write("❷ Ventas totales: ${ano_max['TotalVentas']:.2f}\\n")
27     resultados["ano_mas_ventas"] = {
28         "ano": int(ano_max['Year']),
29         "ventas_totales": round(ano_max['TotalVentas'], 2)
30    }
31
32 return resultados
33
34 def analyze_ventas_3(df, output_file):
35     resultados = {}
36     with open(output_file, "a", encoding="utf-8") as f:
37         f.write("\n\n")
38         f.write("❸ ANÁLISIS ADICIONAL ==\\n")
39
40         # 12. Mes con más ventas
41         ventas_por_mes = df.groupBy(month("Order Date").alias("Month")) \
42             .agg(sum("Sales").alias("TotalVentas"))
43         mes_top = ventas_por_mes.first()
44         if mes_top:
45             f.write("\n❶ [12] Mes con más ventas: {mes_top['Month']}\\n")
46             f.write("❷ Ventas totales: ${mes_top['TotalVentas']:.2f}\\n")
47             resultados["mes_mas_ventas"] = {
48                 "mes": int(mes_top['Month']),
49                 "total_ventas": round(mes_top['TotalVentas'], 2)
50             }
51
52         # 13. Temporada con más ventas
53         ventas_por_temporada = df.groupby("Temporada") \
54             .agg(sum("Sales").alias("TotalVentas"))
55         temporada_top = ventas_por_temporada.first()
56         if temporada_top:
57             f.write("\n❶ [13] Temporada con más ventas: {temporada_top['Temporada']}\\n")
58             f.write("❷ Ventas totales: ${temporada_top['TotalVentas']:.2f}\\n")
59             resultados["meses_mas_ventas"] = {
60                 "mes": int(temporada_top['Temporada']),
61                 "total_ventas": round(temporada_top['TotalVentas'], 2)
62             }
63
64         # 14. Ciudad con más ventas
65         ventas_por_ciudad = df.groupby("City").alias("TotalVentas")) \
66             .orderBy(desc("TotalVentas"))
67         ciudad_top = ventas_por_ciudad.first()
68         if ciudad_top:
69             f.write("\n❶ [14] Ciudad con más ventas: {ciudad_top['City']}\\n")
