Pluralsight Installation

Practical **step-by-step** playbook you can follow inside a Pluralsight cloud server (or any Ubuntu VM) to get a *compatible* stack: **Ubuntu → OpenJDK 17 → Python 3.11 → Scala 2.13 → Hadoop 3.3/3.4 (single-node) → Hive 3.1.x → Spark 3.5.x (prebuilt for Hadoop3 + Scala2.13)**.
I picked these because Spark 3.5 works with Java 8/11/17, Scala 2.12/2.13 and Python 3.8+, and Hive 4's metastore is *not* fully compatible with Spark 3.x (so Hive 3.1.x is the safer choice for Spark-3.5). Apache Spark+2Apache Spark+2

Quick compatibility summary (recommended)

- **OS:** Ubuntu (22.04 LTS or 24.04) on Pluralsight cloud server. help.pluralsight.com
- **Java:** OpenJDK 17 (works with Spark 3.5 and Hive 3.1.x) . Apache Spark
- **Python:** 3.11 (PySpark requires Python ≥ 3.8). Apache Spark
- **Scala:** 2.13 (use Scala 2.13.x matching Spark prebuilt package). Apache Spark
- **Hadoop:** 3.3.x or 3.4.x (use a stable 3.x build). hadoop.apache.org
- **Hive:** 3.1.x (metastore compatible with Spark 3.x). hive.apache.org+1
- **Spark:** 3.5.x prebuilt for Hadoop3 + Scala2.13 (download from Apache Spark downloads). Apache Spark

---

A. Provision an Ubuntu image in Pluralsight (high level)

1. Open **Hands-on → Cloud servers / Cloud sandboxes** in your Pluralsight account and **Create New Server** — choose an Ubuntu image (22.04/24.04) and a machine size with **≥4 vCPU & 8–16GB RAM** and ~50GB disk for decent local HDFS/Spark experimentation. (Pluralsight's hands-on/cloud sandboxes let you launch preconfigured VMs). help.pluralsight.com+1

   If you don't want Pluralsight, the exact same steps work on any Ubuntu VM (EC2 / GCE / VirtualBox).

B. Prepare the Ubuntu VM (common prerequisites)

Open a shell (SSH) to your Ubuntu VM and run:

```
# 1) system update
sudo apt update && sudo apt upgrade -y

# 2) essential packages
sudo apt install -y wget curl git vim ssh build-essential
apt-transport-https ca-certificates

# 3) set timezone / locale if needed (optional)
sudo timedatectl set-timezone Asia/Kolkata
```

C. Install Java (OpenJDK 17), Python 3.11 and pip

```
# Java 17 (OpenJDK)
sudo apt install -y openjdk-17-jdk-headless
java -version   # verify

# Python 3.11 (use deadsnakes PPA if default distro doesn't have 3.11)
sudo apt install -y software-properties-common
```

```
sudo add-apt-repository ppa:deadsnakes/ppa -y
sudo apt update
sudo apt install -y python3.11 python3.11-venv python3.11-dev
python3-pip
python3.11 --version
pip3 --version
```

Set JAVA_HOME in your shell (add to `~/.bashrc`):

```
echo 'export JAVA_HOME=$(dirname $(dirname $(readlink -f $(which
java))))' >> ~/.bashrc
echo 'export PATH=$JAVA_HOME/bin:$PATH' >> ~/.bashrc
source ~/.bashrc
```

(Why Java17? modern, LTS and supported by Spark/Hive builds). Apache Spark

---

D. Install Scala (use SDKMAN — easy & reproducible)

```
# install SDKMAN
curl -s "https://get.sdkman.io" | bash
source "$HOME/.sdkman/bin/sdkman-init.sh"

# install Scala 2.13.x (choose latest 2.13)
sdk install scala 2.13.16   # change to latest 2.13.x if desired
scala -version
```

(You can also use distribution packages, but SDKMAN gives latest 2.13.x easily). scala-lang.org

