SPARK ASSIGNMENT

- HARSHINI V

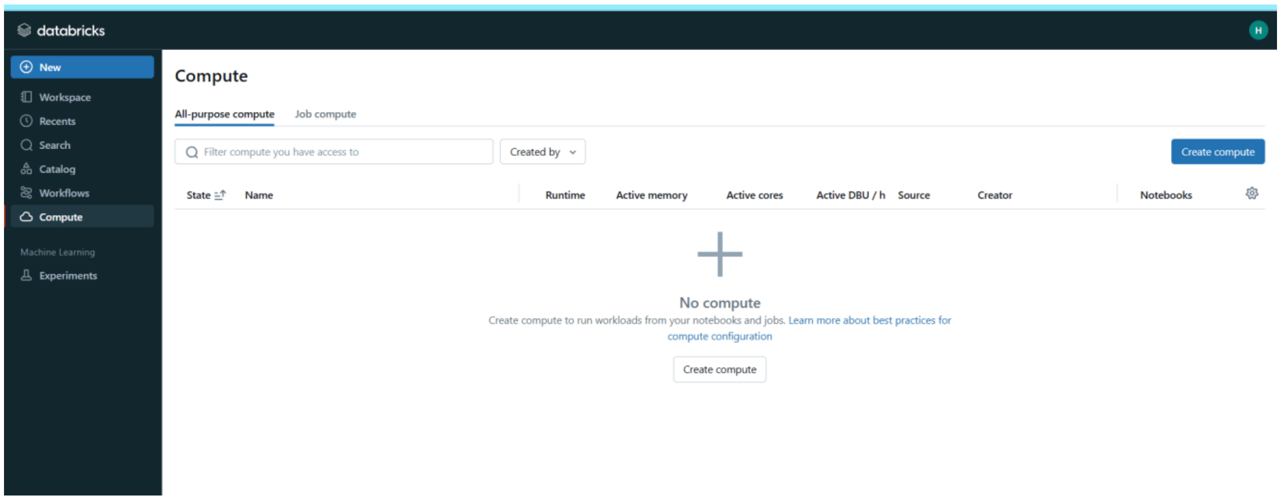
CLUSTER CREATION PROCESS:

The cluster creation process in Databricks is designed to be user-friendly while offering extensive customization options. These clusters serve as the backbone for executing data workflows, making them an essential component of any Databricks environment.

Below are the steps involved in creating a cluster in Databricks,

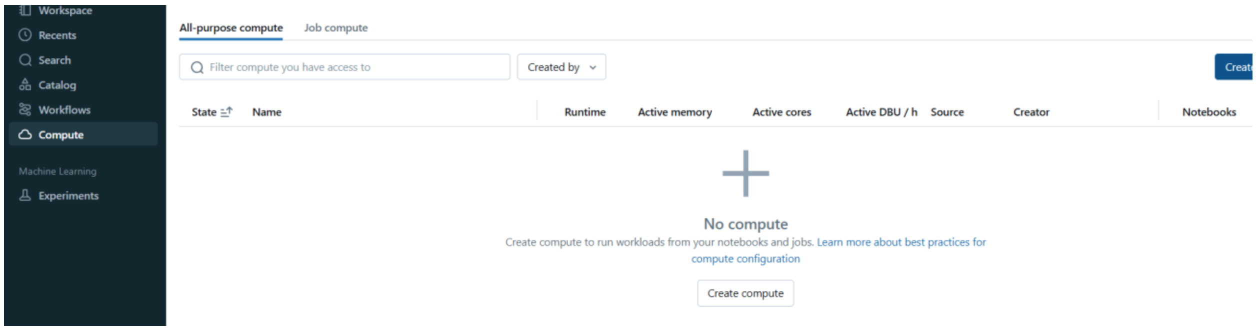
1. Log in to Databricks

To create a cluster, the user must first access the Databricks workspace through its web interface. This requires valid credentials or Single Sign-On (SSO), depending on the organization's setup. The workspace serves as a centralized platform for managing data workflows, including cluster creation.