70             f.write("❷ Ventas totales: ${ciudad_top['TotalVentas']:.2f}\\n")
71             resultados["ciudad_mas_ventas"] = {
72                 "ciudad": ciudad_top['City'],
73                 "total_ventas": round(ciudad_top['TotalVentas'], 2)
74             }
75
76         # 15. Productos con ganancia negativa
77         productos_negativos = df.filter(col("Profit") < 0)
78         productos_negativos_list = []
79         for row in productos_negativos.collect():
80             f.write("❶ [15] Producto con ganancia negativa: (row['Product Name'], \"")
81             f.write("❷ Ganancia: ${row['Profit']}, \"")
82             f.write("❸ Fecha: ${row['Order Date']}\\n")
83             productos_negativos_list.append((row['Product Name'], row['Profit'], str(row['Order Date'])))
84
85         resultados["productos_com_ganancia_negativa"] = productos_negativos_list
86
87         # 16. Región con mayor descuento promedio
88         descuento_por_region = df.groupby("Region") \
89             .agg(sum("Discount").alias("PromedioDescuento")) \
90             .orderBy(desc("PromedioDescuento"))
91         region_top_descuento = descuento_por_region.first()
92         if region_top_descuento:
93             f.write("\n❶ [16] Región con mayor descuento promedio: ({region_top_descuento['Region']})\\n")
94             f.write("❷ Promedio de descuento: ${region_top_descuento['PromedioDescuento']:.2f}\\n")
95             resultados["region_mayor_descuento"] = {
96                 "region": region_top_descuento['Region'],
97                 "promedio_descuento": round(region_top_descuento['PromedioDescuento'], 2)
98             }
99
100         # 17. Región con más unidades vendidas
101         ventas_por_region = df.groupby("Region") \
102             .agg(sum("Quantity").alias("TotalUnidades"))
103         region_top_unidades = ventas_por_region.first()
104         if region_top_unidades:
105             f.write("\n❶ [17] Región con más unidades vendidas: ({region_top_unidades['Region']})\\n")
106             f.write("❷ Total unidades vendidas: ${region_top_unidades['TotalUnidades']:.2f}\\n")
107             resultados["region_mas_unidades"] = {
108                 "region": region_top_unidades['Region'],
109                 "total_unidades": int(region_top_unidades['TotalUnidades'])
110             }
111
112         # 18. Categoría con más ventas
113         ventas_por_categoria = df.groupby("Category") \
114             .agg(sum("Sales").alias("TotalVentas"))
115         categoria_top = ventas_por_categoria.first()
116         if categoria_top:
117             f.write("\n❶ [18] Categoría con más ventas: ({categoria_top['Category']})\\n")
118             f.write("❷ Ventas totales: ${categoria_top['TotalVentas']:.2f}\\n")
119             resultados["categoria_mas_ventas"] = {
120                 "categoria": categoria_top['Category'],
121                 "total_ventas": round(categoria_top['TotalVentas'], 2)
122             }
123
124     return resultados
125
126 def main():
127     # Crear la carpeta /root/server/ si no existe
128     os.makedirs(os.path.dirname(JSON_OUTPUT_PATH), exist_ok=True)
129
130     # Eliminar archivos de salida si existen
131     if os.path.exists(JSON_OUTPUT_PATH):
132         os.remove(JSON_OUTPUT_PATH)
133
134     # Ejecutar análisis y combinar resultados
135     resultados = []
136     resultados.update(analyze_ventas_1(df, OUTPUT_PATH))
137     resultados.update(analyze_ventas_2(df, OUTPUT_PATH))
138     resultados.update(analyze_ventas_3(df, OUTPUT_PATH))
139
140     # Guardar resultados en JSON
141     with open(JSON_OUTPUT_PATH, "w", encoding="utf-8") as json_file:
142         json.dump(resultados, json_file, ensure_ascii=False, indent=4)
143
144     print("Analisis completado. Resultados guardados en: (OUTPUT_PATH)")
145
146     finally:
147         df.unpersist()
