TECHNICAL GUIDE

Migration Guide: Hadoop to Databricks

Data architecture modernization

Introduction

Hadoop is an ecosystem of open source software projects for distributed data storage and processing. Databricks is a cloud- and Apache Spark™–based big data analytics service generally available in Amazon Web Services (AWS), Google Cloud Platform (GCP) and Microsoft Azure. Databricks is the creator of Apache Spark, and Databricks is a managed cloud platform built atop an optimized version of Spark. The Databricks Platform offers a development environment focused on collaboration, streaming and batch data processing for data engineering, data science and BI workloads, and offers a notebook experience as well as integration with several popular IDEs for code development, testing and deployment. This guide will assist you with the migration from Hadoop to Databricks. All the features discussed in this guide are those that are generally available (GA) and production ready.

There are five Databricks notebooks that accompany this guide. The links to these notebooks are in this document in various sections. The folder containing all the notebooks can be downloaded at:

AWS AZURE

Contents

CHAPTER 1
Overview

The lakehouse architecture 5
Hadoop to Databricks component mapping 6

CHAPTER 2
Platform Administration

Databricks deployment 9
Networking and custom configuration 12
Clusters 16
Cluster pools 18
Cluster resource management 19
Cluster monitoring 21
REST API and command line interface 24
Security and governance 25
Data discovery and audit 26

CHAPTER 3
Application Development, Testing and Deployment

Data sources 29
Data migration 32
Hive metastore 34
HiveQL vs. Spark SQL 37
Delta Lake to optimize data pipelines 40
User-defined functions 42
Sqoop 43
Spark code development on Databricks 44
Notebook and IDE for code development 58
Source code management and CI/CD 61
Job scheduling and submission 66

CHAPTER 4
The Path Forward

Next steps 74

CHAPTER

01

Overview

The lakehouse architecture

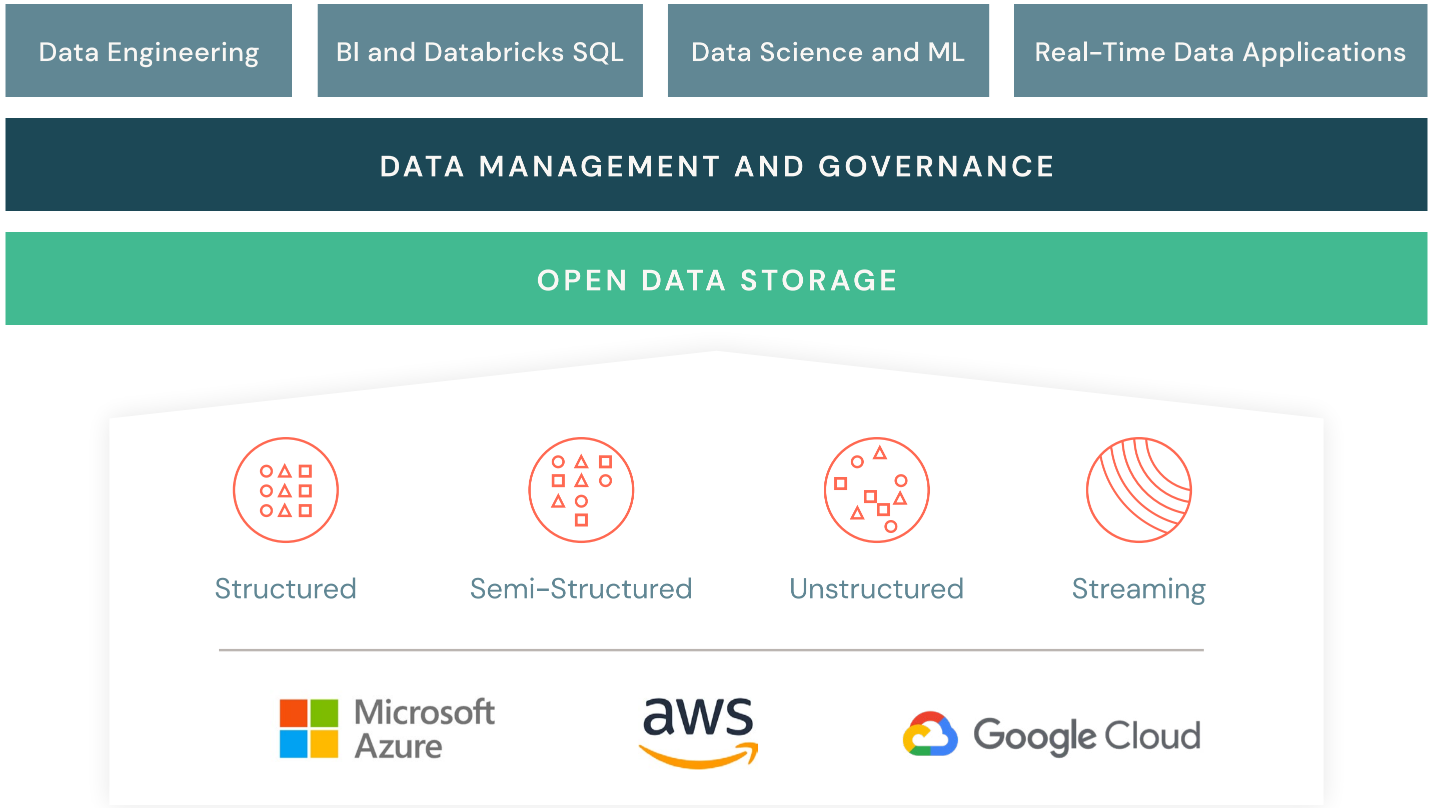
Hadoop to Databricks component mapping

CHAPTER 1: OVERVIEW

The lakehouse architecture

Most Hadoop users, planning the future of their data strategy, are frustrated with the cost, complexity and viability of their existing Hadoop platforms. On-premises Hadoop platforms have failed to deliver on business value due to the lack of data science capabilities, the high cost of operations, lack of agility and poor performance. As a result, enterprises are looking to modernize their existing Hadoop platforms to cloud data platforms.

The Databricks Lakehouse Platform is the cloud-native platform that unifies all your data, analytics and AI workloads. The Lakehouse Platform combines the best elements of data lakes and data warehouses — delivering the data management and performance typically found in data warehouses with the low cost and flexibility of object stores offered by data lakes.



