Labs illustrating PySpark Development

Common Spark development techniques

Spark demonstrations

- PySpark on your computer
  - Run from shell (terminal)
  - Run within IDE (PyCharm)
- PySpark on managed cluster
  - Databricks
  - Google Dataproc
- PySpark in Docker on Kubernetes cluster
  - On your computer
  - On Google Kubernetes engine (GKE)

Why on your local computer?

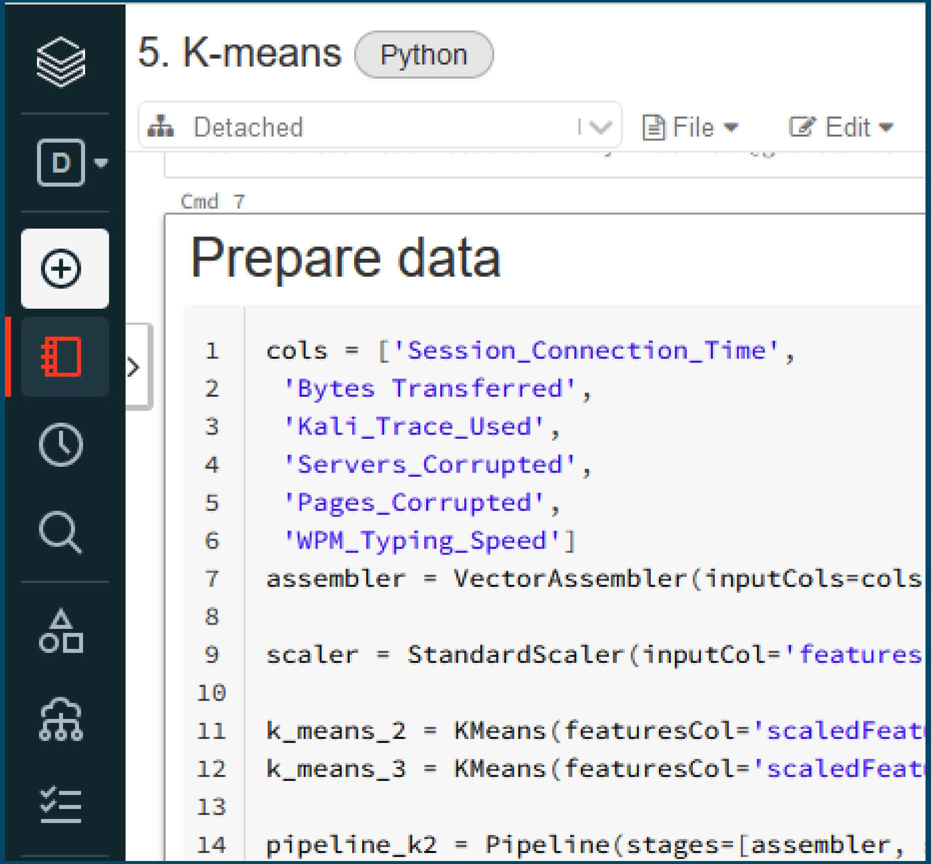
- Typical development life-cycle
    - Create & test with small data on local computer before moving to cluster
- No need for internet resources
    - Test on airplane
    - No expenses
- No need for remote (complex) debugging tools
    - Can use debugger to step through code, including Spark internals
    - Easier for debugging and understanding how it works

Management of Spark cluster

- Managed Spark cluster
  - User requests access to a cluster, perhaps specifying the general size information (e.g., number of nodes)
  - Examples: Databricks, Google Dataproc
- Unmanaged Spark cluster
  - User specifies details of cluster manager and worker nodes, including CPU, memory, OS version, package installations, network configuration, etc.
  - Example: Google Kubernetes Engine or Compute Engine
- Level (higher-level Spark application or lower-level Kubernetes containers)
  - GKE manages Kubernetes, which can run Spark (which is unmanaged)
  - So, GKE automatically manages Kubernetes (managed at the Kubernetes level), but a user must specify (manage) the images and their architecture
- Mostly unmanaged example
  - Use Compute Engine to install Kubernetes (which user must manage)
  - On the installed Kubernetes, run Spark images (which user must manage)
  - Note that Google still manages the virtual computers, which is yet a lower level

Why on managed cluster?

- Running Spark on managed cluster is much simpler to use, costs more, and leaves some architectural considerations to the managing software
    - Databricks
    - Google Dataproc

