B
Free Spark Cloud Offering

So far, we have worked with Spark on your machine only. Since Spark was designed with a *build locally - deploy to cluster* paradigm in mind, it is about time for us to move to the cloud with some of our code.

In this chapter, we will look at two free trial offers from Databricks, and Microsoft's HDInsight. Each of these options is slightly different to work with, but they all share the same underlying capabilities of Spark. Note that there are also other free providers of Apache Spark, including:

- *Amazon EMR*: `http://docs.aws.amazon.com/ElasticMapReduce/latest/ReleaseGuide/emr-spark.html`
- *Google Cloud*: `https://cloud.google.com/hadoop/`
- *IBM Analytics for Apache Spark*: `http://www.ibm.com/analytics/us/en/technology/cloud-data-services/spark-as-a-service`

In this chapter, you will learn:

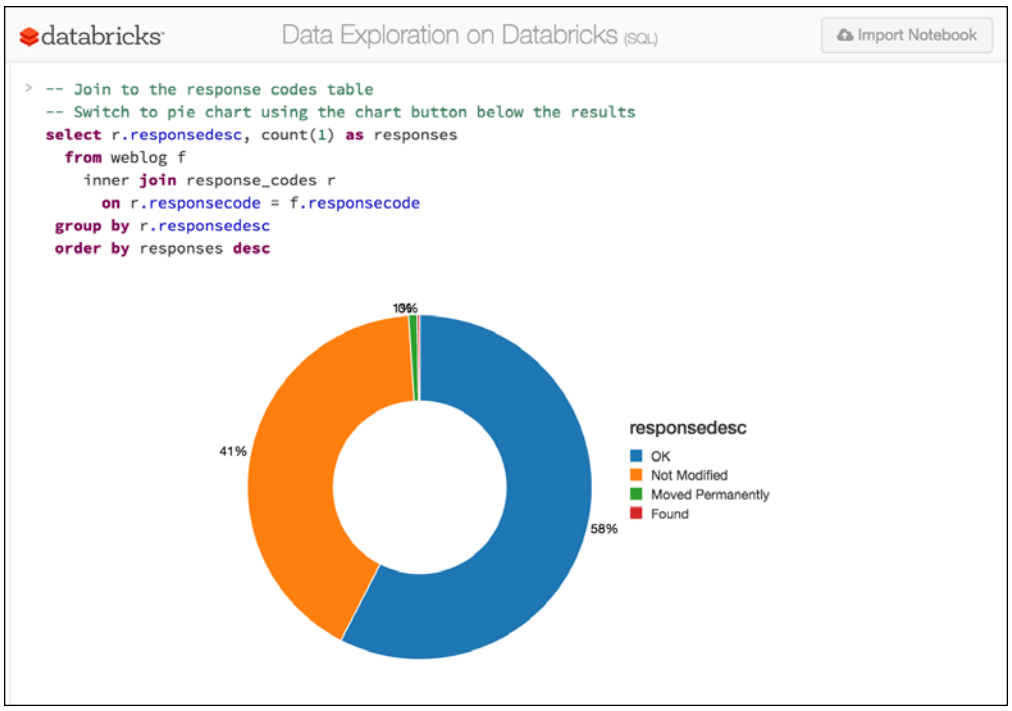
- What the Databricks Community Edition has to offer
- How to sign up, configure and run a cluster on Databricks
- How to monitor the execution of your jobs
- What Microsoft's Spark on HDInsight offers
- What are the steps to sign up and set up your cluster
- How to run and monitor your jobs with HDInsight

Databricks Community Edition

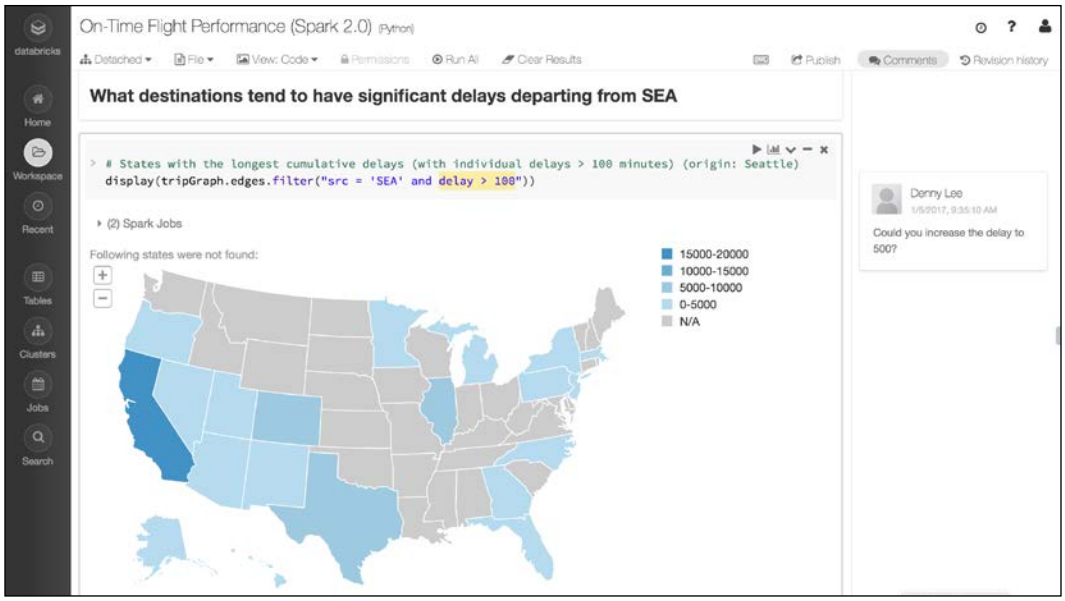
Databricks is a data platform that provides data integration, real-time exploration, and production pipelines as a managed cloud service powered by Apache Spark. The team that created Apache Spark also founded Databricks in 2013. Currently, Databricks is built on top of AWS Cloud Services. The Databricks platform itself provides a wide range of features designed for data engineers and data scientists, including the features noted in the following section.

Notebooks and Dashboards

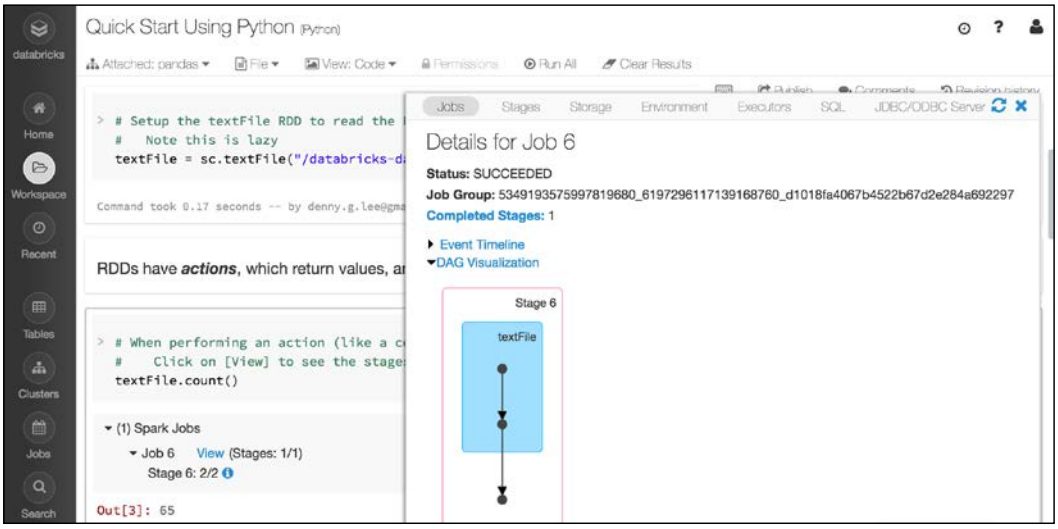
A collaborative interactive workspace that is designed for data scientists and data engineers, Databricks provides an integrated environment that allows you to execute `Python`, `Scala`, `R`, `SQL`, and `Markdown` *within* the same notebook. In addition to its own native visualizations, Databricks notebook allows you to integrate popular visualization libraries including `matplotlib`, `ggplot`, and `D3` (Databricks notebook pie (donut) chart is shown in the following screenshot):



Also, within the same integrated environment, multiple users can collaborate on the same notebook, comment, track with revision history (including GitHub integration), autocomplete, and so on:



To help make it easier to debug your notebooks, the Databricks notebook also includes a real-time progress bar that directly integrates Apache Spark's Web UI into the notebook. The following screenshot shows the Spark Web UI DAG visualization that is embedded directly into the notebook for the `textFile.count()` action:



Connectivity

Databricks allows you to connect to your data via your favorite BI tools and/or REST APIs:

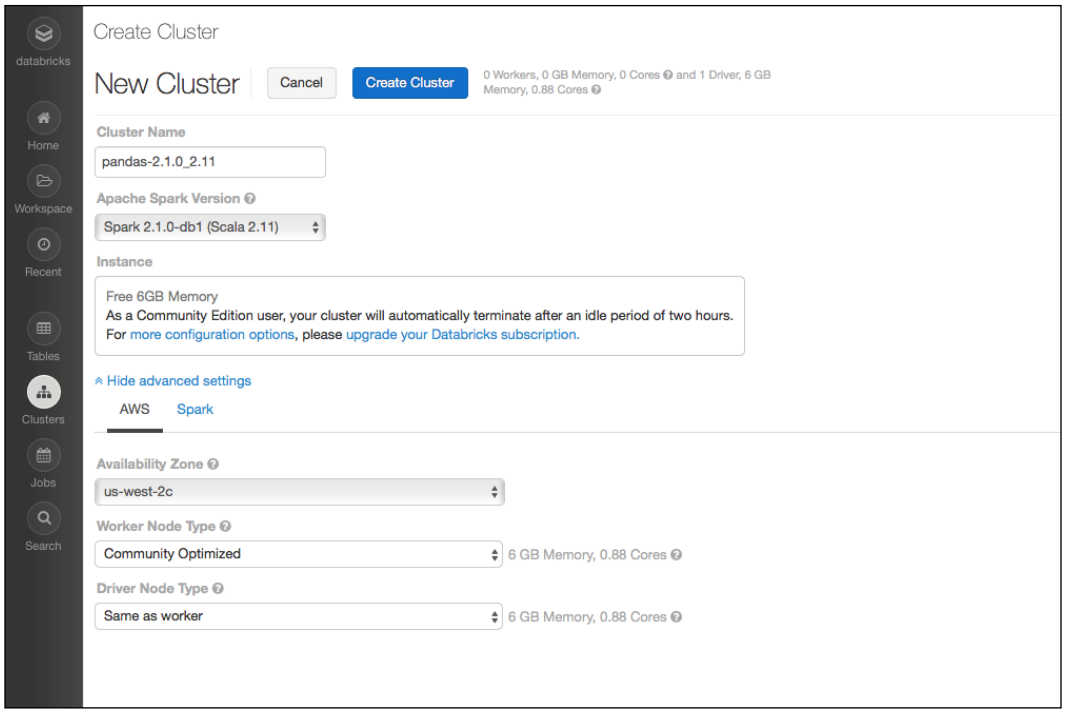
- **Secure SQL Server for BI Tools**: Databricks allows you to securely connect and query your data within Databricks managed Apache Spark clusters using your favorite BI tools, such as Tableau, Qlik, and PowerBI
- **REST API**: From cluster management, to uploading third-party libraries, to executing commands and contexts, you can script out these commands using the Databricks REST API

Jobs and workflows

Databricks has a Jobs and Workflows functionality that allows you to easily take your development notebooks and run them in production. In addition to a flexible scheduler, with Databricks you can run notebooks, Spark JARs, and Jobs. The Jobs feature provides run log history, retries, notifications, and flexible cluster support (for example, reusing existing clusters or launching on-demand clusters).