This unified platform simplifies your data architecture by eliminating the data silos that traditionally separate analytics, data science and machine learning. It's built on open source and open standards to maximize flexibility. And, its native collaborative capabilities accelerate your ability to work across teams and innovate faster.

CHAPTER 1: OVERVIEW

Hadoop to Databricks component mapping

When planning your Hadoop migration, it's important to correctly map legacy Hadoop technologies to modern cloud capabilities. The following table maps key Hadoop platform capabilities to the Databricks Platform.

hadoop	databricks
DATA STORAGE • HDFS atop block storage • Kafka • HBase	EQUIVALENT • Cloud object storage: S3, ADLS, Azure Blob • Cloud-native message bus: Kinesis, Azure Event Hubs, Azure IoT Hub • Cloud-native NoSQL: DynamoDB, CosmosDB
DATA PROCESSING • MapReduce • Pig • HiveQL • Spark	EQUIVALENT • Databricks Delta Engine: Optimized Apache Spark for 10x–100x improvement • Databricks SQL: ANSI SQL 2003 compliant • Code-free ETL: Integrations with Azure Data Factory mapping flows, Prophecy, Talend and more
JOBS • Oozie (workflow automation)	EQUIVALENT • Databricks job scheduler • Native integration with Apache Airflow and Azure Data Factory • Use any scheduler via Databricks APIs
CODE DEVELOPMENT • Apache Zeppelin notebook • Jupyter notebook	EQUIVALENT • Databricks notebook • Support for Zeppelin, Jupyter, any notebook or IDE (Pycharm, IntelliJ, etc.) of your choice via Databricks APIs
INTERACTIVE/AD HOC QUERY • HUE • Impala/Hive LLAP	EQUIVALENT • Databricks SQL workspace • Delta Engine/Photon

Hadoop provides several distributed programming frameworks to process your data. They include the legacy low-level Apache MapReduce API and higher-level frameworks such as Apache Pig and Apache Hive. Hadoop also supports Apache Spark. The Databricks Delta Engine makes data processing easy because the combination of Spark and Databricks delivers optimizations of 10x–100x faster performance improvement over open source Spark. And Spark has APIs to let you code in Java, Scala, Python, SQL and R. Spark SQL is ANSI SQL 2003 compliant. Databricks partner integrations with Azure Data Factory, Prophecy and Talend allow you to develop code-free data pipelines.

The default workflow and job orchestration tool in Hadoop is Oozie. Databricks provides a job scheduler in addition to integration with more advanced scheduling tools, such as Apache Airflow and Microsoft Azure Data Factory. You can use your scheduler of choice with Databricks via the Databricks REST APIs.

For visually interacting with your data, Hadoop lets you connect Apache Zeppelin notebooks to clusters. Databricks has a native notebook interface in the cloud. Databricks also supports Zeppelin and Jupyter notebooks, and lets you connect your favorite notebook or IDE via the Databricks REST APIs.

The Databricks SQL workspace can be used for interactive SQL and ad hoc queries. Databricks SQL is a native SQL interface for running BI and SQL queries on the lakehouse with fast performance and high concurrency. It consists of a user interface with a schema browser, a query editor with autocomplete, and dashboards to create rich visualizations. Users can set up query scheduling with alerts. Databricks SQL automatically and transparently load-balances queries across multiple clusters to provide high-concurrency and low-latency query response. Popular BI tools including Tableau and Microsoft Power BI can connect to the platform using native JDBC/ODBC connectors.

AWS Databricks SQL Guide

Azure Databricks SQL Guide

CHAPTER

02

Platform Administration

Databricks deployment

Networking and custom configuration

Clusters

Cluster pools

Cluster resource management

Cluster monitoring

REST API and command line interface

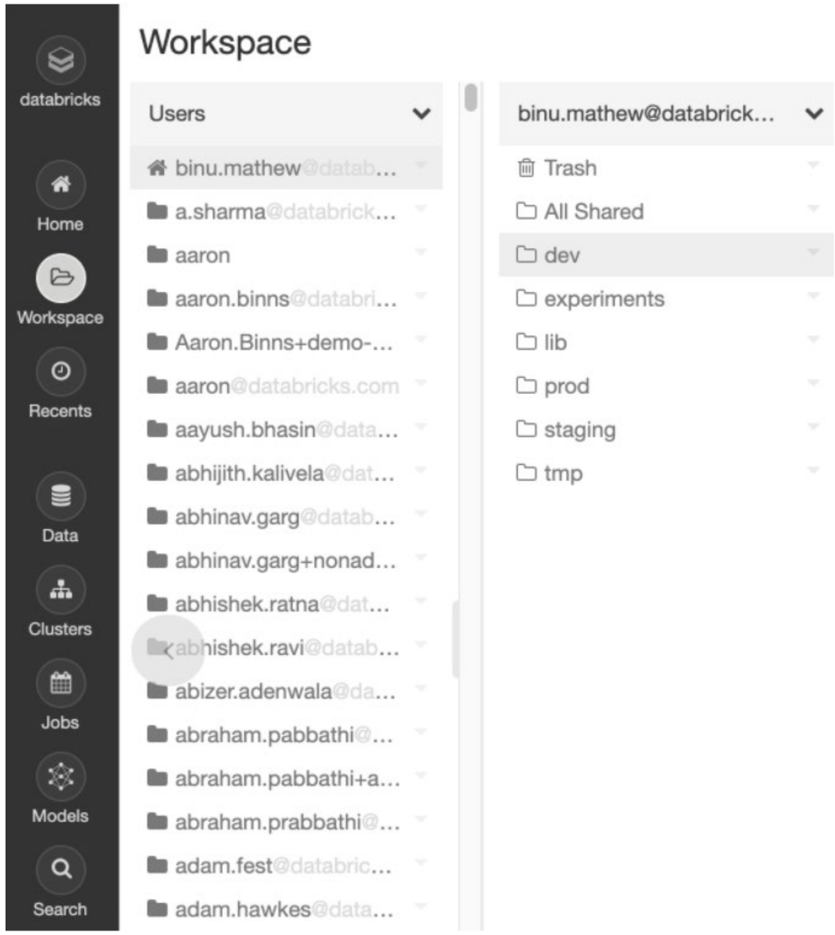
Security and governance

Data discovery and audit

CHAPTER 2: PLATFORM ADMINISTRATION

Databricks deployment

Your Databricks deployment is also referred to as a workspace and has a web UI portal that allows you to administer and manage the platform — and develop, test and deploy your applications. Access to the portal can be authenticated via Single Sign-On (SSO) using multi-factor authentication (MFA) with your organization's identity provider. In Azure Databricks, access to the portal is authenticated via SSO using MFA with your Azure Active Directory (AAD) account. Only users with an AAD account can access the portal. In addition to the web UI, a rich set of REST APIs and a command line interface (CLI) are also available to automate platform administration and application development, testing and deployment.