148
149 if __name__ == "__main__":
150     main()

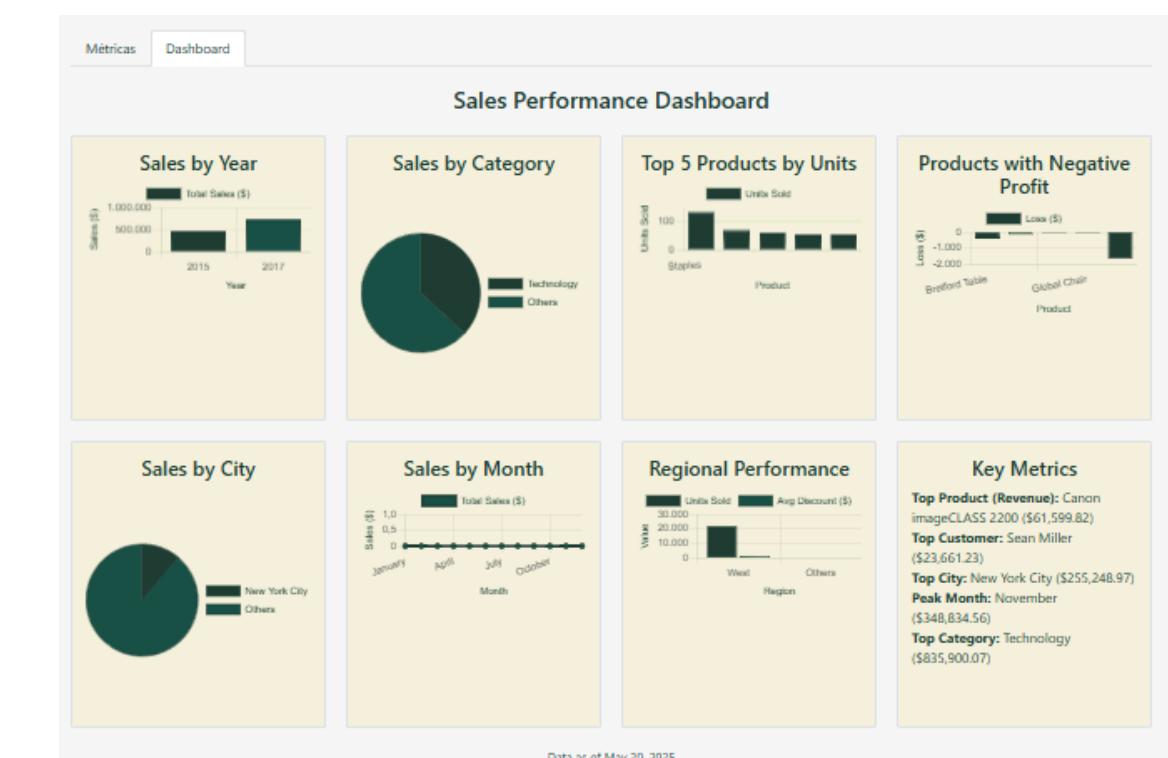
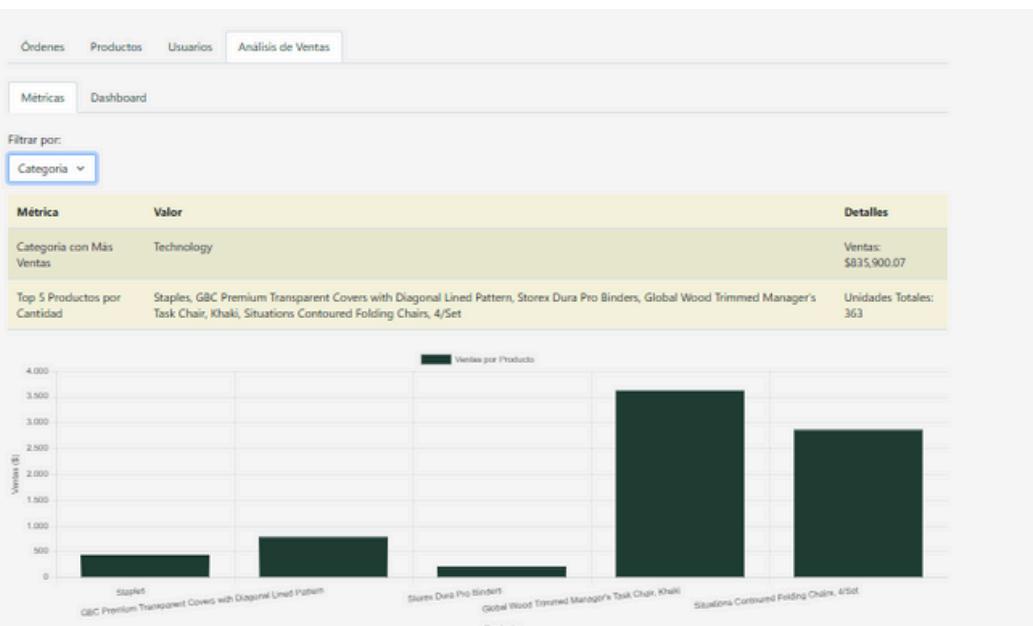
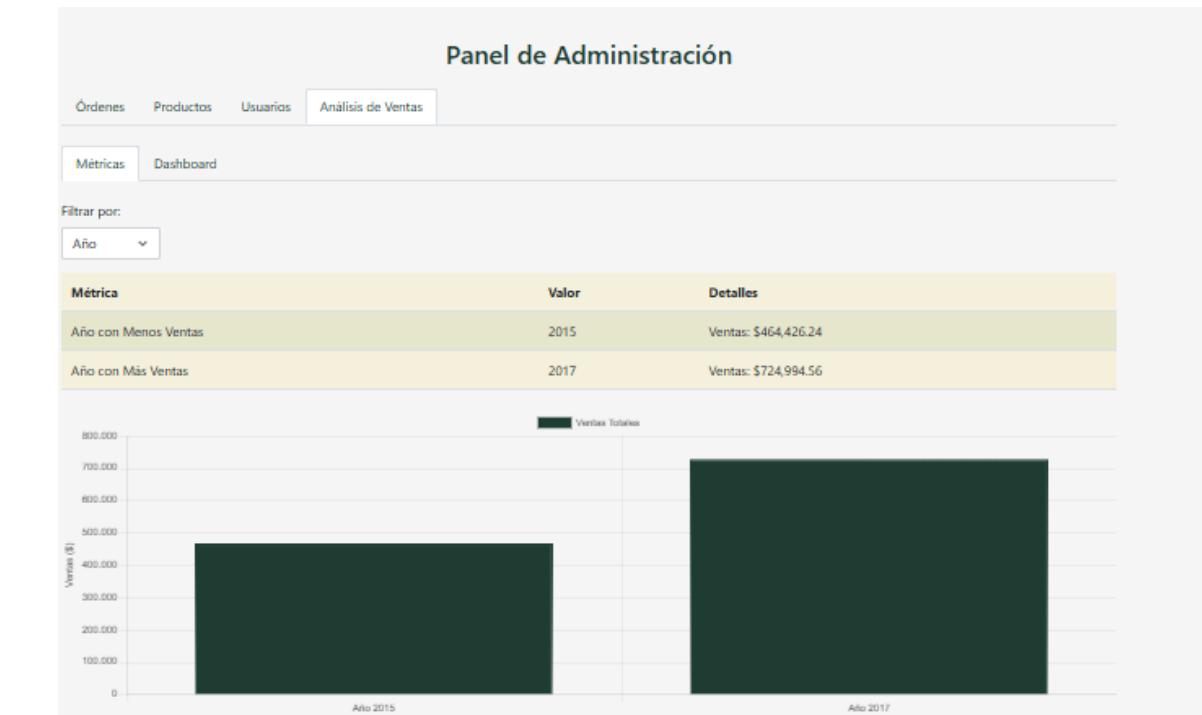
```

```

1 cliente_top = df.valid_descuento.groupBy("Customer Name", "City") \
2     .agg(sum("Sales").alias("TotalCompras")) \
3     .orderBy(desc("TotalCompras")) \
4     .first()
5
6 if cliente_top:
7     f.write("\n❶ [1] Cliente que más compró: {cliente_top['Customer Name']}\\n")
8     f.write("❷ Total comprado: ${float(cliente_top['TotalCompras']):.2f}\\n")
9     resultados["cliente_mas_compró"] = {
10         "cliente": cliente_top['Customer Name'],
11         "ciudad": cliente_top['City'],
12         "total_comprado": round(cliente_top['TotalCompras'], 2)
13     }
14
15 else:
16     f.write("\n❶ [2] No se encontraron datos para el cliente que más compró\\n")
17     resultados["cliente_mas_compró"] = {
18         "mensaje": "No se encontraron datos para el cliente que más compró"
19     }
20
21 # 11. Producto más rentable
22 producto_rentable = df.valid_descuento.groupby("Product Name") \
23     .agg(sum("Profit").alias("TotalGanancia")) \
24     .orderBy(desc("TotalGanancia")) \
25     .first()
26
27 if producto_rentable:
28     f.write("\n❶ [11] Producto más rentable: (producto_rentable['Product Name'])\\n")
29     f.write("❷ Ganancia total: ${float(producto_rentable['TotalGanancia']):.2f}\\n")
30     resultados["producto_mas_rentable"] = {
31         "producto": producto_rentable['Product Name'],
32         "ganancia_total": round(producto_rentable['TotalGanancia'], 2)
33     }
34
35 else:
36     f.write("\n❶ [12] No se encontraron datos para el producto más rentable\\n")
37     resultados["producto_mas_rentable"] = {
38         "mensaje": "No se encontraron datos para el producto más rentable"
39     }
40
41 return resultados
42
43 def analyze_ventas_1(df, output_file):
44     resultados = []