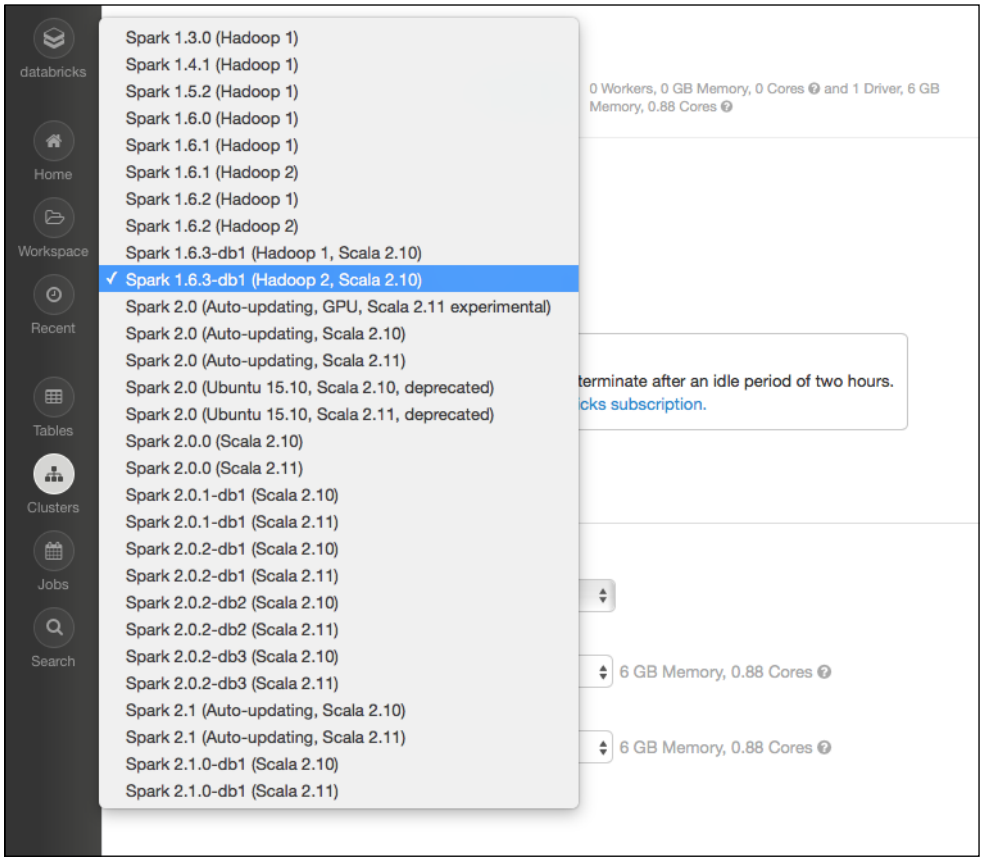
Cluster management

These features are built on top of Databricks managed services with easy-to-use Apache Spark cluster management. You can launch on-demand or spot clusters in a matter of minutes with just a few clicks. Important infrastructure features include high availability, elasticity, 100% Spark Version compatibility, automatic upgrades, and multiple instance types. All of this is supported and tuned for optimal performance by the experts who created Apache Spark. The following is a screenshot of the Databricks Community Edition Cluster Manager. To spin up a cluster, you need only to specify the name and which version of Apache Spark you would like to work with:



With the paid version of Databricks, you also get to choose which worker and driver instance type and an unlimited number of clusters, including the ability to auto-scale those clusters. With the Community Edition, you are provided with a 6 GB mini cluster that can easily handle learning Spark and small proof of concepts.

As noted in the previous section, you can also choose which version of Apache Spark you would like to use. At the time of writing, you can choose from Spark 1.3 to Spark 2.1 with all the major and minor versions in between:



Enterprise security

For those whom are security minded, the Databricks Enterprise Security Framework includes encryption, integrated identity management, role-based access control, access controls, and data governance. From an auditing certification perspective, Databricks has completed SOC 2 Type 1 and offers a HIPAA-compliant service; the service is also available in isolated and secure AWS GovCloud (US).

For more information, please refer to the following links:

- Databricks Product Page: `https://databricks.com/product/databricks`
- *Databricks Primer*: `https://databricks.com/wp-content/uploads/2016/02/Databricks-Primer.pdf`
- *Databricks Feature Primer*: `https://databricks.com/wp-content/uploads/2016/02/Databricks-Feature-Primer.pdf`
- *Databricks Security*: `https://databricks.com/product/security`
- *Protecting Enterprise Data on Apache Spark*: `http://go.databricks.com/protecting-enterprise-data-on-apache-spark-with-databricks`

The free options of Databricks

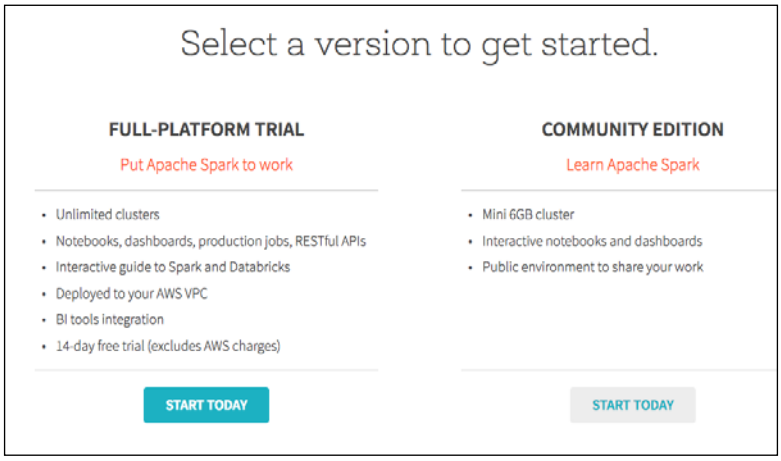
The free option of Databricks is Databricks Community Edition, which provides you with a mini 6GB cluster, interactive notebooks and dashboards, and a public environment to share your work for free. Anyone can sign up for this option and the entire service – including the ability to spin up different versions of Apache Spark on a single mini cluster – is completely free.

For Academic institutions, there is also the Databricks Academic Partners Program designed for both research and instruction. For these institutions, the only costs will be for Amazon cloud services and you can potentially apply for an AWS in Education grant to cover those costs. For more information, please refer to `https://databricks.com/academic`.

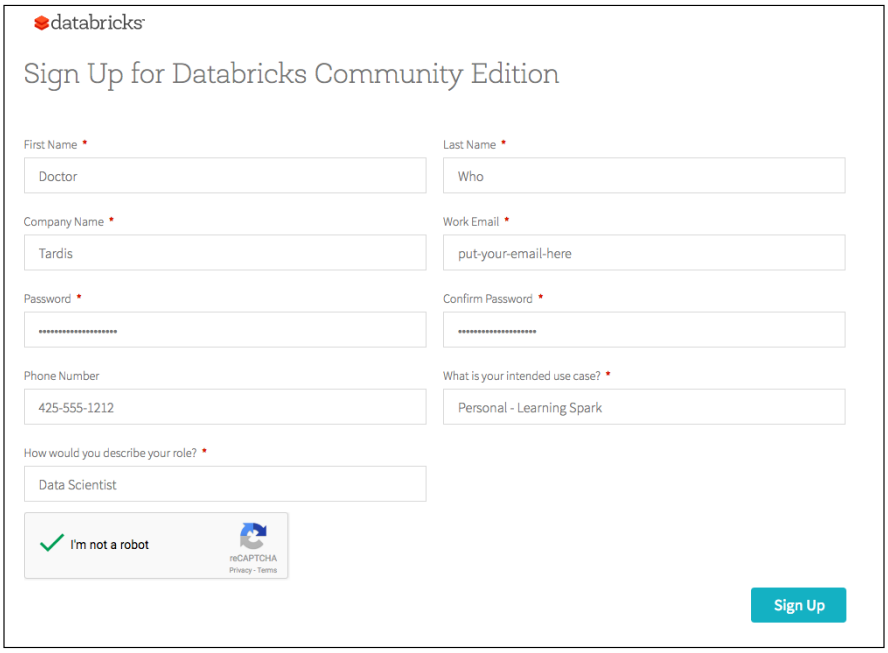
Note that the full platform provides you with important production features including (but not limited to) the ability to spin up an unlimited number of clusters, production jobs and RESTful APIs, BI tools integration, GitHub integration, and advanced security integration. To use the full platform, there is a 14-day free trial excluding AWS charges.

Signing up for the service

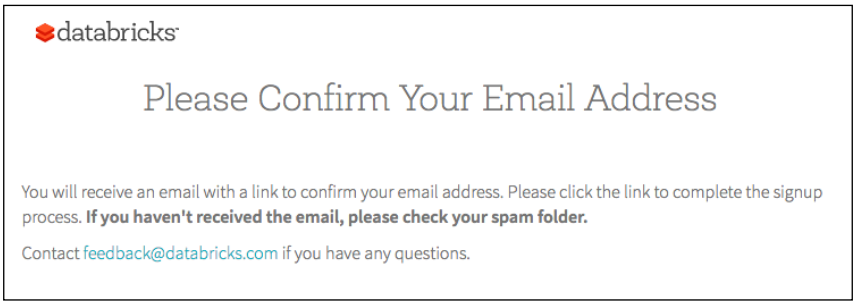
To sign up for the Databricks service, please go to `http://databricks.com/try-databricks`. On this page, you will be given the option to sign up for the full-platform trial (14-day free trial excluding AWS charges) and the *Community Edition*:



To use *Databricks Community Edition*, click on the appropriate **START TODAY** button and it will provide you with the **Sign Up for Databricks Community Edition** page. Fill out the form and click on **Sign Up**, as noted in the following screenshot:



Once you agree to the *Terms of Service*, you will then need to confirm your e-mail address, as indicated in the following screenshot:

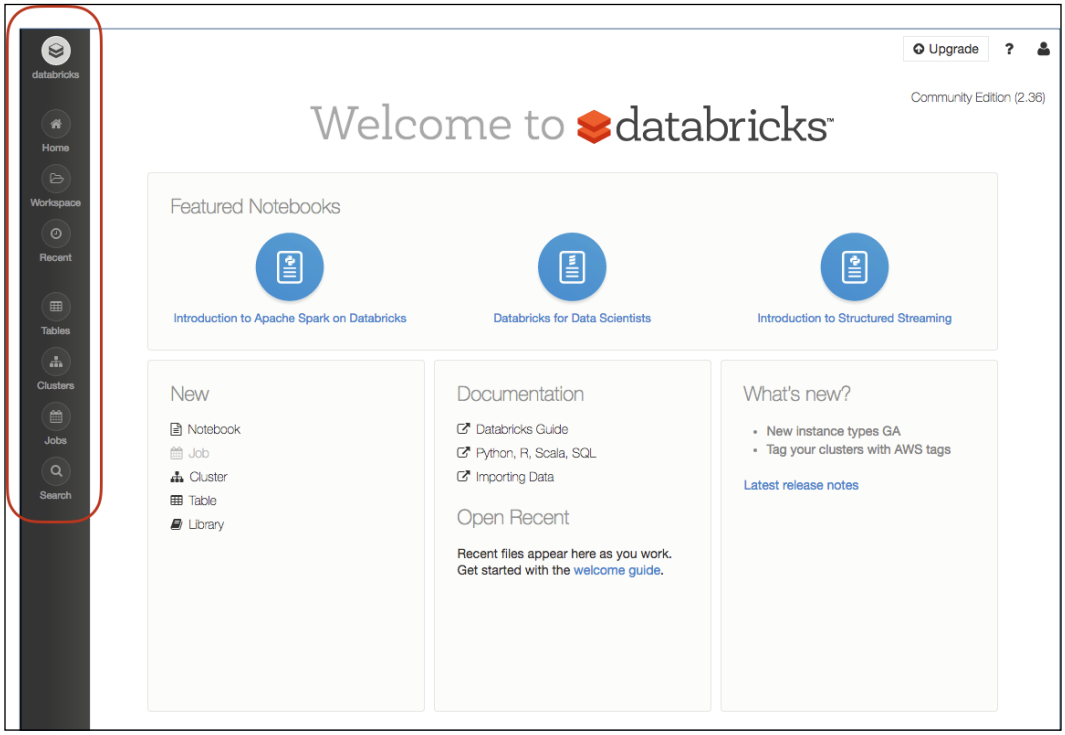


Go to the e-mail address that you provided in the sign-up page, and look for the *Welcome to Databricks! Please verify your e-mail address* e-mail. Once you have verified your e-mail address, you will be redirected to the Databricks login page to log in (as noted in the following screenshot):

In case you forget, the login for Databricks Community Edition can be accessed at `https://community.cloud.databricks.com`. Once you have logged in, you will be presented with the Databricks home page.

