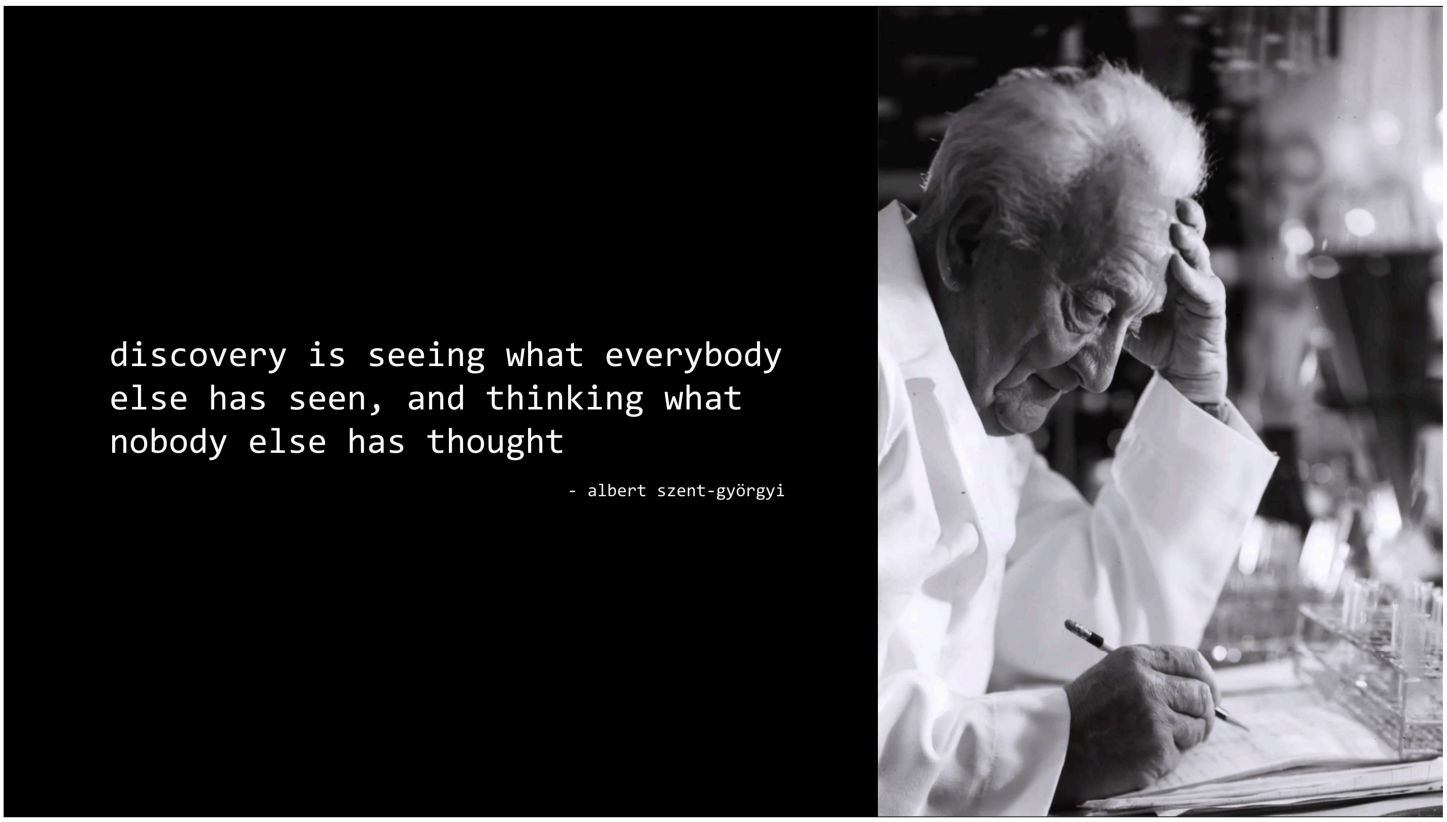
Multi-node Spark Cluster Deployment with Standalone Cluster Manger

Happy MlOps

(λx.x)eranga · Follow

Published in Effectz.AI

8 min read · Sep 17, 2023



Background

In my previous posts, I have discussed various topics about spark, spark cluster deployments, building different ml models with spark etc. In this post I'm gonna discuss deploying multi-node Spark cluster in `Standalone mode`. Apache spark

support multiple resource managers `1) Standalone`, `2) Aprache YARN`, `3) Apache Mesos` and `4) Kubernetes` . `Standalone` is a basic cluster manager that comes with spark compute engine. It provides basic functionalities like Memory management, Fault recovery, Task Scheduling, Interaction with cluster manager. To install Spark Standalone mode, you simply place a compiled version of Spark on each node on the cluster. In Standalone mode it use its own resource manager which Spark distribution comes with. Moreover, Spark allows us to create distributed master-slave architecture, by configuring properties file under $SPARK_HOME/conf directory. Start with a standalone cluster if this is a new deployment. Standalone mode is the easiest to set up and will provide almost all the same features as the other cluster managers if you are only running Spark.

