## ! 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 2 Assignment: Dive into HDFS and MapReduce

# Objective: Understanding HDFS, YARN, and MapReduce

In this assignment, you will be learning about three key technologies in the Hadoop ecosystem: HDFS, YARN, and MapReduce. These technologies are foundational for processing and managing large datasets across distributed systems.

1. **HDFS (Hadoop Distributed File System):** HDFS is the storage layer of Hadoop. It allows you to store large amounts of data across multiple nodes in a reliable and fault-tolerant manner. You will learn how to interact with HDFS, upload data, and verify its replication across the cluster.
2. **YARN (Yet Another Resource Negotiator):** YARN is responsible for resource management in the Hadoop cluster. In this assignment, you will experiment with managing memory resources, which is crucial for running jobs efficiently across a distributed environment.
3. **MapReduce:** MapReduce is a programming model used for processing large data sets in parallel. You will run a MapReduce job to estimate the value of Pi, demonstrating how large-scale computations are handled in a distributed system.

Through this assignment, you will gain hands-on experience with these technologies, learning not only how to execute commands but also understanding the underlying purpose behind each task. By the end, you should be able to navigate HDFS, modify YARN settings, and run MapReduce jobs to process data.

### Objective: Familiarize with the core functionalities of HDFS and get a practical understanding of MapReduce.

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

* Start by navigating to the required directory and initiating the Docker containers:
* cd dsc650-infra/bellevue-bigdata/hadoop-hive-spark-hbase  
    
  docker-compose up -d
* If you’re using Google Cloud, remember to set up port forwarding as outlined in the Week 1 assignment.
* Access the master container:
* docker-compose exec master bash

#### **2. Deep Dive into HDFS**

As you check the HDFS admin report, take a close look at the total space available, used space, and remaining space. You’ll also want to notice how many DataNodes there are, as this will show you how your data is distributed across the cluster. Pay attention to any under-replicated blocks; this could mean the data isn't being copied across nodes as it should, which is important for data redundancy.

* Check the HDFS report:
* hdfs dfsadmin -report

**Deliverable:** Screenshot of the output.

Now that you’re loading the grades.csv file into HDFS, what you’re really doing is transferring local data into the distributed file system. This is important because in real-world scenarios, you’d need to move large datasets into HDFS to process them across many nodes. Once it’s there, you can manage and access it across the cluster.

* Load the grades.csv into HDFS:
* hdfs dfs -put /data/grades.csv /
* Verify that the data has been loaded:
* hdfs dfs -ls /

**Deliverable:** Screenshot proving the data has been loaded.

* Exit the master Docker container:
* CTRL+D or exit

When you check all the worker nodes, the goal is to make sure the file you uploaded (grades.csv) is replicated across the cluster. This shows you how HDFS ensures that your data is fault-tolerant and available even if a node fails. If you see the file on all worker nodes, you know replication is working as it should.

* SSH into each of the 3 worker nodes and verify the data:
* docker-compose exec worker1 bash  
  hdfs dfs -ls /  
  CTRL+D or exit  
    
  docker-compose exec worker2 bash  
  hdfs dfs -ls /  
  CTRL+D or exit
* All worker nodes should display the grades.csv file.
* Re-enter the master container:
* docker-compose exec master bash
* Explore more HDFS commands:
* hdfs dfs -help
* Execute three other HDFS commands of your choice and observe their outputs.

**Deliverable:** Screenshots of the three chosen HDFS command outputs.

#### **3. Introduction to YARN**

* Inside the master container, inspect the YARN nodes:
* yarn node -list

**Deliverable:** Screenshot of the results.

* Understand the yarn.scheduler.maximum-allocation-mb property. This is the maximum memory capacity available for a single container.

Next, you’ll adjust the YARN memory allocation. Increasing this value lets YARN handle bigger jobs by allowing more memory per container. This is something you’ll need to do when running memory-intensive applications on a cluster. It's a great way to see how cluster resources can be tuned for performance.

* Modify the maximum memory allocation:
* sed -i "/<name>yarn.scheduler.maximum-allocation-mb<\/name>/,/<\/property>/s/<value>.\*<\/value>/<value>2048<\/value>/" /usr/program/hadoop/etc/hadoop/yarn-site.xml
* Restart the ResourceManager:
* yarn --daemon stop resourcemanager  
  yarn --daemon start resourcemanager

**Deliverable:** Screenshot from the YARN UI showing the updated maximum memory (2048 MB).

#### **4. Experimenting with MapReduce**

When you run the MapReduce Pi job, you might notice that the Pi result is not exactly 3.14. That’s because the MapReduce job uses a statistical approach (Monte Carlo method), which gives an estimate of Pi. The job teaches you how distributed systems handle large computations, and why sometimes the result can be an approximation rather than an exact value.

* Run the example MapReduce Pi job:
* libjars=$(find /usr/program/hadoop/share/hadoop/mapreduce -name "\*.jar" | tr '\n' ',')
* hadoop jar /usr/program/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.2.3.jar pi -libjars ${libjars} 2 10

**Deliverable:** A summary of the result and why the result of the PI job is not the expected 3.14. This must be a technical discussion.

## 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.