Working with the Databricks Integrated Workspace

After you log in, you will be presented with the Databricks Integrated Workspace. It is a starting point for all the things you can do with Databricks. The Databricks Integrated Workspace is shown in the following screenshot:

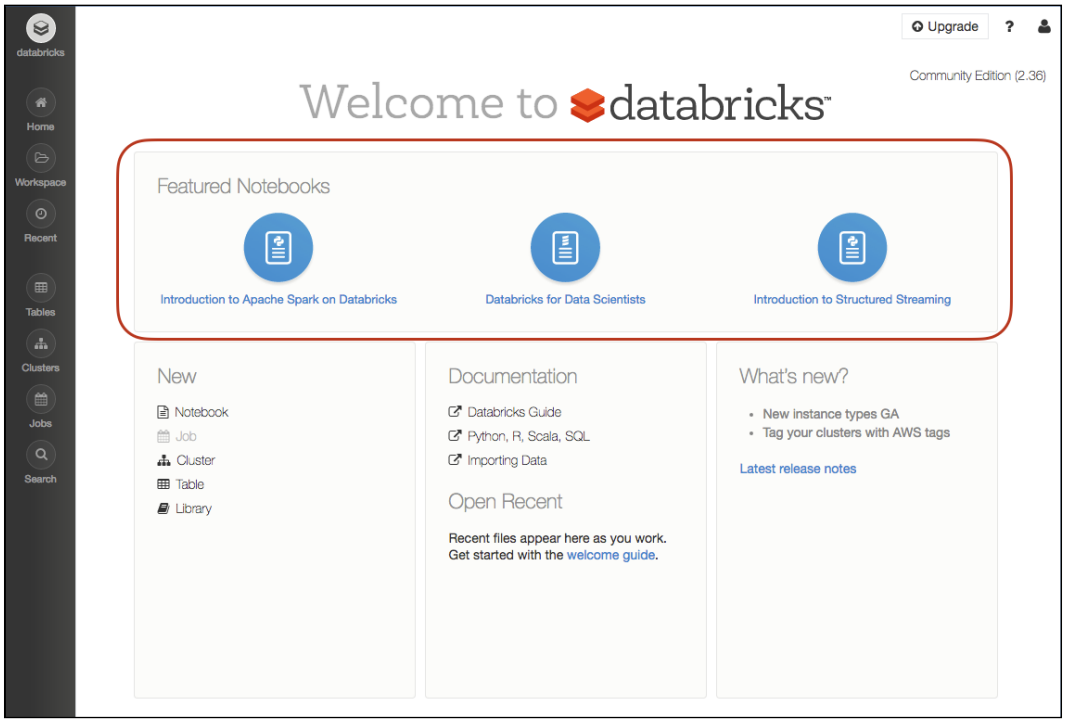


Starting on the left-hand side of Databricks, you have the left-hand navigation bar that allows you to:

- **databricks**: Go back to this main page
- **Home**: Go to your primary workspace – the folder that contains your own items
- **Workspace**: Go to the workspace that you were working in. For example, if you were working with a notebook in a shared folder, clicking on **Workspace** would go to the notebook in that folder, while **Home** would go back to your own personal workspace

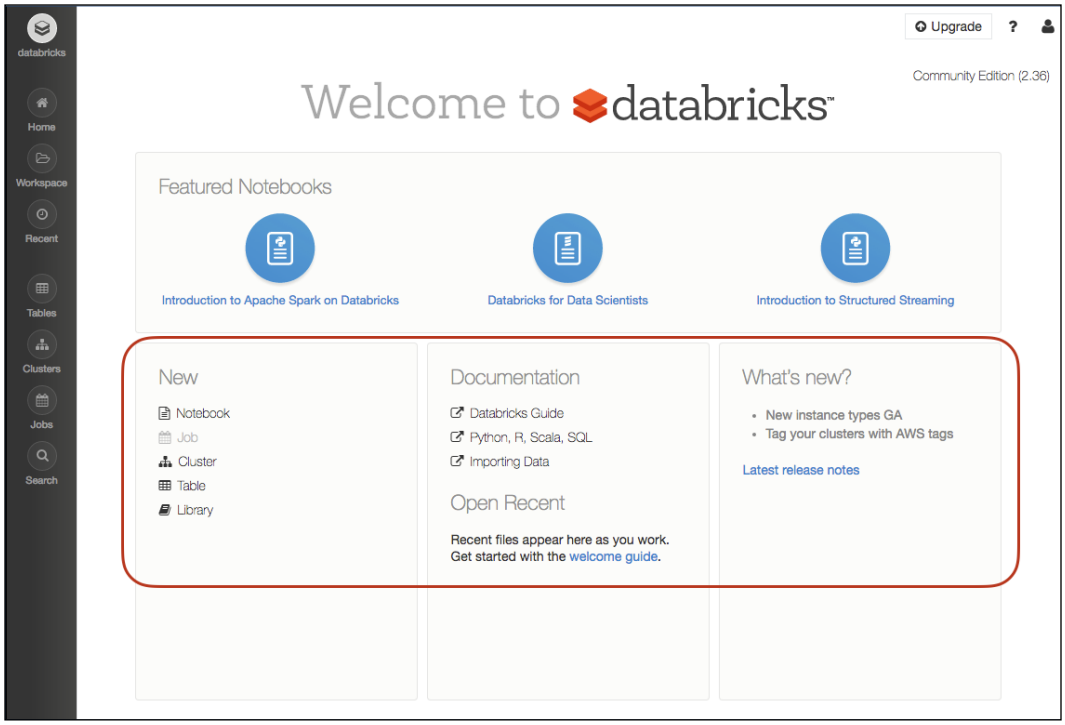
- **Recent**: Review the most recent notebooks that you had opened
- **Tables**: Access any of the tables that you had created
- **Clusters**: Gives you access to the Databricks Cluster Manager to quickly launch, expand, and/or terminate your clusters
- **Jobs**: Easy access to manage your scheduled jobs (available in paid edition only)
- **Search**: Quickly find your notebooks using this handy search feature

The **Featured Notebooks** section in Databricks provides notebooks typically showcasing the latest Spark features, such as (at the time of writing) an **Introduction to Structured Streaming**:



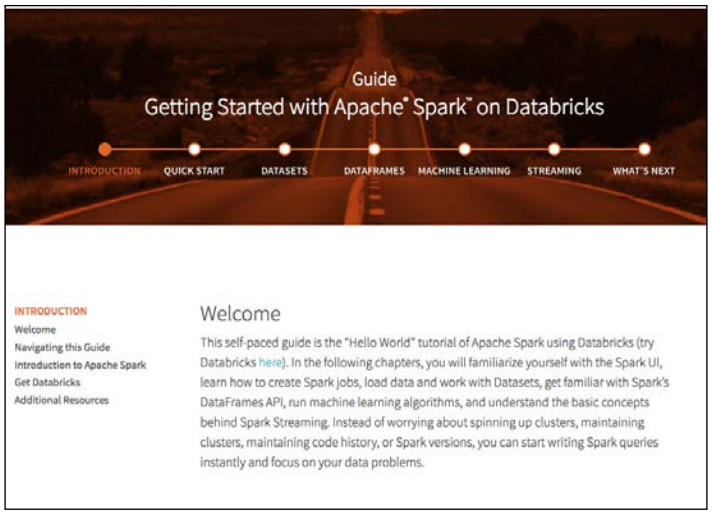
To access any of these notebooks, click on the notebook and it will immediately bring it up and you can execute it. You will learn more on how to work with these notebooks in the next section.

The lower center frame of the Databricks page provides you with easy access to create new items (for example, notebooks, clusters, tables, libraries, and so on), the latest documentation, and any new messages:



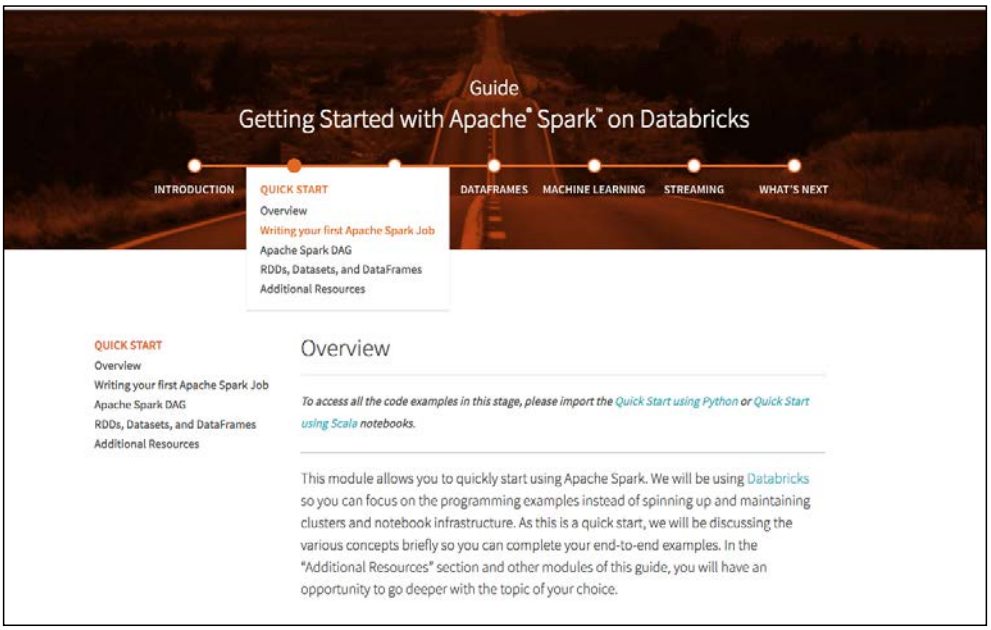
Follow the Getting Started with Apache Spark on Databricks guide

As you can see from the preceding screenshots, there are many different notebooks and documentation, such as the **Databricks Guide**, that you can utilize and refer to. To help you get started from scratch, there is the handy **Getting Started with Apache Spark on Databricks** guide available at: `https://databricks.com/product/getting-started-guide/quick-start` (as shown in the following screenshot):



As noted in the description, this guide is the *Hello World* tutorial of Apache Spark using Databricks. It contains multiple stages including a Quick Start that explains how to quickly start using Apache Spark and separate modules for Datasets, DataFrames, Machine Learning, and Streaming. These are self-contained modules, so you can follow this guide in whichever order you would like to focus on.

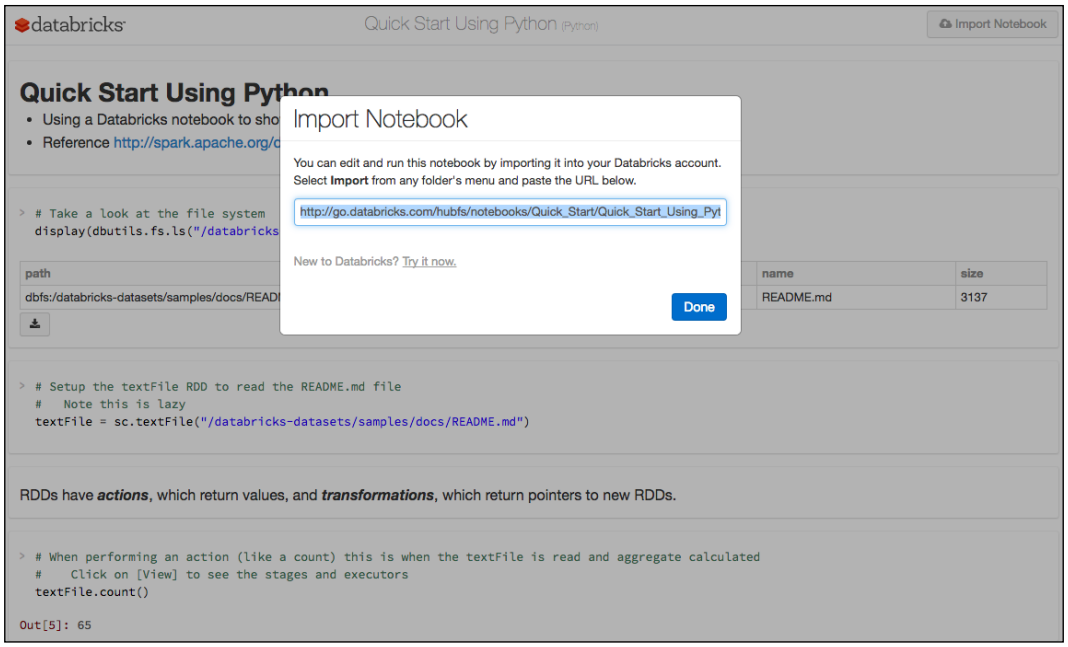
For example, to get to **Writing your first Apache Spark Job**, you can hover over **Quick Start** and jump to the section, as shown in the following screenshot:



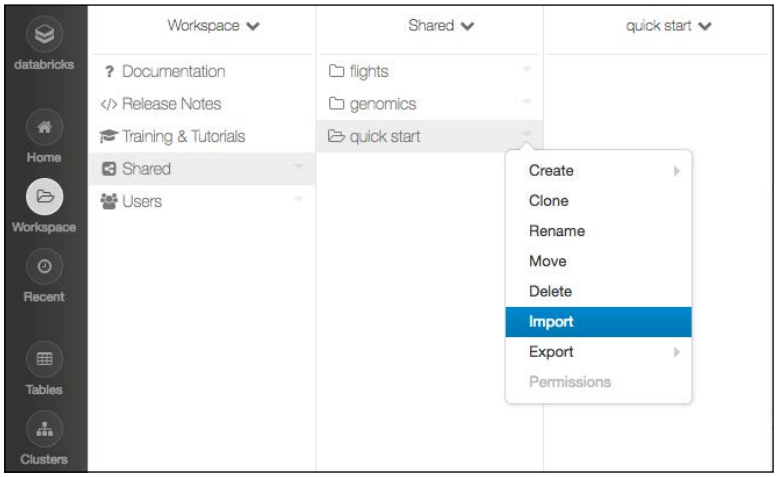
You can access the code examples directly by importing the **Quick Start using Python** notebook. To do this, click on the **Quick Start using Python** link (`http://go.databricks.com/hubfs/notebooks/Quick_Start/Quick_Start_Using_Python.html`) and you will get a view of the notebook.