Standalone offers essential functionalities such as memory management, fault recovery, task scheduling, and interaction with the cluster manager. Setting up Spark in Standalone mode is straightforward — all you need to do is place a compiled version of Spark on each node within the cluster. In this mode, Spark employs its own resource manager, which is included with the Spark distribution.

Furthermore, in Standalone mode, Spark empowers us to establish a distributed master-slave architecture by configuring properties within the `$SPARK_HOME/conf` directory. If you're embarking on a new deployment, starting with Standalone mode is an excellent choice. It's not only the easiest to set up but also provides nearly all the same features as other cluster managers when your primary focus is running Spark workloads.

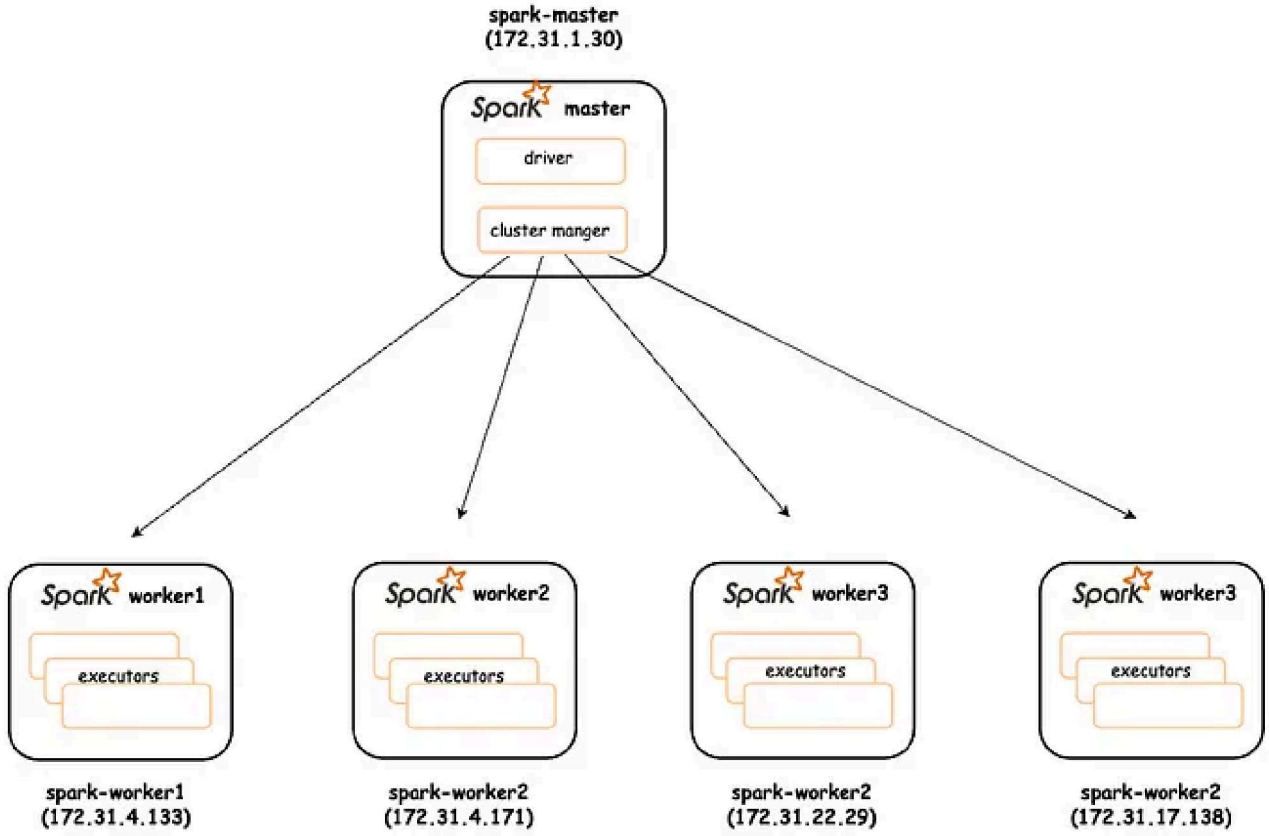
Cluster Architecture

The cluster architecture I'm planning to employ consists of five A`WS EC2 t2.2xlarge` instances. Each of these instances is equipped with `8 cores` and `32GB of memory`. Within this cluster, there is 1 designated master node and 4 worker nodes, ensuring optimal resource allocation and distribution. Here's a breakdown of the host names and IP addresses configured for each node.

```
172.31.1.30     spark-master
172.31.4.133    spark-worker1
172.31.4.171    spark-worker2
172.31.22.29    spark-worker3
172.31.17.138   spark-worker4
```

Within the Spark cluster architecture, each worker node is equipped with a Spark driver process and multiple Spark executor processes. The Spark driver process serves as a Java-based execution environment where the `main()` method of our `Scala`, `Java`, or `Python` program is executed. It is responsible for executing user code and creating essential components like `SparkSession` or `SparkContext`. The `SparkSession`, in particular, plays a pivotal role in creating `DataFrames`, `DataSets`, `RDDs`, executing `SQL` queries, and performing various transformations and actions on the data. These operations are fundamental to Spark's data processing capabilities.