45     with open(output_file, "a", encoding="utf-8") as f:
46         f.write("\n\n")
47         f.write("❸ ANÁLISIS ADICIONAL ==\\n")
48
49         # 12. Mes con más ventas
50         ventas_por_mes = df.valid_descuento.groupBy(month("Order Date").alias("Month")) \
51             .agg(sum("Sales").alias("TotalVentas"))
52         mes_top = ventas_por_mes.first()
53         if mes_top:
54             f.write("\n❶ [12] Mes con más ventas: {mes_top['Month']}\\n")
55             f.write("❷ Ventas totales: ${mes_top['TotalVentas']:.2f}\\n")
56             resultados["meses_mas_ventas"] = {
57                 "mes": int(mes_top['Month']),
58                 "total_ventas": round(mes_top['TotalVentas'], 2)
59             }
60
61         # 13. Temporada con más ventas
62         ventas_por_temporada = df.groupby("Temporada") \
63             .agg(sum("Sales").alias("TotalVentas"))
64         temporada_top = ventas_por_temporada.first()
65         if temporada_top:
66             f.write("\n❶ [13] Temporada con más ventas: {temporada_top['Temporada']}\\n")
67             f.write("❷ Ventas totales: ${temporada_top['TotalVentas']:.2f}\\n")
68             resultados["temporadas_mas_ventas"] = {
69                 "temporada": temporada_top['Temporada'],
70                 "total_ventas": round(temporada_top['TotalVentas'], 2)
71             }
72
73         # 14. Ciudad con más ventas
74         ventas_por_ciudad = df.valid_descuento.groupby("City").alias("TotalVentas")) \
75             .orderBy(desc("TotalVentas"))
76         ciudad_top = ventas_por_ciudad.first()
77         if ciudad_top:
78             f.write("\n❶ [14] Ciudad con más ventas: {ciudad_top['City']}\\n")
79             f.write("❷ Ventas totales: ${ciudad_top['TotalVentas']:.2f}\\n")
80             resultados["ciudad_mas_ventas"] = {
81                 "ciudad": ciudad_top['City'],
82                 "total_ventas": round(ciudad_top['TotalVentas'], 2)
83             }
84