You may have one or more workspaces (deployments), depending on certain conditions:

Separation of environments: For example, different workspaces for development, staging, production and other environments.

Separation of business units: For example, different workspaces for marketing, finance, risk management and other departments.

It's important to understand some basic concepts used in Databricks. You'll see some of these as icons on the left side of the UI, as shown in the following image. These services are available both in the web application UI as well as the REST API and CLI. We will quickly introduce these concepts, and the rest of this guide will cover them in more detail.

Workspace

Workspace helps you organize all the work that you are doing on Databricks. Like a folder structure in your computer, it allows you to save notebooks and libraries and share them with other users. Each user in your organization will have a folder to organize their work into a directory structure. Workspace has permission settings that allow you to control who has access to your work.

- For Azure, please see the online documentation to learn more about Workspace.
- For AWS, click here.

Clusters

Clusters are groups of virtual machines (VMs) that process your data workloads. Clusters allow you to execute code from notebooks, libraries and custom code written via Java/Scala JAR files, and from Python scripts and wheel/egg files. There are three types of clusters:

1. **STANDARD:** Used for single-user workloads and/or to run single jobs, and are ephemeral or short-lived clusters.
2. **HIGH CONCURRENCY:** Shared by multiple users and are meant for long-running clusters.
3. **SINGLE NODE:** Single VM instance for lightweight analytics with or without Spark.

Clusters do not store data. Data is always stored in your cloud storage account and other data sources.

To learn more, see the "Clusters" section of this guide.

Notebooks

Notebooks are a collaborative IDE that allow you to execute commands in Scala, Python, R, SQL or Markdown. Notebooks have a default language, but each cell can be overridden to use another language. Notebooks need to be connected to a cluster in order to be able to execute commands, but they are not permanently tied to a cluster. This allows notebooks to be shared via the web or downloaded onto your local machine. Dashboards can be created from notebooks as a way of displaying the output of cells without the code that generates them. Notebooks can also be scheduled as jobs in one click either to run a data pipeline, update a machine learning model or update a dashboard.

To learn more, see the "Notebook and IDE for code development" section of this guide.

Libraries

Libraries are packages or modules that provide additional functionality that you need to solve your business problems. These may be custom-written Scala or Java JARs, Python egg or wheel files or custom-written packages. You can write and upload libraries manually through the UI, use the Libraries API, or install them directly via package management utilities like PyPi, Maven or CRAN.

Please see the online documentation to learn more about libraries:

AWS AZURE

Data

The data that you interact with in cloud storage can be organized into structured data represented as databases consisting of tables with schemas that have column names and data types. The Data icon from the UI will list your organization's structured data assets. Databricks can work with structured, semi-structured and unstructured data sources.

To learn more, see the "Data sources" section of this guide.

Jobs

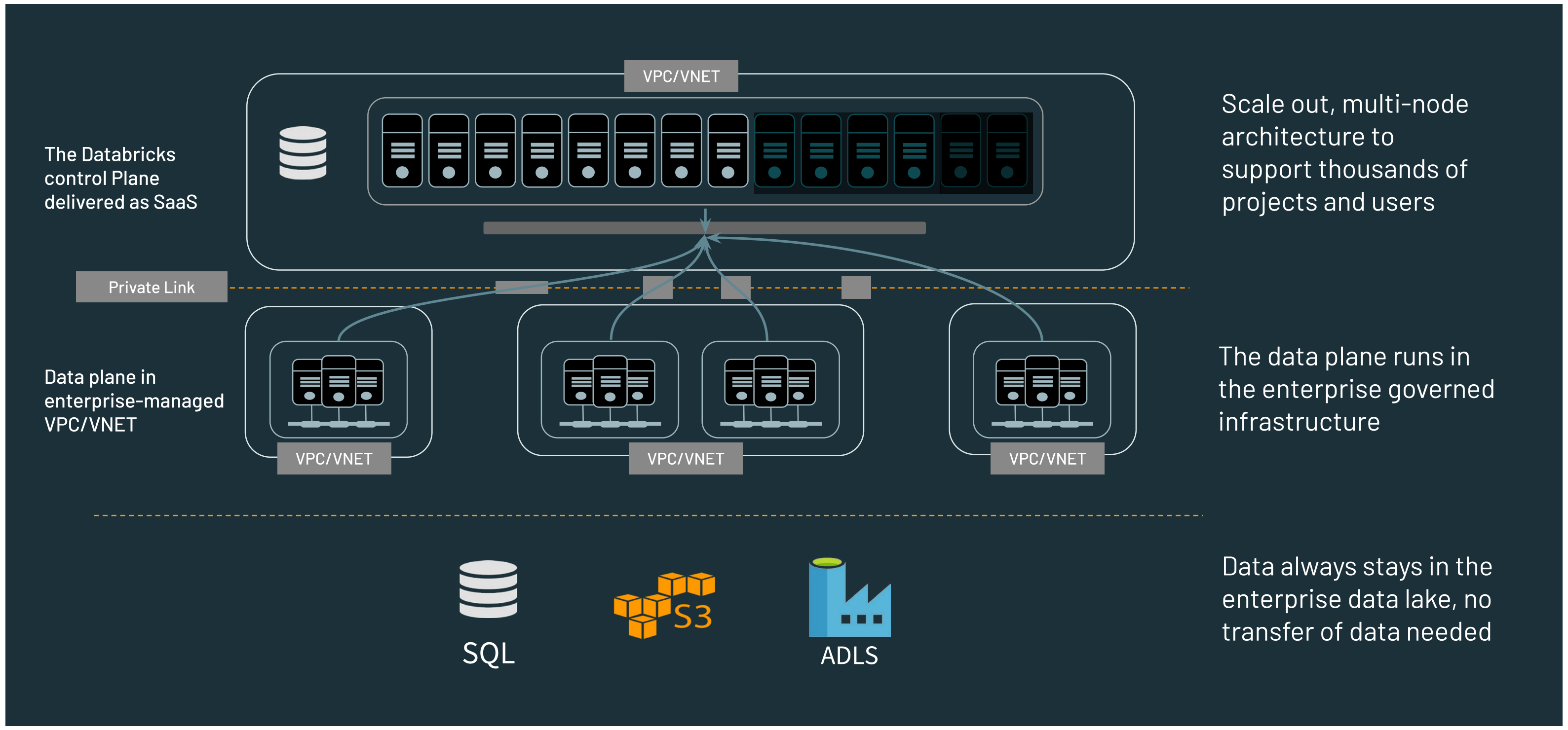
Jobs are how you schedule code execution to occur either on an already existing cluster or a cluster of its own. Jobs can be run from code in notebooks as well as JAR files or Python scripts. They can be created either manually through the UI or via the REST API or command line interface (CLI).

