Azure Databricks concept

Azure Databricks provides a notebook-oriented Apache Spark as-a-service workspace environment. It is the most feature-rich hosted service available to run Spark workloads in Azure. Apache Spark is a unified analytics engine for large-scale data processing and machine learning.

Suppose you work with Big Data as a data engineer or a data scientist, and you must process data that you can describe as having one or more of the following characteristics:

1. High volume - You must process an extremely large volume of data and need to scale out your compute accordingly
2. High velocity - You require streaming and real-time processing capabilities
3. Variety - Your data types are varied, from structured relational data sets and financial transactions to unstructured data such as chat and SMS messages, IoT devices, images, logs, MRIs, etc.

These characteristics are oftentimes called the "3 Vs of Big Data".

When it comes to working with Big Data in a unified way, whether you process it real time as it arrives or in batches, Apache Spark provides a fast and capable engine that also supports data science processes, like machine learning and advanced analytics.

In this module, you will learn:

•Understand the architecture of an Azure Databricks Spark Cluster
•Understand the architecture of a Spark Job

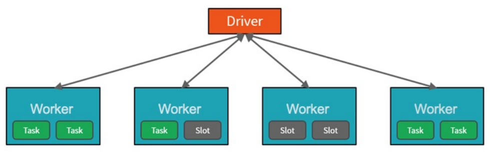
Understand the architecture of an Azure Databricks Spark Cluster

To gain a better understanding of how to develop with Azure Databricks, it is important to understand the underlying architecture. We will look at two aspects of the Databricks architecture: the Azure Databricks service and Apache Spark clusters.

High-level overview

From a high level, the Azure Databricks service launches and manages Apache Spark clusters within your Azure subscription. Apache Spark clusters are groups of computers that are treated as a single computer and handle the execution of commands issued from notebooks. Using a master-worker type architecture, clusters allow processing of data to be parallelized across many computers to improve scale and performance. They

consist of a Spark Driver (master) and worker nodes. The driver node sends work to the worker nodes and instructs them to pull data from a specified data source.
In Databricks, the notebook interface is the driver program. This driver program contains the main loop for the program and creates distributed datasets on the cluster, then applies operations (transformations & actions) to those datasets. Driver programs access Apache Spark through a SparkSession object regardless of deployment location.

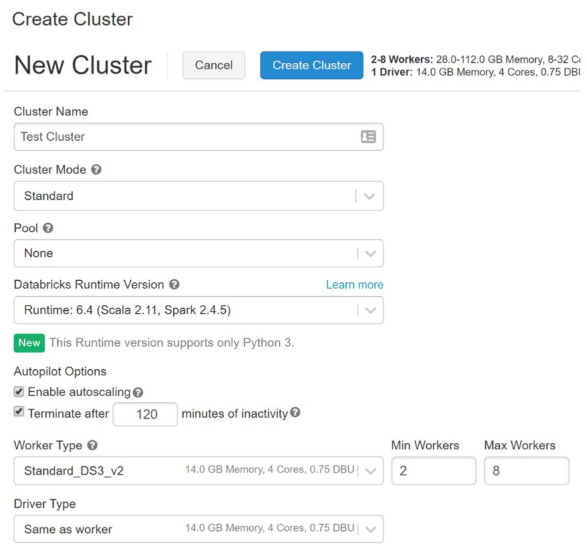


Microsoft Azure manages the cluster, and auto-scales it as needed based on your usage and the setting used when configuring the cluster. Auto-termination can also be enabled, which allows Azure to terminate the cluster after a specified number of minutes of inactivity.

Under the covers

Now let's take a deeper look under the covers. When you create an Azure Databricks service, a "Databricks appliance" is deployed as an Azure resource in your subscription. At the time of cluster creation, you specify the types and sizes of the virtual machines (VMs) to use for both the Driver and Worker nodes, but Azure Databricks manages all other aspects of the cluster.

You also have the option of using a Serverless Pool. A Serverless Pool is self-managed pool of cloud resources that is auto-configured for interactive Spark workloads. You provide the minimum and maximum number of workers and the worker type, and Azure Databricks provisions the compute and local storage based on your usage.



2-8 Workers: 28.0-112.0 GB Memory, 8-32 C
1 Driver: 14.0 GB Memory, 4 Cores, 0.75 DB

Cluster Name: Test Cluster

Cluster Mode: Standard

Pool: None

Databricks Runtime Version: Runtime: 6.4 (Scala 2.11, Spark 2.4.5)

This Runtime version supports only Python 3.