2. Accessing the Compute Tab

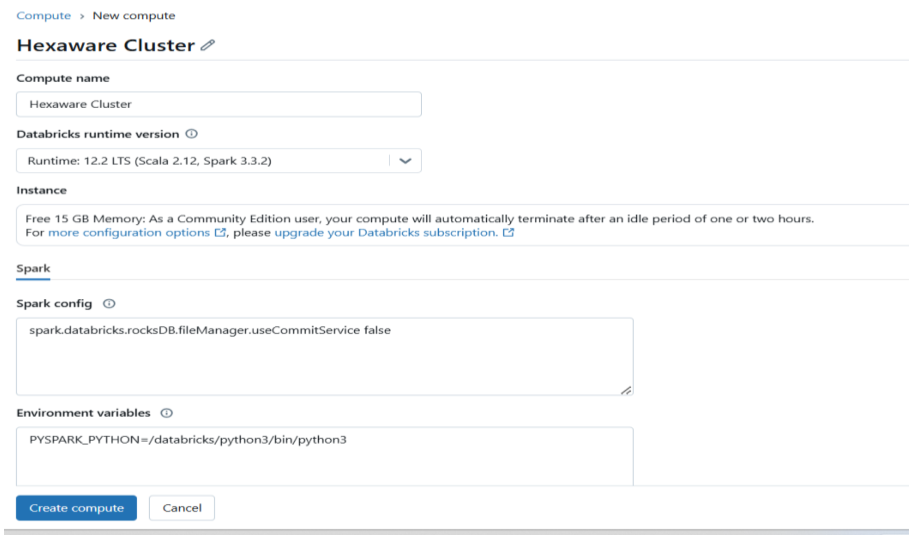
Clusters in Databricks are managed through the Compute tab, which is accessible from the workspace’s navigation menu. This section provides an interface for viewing, managing, and configuring clusters essential for computational tasks. On the left-hand sidebar of the Databricks workspace, locate and click on the "Compute" tab. This section is dedicated to managing clusters within your Databricks environment. In the Compute tab, locate the "Create Compute" button at the top of the page and click on it. This will open the cluster configuration form, where you can define the settings for your cluster.



3. Configure Cluster Settings

The cluster settings involve several components, including the cluster name, runtime version, and node types. Users can choose between different modes such as Standard, High Concurrency, or Single Node to suit their workload. Additional options include setting the number of worker nodes, enabling autoscaling for resource optimization, and specifying the type of instances for the driver and workers.

- **Cluster Name**: Provide a descriptive name for your cluster, such as "Development Cluster" or "Data Analysis Cluster".
- **Cluster Mode**: Select the mode of operation:
    - **Standard**: For regular processing.
    - **High Concurrency**: For multiple users sharing the cluster.
    - **Single Node**: For lightweight tasks.
- **Databricks Runtime Version**: Choose the appropriate runtime version based on your workload requirements. Ensure it supports your desired libraries (e.g., Python, Scala, R).
- **Node Type**: Specify the instance types for driver and worker nodes, such as **Standard_DS3_v2**.
- **Autoscaling** (Optional): Enable this feature to automatically adjust the number of worker nodes based on workload.
- **Worker and Driver Configuration**: Define the number of worker nodes and the type of driver node.


Compute › New compute

Hexaware Cluster

Compute name: Hexaware Cluster

Databricks runtime version: Runtime: 12.2 LTS (Scala 2.12, Spark 3.3.2)

Instance

Free 15 GB Memory: As a Community Edition user, your compute will automatically terminate after an idle period of one or two hours.
For more configuration options, please upgrade your Databricks subscription.

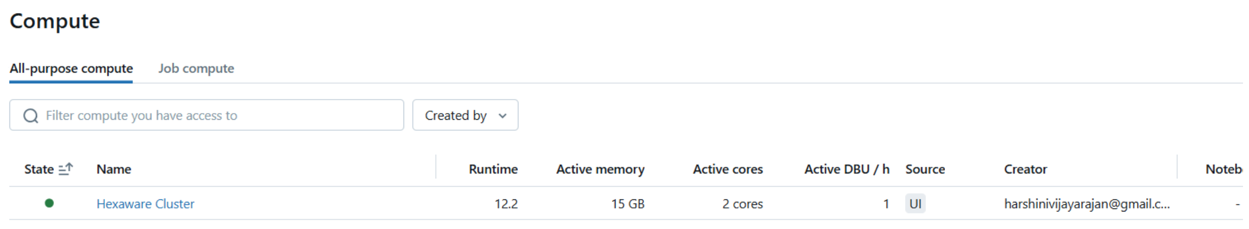
Spark

Spark config:
```
spark.databricks.rocksDB.fileManager.useCommitService false
```