---

E. Download & configure Hadoop (single-node pseudo-distributed)

I recommend **Hadoop 3.3.5 or 3.4.x** (Hadoop 3.x family). Example uses **3.3.5**.

```
# variables
HADOOP_VER=3.3.5
sudo mkdir -p /opt/hadoop
cd /opt/hadoop

# download and extract
sudo wget
https://downloads.apache.org/hadoop/common/hadoop-$HADOOP_VER/hadoop-$
HADOOP_VER.tar.gz
sudo tar -xzf hadoop-$HADOOP_VER.tar.gz
sudo ln -s /opt/hadoop/hadoop-$HADOOP_VER /opt/hadoop/hadoop

# perms (optional)
sudo chown -R $USER:$USER /opt/hadoop

# env variables (add to ~/.bashrc)
cat >> ~/.bashrc <<'EOF'
export HADOOP_HOME=/opt/hadoop/hadoop
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
EOF
source ~/.bashrc
```

Edit `hadoop-env.sh` to point to JAVA_HOME (replace path if different):

```
sed -i "s|^# export JAVA_HOME=.*|export JAVA_HOME=$JAVA_HOME|"
$HADOOP_HOME/etc/hadoop/hadoop-env.sh
```

Create minimal single-node configs (example — place into the files in `$HADOOP_HOME/etc/hadoop`):

`core-site.xml`

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
```

```
  </property>
</configuration>
```

`hdfs-site.xml`

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:///home/$USER/hadoopdata/hdfs/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:///home/$USER/hadoopdata/hdfs/datanode</value>
  </property>
</configuration>
```

`mapred-site.xml` (create from template `mapred-site.xml.template`):

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

`yarn-site.xml`

```
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>
```

Prepare directories, format and start:

```
mkdir -p ~/hadoopdata/hdfs/namenode ~/hadoopdata/hdfs/datanode
$HADOOP_HOME/bin/hdfs namenode -format

# start HDFS + YARN
$HADOOP_HOME/sbin/start-dfs.sh
$HADOOP_HOME/sbin/start-yarn.sh

# verify (use jps or hdfs dfs commands)
jps      # NameNode, DataNode, ResourceManager, NodeManager should
show
hdfs dfs -mkdir -p /user/$(whoami)
```

(Official Hadoop docs explain compatibility/requirements and the recommended practice that minor Hadoop releases remain compatible). [hadoop.apache.org+1]

F. Install Hive (3.1.x) — simple local metastore (derby) for experimentation

NOTE: Derby metastore is fine for dev / single-user testing. In any multi-user or persistent environment use MySQL/Postgres for metastore.

```
# download and extract Hive 3.1.3 (example)
HIVE_VER=3.1.3
sudo mkdir -p /opt/hive
cd /opt/hive
sudo wget
https://downloads.apache.org/hive/hive-$HIVE_VER/apache-hive-$HIVE_VER
-bin.tar.gz
sudo tar -xzf apache-hive-$HIVE_VER-bin.tar.gz
sudo ln -s /opt/hive/apache-hive-$HIVE_VER-bin /opt/hive/hive
sudo chown -R $USER:$USER /opt/hive

# env
cat >> ~/.bashrc <<'EOF'
```

```
export HIVE_HOME=/opt/hive/hive
export PATH=$PATH:$HIVE_HOME/bin
EOF
source ~/.bashrc

# create HDFS warehouse dir used by Hive
hdfs dfs -mkdir -p /user/hive/warehouse
hdfs dfs -chmod 1777 /user/hive/warehouse

# init default metastore (derby)
$HIVE_HOME/bin/schematool -dbType derby -initSchema

# start metastore & HiveServer2 (background)
nohup $HIVE_HOME/bin/hive --service metastore > ~/hive-metastore.log
2>&1 &
nohup $HIVE_HOME/bin/hive --service hiveserver2 > ~/hiveserver2.log
2>&1 &
# test with beeline
beeline -u jdbc:hive2://localhost:10000 -n user -p pass
--showHeader=false -e "SHOW DATABASES;"
```

Hive 3.x downloads & release notes are on the Hive site. For production you should configure a MySQL/Postgres metastore. hive.apache.org+1