Autopilot Options
- [✓] Enable autoscaling
- [✓] Terminate after 120 minutes of inactivity

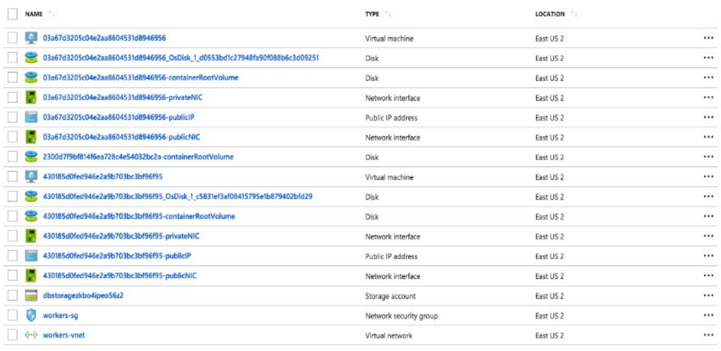
Worker Type: Standard_DS3_v2 14.0 GB Memory, 4 Cores, 0.75 DBU

Min Workers: 2

Max Workers: 8

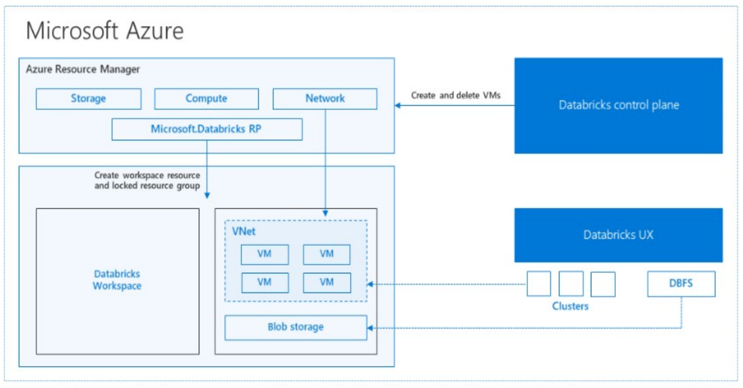
Driver Type: Same as worker 14.0 GB Memory, 4 Cores, 0.75 DBU

The "Databricks appliance" is deployed into Azure as a managed resource group within your subscription. This resource group contains the Driver and Worker VMs, along with other required resources, including a virtual network, a security group, and a storage account. All metadata for your cluster, such as scheduled jobs, is stored in an Azure Database with geo-replication for fault tolerance.



NAME	TYPE	LOCATION
03a67d3205c04e2aa8604531d8946956	Virtual machine	East US 2
03a67d3205c04e2aa8604531d8946956_OsDisk_1_d0553bd1c27948fa90f088b6c3d09251	Disk	East US 2
03a67d3205c04e2aa8604531d8946956-containerRootVolume	Disk	East US 2
03a67d3205c04e2aa8604531d8946956-privateNIC	Network interface	East US 2
03a67d3205c04e2aa8604531d8946956-publicIP	Public IP address	East US 2
03a67d3205c04e2aa8604531d8946956-publicNIC	Network interface	East US 2
2300d7f9bf814f6ea728c4e54032bc2a-containerRootVolume	Disk	East US 2
430185d0fed946e2a9b703bc3bf96f95	Virtual machine	East US 2
430185d0fed946e2a9b703bc3bf96f95_OsDisk_1_c5831ef3af08415795e1b879402bfd29	Disk	East US 2
430185d0fed946e2a9b703bc3bf96f95-containerRootVolume	Disk	East US 2
430185d0fed946e2a9b703bc3bf96f95-privateNIC	Network interface	East US 2
430185d0fed946e2a9b703bc3bf96f95-publicIP	Public IP address	East US 2
430185d0fed946e2a9b703bc3bf96f95-publicNIC	Network interface	East US 2
dbstoragezkbo4ipeo56z2	Storage account	East US 2
workers-sg	Network security group	East US 2
workers-vnet	Virtual network	East US 2

Internally, Azure Kubernetes Service (AKS) is used to run the Azure Databricks control-plane and data-planes via containers running on the latest generation of Azure hardware (Dv3 VMs), with NvMe SSDs capable of blazing 100us latency on IO. These make Databricks I/O performance even better. In addition, accelerated networking provides the fastest virtualized network infrastructure in the cloud. Azure Databricks utilizes these features to further improve Spark performance. Once the services within this managed resource group are ready, you will be able to manage the Databricks cluster through the Azure Databricks UI and through features such as auto-scaling and auto-termination.