As this is an `HTML` notebook, you can scroll through the notebook to view the code and results, but it is not currently active:

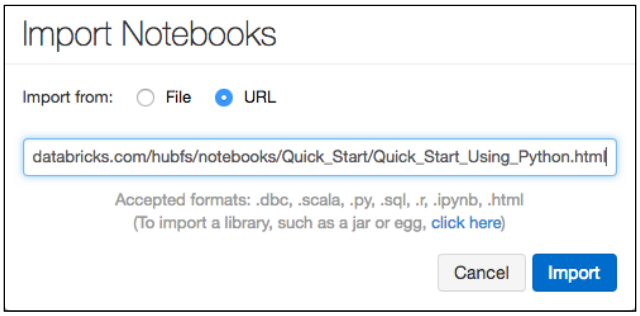
To make it active, click on the **Import Notebook** button at the top right and the **Import Notebook** dialog will appear (as shown in the following screenshot):



Copy the URL in the dialog box and go back to your Databricks Community Edition workspace. From here, go to any folder (in the following screenshot, we're using a `Shared` folder, but you can put this anywhere you want), right-click on the folder to activate the menu, and click on **Import**:



From here, the Databricks **Import Notebooks** dialog will appear. Click on the **URL** button and paste the URL you just copied from the original notebook:



Once you click **Import**, the notebook will be copied into your workspace in the folder you had specified so you can execute it.

Similar to the `HTML` page, you can scroll through the page and review the results. But, now that the notebook is in your workspace, you can execute the notebook against an Apache Spark cluster. Let's start exploring this notebook by double-clicking the top cell with the title **Quick Start with Python**.

Notice how the cell changes to edit mode so you can see the underlying Markdown code as indicated by the `%md` in the first line of the cell. A markdown cell follows most of the basic Markdown language syntax, allowing you to provide a cell dedicated to text, descriptions, and supplemental media such as images. Once you click outside of the markdown cell (for example, click another cell), the markdown cell will immediately resolve.

Next, let's click on the cell with the following code snippet:

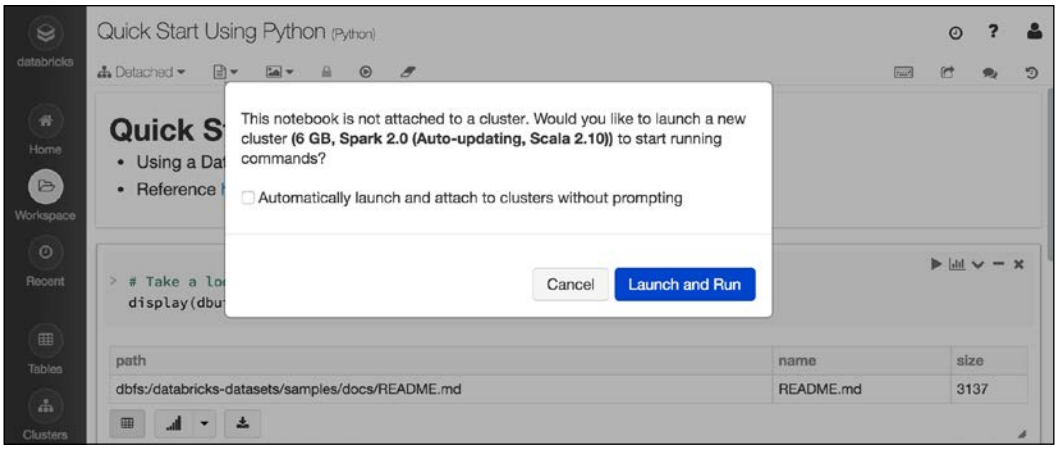
```
# Take a look at the file system
display(dbutils.fs.ls("/databricks-datasets/samples/docs/"))
```

This is PySpark code that uses the Databricks commands `display` and `dbtils.fs.ls`. The `display` command is a powerful command that converts Spark DataFrames into native visualizations (for example, formatted tables, bar charts, maps, and so on) as well as visualizes various Spark ML algorithms. The `dbutils.fs.ls` command is basically a `ls` command for any native `DBFS` (Databricks File System) or AWS `S3` mounts to your cluster. The execution of this code means you want to have a formatted table view of the files in the `/databricks-datasets/samples/docs/` folder.

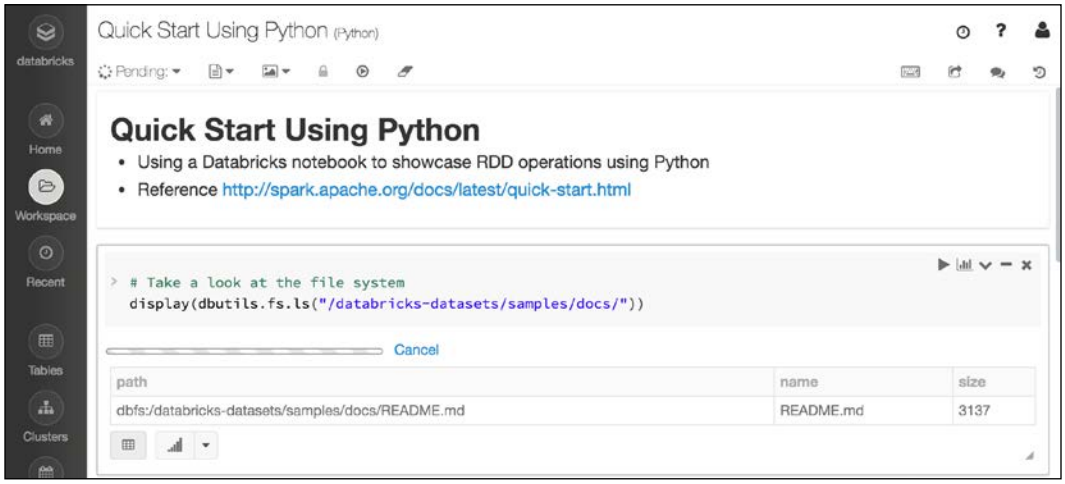
> While you can import your own data into Databricks, to help you get quickly started there is a wide variety of datasets available in the `/databricks-datasets` mount that you can work with. To import your own data, please follow the Databricks *Data Import How-To Guide* at `https://databricks.com/wp-content/uploads/2015/08/Databricks-how-to-data-import.pdf`.

To execute this command, you can either click the play button located in the top right of the cell or using your keyboard, click `<shift><enter>`:

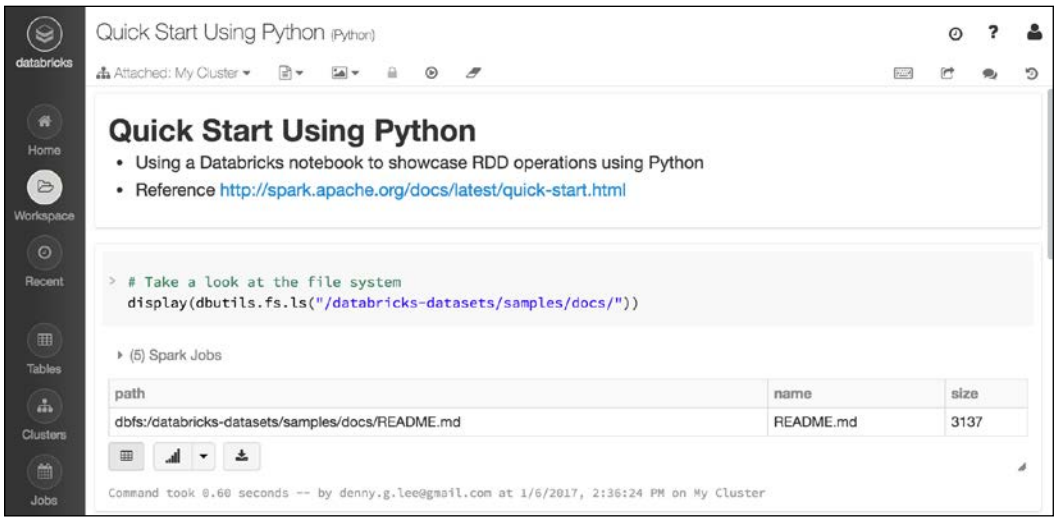
But wait, we forgot to start a Spark cluster so we can execute our notebook! That's okay, the Databricks workspace includes the rather nifty feature that automatically launches a cluster and attaches your notebook to it:



In this example, Databricks is automatically starting a Spark 2.0 (Scala 2.10) mini-cluster. If you want to start a cluster with a different version of Apache Spark, click **Cancel**, go to the Cluster Manager on the left-hand navigation so you can customize your settings if you are ok with the cluster configuration, and then click **Launch and Run** and a cluster will be created in the background, as shown in the following screenshot (notice the pending state in the upper left):



Once the cluster has been created, the pending state will switch to the cluster name (in this case, it is **My Cluster**) and the first cell (the `display` and `dbutils.fs.ls` commands) will automatically execute, as shown in the following screenshot:



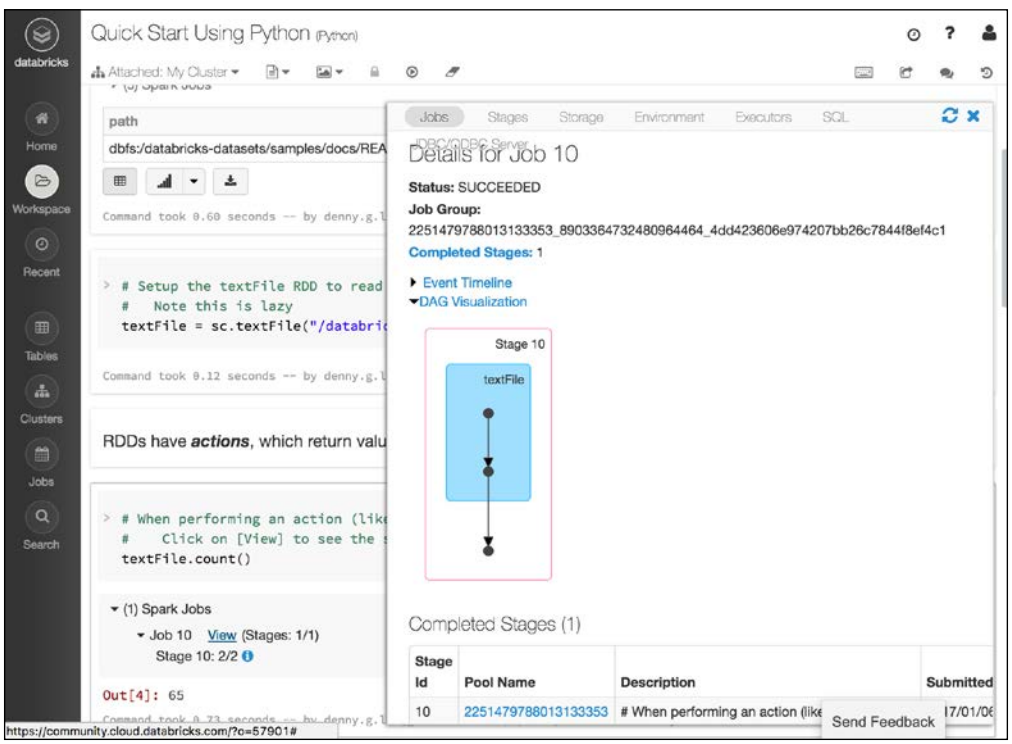
From this point onwards, you can execute the next few cells (either via the **Run** button or by typing `<shift><enter>` in the cell) to run the following commands:

```
# Setup the textFile RDD to read the README.md file
#   Note this is lazy
textFile = sc.textFile("/databricks-datasets/samples/docs/README.md")

# When performing an action (like a count) this is when the textFile
is read and aggregate calculated
#    Click on [View] to see the stages and executors
textFile.count()
```

As noted in the text within the notebook (as well as in the code comments), this is a simple row count example, where the first command is an RDD transformation to create the `textFile` RDD by reading the `README.md` file in the `/databricks-datasets/samples/docs` folder. The second action performs an RDD action to execute the row count.

As shown in the following screenshot, upon executing the action, a `Spark Jobs` dialog appears, which provides the real-time progress of the jobs and associated stages executed to complete the `textFile.count()`. Once you click **View**, you will see the Spark UI DAG of jobs and stages embedded directly in your notebook, so you can easily debug your Spark job:



Next steps

And, just like that, you have executed your first Spark job within Databricks. Continue working with this notebook and try out other datasets within the `/databricks-datasets` folder. Don't forget, your notebook has revision history, so if you make a mistake and need to revert to an older version of the notebook, just click on **Revision History** and restore the version you want.

