Certainly! In Apache Spark, `SparkSession` is the entry point for reading data, executing SQL queries, and managing the Spark cluster resources. It encapsulates the functionality provided by the earlier entry points (`SparkConf`, `SparkContext`, `SQLContext``) in a unified interface. Here, I'll provide detailed explanations along with two code examples for each of the specified points:

### 1. Configuring SparkSession:

#### Example 1: Basic SparkSession Configuration
```python
from pyspark.sql import SparkSession

# Create a SparkSession
spark = SparkSession.builder \
    .appName("example_app") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

# Now you can use 'spark' to perform various Spark operations
```

In this example:
- We create a `SparkSession` using the `SparkSession.builder` API.
- The `.appName("example_app")` sets the application name.
- The `.config("spark.some.config.option", "some-value")` sets a configuration option. You can add more such configurations as needed.

#### Example 2: Setting Master and Executor Memory
```python
from pyspark.sql import SparkSession

# Create a SparkSession with specific master and executor memory settings
spark = SparkSession.builder \
    .appName("example_app") \
    .config("spark.master", "local[2]") \
    .config("spark.executor.memory", "2g") \
    .getOrCreate()

# Now you can use 'spark' with the specified configurations
```

In this example:
- We set the master to run locally with 2 cores (`local[2]`).
- We allocate 2 gigabytes of memory for each executor (`spark.executor.memory`).

### 2. SparkSession vs SparkContext:

#### Example 1: Using SparkSession for SQL Operations
```python
from pyspark.sql import SparkSession

# Create a SparkSession
spark = SparkSession.builder \
    .appName("example_app") \
    .getOrCreate()

# Read a CSV file into a DataFrame using SparkSession
df = spark.read.csv("path/to/data.csv", header=True, inferSchema=True)

# Perform SQL operations using SparkSession
result = df.select("column1").filter(df["column2"] > 10).groupBy("column3").count()
result.show()
```

In this example:
- We use `SparkSession` to read a CSV file into a DataFrame.
- We perform SQL-like operations on the DataFrame using the `select`, `filter`, and `groupBy` functions.

#### Example 2: Using SparkContext for RDD Operations
```python
from pyspark import SparkContext

# Create a SparkContext
sc = SparkContext(appName="example_app")

# Create an RDD from a list
data = [1, 2, 3, 4, 5]
rdd = sc.parallelize(data)

# Perform RDD transformations and actions using SparkContext
result = rdd.map(lambda x: x * 2).filter(lambda x: x > 5).collect()
print(result)
```

In this example:
- We use `SparkContext` to create an RDD from a list.

- We perform RDD transformations (`map` and `filter`) and an action (`collect`) using `SparkContext`.

Note: In modern Spark applications, it's more common to use `SparkSession` for DataFrame and SQL operations as it provides higher-level abstractions, while `SparkContext` is often used for lower-level RDD operations. However, `SparkSession` internally uses `SparkContext`.