The Spark executors, on the other hand, are distributed across the cluster, ensuring efficient data processing. Each executor is allocated a bandwidth, often referred to as a `core`, for data processing tasks. The specific configuration details, including `core` allocation and other executor settings, are discussed in Section 2. Executors receive tasks from the driver, which are then executed to process the logic of your code on the available data. They have the ability to store data in memory or disk storage as needed. Moreover, Spark Executors can seamlessly read data from both internal and external storage systems, making them versatile data processing components. It's worth noting that in the Standalone mode of Spark cluster deployment, the Cluster Manager resides within the master node. This Cluster Manager efficiently manages and allocates resources according to the specific requirements of various Spark applications, ensuring optimal performance and resource utilization.



The following are the steps I have used to set up the master node, including the installation of Java, Scala, and Spark.

1. Setup Master Node

Below is the method I used to set up the Master node, which involved installing `Java`, `Scala`, `Spark` and configurations.

```
# add node host names into /etc/hosts
sudo vim /etc/hosts
172.31.1.30     spark-master
172.31.4.133    spark-worker1
172.31.4.171    spark-worker2
172.31.22.29    spark-worker3
172.31.17.138   spark-worker4

# install asdf
git clone https://github.com/asdf-vm/asdf.git ~/.asdf --branch v0.13.0

# configure asdf
vim ~/.bashrc
. "$HOME/.asdf/asdf.sh"
. "$HOME/.asdf/completions/asdf.bash"

# install java, scala
```

```
asdf plugin add java
asdf install java openjdk-11
asdf global java openjdk-11
asdf plugin add scala
asdf install scala 2.13.10
asdf global scala 2.13.10

# set java home
. ~/.asdf/plugins/java/set-java-home.bash

# install spark
wget https://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.0-bin-hadoop2.
tar xvf spark-3.0.0-bin-hadoop2.7.tgz
mv spark-3.0.0-bin-hadoop2.7 spark-3.0.0
cd spark-3.0.0

# set spark home and jave home
vim ~/.bashrc
export SPARK_HOME=$HOME/spark-3.0.0
export PATH=$PATH:$SPARK_HOME/bin

# configure spark env in `conf/spark-env.sh` file
# there is template of the `spark-env.sh` file `conf/spark-env.sh.template`
cp conf/spark-env.sh.template conf/spark-env.sh

# define master node host and java home
vim conf/spark-env.sh
export SPARK_MASTER_HOST=172.31.1.30
export JAVA_HOME=/home/ubuntu/.asdf/installs/java/openjdk-11

# define slave node details in `conf/slaves` file
vim conf/slaves
spark-worker1
spark-worker2
spark-worker3
spark-worker4

# generate ssh key pair for master node
# following is the public key
cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCvuLBzWJmQEAhqymY+QyyWcGVoyhYqE3cohZsDH5L
```

2. Setup Worker Nodes

Here's how I set up the Worker nodes in the cluster. It included installing Java, Scala, Spark, and configurations.

```
# add node host names into /etc/hosts
sudo vim /etc/hosts
172.31.1.30     spark-master
172.31.4.133    spark-worker1
172.31.4.171    spark-worker2
172.31.22.29    spark-worker3
172.31.17.138   spark-worker4

# install asdf
git clone https://github.com/asdf-vm/asdf.git ~/.asdf --branch v0.13.0

# configure asdf
vim ~/.bashrc
. "$HOME/.asdf/asdf.sh"
. "$HOME/.asdf/completions/asdf.bash"

# install java, scala
asdf plugin add java
asdf install java openjdk-11
asdf global java openjdk-11
asdf plugin add scala
asdf install scala 2.13.10
asdf global scala 2.13.10

# set java home
. ~/.asdf/plugins/java/set-java-home.bash

# install spark
wget https://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.0-bin-hadoop2.
tar xvf spark-3.0.0-bin-hadoop2.7.tgz
mv spark-3.0.0-bin-hadoop2.7 spark-3.0.0
cd spark-3.0.0

# set spark home and jave home
vim ~/.bashrc
export SPARK_HOME=$HOME/spark-3.0.0
export PATH=$PATH:$SPARK_HOME/bin

# configure spark env in `conf/spark-env.sh` file
# there is template of the `spark-env.sh` file `conf/spark-env.sh.template`
cp conf/spark-env.sh.template conf/spark-env.sh

# define master node host and java home
vim conf/spark-env.sh
export SPARK_MASTER_HOST=172.31.1.30
export JAVA_HOME=/home/ubuntu/.asdf/installs/java/openjdk-11

# define slave node details in `conf/slaves` file
vim conf/slaves
spark-worker1
spark-worker2
```

```
spark-worker3
spark-worker4

# generate ssh key pair for master node
# following is the public key
cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCvuLBzWJmQEAhqymY+QyyWcGVoyhYqE3cohZsDH5L

# copy the public key of the master node into each worker nodes' `~/.ssh/author
# following is the content of the `~/.ssh/authorized_keys` file in each worker
vim ~/.ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCvuLBzWJmQEAhqymY+QyyWcGVoyhYqE3cohZsDH5L
```