Other great resources include:

- *Getting Started with Apache Spark on Databricks Guide*: `https://databricks.com/product/getting-started-guide/quick-start`
- Databricks guide *Getting Started*: `https://docs.databricks.com/user-guide/getting-started.html`
- *Introduction to Databricks* [Video]: `https://vimeo.com/130273206`
- *Databricks Cluster Manager and Jobs* [Video]: `https://vimeo.com/156886719`
- *Data Visualizations in Databricks* [Video]: `https://vimeo.com/156886721`
- *Collaboration in Databricks* [Video]: `https://vimeo.com/156886720`
- *Data Exploration in Databricks* [Video]: `https://vimeo.com/137874931`

Using HDInsight on Microsoft Azure

With Databricks' Spark notebooks you get the most recent incarnations of Spark as they are unveiled and pass the beta phase. However, Microsoft's HDInsight product offers plenty of innovation as well, at the expense of some lead-time to get the latest Spark release.

With the Spark for Azure HDInsight product, you get the Jupyter notebooks preinstalled, as well as everything that the Anaconda distribution of Python has to offer. Also, with the recent purchase of Revolution Analytics by Microsoft, HDInsight integrates R Server, exposing the largest R-compatible parallel analytics and machine-learning library. In addition, HDInsight API allows your apps to connect to Azure Data Lake Store which, in turn, lets you store trillions of files, each of which can be petabytes in size.

> Read more about the Azure *Data Lake Store* here `https://azure.microsoft.com/en-us/services/data-lake-store/`.

The free options on HDInsight

When signing up for Spark on Azure HDInsight, the offers are different depending on whether you are a simple Joe, a startup, a student or an academic.

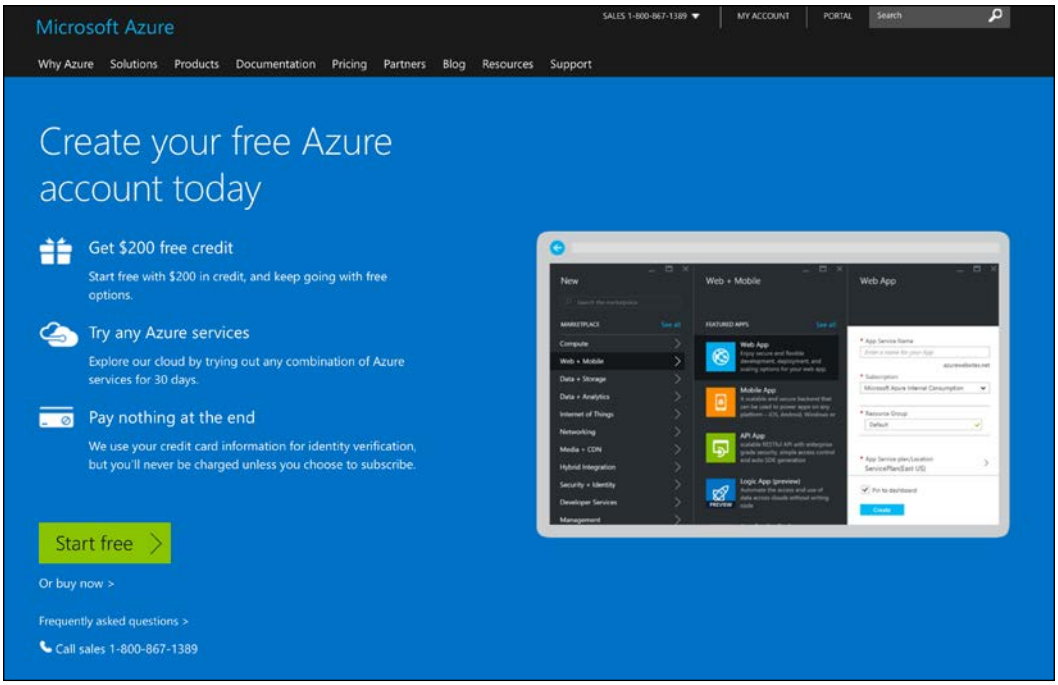
As an individual, when you sign up, you get $200 credit towards services you use and 30-day trial of any Azure services. Even though you will need to provide credit card information it will not be charged at the end of your trial should you not decide to convert to a paid offer at the end of the 30-day period.

For startups, Microsoft offers $150 a month of free Azure cloud services and free software (such as the Visual Studio and Office package). The offer is available to tech companies that are no older than five years and make less than $1 million.

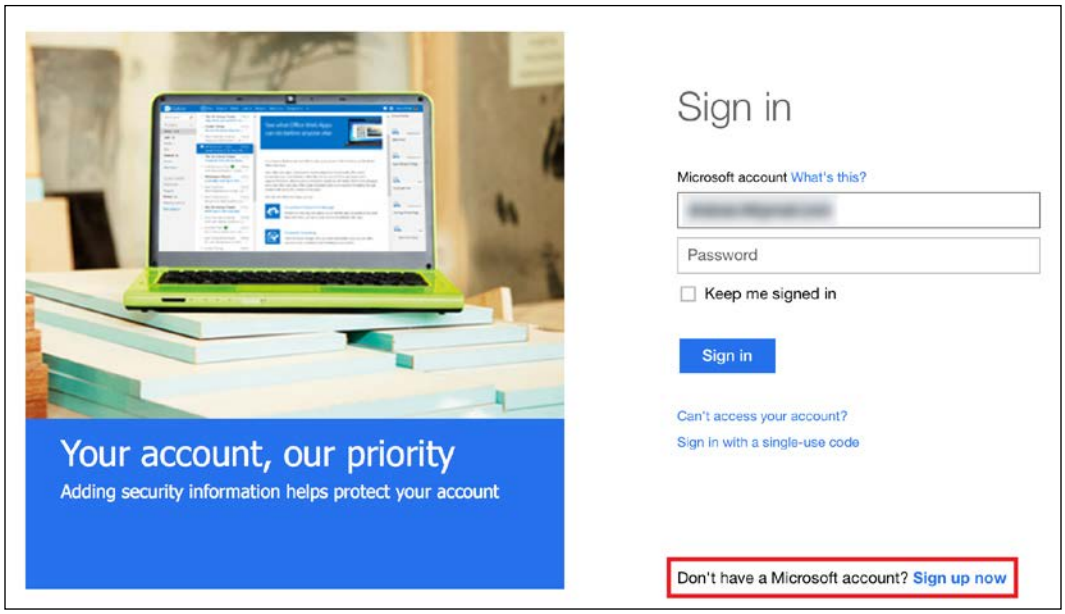
If you are a student or an educator, you can go to `https://www.microsoftazurepass.com/azureu` and access the free services through that portal (as a student) or apply for free credits (if you are a researcher).

Signing up for the service

Let's finally sign up for the service. Go to `https://azure.microsoft.com/en-us/services/hdinsight/` and click on the **FREE ACCOUNT** link. You should see a screen similar to the following:



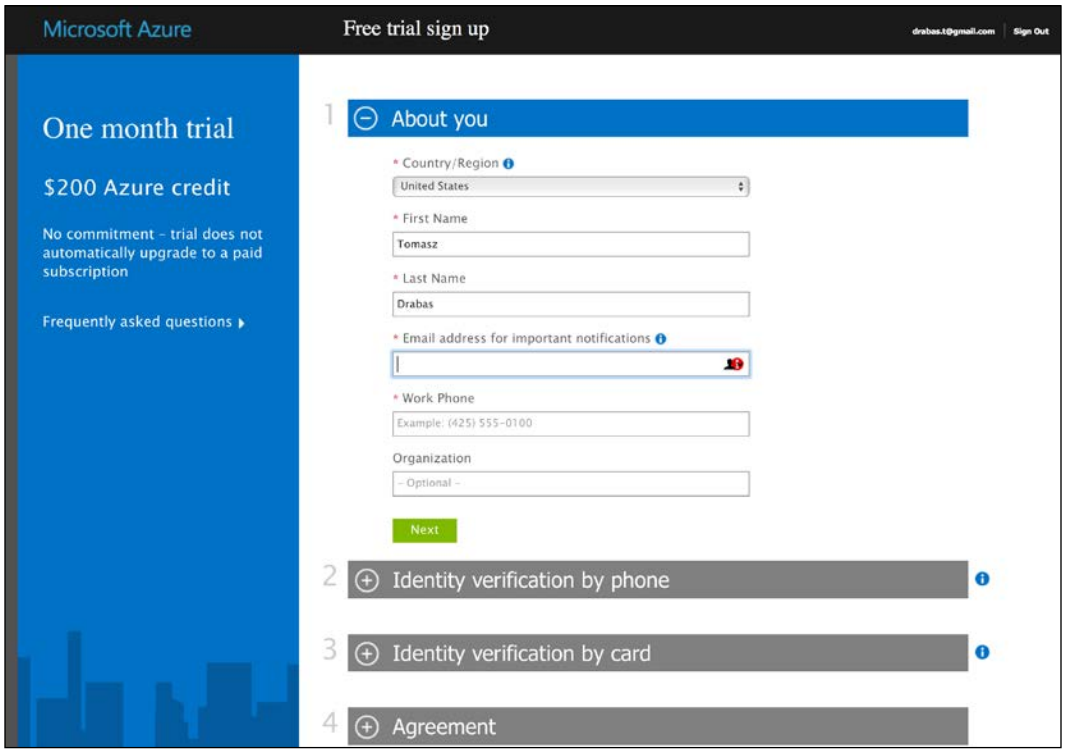
After clicking on the **Start free** button you will be taken to the sign-in page. You need to sign in with your Microsoft Account, that is, an account you registered with Microsoft. If you do not have a Microsoft Account there's a link you can use to create one (see the link highlighted in the following screenshot):



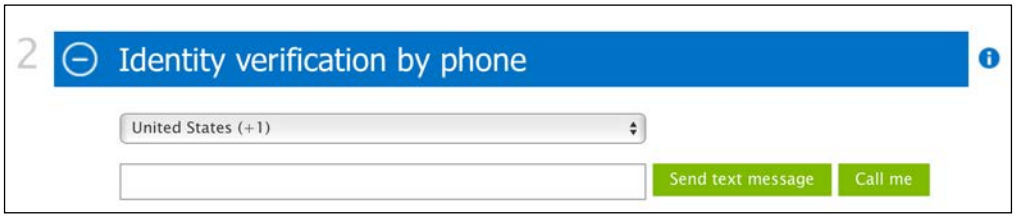
Once you sign in, you should be taken to a page that will allow you to sign up for the service.

> As you can see in the following screenshot, even though you provide the credit card details, your card will not be charged unless you specifically transition to a paid option.

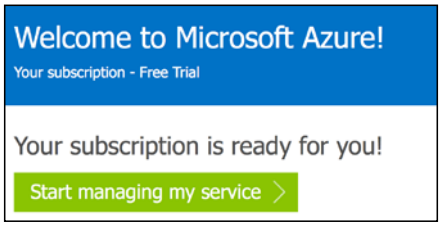
First, you need to fill in some personal information:



Once you have filled it in, you will go through a two-factor verification: first, you need to provide a phone number that the company will either call you on or send a text message to with a verification code:

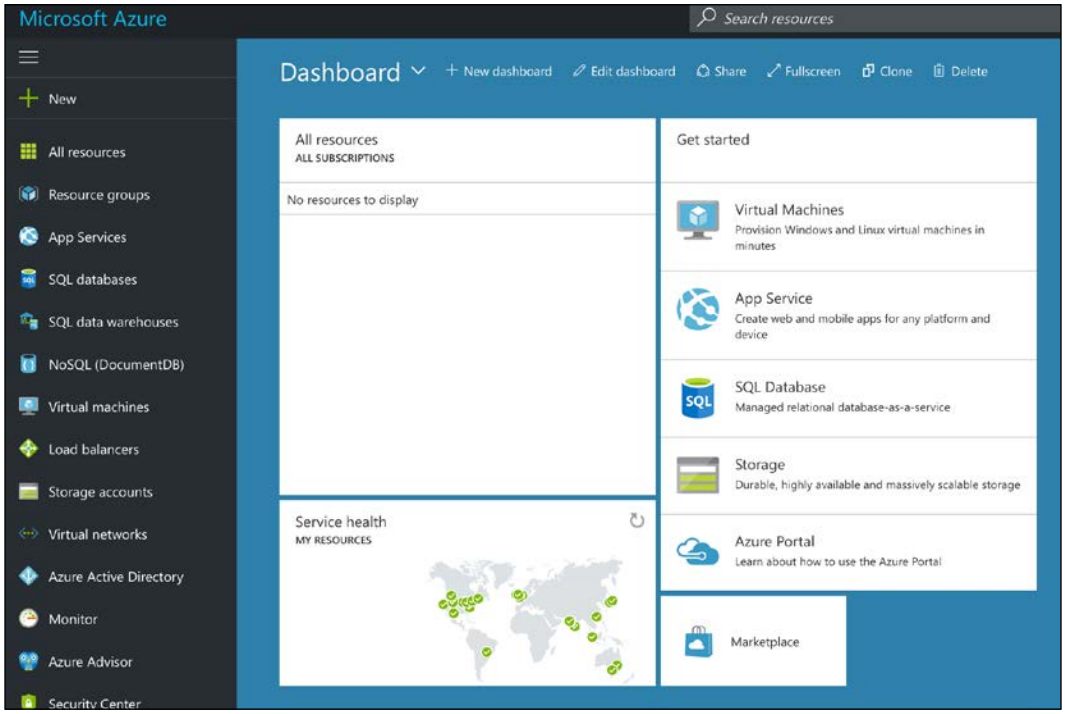


Once you enter the verification code you will be taken to the next screen that will prompt you for the credit card details. After entering the information you will be taken to the last step, which is acknowledging the subscription agreement. Once you sign up you should see the following message; upon clicking on **Start managing my service** you will be taken to the Azure Dashboard:



Microsoft Azure dashboard

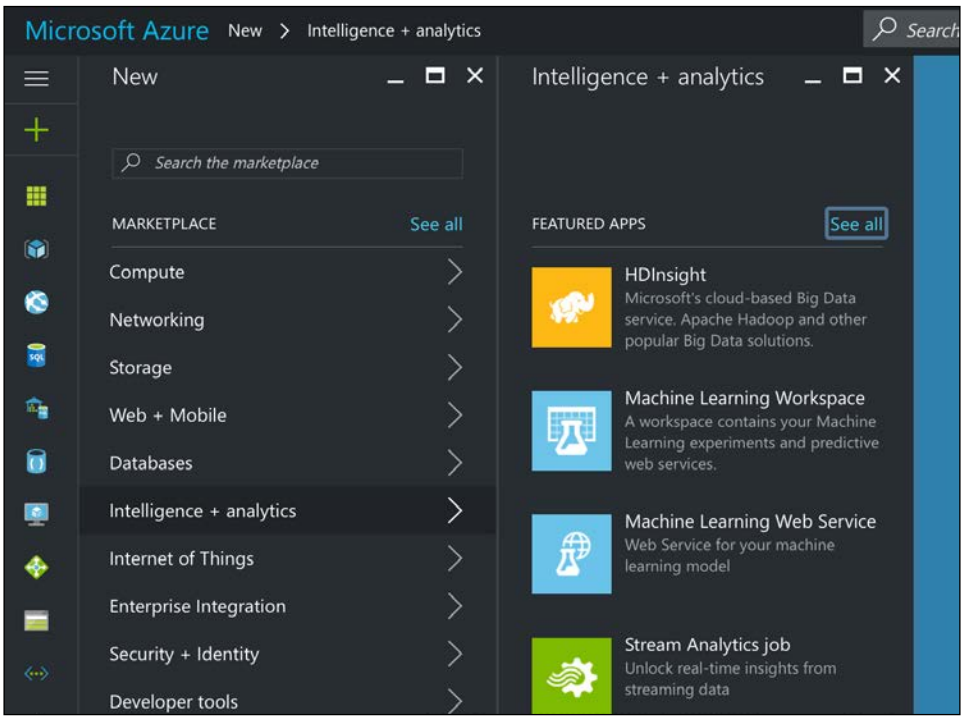
The Azure Dashboard is a one-stop shop for all things Azure: if you want to set up a new cluster, add a storage account or scale up/down your cluster - this is where you do it:



On the left-hand side of the screen you have the full palette of services you can set up. In the middle, you will be shown all the resources available in your subscription; for now there is nothing, but this will change soon.

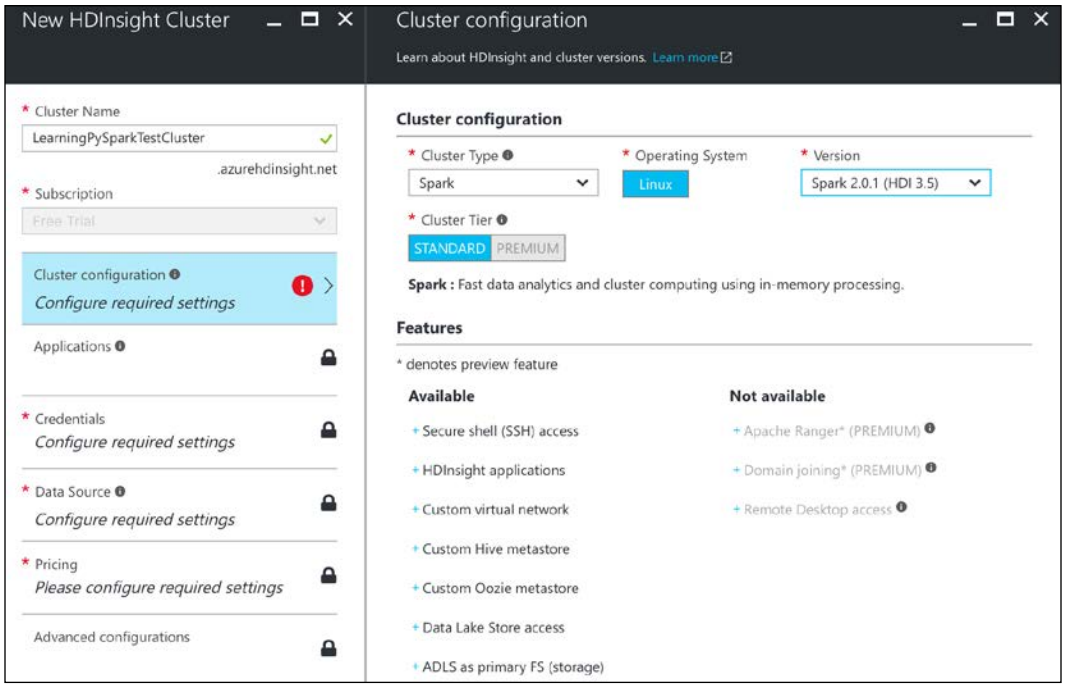
Setting up an HDInsight Spark cluster

Right now, we do not have any cluster running within our subscription, hence we did not see any resources on our dashboard. Let's finally create our Spark cluster. Once you click on the **New** button you will be shown the following options. You can either type `HDInsight` in the search box, or scroll to the **Intelligence + analytics** option:



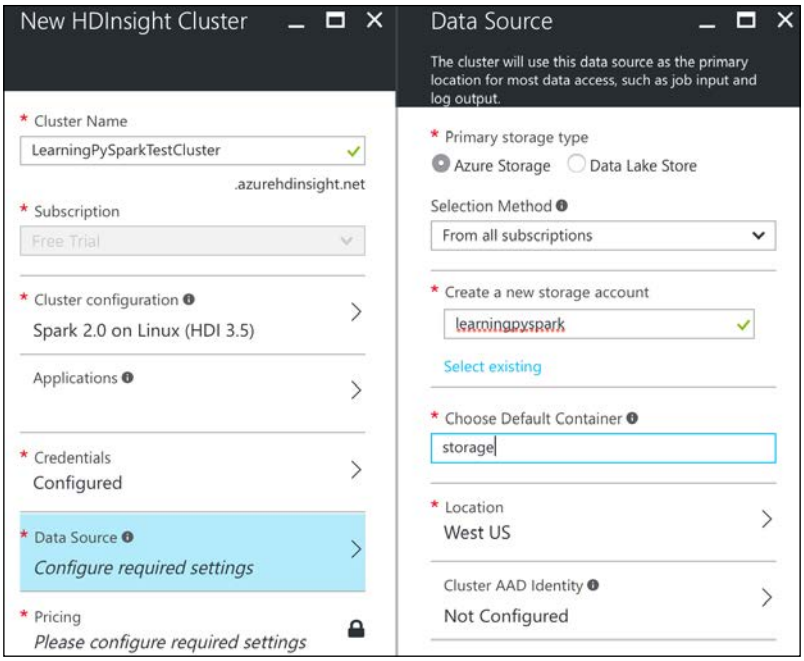
Once you select the **HDInsight** app, we can start configuring the cluster. First, create a name for your cluster.

> Note that you cannot use the same name that we present in the following example.



Next, move to the **Cluster Configuration**. From the **Cluster Type** dropdown, select *Spark*, the **Version** should be *Spark 2.x.y* (where *x* and *y* denote the latest version of Spark), and the **Cluster Tier** we will keep as **STANDARD**. Click on **Select**, or move to the **Credentials** on the left-hand side.

There, you need to create the logins for accessing the cluster or connecting to the cluster via SSH; create these as you please. Once created, you will be taken to the **Data Source** configuration:

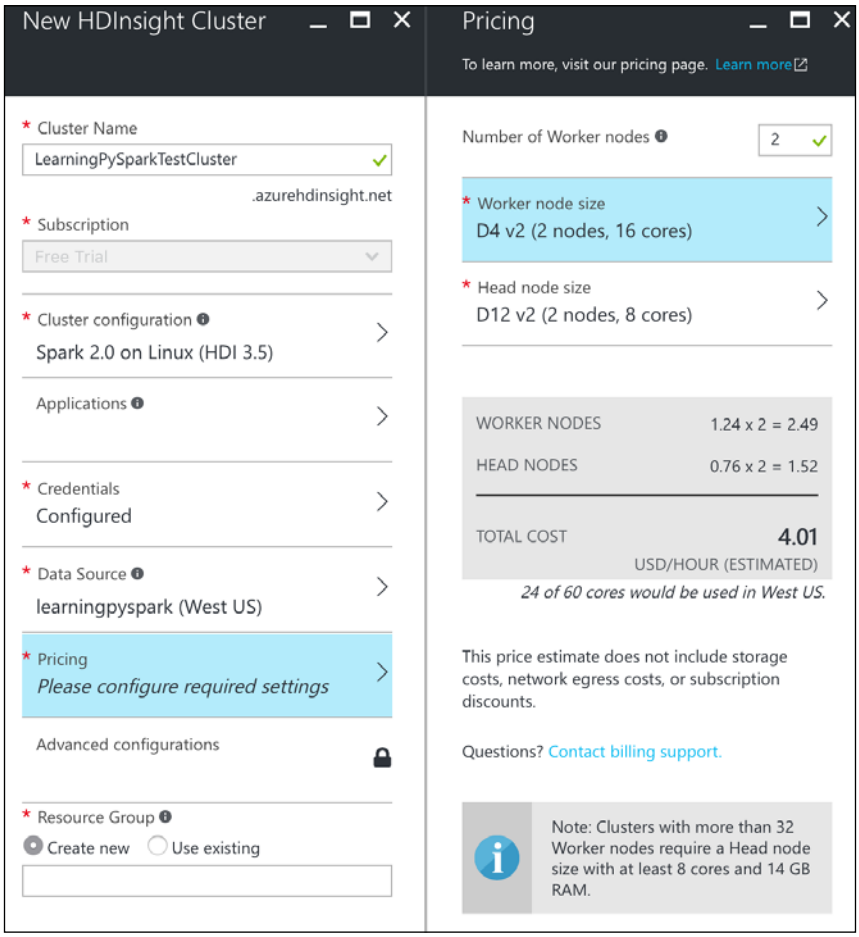


In this example, we will stick with the defaults; that is, we will use **Azure Storage**. We will **Create a new storage account**.

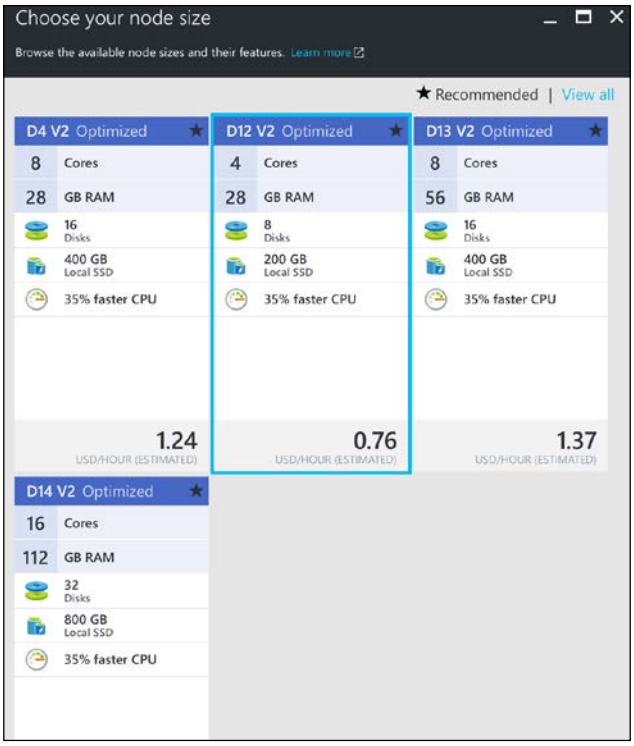
> Note that, as with the cluster name, you cannot reuse the name as presented in the preceding screenshot.

The **Choose Default Container** option should be treated as a default name for the folder to store your data in; this name you can reuse.