To learn more, see the "Job scheduling and submission" section of this guide.

CHAPTER 2: PLATFORM ADMINISTRATION

Networking and custom configuration

At a high level, the Databricks deployment architecture consists of a control plane that runs in the Databricks subscription, and a data plane that runs in the customer subscription. The control plane includes the back-end services that Databricks manages in its own account. The data plane is managed by your account and is where your data resides and where data is processed.


VPC/VNET
The Databricks control Plane delivered as SaaS
Scale out, multi-node architecture to support thousands of projects and users
Private Link
Data plane in enterprise-managed VPC/VNET
VPC/VNET
VPC/VNET
VPC/VNET
The data plane runs in the enterprise governed infrastructure
SQL
S3
ADLS
Data always stays in the enterprise data lake, no transfer of data needed

To learn more about this architecture, refer to the online documentation:

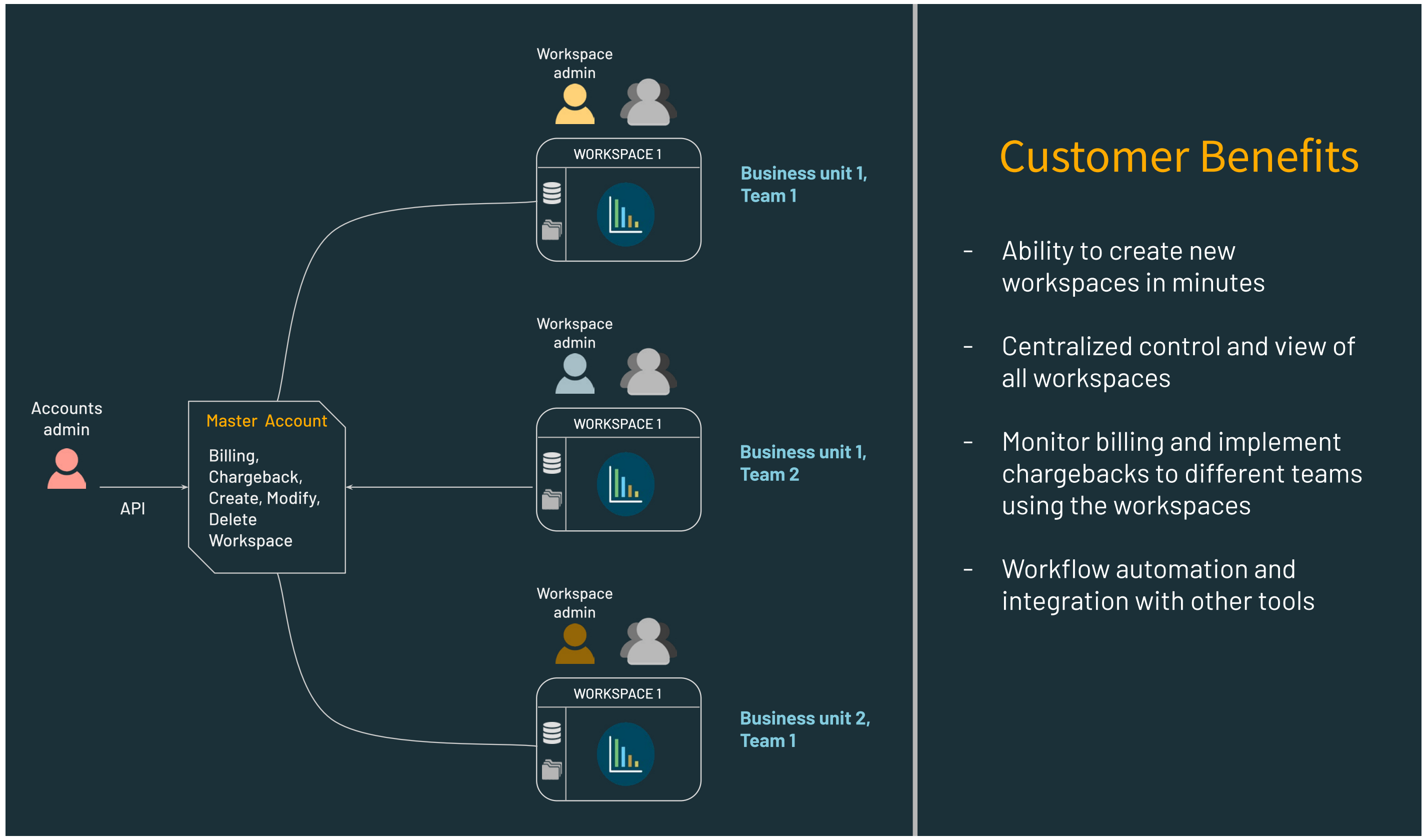
AWS

AZURE

Some key features of this architecture:

- Multiple workspaces
- Bring your own VPC/VNET
- Bring your own key
- No public IPs
- IP access lists