3. Start Spark Cluster

The cluster manager scripts reside in the `$SPARK_HOME/sbin` directory. Here is how you can start the Spark cluster in standalone mode.

```
# start cluster
# this command will start 1 master node and 4 worker nodes according to the `sl
./sbin/start-all.sh

# master logs resides in SPARK_HOME/logs directory of master node
ls logs/
spark-ubuntu-org.apache.spark.deploy.master.Master-1-ip-172-31-1-30.out

# worker logs resides in SPARK_HOME/logs directory of each worker node
ls logs/
spark-ubuntu-org.apache.spark.deploy.worker.Worker-1-ip-172-31-4-133.out

# stop cluster
./sbin/stop-all.sh
```

4. Submit Spark Job

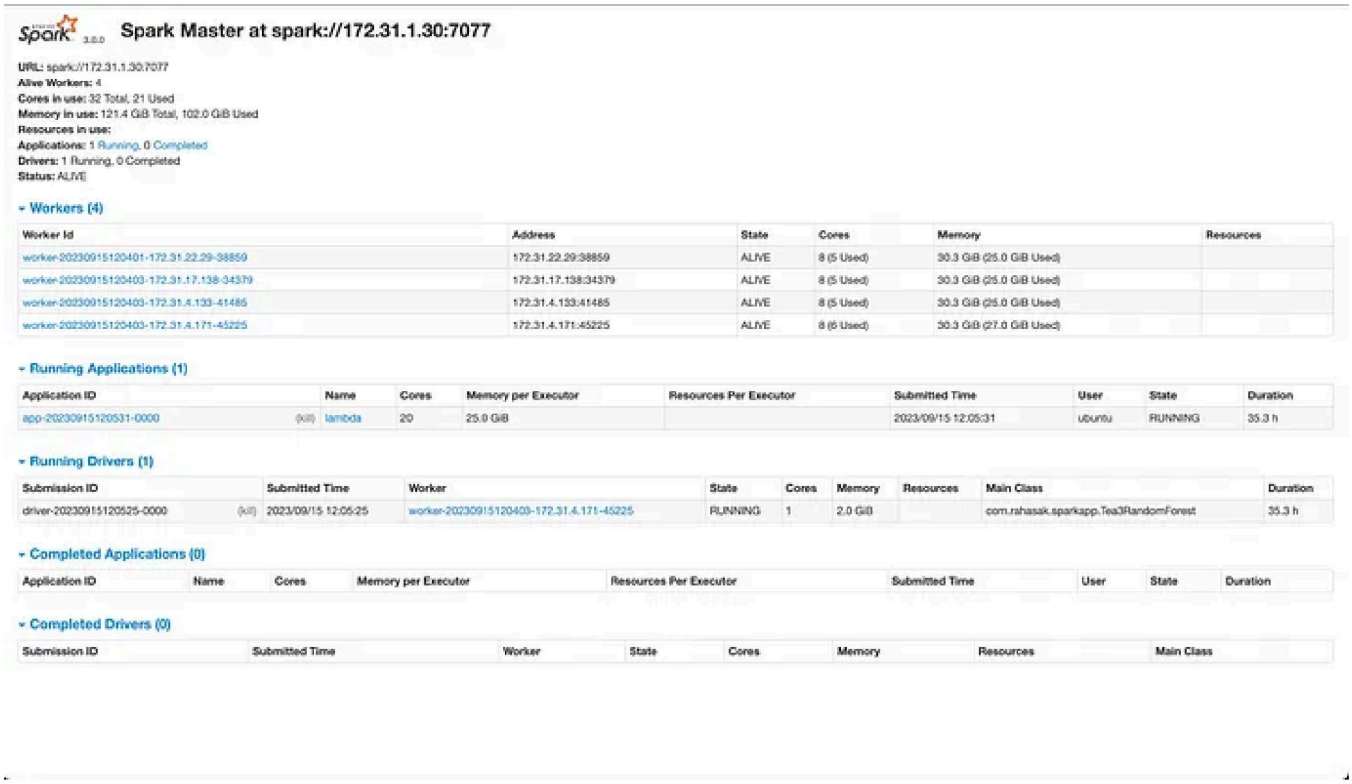
I've created a `.jar` file containing a Spark job and I'm going to submit this Spark job (as a `.jar` file) to the deployed Spark cluster using the `spark-submit` command. When submitting the job, we have the option to specify various configurations related to the Spark driver and executors. Below, you'll find the `spark-submit` command along with different configurations.

```
# --master spark://172.31.1.30:7077: This parameter specifies the URL of the Sp
# --deploy-mode cluster: This parameter specifies the deployment mode for the S
# --driver-memory 2g: This parameter sets the amount of memory to be allocated
# --num-executors 2: This parameter specifies the number of Spark executors to
# --executor-cores 5: This parameter sets the number of CPU cores to be allocat
# --executor-memory 25g: This parameter specifies the amount of memory to be al
# --class com.rahasak.sparkapp.Tea3RandomForest: This parameter specifies the m
# http://172.31.1.30:8000/sjobs.jar: This is the path to the JAR file containir
spark-submit --master spark://172.31.1.30:7077 \
 --deploy-mode cluster \
 --driver-memory 2g \
 --num-executors 2 \
 --executor-cores 5 \
 --executor-memory 25g \
 --class com.rahasak.sparkapp.Tea3RandomForest http://172.31.1.30:8000/sjobs.ja
```