Once you have configured your storage, you will be presented with the **Pricing** page:



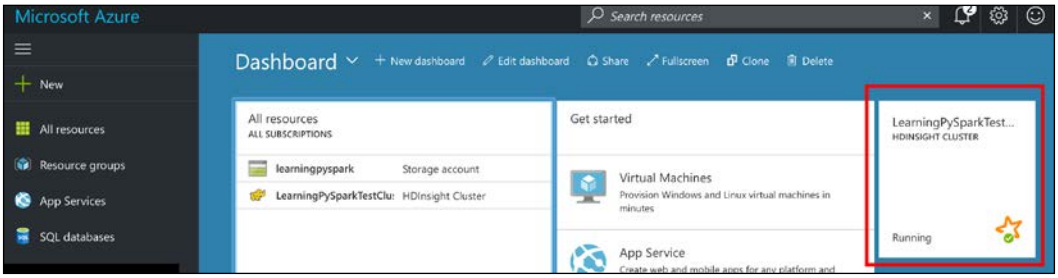
The options normally default to four workers and two head nodes; in our example, we will use only two workers. Note that you can select from other machine configurations with different price tags:



Having chosen your pricing tier, all that is left is to create a new resource group and create our cluster.

> For convenience, check the **Pin to dashboard** checkbox; this will help with accessing your cluster, as it will show every time you log in to your Azure dashboard.

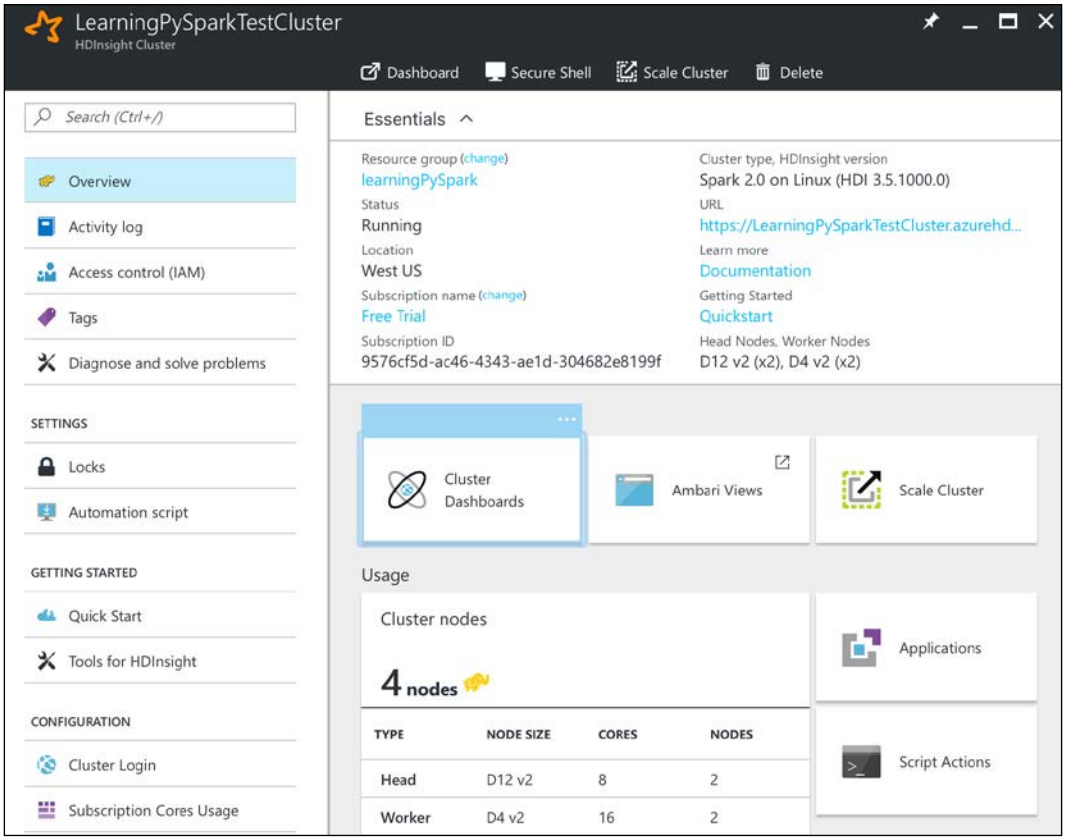
Once created (it might take anything between 10-30 minutes), you will see a screen similar to the following:



Notice the running HDInsight cluster? We're ready to go!

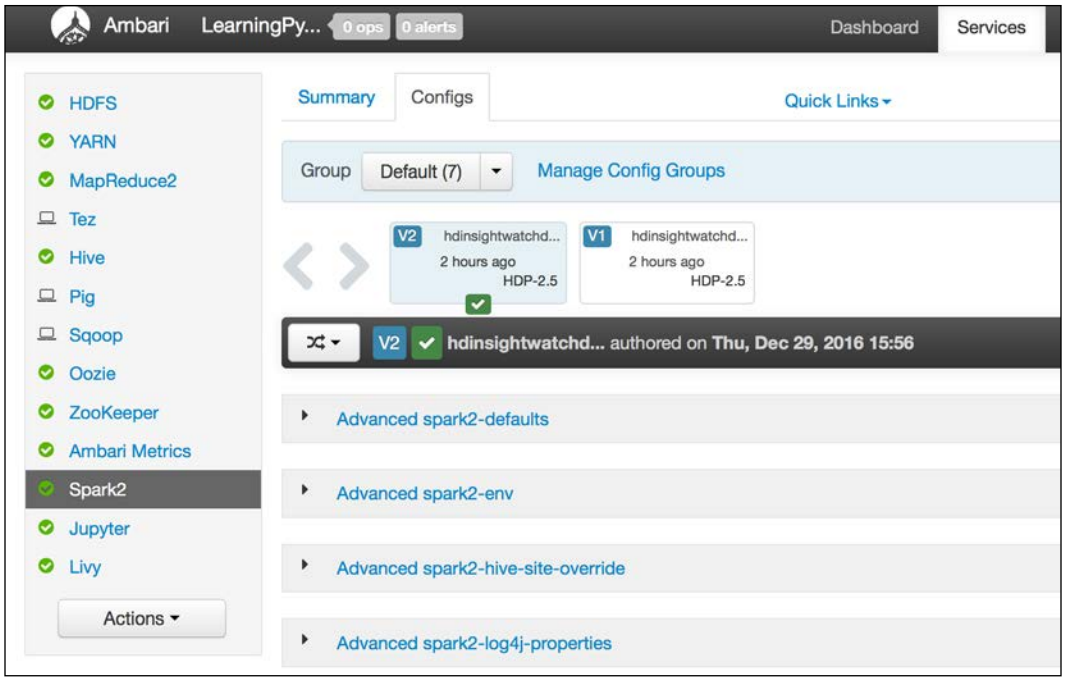
Running Spark code

Once you click on the tile, you will be taken to the cluster overview page:



Here, you can learn everything about your cluster: its **Status**, **Location**, **Subscription name**, and **Cluster type, HDInsight version**. It also shows how many machines there are in your cluster in the **Usage** section. From here, you can scale your cluster and access all the features of Spark. If you click on the **Cluster Dashboards**, you will be shown the following options:

- **HDInsight Cluster Dashboard**: It takes you to the Ambari view of your cluster. Here, you can change your cluster configuration at the very granular level. For example, you get the access to all of the Spark / HDInsight configuration options:



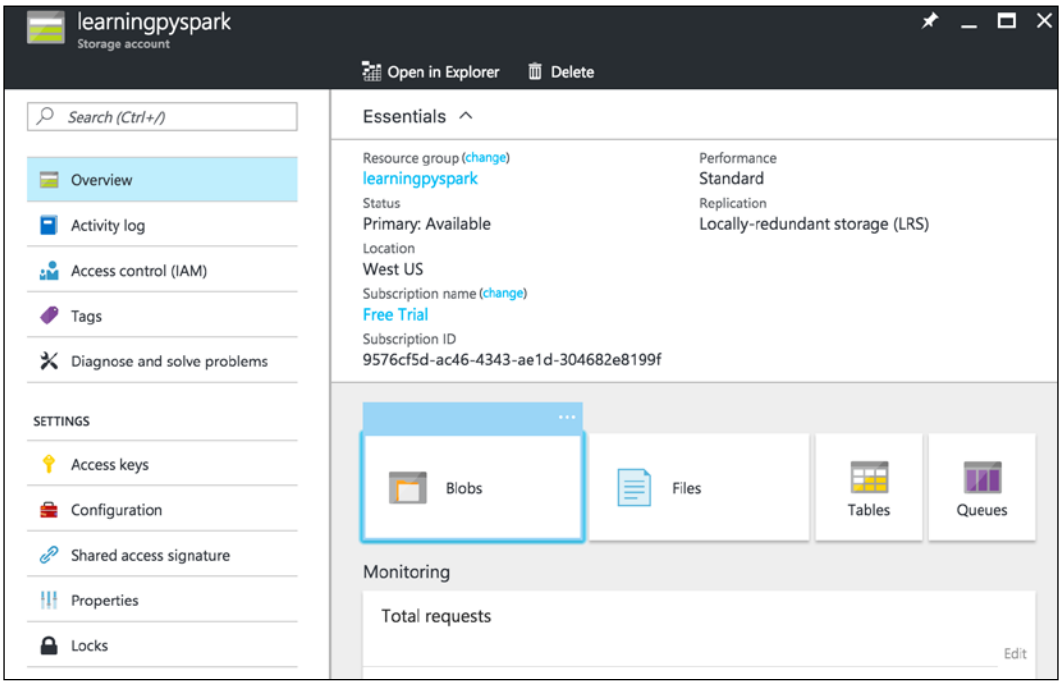
For an overview of all the Spark options, refer to Spark's documentation `http://spark.apache.org/docs/latest/configuration.html`.

- **Jupyter Notebook** is the place we will spend most of our time as it brings us to the Jupyter main page for our cluster
- **Spark History Server** takes you to the logs for applications
- **Yarn** is a scheduler of jobs. We will use it to monitor our jobs

Running any Spark job, however, without data makes little sense. Thus, let's move the data we used in *Chapter 6, Introducing the ML Package*, over to the Azure Data Storage.

Managing data

When we created the Spark cluster, we also created a storage account. If you go to your Microsoft Azure Dashboard main view (just click on **Microsoft Azure** in the top left corner of the page), under the **All resources** tile, you should now see two options: the **HDInsight cluster** and the **Storage account**. After clicking on the **Storage account** option, you should see a view similar to the following screenshot:



Clicking on the **Blobs** tile will get you to the blob storage. The blob storage is an offering that is completely data agnostic: it can store data in literally any format, such as text, CSV, parquet, or JSON (to name just a few); the data can be structured or unstructured.

Click on the container name in the subsequent window to get to a long list of files already present in your container. On top, click on **Upload**, select the file, and click on the **Upload** button on the bottom of the tab.

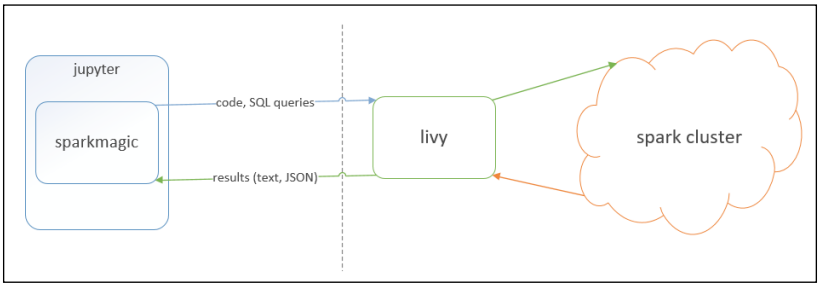
Now, let's run some code.

Configuring your session

Before we do that, however, we need to configure our session: right now we have two workers, each with 16 cores and 28 GB of RAM so we can fine-tune the job to the data better. First, navigate to your dashboard, click on your **HDInsight Cluster**, then **Cluster Dashboards**, and finally on **Jupyter Notebook**. This will open the main screen for Jupyter notebooks. You should see two folders: PySpark and Scala.

Go ahead and create a new folder by clicking on **New** and selecting **Folder**. Next, create a new PySpark notebook inside the newly created folder.

HDInsight uses **sparkmagic**, which communicates with your Spark cluster through **livy**. Livy is Spark's REST server that allows you to communicate with your cluster from anywhere in the world. **sparkmagic** exposes a host of, so called, magic: a set of commands to simplify interacting with Spark. At the general level, the communication looks similar to what is shown in the following diagram (source: `http://bit.ly/2hDNCY0`):



First, let's configure our Spark session. In order to do so, inside your notebook, type the following:

```
%%configure -f
{
    "name": "learningPySpark_Example",
    "numExecutors": 2,
    "executorCores": 4,
    "executorMemory": "2GB"
}
```

- The `%%` indicates the magic: In this case it is the `configure` command. The `-f` flag will force drop the session if it has already been created and will create a new one. The configuration string is JSON-formatted; you can pass any of the following most-used commands (sorted in a somewhat arbitrary order from what we deem most to least likely used):`name`: The name of your application `numExecutors`: Number of executors `executorCores`:

Number of executor cores `executorMemory`: Amount of memory requested for the application `pyFiles`: List of other Python files to be used during the session, can be `single.py` file(s) (separated by commas) or `.egg/.zip` whole modules `kind`: A kind of the kernel to use, can be `pyspark`, `pyspark3`, `spark`, or `sparkr`; this parameter is automatically passed by the Jupyter notebook when you run the `%%configure` so you do not have to specify this `driverMemory`: How much memory to reserve on the driver `driverCores`: How many cores to use on the driver `heartbeatTimeoutInSecond`: Indicates the longest interval (in seconds) between heartbeat communication to/from the Spark server.