Multiple workspaces

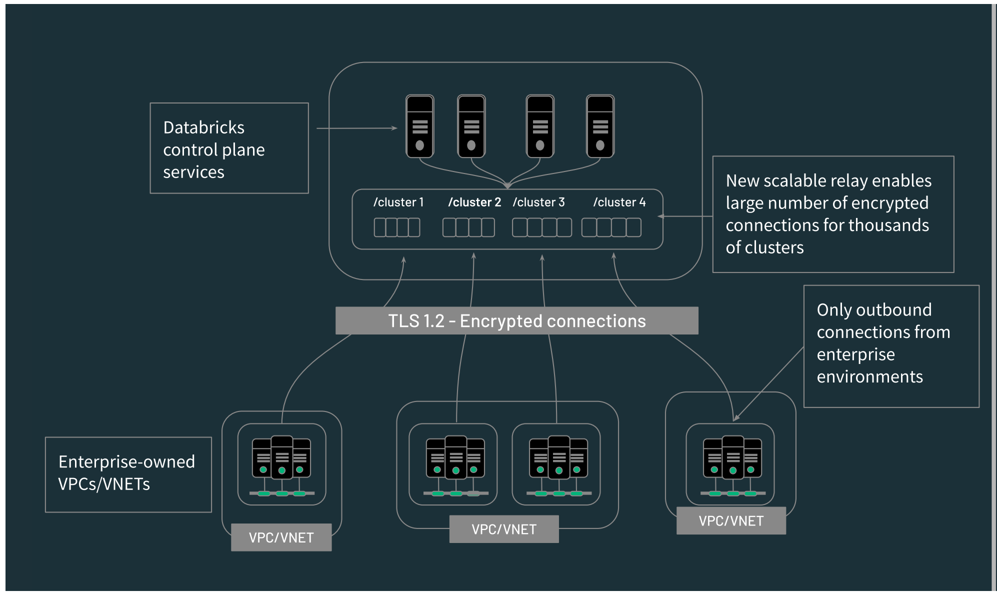


Customer Benefits

- Ability to create new workspaces in minutes
- Centralized control and view of all workspaces
- Monitor billing and implement chargebacks to different teams using the workspaces
- Workflow automation and integration with other tools

- AWS documentation
- For Azure, you would manage multiple workspaces from your Azure account portal

No public IPs


Databricks control plane services
/cluster 1
/cluster 2
/cluster 3
/cluster 4
New scalable relay enables large number of encrypted connections for thousands of clusters
TLS 1.2 - Encrypted connections
Only outbound connections from enterprise environments
Enterprise-owned VPCs/VNETs
VPC/VNET
VPC/VNET
VPC/VNET

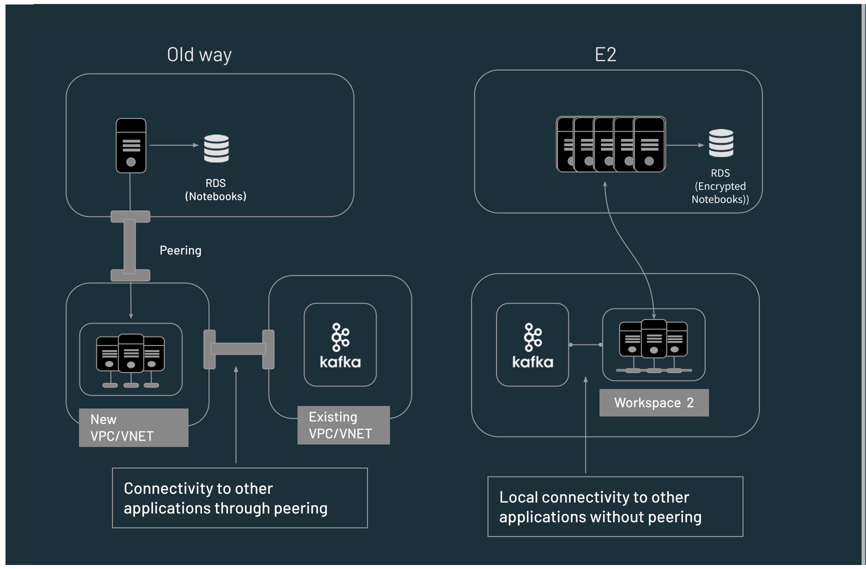
Customer Benefits

- Improved security
- Easy network administration
- Easier information security approval

AWS documentation

Azure documentation

Bring your own VPC/VNET


Old way
RDS (Notebooks)
Peering
kafka
New VPC/VNET
Existing VPC/VNET
Connectivity to other applications through peering
E2
RDS (Encrypted Notebooks))
kafka
Workspace 2
Local connectivity to other applications without peering

Customer Benefits

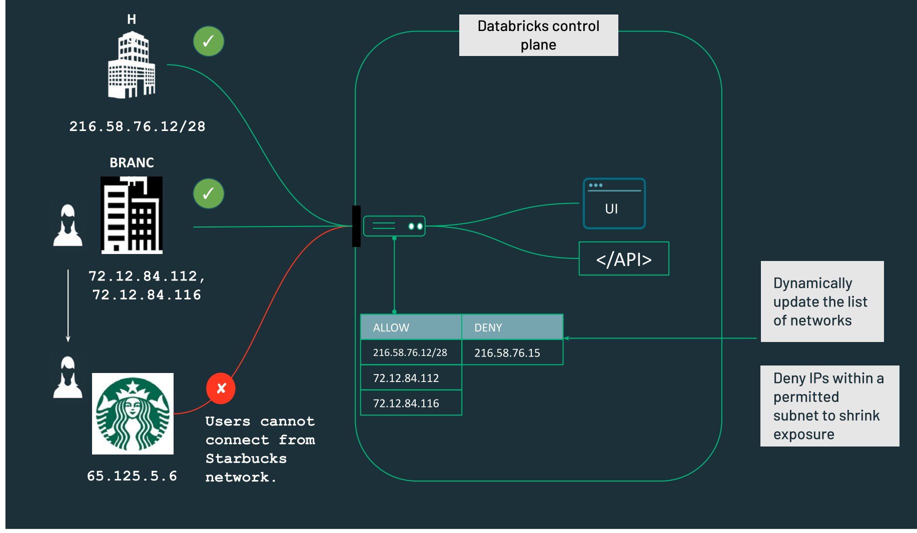
- Limits outgoing connections
- Simplified network operations
- Consolidation of VPCs
- Limit outgoing connections

AWS documentation

Azure documentation

IP access lists

Control the networks that can access Databricks



ALLOW	DENY
216.58.76.12/28	216.58.76.15
72.12.84.112	
72.12.84.116	

Users cannot connect from Starbucks network.