One key aspect to pay attention to in this `spark-submit` command is the configuration parameters, namely `driver-memory`, `num-executors`, `executor-cores`, and `executor-memory`. These settings should be carefully determined based on the CPU and memory limits of the cluster nodes. For a detailed explanation on how to decide these configurations, you can refer to this video resource, which provides valuable insights. Additionally, I've published the `.jar` file containing the Spark job on a simple HTTP web server and specified its address as the path in the `spark-submit` command. You can find more information on this approach in this blog post.

5. Access Spark Web UI

Apache Spark offers a comprehensive set of web user interfaces (UIs) designed for monitoring various aspects of your Spark cluster and jobs. By default, the Spark web UI is accessible on port `8080` of the Spark master node. In my setup, you can access it using the address `172.31.1.30:8080` or by using the public IP of the machine. This web interface provides detailed information about the cluster's workers, currently running applications, and Spark driver specifics. Notably, you can track the progress, view any errors, and access standard outputs generated by your Spark job directly from the web UI.



Reference

1. https://www.innovationmerge.com/2021/06/26/Setting-up-a-multi-node-Apache-Spark-Cluster-on-a-Laptop/
2. https://aws.plainenglish.io/how-to-setup-install-an-apache-spark-3-1-1-cluster-on-ubuntu-817598b8e198
3. https://www.linkedin.com/pulse/how-setup-install-apache-spark-311-cluster-ubuntu-shrivastava
4. https://data-flair.training/blogs/install-apache-spark-multi-node-cluster/
5. https://www.cloudduggu.com/spark/installation-multi-node/
6. https://medium.com/codex/setting-up-a-multi-node-apache-spark-cluster-on-apache-hadoop-and-apache-hive-412764ab6881
7. https://sparkbyexamples.com/spark/spark-submit-command/
8. https://medium.com/rahasak/tagged/spark
9. https://medium.com/rahasak/random-forest-price-prediction-model-with-spark-kubernetes-operator-and-minio-object-storage-f11f10ed7f5c
10. https://medium.com/rahasak/spark-cluster-deployment-with-kubernetes-1848d061cfc9