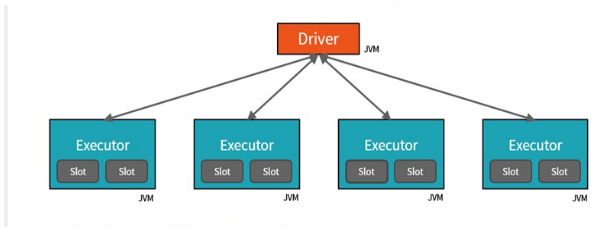


Understand the architecture of a Spark Job

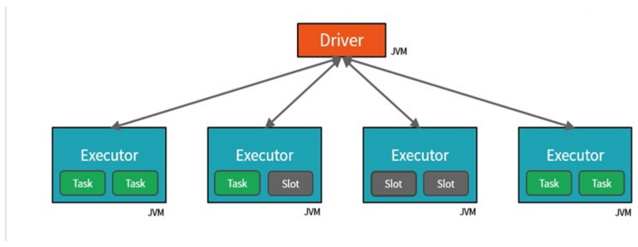
Let's dive a little deeper into how the cluster relates to Spark Jobs.

Before we break down these components in detail, let us summarize the fundamentals of a Spark architecture. As you may know, Spark is a Distributed computing environment. The unit of distribution is a Spark Cluster. Every Cluster has a Driver and one or more executors. Work submitted to the Cluster is split into as many independent Jobs as needed. This is how work is distributed across the Cluster's nodes. Jobs are further subdivided into tasks. The input to a job is partitioned into one or more partitions. These partitions are the unit of work for each slot. In between tasks, partitions may need to be re-organized and shared over the network.

The cluster: Drivers, executors, slots & tasks



- •The **Driver** is the JVM in which our application runs.
- •The secret to Spark's awesome performance is parallelism.
  - •Scaling vertically is limited to a finite amount of RAM, Threads and CPU speeds.
  - •Scaling horizontally means we can simply add new "nodes" to the cluster almost endlessly.
- •We parallelize at two levels:
  - •The first level of parallelization is the **Executor** - a Java virtual machine running on a node, typically, one instance per node.
  - •The second level of parallelization is the **Slot** - the number of which is determined by the number of cores and CPUs of each node.
- •Each **Executor** has a number of **Slots** to which parallelized **Tasks** can be assigned to it by the **Driver**.



- • The JVM is naturally multithreaded, but a single JVM, such as our **Driver**, has a finite upper limit.
- •By creating **Tasks**, the **Driver** can assign units of work to **Slots** for parallel execution.
- •Additionally, the **Driver** must also decide how to partition the data so that it can be distributed for parallel processing (not shown here).

•Consequently, the **Driver** is assigning a **Partition** of data to each task - in this way each **Task** knows which piece of data it is to process.
•Once started, each **Task** will fetch from the original data source the **Partition** of data assigned to it.

Jobs & stages

•Each parallelized action is referred to as a **Job**.
•The results of each **Job** (parallelized/distributed action) is returned to the **Driver**.
•Depending on the work required, multiple **Jobs** will be required.
•Each **Job** is broken down into **Stages**.
•This would be analogous to building a house (the job)
  •The first stage would be to lay the foundation.
  •The second stage would be to erect the walls.
  •The third stage would be to add the room.
  •Attempting to do any of these steps out of order just won't make sense, if not just impossible.

Cluster management

- At a much lower level, Spark Core employs a **Cluster Manager** that is responsible for provisioning nodes in our cluster.
  - Databricks provides a robust, high-performing **Cluster Manager** as part of its overall offerings.
- In each of these scenarios, the **Driver** is [presumably] running on one node, with each **Executors** running on N different nodes.
- For the sake of this learning path, we don't need to concern ourselves with cluster management, thanks to Azure Databricks.
- From a developer's and learner's perspective my primary focus is on...
  - The number of **Partitions** my data is divided into.
  - The number of **Slots** I have for parallel execution.
  - How many **Jobs** am I triggering?