Dynamically update the list of networks

Deny IPs within a permitted subnet to shrink exposure

Customer Benefits

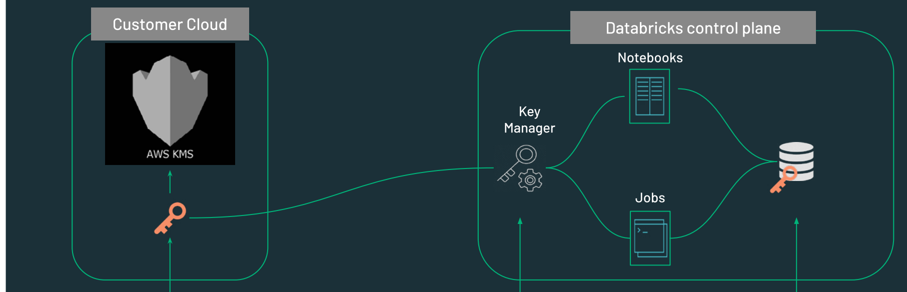
- Reduce risk of external attacks and potential data leaks within an enterprise
- Track and log malicious internal attacks on Databricks within the org

AWS documentation

Azure documentation

Bring your own key

Encryption with enterprise root keys (BYOK)



Customer creates KMS key for an account or a workspace

Databricks creates data encrypting keys rooted in customer key

Applications use customer key to encrypt all the data

Customer Benefits

- Get full control over keys used to encrypt data in the control plane. Revoking key revokes data access.
- Key hierarchy enables use of different keys for different notebooks
- An audit log of key operations makes for easy reporting and audit requests

AWS documentation

Azure documentation

CHAPTER 2: PLATFORM ADMINISTRATION

Clusters

Databricks is a fully managed PaaS offering that requires no infrastructure administration, management or maintenance. Users and processes run code on clusters of VMs for data engineering, data science and data analytics workloads. This includes batch and real-time production ETL pipelines, streaming analytics, ad hoc analytics, machine learning, deep learning and graph analytics. Clusters can be fixed-size clusters or autoscaling and, by default, they auto-terminate after 120 minutes of inactivity (this is configurable).

- Fixed-size clusters stay constant for the duration of the cluster lifecycle. This is a good option to choose when you know the exact compute capacity required (CPU cores and RAM), as no time will be spent allocating and starting additional VMs.
- Autoscaling clusters will dynamically scale out from a minimum number of cluster VM nodes to a maximum that you configure. This option is recommended when you cannot easily predict the compute capacity required (CPU cores and RAM) — for instance, because of increasing data volumes or data skew.

Clusters are divided into three distinct types

1. STANDARD CLUSTERS
2. HIGH CONCURRENCY CLUSTERS
3. SINGLE NODE CLUSTERS

Standard clusters

Standard clusters are recommended for a single user. They can be fixed-size or autoscaling clusters. They are typically short-running ephemeral clusters used for running jobs; however, in the case of streaming jobs, the cluster might be always-on and long-running. Standard clusters can run workloads developed in any language: Java, Python, R, Scala or SQL. Standard clusters are also used to run individual jobs — for example, streaming, ETL or machine learning. Since standard clusters run a single user job and not jobs from multiple users and/or processes, stronger resource isolation, SLA guarantees and security can be provided.

High concurrency clusters

A high concurrency cluster is ideal for multiple users accessing a single cluster to run interactive or automated jobs. They can be fixed-size or autoscaling clusters. By default, high concurrency clusters are set to autoscale. These clusters only support SQL, Python and R. The key benefits of high concurrency clusters are that they provide Apache Spark–native fine-grained sharing for maximum resource utilization and minimum query latencies so that all users on the cluster can run jobs by sharing total compute resources (CPU and RAM) among all the users on the cluster. High concurrency clusters can help reduce costs for a shared user work environment, experimentation, testing and execution of some production workloads.

Single node clusters

A single node cluster consists of a Spark driver and no Spark workers. It supports Spark jobs and all Spark data sources. In contrast, standard clusters require at least one Spark worker to run Spark jobs. Single node clusters can be useful for:

- Running single-node machine learning workloads that need Spark to load and save data
- Lightweight exploratory data analysis (EDA)

All cluster types can be created and configured via the UI, REST API or command line interface (CLI). Databricks supports a variety of VM types for different workloads: memory-optimized, CPU-optimized, storage-optimized and GPU-accelerated VMs. Databricks also supports custom containers to launch clusters with predefined settings. For example, if you have several libraries that need to be used on the cluster, you can reduce the cluster start-up time by creating a custom Docker image that contains all of your dependencies and then using this image to launch the Databricks cluster.

Please refer to the online documentation to learn more about clusters:

AWS

AZURE

Here are some cluster best practices:

- Use autoscaling clusters when the compute capacity required is unknown
- Set automatic termination when applicable
- Use the latest Databricks Runtime version to take advantage of recent performance and other optimizations
- Use high concurrency cluster mode for data analysis by a team of users via notebooks or a BI tool, or if you want to enforce data protection via table ACLs
- Use cluster tags for project- or team-based chargeback
- Custom Spark configuration settings can be applied to all nodes in a cluster if needed
- Use the Cluster Event Log and Spark UI to analyze cluster activities and submitted job performance
- Configure Cluster Log Delivery to deliver Spark driver and worker logs to cloud storage
- Use Cluster Access Control to configure permissions for users and groups
- Use Initialization Scripts or Databricks Container Services to launch clusters with preinstalled software and libraries. Databricks Container Services allows you to create your own Docker image and then launch a Databricks cluster using this Docker image.
- User Cluster Policies to limit the cluster types that users can launch

CHAPTER 2: PLATFORM ADMINISTRATION

Cluster pools

Cluster start-up time and autoscaling time are gated by the time it takes to acquire VM instances from the cloud provider. If this begins to impact SLAs, we recommend using Databricks cluster pools. Pools reduce cluster start and autoscaling times by maintaining a set of idle, ready-to-use instances. When a cluster attached to a pool needs additional VMs, it first attempts to allocate one of the pool's idle VMs. That way, the VMs can instantly be attached to the cluster with no latency, allowing stronger guarantees to meet SLAs. There will be an associated cloud cost associated with VMs allocated to the pool. However, there will be no Databricks charge for those instances.

Please refer to the online documentation to learn more about clusters:

AWS

AZURE

CHAPTER 2: PLATFORM ADMINISTRATION

Cluster resource management

Hadoop users are familiar with Apache YARN for application resource management and job scheduling. Using YARN, multiple users and processes can share a single Hadoop cluster so that cluster resources, such as CPU and RAM, can be shared.