Spark Machine Learning Mlops Spark Mllib AI

Follow

Written by (λx.x)eranga

2.5K Followers · Editor for Effectz.AI

Ego = 1/Knowledge

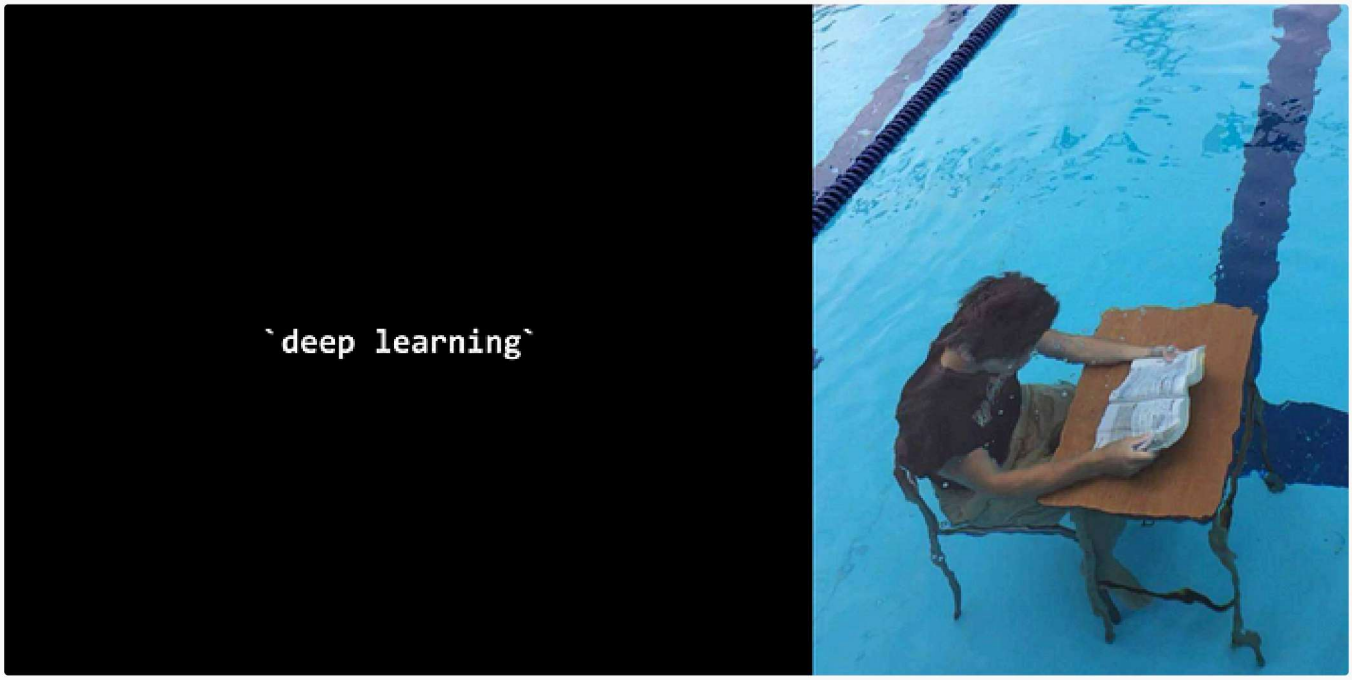
More from (λx.x)eranga and Effectz.AI

(λx.x)eranga in Effectz.AI

Build Your Own LLM with LLM Fine-Tuning on MacOS Using MLX

Privacy-preserving LLM

Jun 21 130 3

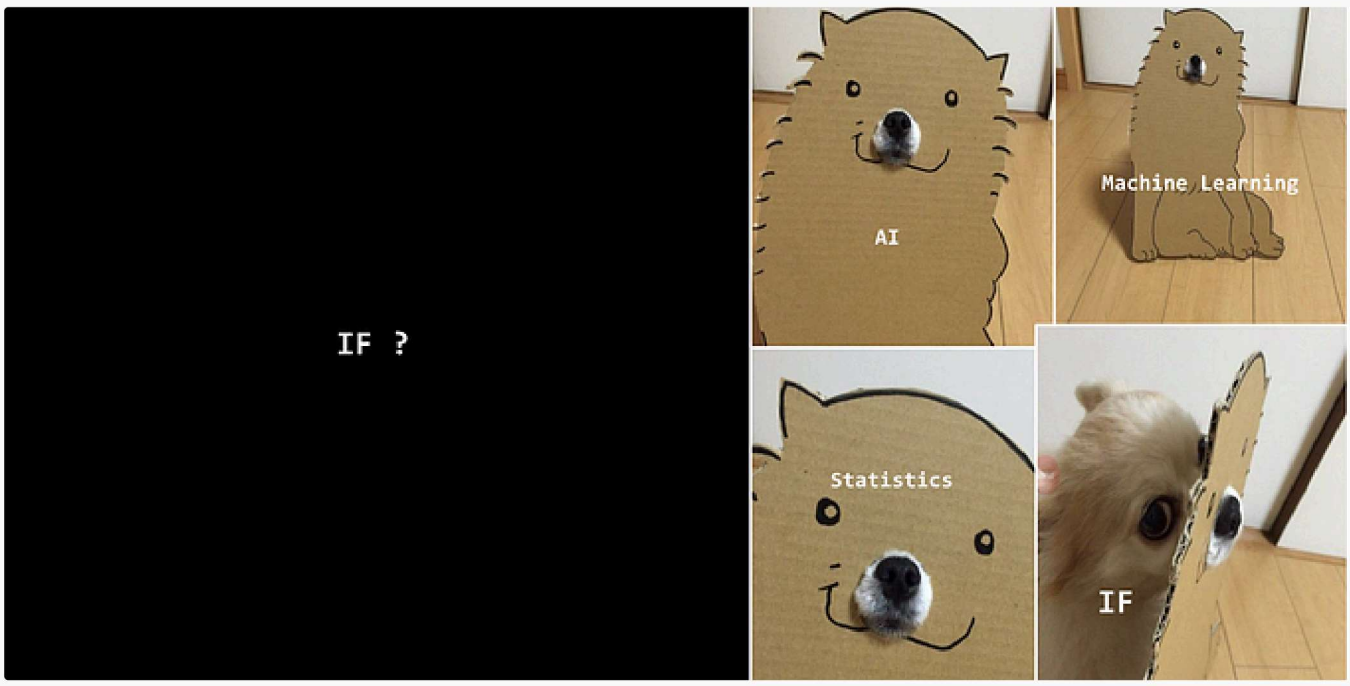


(λx.x)eranga in Effectz.AI

Build RAG Application Using a LLM Running on Local Computer with Ollama and Langchain

