**What you’ll do (high level)**

1. Start your **Cloudera Quickstart VM** in Oracle VirtualBox.
2. Open a Terminal inside the VM.
3. Start a tiny **TCP server** (with nc) that will accept lines of text on **port 9999**.
4. Use **WinSCP** to create a Python script on the VM: a minimal Spark Streaming **word count**.
5. Run the script with spark-submit (in **local[2]** mode).
6. Type lines into the nc window; watch word counts appear in Spark every 2 seconds.
7. Stop both processes cleanly.

**A) Launch the VM**

1. Open **Oracle VM VirtualBox** on your host machine.
2. Select **Cloudera Quickstart** VM → click **Start** (Normal Start).
3. Wait until the desktop is fully loaded (you’ll see a Linux desktop with terminal and apps).

No commands here — just GUI actions.

**B) Open a terminal and find your IP (optional)**

You don’t actually need the VM’s IP for this task because we’ll connect to **localhost**. Still, here’s how to see networking is fine:

**Command (inside VM):**

ifconfig

**Explanation:**

* ifconfig — shows network interfaces and their IPs.  
  You’ll see an entry like eth0 or enp0s3 with an inet address, e.g., 192.168.56.101.  
  We won’t use this IP in the command; it’s just a sanity check.

**C) Start a simple TCP server (the data source)**

Open **Terminal #1** (inside the VM) and run:

Command : nc -lk 9999

Explanation

nc = netcat, a tiny TCP tool.

-l = “listen mode” (act like a simple TCP server).

-k = keep listening for more input/clients instead of exiting after the first one.

9999 = the TCP port. Your Spark code connects to localhost:9999.

If your nc doesn’t support -k, use a shell loop to keep relistening:

Command :

while true; do nc -l 9999; done

Explanation

while true; do …; done = run nc -l 9999 repeatedly. If nc exits after one session, the loop immediately starts it again, so Spark can reconnect.

If your nc needs an explicit port flag:

Command :

nc -lk -p 9999

Explanation

-p 9999 = specify the port with -p (some nc flavors require it).

Keep this terminal open and running. Do not press Ctrl+C again.

Note :  
If you get “command not found”, install netcat using the following command and then type the above 3 commands :

sudo yum install -y nc

* sudo — run the command as admin.
* yum — the CentOS package manager.
* install -y — install, auto-answer “yes”.
* nc — the package providing netcat.

**D) Create the Spark Streaming script with WinSCP (no nano)**

1. Open **WinSCP** on your host machine.
2. **Session**:
   * **File protocol**: SFTP (or SCP)
   * **Host name**: your VM’s IP (from ifconfig, e.g., 192.168.56.101)
   * **User name**: cloudera
   * **Password**: cloudera
   * Click **Login** (accept the host key if prompted).
3. In the **right pane** (remote side), go to /home/cloudera/.
4. Right-click → **New** → **File** → name it:

task11\_socket\_streaming.py

1. Double-click the new file to open WinSCP’s editor. **Paste** the code below, then **Save**.

# task11\_socket\_streaming.py

# Minimal Spark Streaming word count from a TCP socket (Spark 1.6)

from \_\_future\_\_ import print\_function

from pyspark import SparkContext

from pyspark.streaming import StreamingContext

# Create a SparkContext (app name shows in logs/UI)

sc = SparkContext(appName="Task11\_SocketStreaming")

# Create a StreamingContext with 2-second micro-batches

ssc = StreamingContext(sc, 2)

# Define the input stream: connect to TCP server on localhost:9999

lines = ssc.socketTextStream("localhost", 9999)

# Split each incoming line into words

words = lines.flatMap(lambda line: line.split())

# Map each word to (word, 1)

pairs = words.map(lambda w: (w, 1))

# Reduce by key per batch to get counts

counts = pairs.reduceByKey(lambda a, b: a + b)

# Print the top few word counts for each 2-second batch

counts.pprint()

# Start streaming and wait forever (until Ctrl+C)

ssc.start()

ssc.awaitTermination()

**What each key line does:**

* SparkContext(...) — starts Spark for this app (driver process).
* StreamingContext(sc, 2) — makes a streaming context with **2-second** batch size.
* socketTextStream("localhost", 9999) — reads lines from our nc listener.
* flatMap(...split()) — splits each line into words by whitespace.
* map(lambda w: (w, 1)) — tags each word with count 1.
* reduceByKey(...) — sums counts **per word** in each batch.
* pprint() — prints the counts for the batch to the driver console.
* start() — begins receiving/processing.
* awaitTermination() — blocks until you stop the job.

**E) Run the streaming job**

Open **Terminal #2** (a new terminal inside the VM), and run:

spark-submit --master local[2] /home/cloudera/task11\_socket\_streaming.py

**Explanation (piece by piece):**

* spark-submit — the launcher that runs your Spark application.
* --master local[2] — run Spark **locally** with **2 threads**:
  + 1 thread for the **receiver** (listens to the socket),
  + 1 thread for **processing** the batches.  
    (local[1] would starve the receiver or the processing; streaming needs at least 2.)
* /home/cloudera/task11\_socket\_streaming.py — absolute path to your script (you created it with WinSCP in that folder).

**F) Send data and see results**

1. Go back to **Terminal #1** (where nc -lk 9999 is running).
2. Type a few lines, pressing **Enter** after each line, for example:

hello spark

hello world

spark spark streaming

1. Watch **Terminal #2** (Spark app) — every **2 seconds** it prints the counts it saw in that batch, e.g.:

-------------------------------------------

Time: 2025-09-03 22:45:02

-------------------------------------------

('hello', 2)

('spark', 1)

('world', 1)

-------------------------------------------

Time: 2025-09-03 22:45:04

-------------------------------------------

('spark', 2)

('streaming', 1)

* Each “Time:” block is a **batch** (every 2 seconds).
* Only words typed during that 2-second window are counted and printed in that block.

**G) Stop everything cleanly**

* In **Terminal #2** (Spark), press **Ctrl + C** once or twice to stop the app.
* In **Terminal #1** (nc), press **Ctrl + C** to stop the listener.