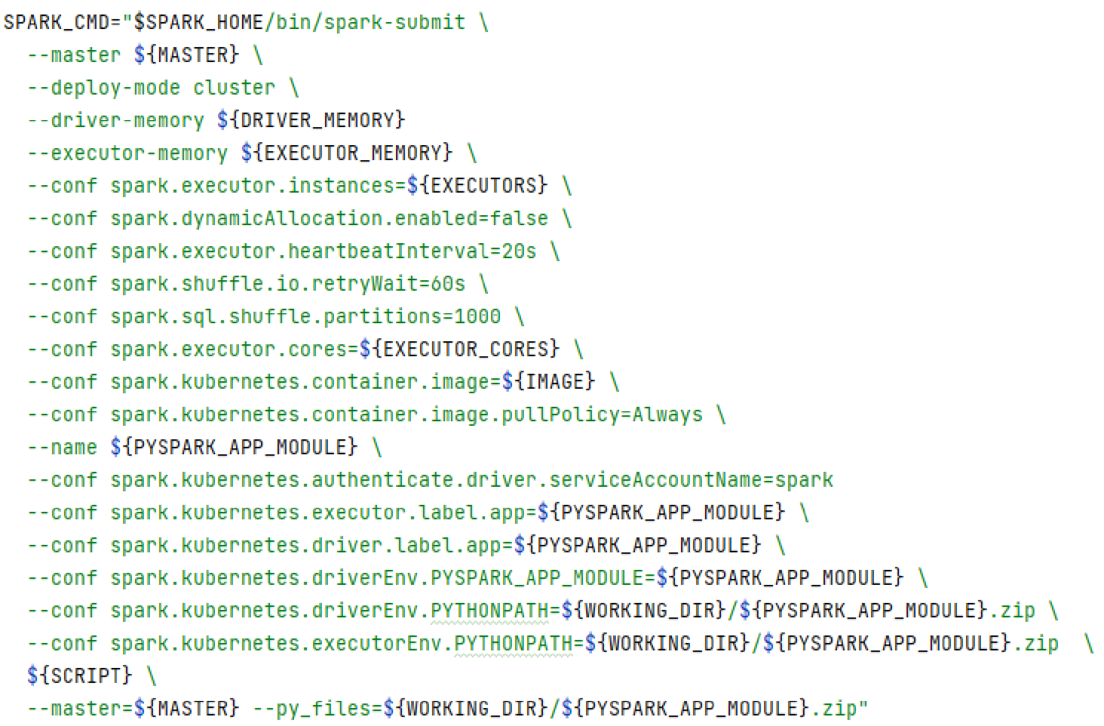


Prepare data

```
cols = ['Session_Connection_Time',
 'Bytes Transferred',
 'Kali_Trace_Used',
 'Servers_Corrupted',
 'Pages_Corrupted',
 'WPM_Typing_Speed']
assembler = VectorAssembler(inputCols=cols

scaler = StandardScaler(inputCol='features

k_means_2 = KMeans(featuresCol='scaledFeat
k_means_3 = KMeans(featuresCol='scaledFeat

pipeline_k2 = Pipeline(stages=[assembler,
```

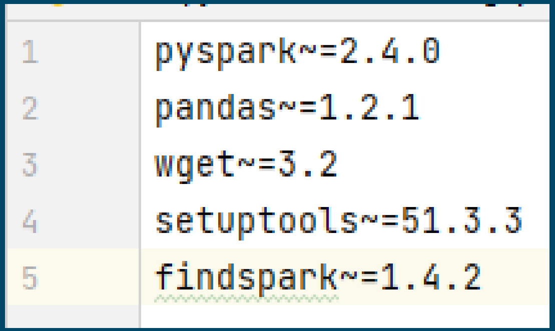
Why on unmanaged cluster?

- Running Spark on unmanaged cluster is more complex, costs less, and leaves architectural considerations to the user
  - Google Kubernetes Engine
  - Compute Engine



```
SPARK_CMD="$SPARK_HOME/bin/spark-submit \
  --master ${MASTER} \
  --deploy-mode cluster \
  --driver-memory ${DRIVER_MEMORY}
  --executor-memory ${EXECUTOR_MEMORY} \
  --conf spark.executor.instances=${EXECUTORS} \
  --conf spark.dynamicAllocation.enabled=false \
  --conf spark.executor.heartbeatInterval=20s \
  --conf spark.shuffle.io.retryWait=60s \
  --conf spark.sql.shuffle.partitions=1000 \
  --conf spark.executor.cores=${EXECUTOR_CORES} \
  --conf spark.kubernetes.container.image=${IMAGE} \
  --conf spark.kubernetes.container.image.pullPolicy=Always \
  --name ${PYSPARK_APP_MODULE} \
  --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark
  --conf spark.kubernetes.executor.label.app=${PYSPARK_APP_MODULE} \
  --conf spark.kubernetes.driver.label.app=${PYSPARK_APP_MODULE} \
  --conf spark.kubernetes.driverEnv.PYSPARK_APP_MODULE=${PYSPARK_APP_MODULE} \
  --conf spark.kubernetes.driverEnv.PYTHONPATH=${WORKING_DIR}/${PYSPARK_APP_MODULE}.zip \
  --conf spark.kubernetes.executorEnv.PYTHONPATH=${WORKING_DIR}/${PYSPARK_APP_MODULE}.zip  \
  ${SCRIPT} \
  --master=${MASTER} --py_files=${WORKING_DIR}/${PYSPARK_APP_MODULE}.zip"
```

Illustrative problem for Spark developer



```
pyspark~=2.4.0
pandas~=1.2.1
wget~=3.2
setuptools~=51.3.3
findspark~=1.4.2
```

- Program requires other software to run
  - Python modules, Java jars, etc.
- Program & modules must be distributed over cluster
  - PySpark .py file or Java jar sent
  - Dependencies are a problem, because many (large) files may be required
- Managed cluster
  - First, install dependencies on cluster nodes (large files copied)
  - Then, distribute the program (--py_files)
- Unmanaged cluster
  - Create Docker image of program & dependencies
  - Distribute image to cluster nodes (very efficient, only updates sent)

Important to remember

- Labs illustrate common Spark development techniques
  - Development on local computer
  - Development on managed cluster
    - Modules installed on nodes before program runs
  - Development on unmanaged cluster
    - Docker image created & distributed to cluster