- Fair scheduling that evenly distributes resources among all jobs
- Capacity scheduling to define queues with weighted percentages and assign users and groups to these queues so that their jobs run within those queues

YARN will launch jobs and monitor them for their duration so that if any failures are encountered, YARN will attempt to restart those jobs.

Databricks takes a different approach to application resource management and job scheduling for standard and high concurrency clusters and uses its own resource manager, which is more similar to the Apache Spark stand-alone resource manager. A single SparkContext is shared among multiple sessions on the cluster. Each user on the cluster will have their own separate SparkSession, but all users will share the same SparkContext. The SparkContext allows your Spark application to access the cluster with the help of the resource manager.

The following Spark configuration settings apply to both standard and high concurrency clusters:

- The default functionality of Databricks clusters is to use fair scheduling that evenly distributes CPU and RAM compute resources among all jobs
  `spark.scheduler.mode FAIR`
- Databricks, by default, does not have capacity queues similar to those in YARN
- Databricks does use preemption to prevent overallocation of resources, ensuring that all jobs have an equal share of resources
  `spark.databricks.preemption.enabled true`
- Each user has their own SparkSession
  `spark.databricks.session.share false`

Resource sharing on standard vs. high concurrency clusters

Standard clusters are meant to be used for a single processing job. This could be an interactive session in which the user submits jobs via the notebook or it could be an automated job. We don't recommend sharing a standard cluster to execute jobs from multiple users and/or processes. High concurrency clusters are meant to be shared. Both standard and high concurrency clusters enable preemption by default, but in high concurrency clusters, jobs from each user run in a separate fair scheduler pool with preemption configured to ensure a fair allocation of resources. High concurrency clusters also create fault isolation for each user. With fault isolation, each user's environment is isolated from others so that a single user's process cannot impact the entire cluster.

A common problem when multiple users share a cluster is that one user's faulty code can crash the Spark driver, bringing down the cluster for all users. In these types of situations, the Databricks resource manager provides fault isolation by sandboxing the driver processes belonging to different users from one another so that a user can safely run commands that might otherwise crash the driver, eliminating concern about impacting the experience of other users.

With preemption on high concurrency clusters, the following Spark settings can be configured to share a cluster among multiple jobs in the way that you'd like:

`spark.databricks.preemption.threshold 0.5`
The fair share fraction to guarantee per user. Setting this to 1.0 means the scheduler will aggressively attempt to guarantee perfect fair sharing. Setting this to 0.0 effectively disables preemption. The default setting is 0.5, which means at worst a user will get half their fair share.

`spark.databricks.preemption.timeout 30s`
How long a user must remain starved before preemption kicks in. Setting this to lower values will provide more interactive response times at the cost of cluster efficiency. Recommended values are from 1 to 100 seconds.

`spark.databricks.preemption.interval 5s`
How often the scheduler will check for task preemption. This should be set to less than the preemption time-out.

Please refer to the online documentation for more information on preemption:
AWS Azure

In both standard and high concurrency clusters, by default, all queries started in a notebook run in the same fair scheduling pool. Therefore, jobs generated by triggers from all the streaming queries in a notebook run one after another in first in, first out (FIFO) order. This can cause unnecessary delays in the queries, because they are not efficiently sharing the cluster resources. To enable all streaming queries to execute jobs concurrently and to share the cluster efficiently, you can set the queries to execute in separate scheduler pools.

Please refer to the online documentation for more information:
AWS Azure

High concurrency clusters allow for multiple users to submit different SQL queries. Databricks can prevent rogue queries from monopolizing cluster resources by examining the most common causes of large queries and terminating queries that pass a threshold. This is a feature called Query Watchdog.

`spark.databricks.queryWatchdog.enabled true`
This will enable Query Watchdog

`spark.databricks.queryWatchdog.outputRatioThreshold 1000L`
To prevent a query from creating too many output rows for the number of input rows, you can enable Query Watchdog and configure the maximum number of output rows as a multiple of the number of input rows. "1000L" declares that any given task should never produce more than 1,000 times the number of input rows.

Please refer to the online documentation for more information on Query Watchdog:
AWS Azure

CHAPTER 2: PLATFORM ADMINISTRATION

Cluster monitoring

Databricks provides Ganglia metrics to monitor clusters as jobs run on those clusters. This is native to the Databricks platform, and no additional setup or integration is required.

Please refer to the online documentation for more information:

AWS AZURE

Advanced monitoring in AWS

You can also implement more advanced application and infrastructure monitoring by using AWS CloudWatch. This can be done by installing AWS CloudWatch agents on the Databricks EC2 nodes so metrics can be sent to CloudWatch. Databricks allows the installation of third-party software, and customers are responsible for the maintenance and support of third-party software. Init scripts or Databricks Container Services can be used to install CloudWatch agents on all the nodes in a Databricks cluster.

- Please refer to the online documentation for more information about init scripts
- Please refer to the online documentation for more information about Databricks Container Services

The CloudWatch agent can easily be installed on a Databricks node. The Databricks EC2 instances allow for all outbound traffic, and you should be able to configure the CloudWatch agent to send metrics to CloudWatch. After installation, the CloudWatch agent needs to be configured.

Init script to install CloudWatch agent

Here’s an example of what the init script could look like. This is a working example. The init script would be made a global init script if you needed the CloudWatch agent installed on every Databricks cluster node launched:

```
dbutils.fs.put(“/databricks/init/cloud-watch-agent-install.sh”,”””
#!/bin/bash
# install CloudWatch agent
wget https://s3.amazonaws.com/amazoncloudwatch-agent/linux/amd64/latest/AmazonCloudWatchAgent.zip
unzip AmazonCloudWatchAgent.zip
sudo ./install.sh
# copy configuration files from root S3 bucket to local file system on Ubuntu nodes
cp /tmp/bmathew/test_script/common-config.toml /opt/aws/amazon-cloudwatch-agent/etc/common-config.toml
cp /tmp/bmathew/test_script/amazon-cloudwatch-agent-schema.json /opt/aws/amazon-cloudwatch-agent/doc/
amazon-cloudwatch-agent-schema.json
# start the CloudWatch Agent
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -c file:/
opt/aws/amazon-cloudwatch-agent/doc/amazon-cloudwatch-agent-schema.json -s
“””, True)
```

Configuring CloudWatch agent

1. **OPTIONAL:** Configure common-config.toml to modify the Common Configuration and Named Profile for CloudWatch Agent. Modify this file only if you need to specify proxy settings or if you need the agent to send metrics to CloudWatch in a different region than where the instance is located.