Privacy-preserving LLM without GPU

Mar 17 744 7

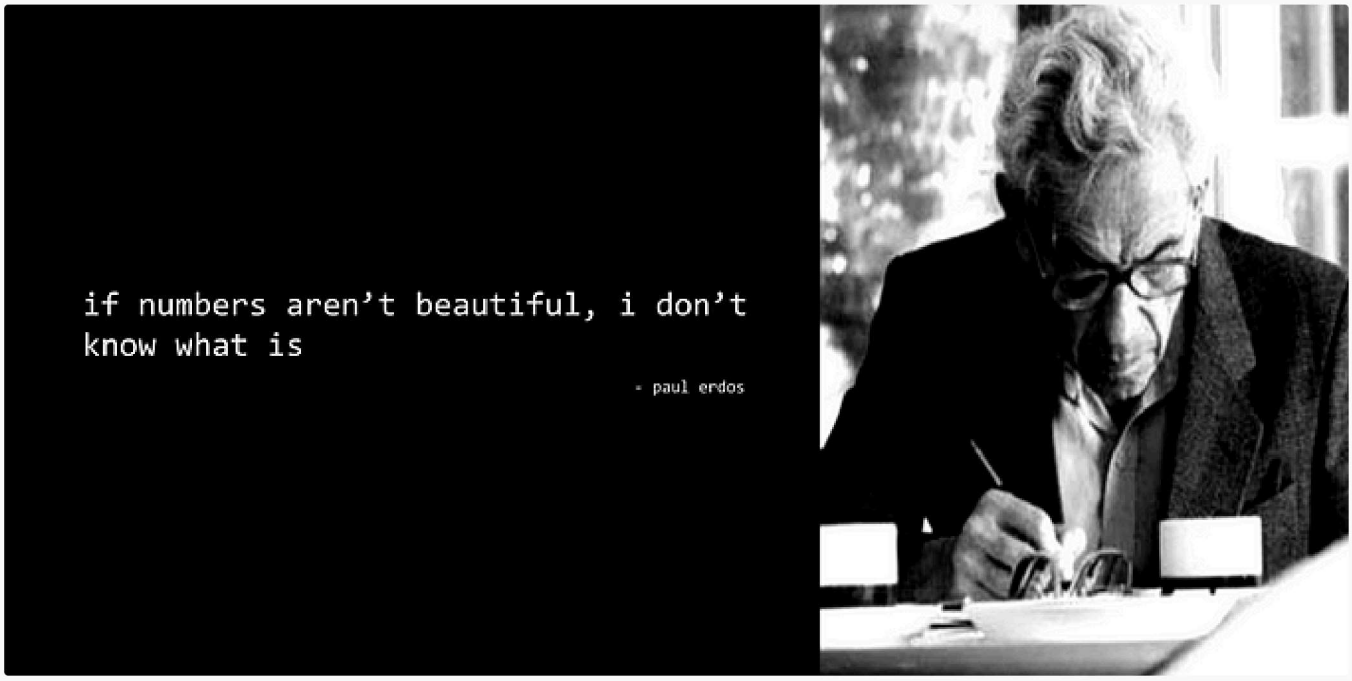


(λx.x)eranga in Effectz.AI

Build RAG Application Using a LLM Running on Local Computer with GPT4All and Langchain

Privacy-preserving LLM without GPU

Mar 11 592 8



(λx.x)eranga in Effectz.AI

Build RAG Application Using a LLM Running on Local Computer with Ollama Llama2 and LlamaIndex