> The remaining parameters are less likely used and you can look them up online if required.

Now that we have our session configured, you should see a similar output from running the code:

```
Current session configs: {u'executorCores': 4, u'numExecutors': 2, u'executorMemory': u'2GB', u'name':
u'learningPySpark_Example', u'kind': 'pyspark'}

No active sessions.
```

Other magic words include `%%help`, which will show all the commands that you can run:

Magic	Example	Explanation
info	%%info	Outputs session information for the current Livy endpoint.
cleanup	%%cleanup -f	Deletes all sessions for the current Livy endpoint, including this notebook's session. The force flag is mandatory.
delete	%%delete -f -s 0	Deletes a session by number for the current Livy endpoint. Cannot delete this kernel's session.
logs	%%logs	Outputs the current session's Livy logs.
configure	%%configure -f {"executorMemory": "1000M", "executorCores": 4}	Configure the session creation parameters. The force flag is mandatory if a session has already been created and the session will be dropped and recreated. Look at Livy's POST /sessions Request Body for a list of valid parameters. Parameters must be passed in as a JSON string.
sql	%%sql -o tables -q SHOW TABLES	Executes a SQL query against the variable sqlContext (Spark v1.x) or spark (Spark v2.x). Parameters: • -o VAR_NAME: The result of the query will be available in the %%local Python context as a Pandas dataframe. • -q: The magic will return None instead of the dataframe (no visualization). • -m METHOD: Sample method, either `take` or `sample`. • -n MAXROWS: The maximum number of rows of a SQL query that will be pulled from Livy to Jupyter. If this number is negative, then the number of rows will be unlimited. • -r FRACTION: Fraction used for sampling.
local	%%local a = 1	All the code in subsequent lines will be executed locally. Code must be valid Python code.

We will try some of them soon.

Running code

Having the session configured, let's start it and import our data from the storage account. Run the following code in your notebook (and substitute your names in place of the `storage` and `container`):

```
storage = 'learningpyspark'
container = 'storage'
f = 'births_transformed.csv.gz'

conn = 'wasb://{0}@{1}.blob.core.windows.net/{2}'.format(
    container,
    server,
    f
)

births = spark.read.csv(
    conn,
    header=True, inferSchema=True)
```

The string for connecting to the Azure Blob Storage has the following format: `wasb[s]://<container_name>@<storage_account_name>.blob. core.windows.net/<path>`.

Running the preceding code should create the session and load the data into our notebook. You should see something similar to the following:

```
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
5	application_1483055828481_0010	pyspark	idle	Link	Link	✔

```
SparkSession available as 'spark'.
```

Now that we have loaded our data, we can run some simple code. Let's aggregate the data by `BIRTH_PLACE` in a way we already know:

```
for col in births \
        .groupby('BIRTH_PLACE') \
        .count() \
        .sort('BIRTH_PLACE') \
        .collect():
    print(col)
```

The preceding code produces the following output:

```
Row(BIRTH_PLACE=1, count=44558)
Row(BIRTH_PLACE=2, count=136)
Row(BIRTH_PLACE=3, count=224)
Row(BIRTH_PLACE=4, count=327)
Row(BIRTH_PLACE=5, count=74)
Row(BIRTH_PLACE=6, count=11)
Row(BIRTH_PLACE=7, count=91)
Row(BIRTH_PLACE=9, count=8)
```

The same (and more) can be attained with the `%%sql` magic:

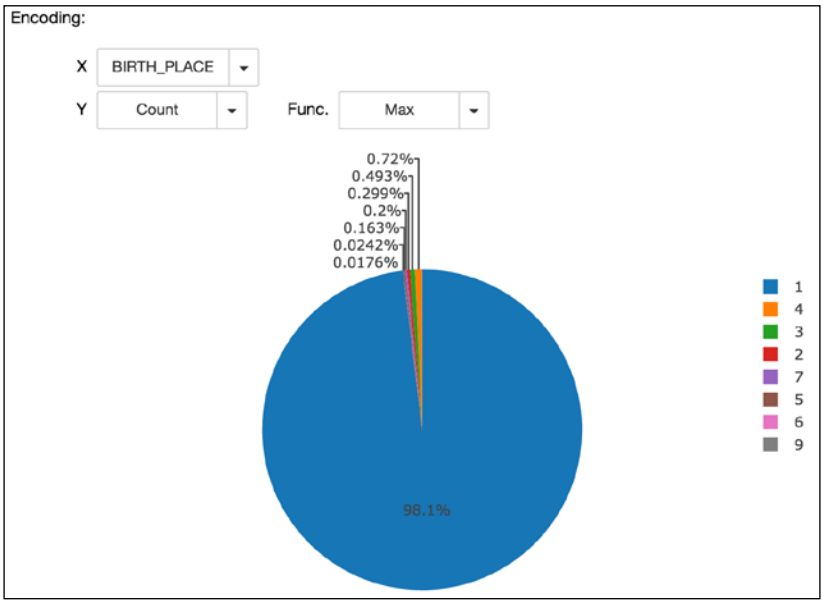
```
%%sql -o birthPlace
SELECT BIRTH_PLACE,
    COUNT(*) AS Count
FROM births_sql
GROUP BY BIRTH_PLACE
ORDER BY BIRTH_PLACE
```

What follows the `%%sql` is a pure ANSI SQL syntax that aggregates the data at the **BIRTH_PLACE** level and produces the following, nicely formatted table:

BIRTH_PLACE	Count
1	44558
2	136
3	224
4	327
5	74
6	11
7	91
9	8

The output, however, can easily be changed at a click of a button to either of these options:

This is what a pie chart looks like:



The `-o` flag we used in the `%%sql` statement instructs the magic to expose the result *locally* as a pandas DataFrame.

> Use the `-o` flag carefully. As stated previously, the results are exposed locally, that is, the data is moved back to the head node. If the result is big this will not be the most efficient way to look at your data.

Now you can access and play with the data in a more *pythonic* way:

```
%%local
birthPlace.head()
```

Note the `%%local` magic - you can now load, for example, matplotlib library and visualize your data that way.

The preceding code will produce a very similar output:

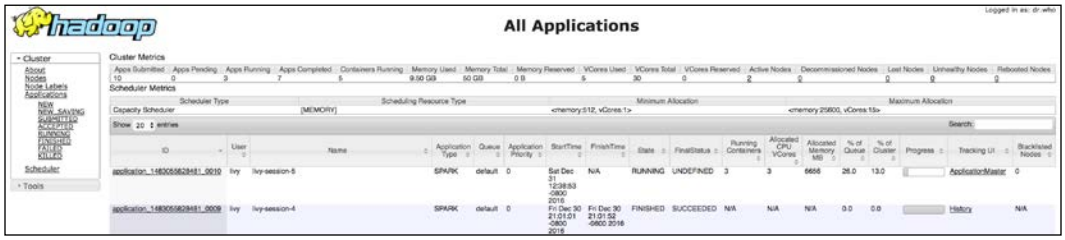
Type: Table Pie Scatter Line Area Bar

BIRTH_PLACE	Count
1	44558
6	11
3	224
5	74
9	8

So far we have played with Spark in our local machine so execution was with a single node only. Let's check how this works on the cluster.

Monitoring jobs execution with Yarn

First, open the Yarn UI (User Interface): go to the **Cluster Dashboards** on your HDInsight cluster, and click on **Yarn**. A window similar to the following should pop up:



It lists all the running and finished applications. Clicking on the **ApplicationMaster** under the **Tracking UI** column will open the execution log for your application:

Spark 2.0.0.2.5.2.1-1 | Jobs | Stages | Storage | Environment | Executors | SQL | livy-session-7 application UI

Spark Jobs (?)

User: yarn
Total Uptime: 8.0 min
Scheduling Mode: FIFO
Completed Jobs: 9

Event Timeline

Completed Jobs (9)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
8	runJob at PythonRDD.scala:441	2016/12/31 22:54:49	75 ms	1/1 (2 skipped)	4/4 (201 skipped)
7	runJob at PythonRDD.scala:441	2016/12/31 22:54:49	67 ms	1/1 (2 skipped)	4/4 (201 skipped)
6	runJob at PythonRDD.scala:441	2016/12/31 22:54:48	0.9 s	2/2 (1 skipped)	201/201 (1 skipped)
5	toJSON at NativeMethodAccessorImpl.java:-2	2016/12/31 22:54:47	0.8 s	2/2	201/201
4	collect at <stdin>:4	2016/12/31 22:54:43	0.6 s	2/2 (1 skipped)	209/209 (1 skipped)
3	collect at <stdin>:4	2016/12/31 22:54:41	2 s	2/2	201/201
2	csv at NativeMethodAccessorImpl.java:-2	2016/12/31 22:54:35	2 s	1/1	1/1
1	csv at NativeMethodAccessorImpl.java:-2	2016/12/31 22:54:35	0.1 s	1/1	1/1
0	csv at NativeMethodAccessorImpl.java:-2	2016/12/31 22:54:33	1 s	1/1	1/1

The view lists all the jobs completed during the execution of the app. You can see that some of the stages were skipped: this is because those stages were executed earlier and Spark was intelligent enough to recognize that and not run the same jobs multiple times.

You can also glimpse inside each job and see what stages were executed by what executors, and other interesting statistics for each job stage:

Spark 2.0.0.2.5.2.1-1 | Jobs | Stages | Storage | Environment | Executors | SQL | livy-session-7 application UI

Details for Stage 6 (Attempt 0)

Total Time Across All Tasks: 1.0 s
Locality Level Summary: Node local: 8; Process local: 192
Shuffle Read: 504.0 B / 8
Shuffle Write: 504.0 B / 8

DAG Visualization
Show Additional Metrics
Event Timeline

Summary Metrics for 200 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	1 ms	2 ms	3 ms	5 ms	27 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Shuffle Read Size / Records	0.0 B / 0	0.0 B / 0	0.0 B / 0	0.0 B / 0	63.0 B / 1
Shuffle Write Size / Records	0.0 B / 0	0.0 B / 0	0.0 B / 0	0.0 B / 0	63.0 B / 1

Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Succeeded Tasks	Shuffle Read Size / Records	Shuffle Write Size / Records
1	10.0.0.7:44793	2 s	150	0	150	504.0 B / 8	504.0 B / 8
2	10.0.0.6:46439	2 s	50	0	50	0.0 B / 0	0.0 B / 0

Tasks (200)

Page: 1 2 > 2 Pages. Jump to 1 . Show 100 items in a page. Go

Index ▲	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Shuffle Read Size / Records	Write Time	Shuffle Write Size / Records	Errors
0	208	0	SUCCESS	PROCESS_LOCAL	2 / 10.0.0.6	2016/12/31 22:54:43	24 ms		0.0 B / 0		0.0 B / 0	
1	209	0	SUCCESS	PROCESS_LOCAL	2 / 10.0.0.6	2016/12/31 22:54:43	11 ms		0.0 B / 0		0.0 B / 0	
2	210	0	SUCCESS	PROCESS_LOCAL	2 / 10.0.0.6	2016/12/31 22:54:43	15 ms		0.0 B / 0		0.0 B / 0	
3	211	0	SUCCESS	PROCESS_LOCAL	2 / 10.0.0.6	2016/12/31 22:54:43	9 ms		0.0 B / 0		0.0 B / 0	
4	216	0	SUCCESS	PROCESS_LOCAL	2 / 10.0.0.6	2016/12/31 22:54:43	8 ms		0.0 B / 0		0.0 B / 0	
5	217	0	SUCCESS	PROCESS_LOCAL	2 / 10.0.0.6	2016/12/31 22:54:43	7 ms		0.0 B / 0		0.0 B / 0	

As you can see, each task took between 1 ms to 27 ms to run with the median being 3 ms. GC Time is the Garbage Collection time. The view lists both of our executors (as requested), the time it took to run each task, and how many tasks failed and succeeded (among other things). At the following link, you can check information about each task execution: status, executor ID, when it was launched, and how long it took to execute.

If you want to learn more about the locality level you can read more here: `https://spark.apache.org/docs/latest/tuning.html#data-locality`.

Summary

In this chapter we provided you with an introduction to two great (and free) offerings of Spark in the cloud: the (always free) Databricks Community Edition, and Microsoft's HDInsight free 30-day trial offer. We presented how you can sign up, configure and get started with these two offers. This is by no means an exhaustive description of the offers but rather an introduction to get you started. Also, for space reasons, we did not cover other Spark offers from other players such as Google or Amazon. These, however, you can look up by following the links we provided at the beginning of the chapter.