85         # 15. Productos con ganancia negativa
86         productos_negativos = df.filter(col("Profit") < 0)
87         productos_negativos_list = []
88         for row in productos_negativos.collect():
89             f.write("❶ [15] Producto con ganancia negativa: (row['Product Name'], \"")
90             f.write("❷ Ganancia: ${row['Profit']}, \"")
91             f.write("❸ Fecha: ${row['Order Date']}\\n")
92             productos_negativos_list.append((row['Product Name'], row['Profit'], str(row['Order Date'])))
93
94         resultados["productos_com_ganancia_negativa"] = productos_negativos_list
95
96         # 16. Región con mayor descuento promedio
97         descuento_por_region = df.groupby("Region") \
98             .agg(sum("Discount").alias("PromedioDescuento")) \
99             .orderBy(desc("PromedioDescuento"))
100         region_top_descuento = descuento_por_region.first()
101         if region_top_descuento:
102             f.write("\n❶ [16] Región con mayor descuento promedio: ({region_top_descuento['Region']})\\n")
103             f.write("❷ Promedio de descuento: ${region_top_descuento['PromedioDescuento']:.2f}\\n")
104             resultados["region_mayor_descuento"] = {
105                 "region": region_top_descuento['Region'],
106                 "promedio_descuento": round(region_top_descuento['PromedioDescuento'], 2)
107             }
108
109         # 17. Región con más unidades vendidas
110         ventas_por_region = df.groupby("Region") \
111             .agg(sum("Quantity").alias("TotalUnidades"))
112         region_top_unidades = ventas_por_region.first()
113         if region_top_unidades:
114             f.write("\n❶ [17] Región con más unidades vendidas: ({region_top_unidades['Region']})\\n")
115             f.write("❷ Total unidades vendidas: ${region_top_unidades['TotalUnidades']:.2f}\\n")
116             resultados["region_mas_unidades"] = {