Privacy-preserving LLM without GPU

Mar 25 592 3

See all from (λx.x)eranga

See all from Effectz.AI

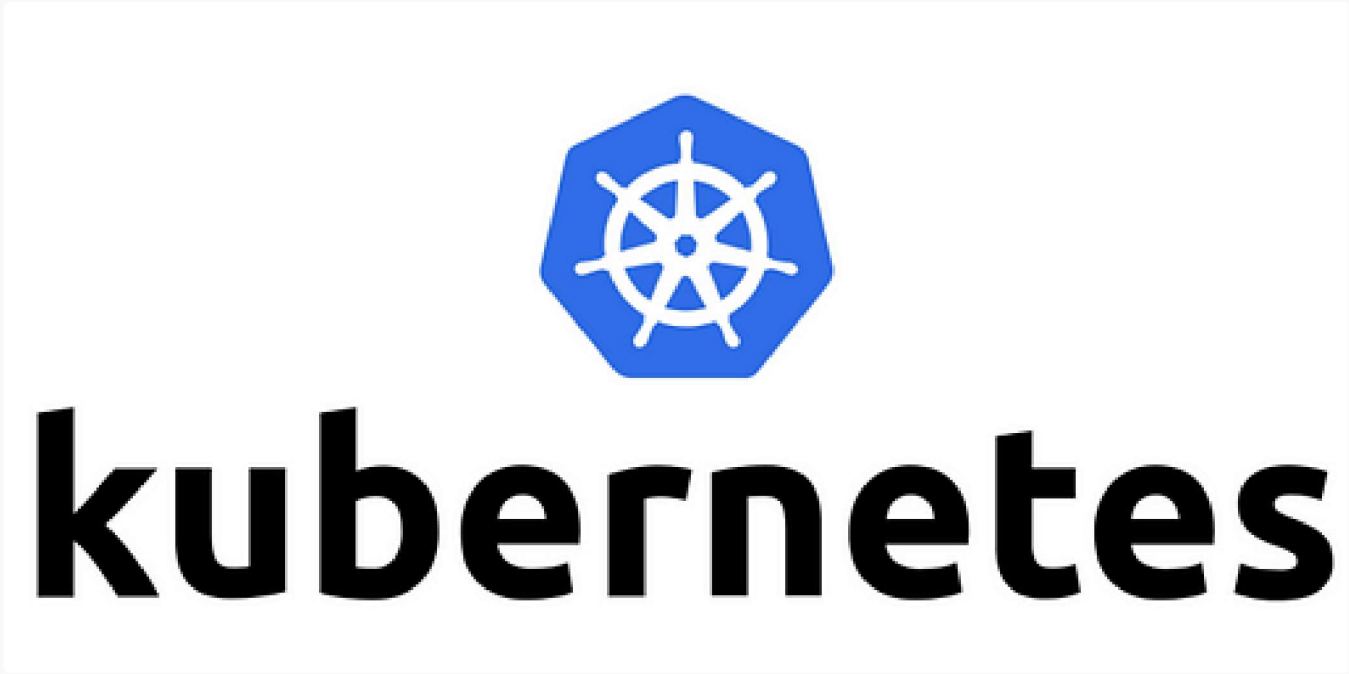
Recommended from Medium

Vishal Barvaliya

20 Real-Time Spark Scenario-Based Questions for Data Engineers

Apache Spark is a powerful tool for processing large datasets in real-time. If you're new to Spark or looking to understand how it can be...

Jul 16 128



Sai Parvathaneni in Towards Dev

Data Engineering with Kubernetes: End-to-End Data Pipeline

In this article, we dive into setting up Kubernetes for modern data engineering tasks. We'll cover everything from installation to running...

3d ago 8

Lists

Predictive Modeling w/ Python

20 stories · 1413 saves

The New Chatbots: ChatGPT, Bard, and Beyond

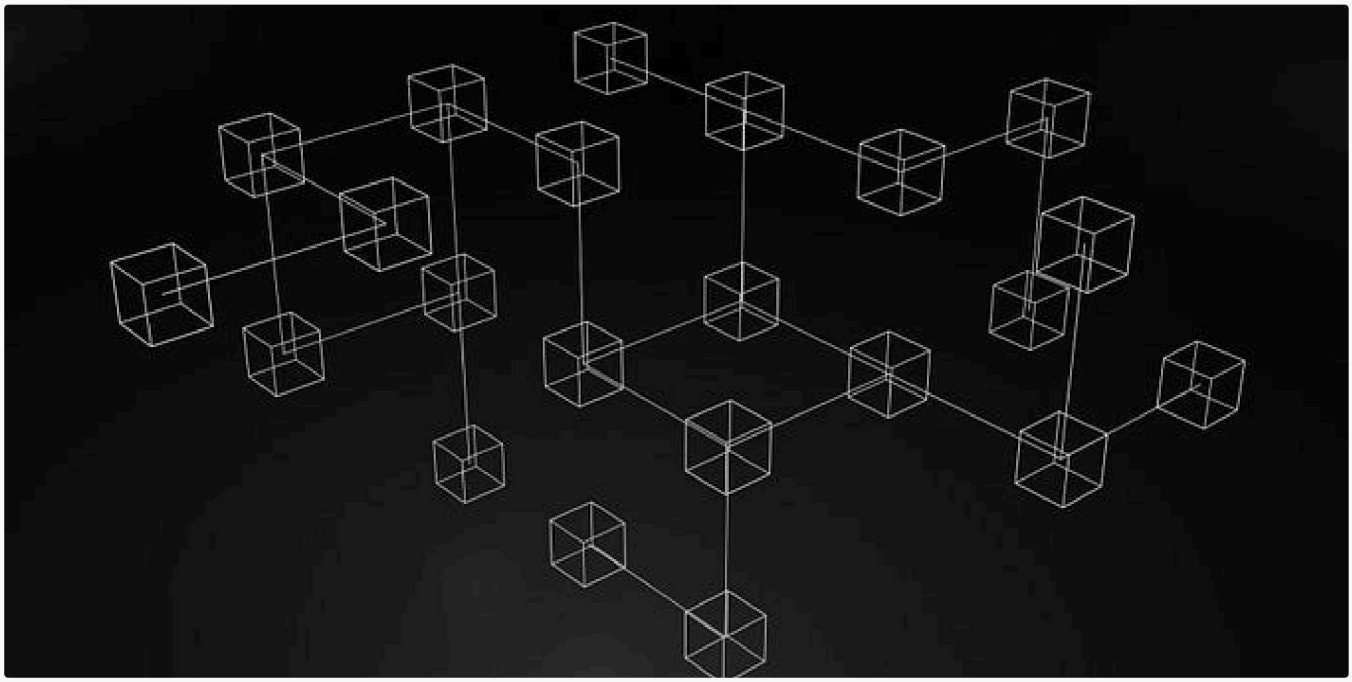
12 stories · 431 saves

Practical Guides to Machine Learning

10 stories · 1713 saves

Natural Language Processing

1607 stories · 1169 saves



Adam Szpilewicz in Level Up Coding

Setting Up a Local Spark Playground with Docker Compose

In today's data-driven environment, Apache Spark stands out as an essential tool for big data processing, offering powerful capabilities...

Feb 1 50

Akshay Goyal

How to Install Apache Spark on Local Machine: A Step-by-Step Guide for Mac, Linux and Windows Users

Mar 8 3

Keivan Soleimani

Spark On Kubernetes

Spark On Kubernetes Cluster

Mar 28 5

Christopher Chung in Polymath Data Lab

Apache Spark Lazy Evaluation: Transformations vs. Actions

One crucial aspect of using Spark effectively is understanding the distinction between transformations and actions. One crucial aspect of...