G. Install Spark (3.5.x prebuilt for Hadoop3 + Scala 2.13)

Download prebuilt Spark 3.5.x (pick the `Pre-built for Apache Hadoop 3.3 and later (Scala 2.13)` package).

```
SPARK_VER=3.5.3
cd /opt
sudo wget
https://downloads.apache.org/spark/spark-$SPARK_VER/spark-$SPARK_VER-b
in-hadoop3-scala2.13.tgz
```

```
sudo tar -xzf spark-$SPARK_VER-bin-hadoop3-scala2.13.tgz
sudo ln -s /opt/spark-$SPARK_VER-bin-hadoop3-scala2.13 /opt/spark
sudo chown -R $USER:$USER /opt/spark

# env
cat >> ~/.bashrc <<'EOF'
export SPARK_HOME=/opt/spark
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
EOF
source ~/.bashrc
```

Configure Spark to use your Hive metastore & HDFS warehouse:

1. Copy `hive-site.xml` into Spark config so Spark picks up the metastore settings:

```
cp $HIVE_HOME/conf/hive-site.xml $SPARK_HOME/conf/
```

2. Add minimal `spark-defaults.conf` (in `$SPARK_HOME/conf/spark-defaults.conf`):

```
spark.sql.catalogImplementation  hive
spark.sql.warehouse.dir
hdfs://localhost:9000/user/hive/warehouse
```

3. Edit `$SPARK_HOME/conf/spark-env.sh` to set JAVA_HOME and HADOOP_CONF_DIR:

```
echo "export JAVA_HOME=$JAVA_HOME" >> $SPARK_HOME/conf/spark-env.sh
echo "export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop" >>
$SPARK_HOME/conf/spark-env.sh
```

Start a quick local test:

```
# Scala shell
```

```
$SPARK_HOME/bin/spark-shell --master local[*]     # should start REPL

# pyspark
$SPARK_HOME/bin/pyspark --master local[*]

# test Spark <-> Hive
$SPARK_HOME/bin/spark-sql --conf spark.sql.catalogImplementation=hive
-e "SHOW DATABASES;"
```

(Download page & prebuilt options: Apache Spark downloads page). [Apache Spark+1]

H. Optional: install PySpark via pip (for using PySpark in virtualenv)

If you prefer to use `pip` and virtualenv:

```
python3.11 -m venv ~/pyspark-venv
source ~/pyspark-venv/bin/activate
pip install --upgrade pip
pip install pyspark==3.5.3   # match Spark version
# run pyspark via local Spark or via installed wheel
```

(When using pip-installed pyspark, ensure `JAVA_HOME` and `HADOOP_CONF_DIR` are set so it talks to HDFS/Hive correctly.)

I. Verify everything (sanity checks)

```
java -version
python3.11 --version
scala -version
hadoop version
hdfs dfs -ls /
```

```
spark-shell --version
$HIVE_HOME/bin/hive --version
beeline -u jdbc:hive2://localhost:10000 -e "show databases;"
```

J. Short troubleshooting & production tips

- **Hive 4 metastore**: Spark 3.x may not be compatible with Hive 4 metastore — that's why I recommended Hive 3.1.x for Spark 3.5.x. If you need Hive 4 features, you should evaluate Spark 4.x. Google Groups

- **Metastore DB**: use **MySQL/Postgres** for a real shared metastore (Derby is single-user only).

- **Classpath / JAR conflicts**: Spark + Hive + Hadoop can suffer from jar/shim conflicts. Keep Spark's `jars/` clean and use the prebuilt matching Spark+Hadoop package to avoid mismatches. Apache Spark

- **Memory**: For Hadoop/Spark on a single VM keep options low (e.g. container memory limits) to avoid OOM.

- **Ports/Firewall**: ensure SSH and required ports (HDFS UI 9870, YARN 8088, Spark UI 4040, HiveServer2 10000) are reachable if you want to connect from your host.

- **If Pluralsight sandbox restricts installs**: some managed lab sandboxes are locked; you might prefer an EC2/GCE/Local VM for full control.

K. Useful authoritative links (docs I used)

- Pluralsight Hands-on / Cloud servers docs (create cloud server / sandboxes). help.pluralsight.com+1

- Spark 3.5.x docs (Java / Scala / Python compatibility). Apache Spark

- Spark downloads / prebuilt packages for Hadoop/Scala. Apache Spark

- Hadoop compatibility & docs. hadoop.apache.org
- Hive downloads & notes (Hive 3.x). hive.apache.org+1