117                 "region": region_top_unidades['Region'],
118                 "total_unidades": int(region_top_unidades['TotalUnidades'])
119             }
120
121         # 18. Categoría con más ventas
122         ventas_por_categoria = df.groupby("Category") \
123             .agg(sum("Sales").alias("TotalVentas"))
124         categoria_top = ventas_por_categoria.first()
125         if categoria_top:
126             f.write("\n❶ [18] Categoría con más ventas: ({categoria_top['Category']})\\n")
127             f.write("❷ Ventas totales: ${categoria_top['TotalVentas']:.2f}\\n")
128             resultados["categoria_mas_ventas"] = {
129                 "categoria": categoria_top['Category'],
130                 "total_ventas": round(categoria_top['TotalVentas'], 2)
131             }
132
133     return resultados
134
135 def main():
136     # Crear la carpeta /root/server/ si no existe
137     os.makedirs(os.path.dirname(JSON_OUTPUT_PATH), exist_ok=True)
138
139     # Eliminar archivos de salida si existen
140     if os.path.exists(JSON_OUTPUT_PATH):
141         os.remove(JSON_OUTPUT_PATH)
142
143     spark = initialize_spark("AnálisisFinalCompleto")
144     df = load_and_prepare_data(spark, INPUT_PATH)
145
146     try:
147         # Ejecutar análisis y combinar resultados
148         resultados = []
149         resultados.update(analyze_ventas_1(df, OUTPUT_PATH))
150         resultados.update(analyze_ventas_2(df, OUTPUT_PATH))
151         resultados.update(analyze_ventas_3(df, OUTPUT_PATH))
152
153         # Guardar resultados en JSON
154         with open(JSON_OUTPUT_PATH, "w", encoding="utf-8") as json_file:
155             json.dump(resultados, json_file, ensure_ascii=False, indent=4)
156
157         print("Analisis completado. Resultados guardados en: (OUTPUT_PATH)")
158
159         finally:
160             df.unpersist()
161
162             # Finalizar
163             spark.stop()
164
165 if __name__ == "__main__":
166     main()