Environment variables:
```
PYSPARK_PYTHON=/databricks/python3/bin/python3
```

4. Review and Create the Cluster

Review the cluster configurations you have entered to ensure they meet your requirements. Click on the "Create Cluster" button at the bottom of the form. Databricks will start provisioning your cluster. The process might take a few minutes, and the cluster's status will change to Running once it's ready.



State	Name	Runtime	Active memory	Active cores	Active DBU / h	Source	Creator	Noteb
●	Hexaware Cluster	12.2	15 GB	2 cores	1	UI	harshinivijayarajan@gmail.c...	-

Spark Architecture Overview

Apache Spark's architecture is designed to handle large-scale data processing by leveraging in-memory computation and distributed data processing. It provides high performance for batch and streaming data processing, making it a versatile tool for big data applications.

Core Components of Spark Architecture

The Spark architecture consists of several key components, each playing a crucial role in its operation:

1. **Driver Program**:
   The driver program is the central orchestrator of a Spark application. It runs the user's main function, creates the SparkContext, and coordinates the execution of tasks. The driver is responsible for translating high-level code into a directed acyclic graph (DAG) of stages and scheduling these stages on worker nodes.
2. **Cluster Manager**:
   Spark can run on various cluster managers such as Standalone, Apache Mesos, Hadoop YARN, or Kubernetes. The cluster manager handles resource allocation and task scheduling across the cluster nodes.
3. **Worker Nodes**:
   Worker nodes are the machines where tasks are executed. Each worker node runs one or more executors, which are responsible for executing the tasks assigned to them and storing data partitions in memory or disk as required.
4. **Executors**:
   Executors are processes launched on worker nodes that perform the actual computation. They also manage the storage of intermediate data for fault tolerance and caching.
5. **Spark Context**:
   SparkContext is the entry point for any Spark application. It connects the driver program to the cluster manager and initializes the application by providing configuration information and access to the cluster's resources.

Execution Workflow in Spark Architecture

1. **Job Submission**:
   A Spark application is submitted to the driver, which defines the computation logic using RDDs, DataFrames, or Datasets.
2. **DAG Creation**:
   The driver creates a Directed Acyclic Graph (DAG) that represents the sequence of transformations on the data. The DAG is then divided into stages based on wide and narrow dependencies.
3. **Task Scheduling**:
   The DAG Scheduler breaks down the stages into tasks and sends them to the cluster manager for execution.
4. **Task Execution**:
   The cluster manager allocates resources to worker nodes, where executors execute the tasks. Tasks process data from their assigned partitions.
5. **Result Collection**:
   Once tasks are completed, results are sent back to the driver program. Actions in Spark, such as collect() or saveAsTextFile(), finalize the computation and provide the output.

Key Features of Spark Architecture

1. **In-Memory Computation**:
   Spark minimizes I/O operations by caching data in memory, which significantly enhances performance for iterative computations.
2. **Fault Tolerance**:
   Spark ensures fault tolerance through lineage information. If a node fails, Spark recomputes the lost data from its source.
3. **Scalability**:
   Spark's distributed nature allows it to scale seamlessly across clusters with thousands of nodes.
4. **Support for Diverse Workloads**:
   Spark supports batch processing, interactive querying, real-time analytics, machine learning, and graph processing, all within the same architecture.
5. **Unified API**:
   Spark provides a consistent API for RDDs, DataFrames, and Datasets, simplifying development and ensuring compatibility across its components.

Apache Spark's architecture is a blend of simplicity, flexibility, and efficiency. Its ability to process massive datasets in a distributed and fault-tolerant manner makes it one of the most powerful tools for modern data analytics and machine learning applications.