## ! STOP!

## You are required to watch the walkthrough video for this week's assignment. In the video, I guide you through each command, showing both the input and the expected output. Since you are working with new technology for the first time, it’s important not to just run a command and assume it worked. You will lose points if the commands are not executed correctly. Watching the assignment walkthrough video is mandatory.

## Week 4 Assignment: Introduction to Apache Spark using Scala and PySpark

# Conceptual Foundations

Before beginning the assignment, watch the instructor-led fundamentals video, which introduces and explains the key concepts for this week:  
<https://youtu.be/Sw8KP0TGTfU>

**Deliverable:** Write a 3–4 paragraph summary that demonstrates your understanding of the material presented in the video. Your writeup should explain the main ideas in your own words, highlight why these concepts are important, and connect them to the technologies you will be working with in the assignment.

#### **Objective: Getting Hands-on with Apache Spark using PySpark and Scala**

Apache Spark is an open-source, distributed computing system that enables lightning-fast data processing across large datasets. It’s designed to handle both batch and real-time data, making it one of the most versatile tools in the big data ecosystem. Spark’s in-memory processing capabilities drastically improve the speed of applications compared to traditional disk-based engines like Hadoop MapReduce, allowing for faster data analytics, machine learning, and graph processing.

In this assignment, you will:

* **Explore Spark's Core Concepts**: Learn how Spark leverages **Resilient Distributed Datasets (RDDs)** to efficiently distribute and process data across a cluster of machines.
* **Work with PySpark and Scala**: Gain hands-on experience with both Python and Scala APIs for Spark, two of the most commonly used languages for big data processing.
* **Understand Distributed Data Processing**: See how Spark splits tasks into smaller units, distributes them across different nodes in a cluster, and processes them in parallel for maximum efficiency.
* **Perform Data Transformations**: Use Spark’s powerful transformation functions like map and filter to manipulate datasets and observe how Spark executes these transformations in a distributed manner.
* **Run Jobs Using YARN**: Interact with **YARN**, the cluster manager responsible for resource allocation, to see how Spark jobs are scheduled and executed in a distributed environment.

By the end of this assignment, you will understand why Spark is a cornerstone of modern big data architectures and how to leverage its capabilities for efficient and scalable data processing.

### Objective: Scale Google Cloud Enviornment

### Objective: Familiarize with Spark Submit, PySpark, and Scala

#### **1. Environment Initialization**

* As you did in the Week 2 and Week 3 assignments, begin by navigating to the appropriate directory:
* cd dsc650-infra/bellevue-bigdata/hadoop-hive-spark-hbase
* Start your Docker containers:
* docker-compose up -d
* Access the master container:
* docker-compose exec master bash

#### **2. Running a Built-in Spark Example with PySpark**

Here, you will run the built-in **SparkPi** example using PySpark. This simple example calculates an approximation of Pi by using PySpark to distribute the computation across the cluster. This step provides a basic introduction to running a Spark job and observing how Spark manages resources under the hood.

To get hands-on experience with PySpark, we will execute the provided SparkPi example using the following command in the master container:

spark-submit --class org.apache.spark.examples.SparkPi \  
 --master yarn \  
 --deploy-mode client \  
 --driver-memory 2g \  
 --executor-memory 1g \  
 --executor-cores 1 \  
 $SPARK\_HOME/examples/jars/spark-examples\*.jar \  
 10

**Deliverable:** Screenshot of the SparkPi output.

#### **3. Starting the Spark Scala Shell**

After running the PySpark job, you will now start the **Spark Scala shell**. Spark was originally developed in Scala, and this shell gives you the ability to interact with Spark using Scala commands. Here, you will begin experimenting with Spark’s core features in its native language.

In the master container’s terminal, initiate the Spark Scala shell:

$SPARK\_HOME/bin/spark-shell --master yarn --driver-memory 2g --executor-memory 1g --executor-cores 1

#### **4. Generating and Printing Random Numbers in the Scala Shell**

In the Scala shell, you will generate a list of random numbers and distribute them across your Spark cluster. By using the parallelize function, you are distributing the data to be processed in parallel, which is one of Spark’s key features. Printing a subset of these numbers will confirm that the data has been distributed correctly.

Execute the following commands in the Spark Scala shell:

val numNumbers = 10000  
val numbers = (1 to numNumbers).map(\_ => scala.util.Random.nextInt(1000))  
val numbersRDD = sc.parallelize(numbers)  
numbersRDD.take(100).foreach(println)

**Deliverable:** Screenshot of the first 100 generated random numbers.

#### **5. Generating and Transforming Random Sentences in the Scala Shell**

In this final section, you will generate random sentences from a predefined list of words and apply transformations to the data. By using Spark’s map function, you’ll gain experience in applying transformations to distributed data. This demonstrates how Spark allows you to manipulate large datasets in parallel using simple functional programming constructs.

Modify the transformation placeholder in the following command to apply your custom logic (e.g., convert sentences to uppercase, reverse the words, etc.).

Generate random sentences and apply a transformation of your choice:

val numberOfSentences = 1000  
val words = List("apple", "banana", "cherry", "date", "elderberry", "fig", "grape", "honeydew")  
val sentences = (1 to numberOfSentences).map(\_ => scala.util.Random.shuffle(words).take(scala.util.Random.nextInt(6) + 1).mkString(" ") + ".")  
val sentencesRDD = sc.parallelize(sentences)  
  
// Apply your custom transformation here  
val transformedSentences = sentencesRDD.map(sentence => /\* Your transformation code here \*/)  
transformedSentences.take(100).foreach(println)

**Instructions:** Modify the placeholder /\* Your transformation code here \*/ to apply a unique transformation to each sentence.

**Deliverable:** Screenshot of a segment of the generated transformed sentences and an explanation of your unique transformation.

## Shutting Down

Ensure all Docker containers are turned off with docker-compose down for each directory. If you’re using google cloud, please shut down your virtual machine to preserve cloud costs.