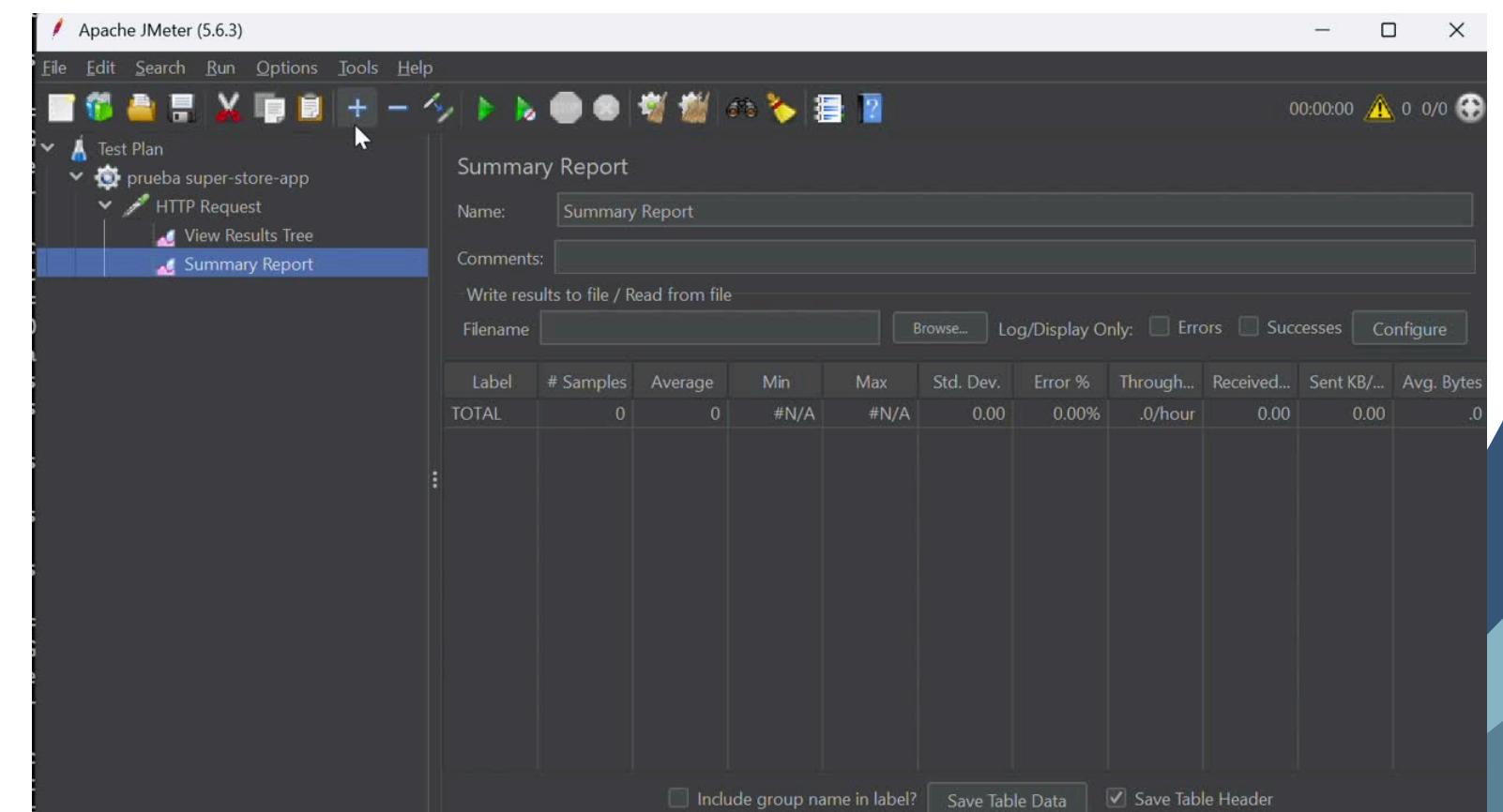
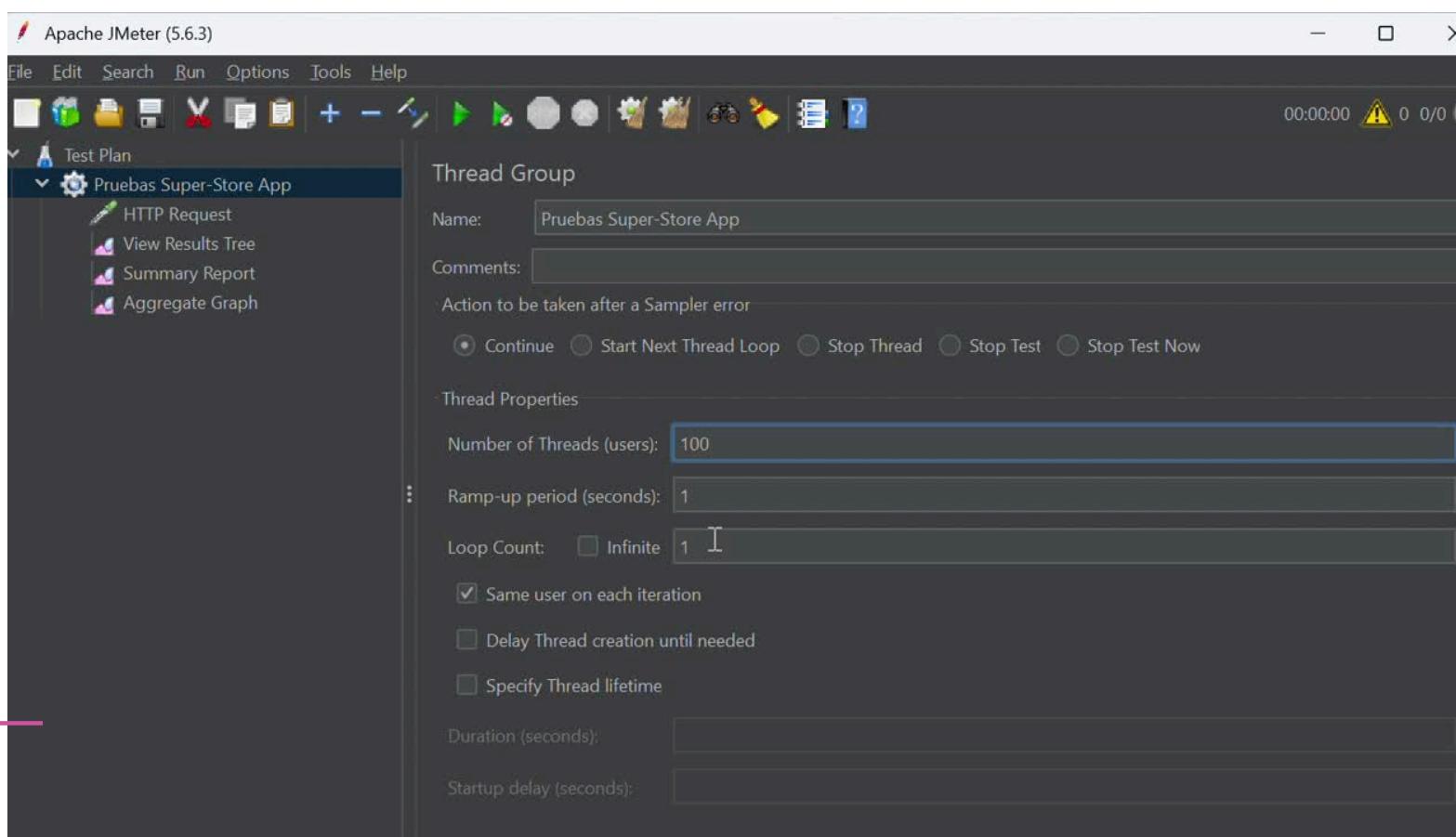
```

Resultados del análisis

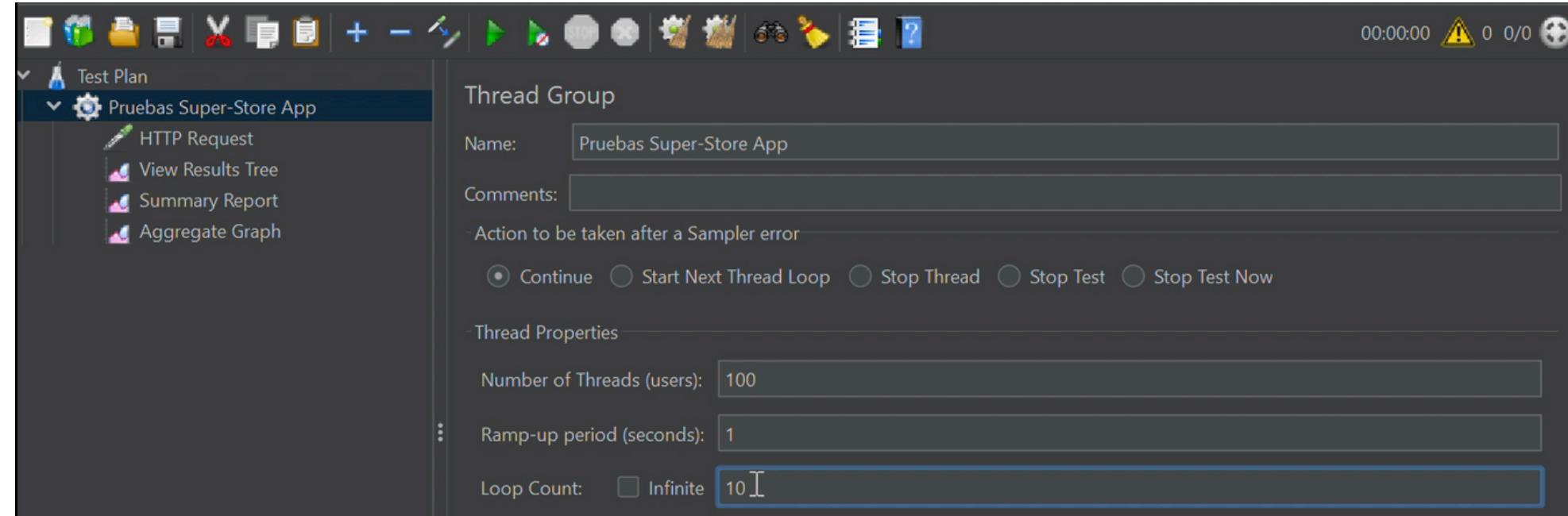


PRUEBAS DE ESCALABILIDAD

```
root@servidorUbuntu:~# docker service ls
ID          NAME      MODE      REPLICAS  IMAGE
wdecfgn3kjda  super-store_carrito  replicated  3/3        angelarui2006/super-store-carrito:v1
ihwjwpz6xtky  super-store_frontend  replicated  2/2        angelarui2006/super-store-front:v1
2arwaes0rg3a   super-store_mysql    replicated  2/2        angelarui2006/super-store-mysql:v1
hbzk04bx5e0    super-store_ordenes  replicated  3/3        angelarui2006/super-store-ordenes:v1
m5oxbrxvnzot  super-store_productos replicated  3/3        angelarui2006/super-store-productos:v1
r6izqf0nbpsy   super-store_servidor  replicated  2/2        angelarui2006/super-store-servidor:v1
mikq2yxwl9w2   super-store_usuarios replicated  2/2        angelarui2006/super-store-usuarios:v1
PORTS
*:3308->3308/tcp
*:8080->80/tcp
*:3307->3306/tcp
*:3010->3010/tcp
*:3001->3001/tcp
*:3020->3020/tcp
*:3009->3009/tcp
root@servidorUbuntu:~# |
```

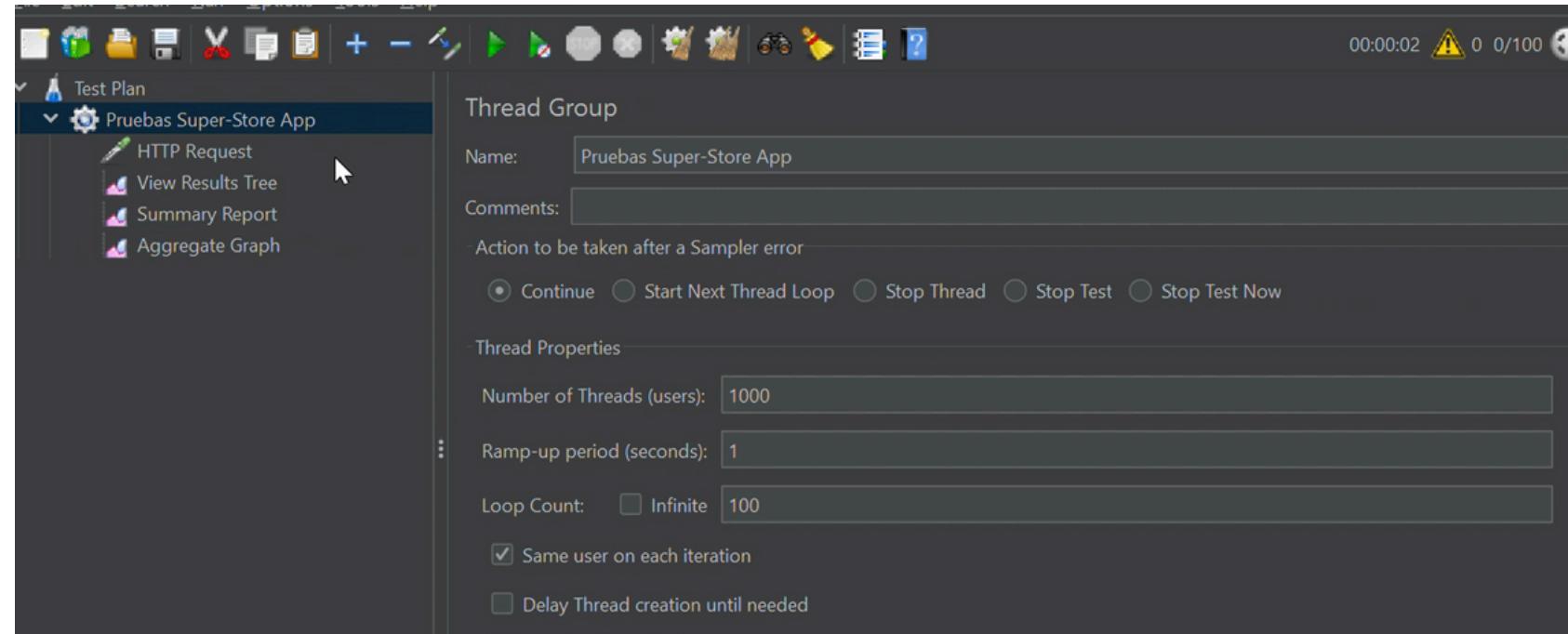


PRUEBAS DE ESCALABILIDAD



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Through...	Received...	Sent KB/...	Avg. Bytes
HTTP Re...	1000	77	1	1084	188.78	0.00%	486.6/sec	2876.04	57.03	6052.1
TOTAL	1000	77	1	1084	188.78	0.00%	486.6/sec	2876.04	57.03	6052.1

PRUEBAS DE ESCALABILIDAD



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Through...	Received...	Sent KB/...	Avg. Bytes
HTTP Re...	101000	736	0	77494	4223.99	0.00%	419.8/sec	2481.16	49.20	6052.0
TOTAL	101000	736	0	77494	4223.99	0.00%	419.8/sec	2481.16	49.20	6052.0

PRUEBAS DE ESCALABILIDAD

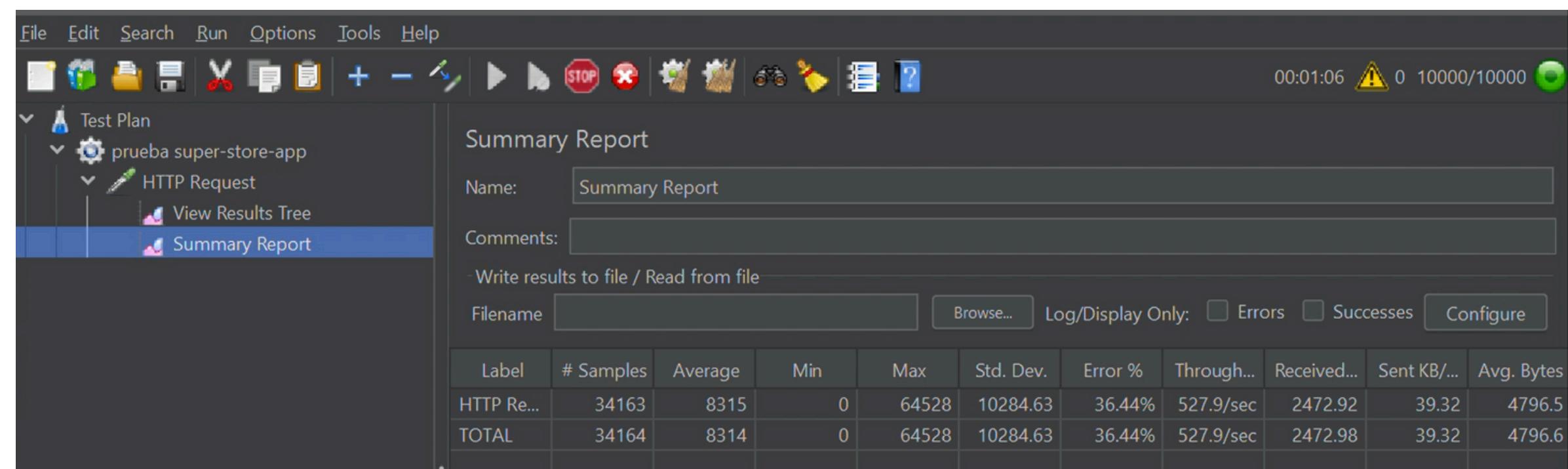
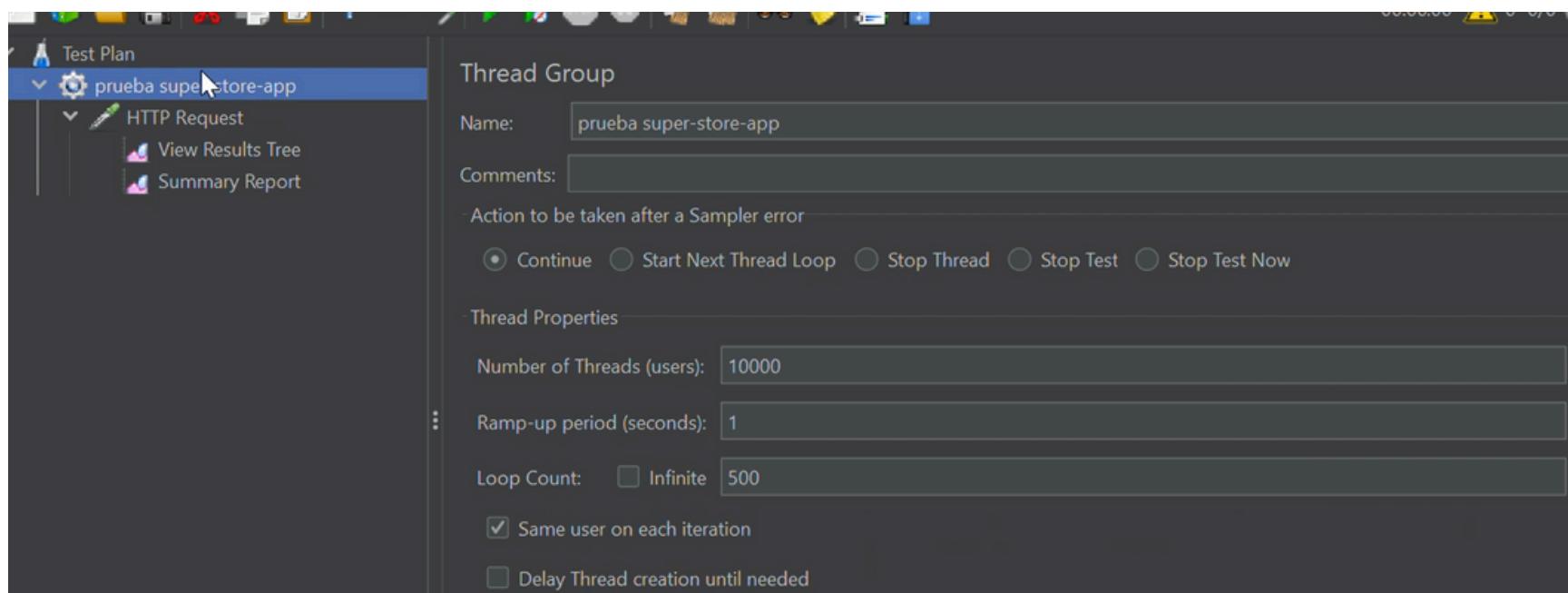


Diagrama de Despliegue

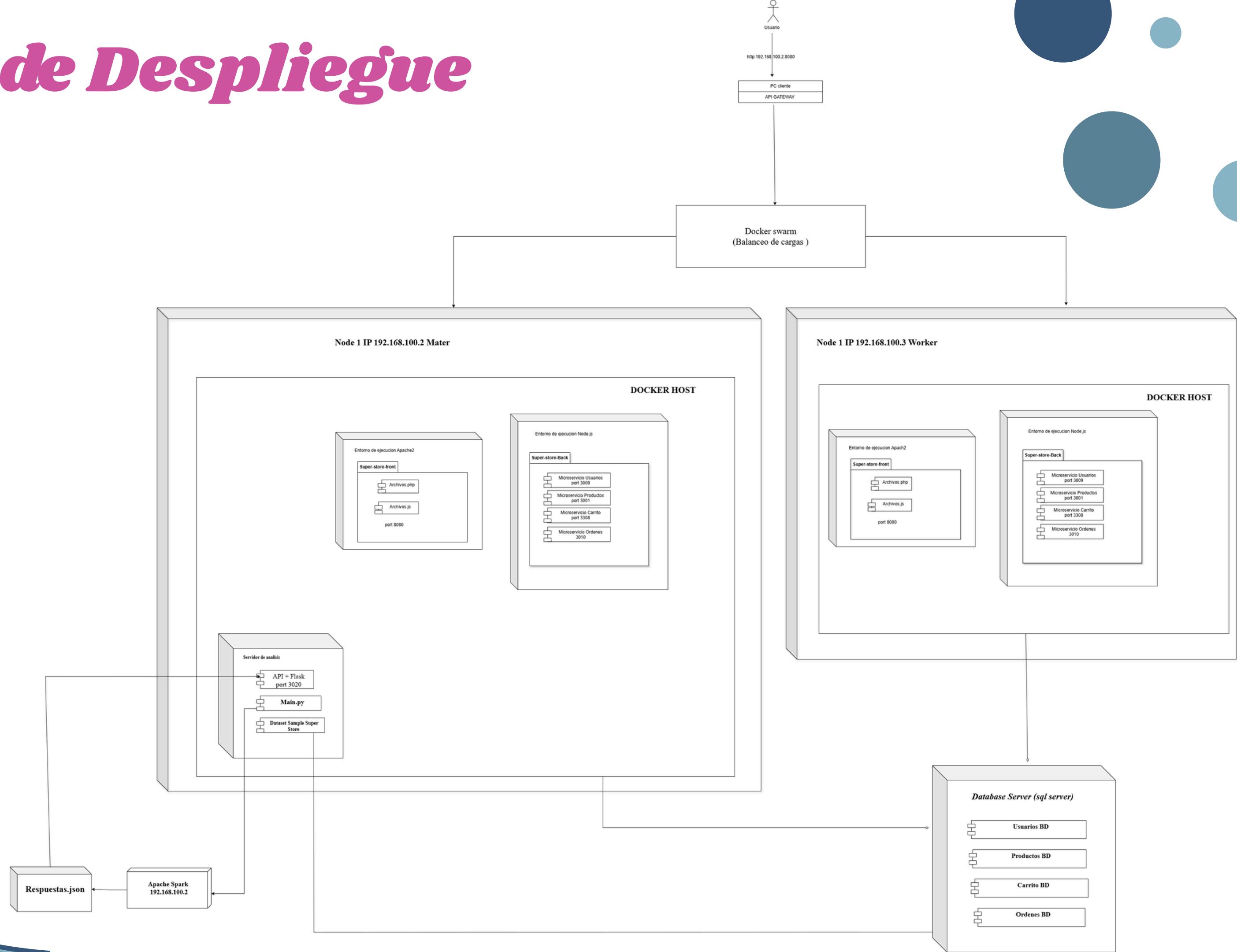
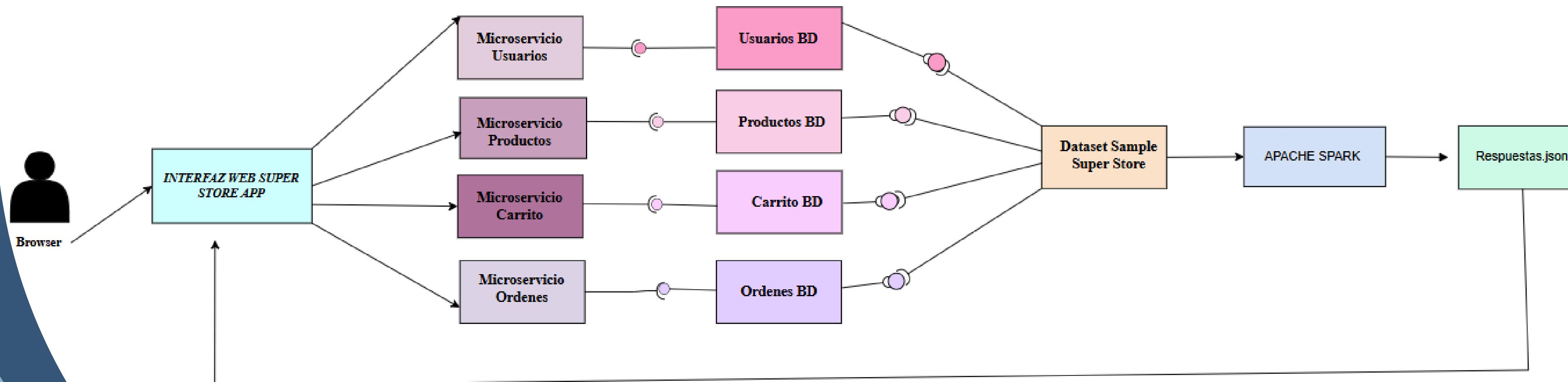


Diagrama de Componentes



GRACIAS