   All Databricks EC2 instances have outbound access. All Databricks clusters will launch into the same region (although you can specify different availability zones within the region). Assuming that CloudWatch is installed in the same region as your Databricks shard (deployment), this file might not need to be modified. The IAM role attached to the Databricks cluster needs the proper permissions to be able to send metrics to CloudWatch.

   If this file does need to be modified, then this can be done through the init script:

   The init script would copy the edited/final version of this config file from the root S3 bucket (dbfs — databricks file system) and place it into the proper directory on the Databricks EC2 node (local Linux file system).

2. **REQUIRED:** JSON configuration file. You need to create a configuration file before you start the agent on any servers. The agent configuration file is a JSON file that specifies the metrics and logs that the agent is to collect. The init script would copy the edited/final version of the agent configuration file from the root S3 bucket (dbfs — databricks file system) and place it into the proper directory on the Databricks EC2 node (local Linux file system).

Please see the official AWS documentation on configuring the CloudWatch agent.

Advanced monitoring in Azure

In Azure, there is also integration with Azure Monitor, which is a more familiar solution for Azure customers, as they are already using Azure Monitor to observe all the activity in their Azure accounts. With the Azure Databricks and Azure Monitor integration, you can send job/application logs to Azure Monitor for detailed monitoring of cluster metrics.

Please refer to the online documentation to complete this integration.

You can also integrate with the Azure Log Analytics workspace to easily analyze metrics collection data. Please refer to the online documentation.

CHAPTER 2: PLATFORM ADMINISTRATION

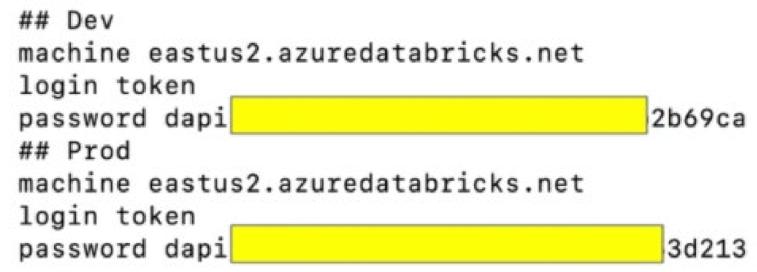
REST API and command line interface

Databricks provides a rich set of REST APIs and a powerful command line interface (CLI) for platform interaction to perform tasks such as:

- Cluster administration, job submission, library management, user/group management
- Submit JAR files (Java or Scala) and Python code (scripts, egg, wheel files) to schedule and run as streaming or batch jobs
- Develop locally with your favorite editor/IDE and connect to Databricks: Visual Studio, PyCharm, IntelliJ, RStudio, Jupyter, Zeppelin and more
- Source code management (SCM) integration: Develop locally on your laptops, import/export to Databricks, check-in/check-out with your SCM tool
- Continuous Integration/Continuous Delivery (CI/CD)
- Integrate with third-party schedulers for more advanced DAG/workflow creation (e.g., Azure Data Factory, Apache Airflow)

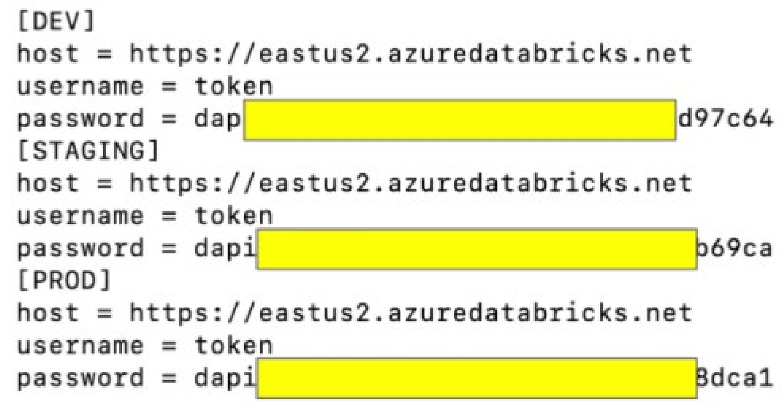
Users authenticate with the REST API and CLI via tokens. Tokens can be generated from the UI and also by using the Tokens API. You can store tokens locally in a file: in a .netrc file and/or a .databrickscfg file. You might be accessing multiple deployments so your files could have multiple entries. For example:

.netrc



```
## Dev
machine eastus2.azuredatabricks.net
login token
password dapi[redacted]2b69ca
## Prod
machine eastus2.azuredatabricks.net
login token
password dapi[redacted]3d213
```

.databrickscfg



```
[DEV]
host = https://eastus2.azuredatabricks.net
username = token
password = dap[redacted]d97c64
[STAGING]
host = https://eastus2.azuredatabricks.net
username = token
password = dapi[redacted]b69ca
[PROD]
host = https://eastus2.azuredatabricks.net
username = token
password = dapi[redacted]8dca1
```

Please refer to the online documentation for more information on using the REST API:

AWS AZURE

Please refer to the online documentation for more information on installing and using the command line interface:

AWS AZURE

CHAPTER 2: PLATFORM ADMINISTRATION

Security and governance

Databricks lets you centrally manage and govern platform access from the UI or via the REST API to grant, deny and revoke user access:

- Control data access to databases, tables and views using RBAC
- Cloud IAM integration for file access
- Audit logs to monitor user activity
- Third-party data catalog integration for master data management, lineage, data discovery, data profiling and data classification. Vendor specific and open source.

Alation Amundsen IMMUTA

PRIVACERA Prophecy talend

Coming from the Hadoop ecosystem, you are probably already familiar with role-based access control (RBAC) and attribute-based access control (ABAC). Databricks provides RBAC for both the platform and your data. In Databricks, you can use access control lists (ACLs) to configure permissions to access the workspace, notebooks, clusters, pools, jobs and Spark SQL tables.

At a platform level, we recommend that you enable audit logs to track all user interaction with the platform.

Please refer to the online documentation for more information:

AWS AZURE

You might be using Apache Ranger today for RBAC and ABAC access to your data. Similar RBAC functionality is available in Databricks using database views and then applying access permissions on those views. For example, if you have a base Spark SQL table with sensitive information and you want to control who has access to it, then you can create one or more Spark SQL views. The views give you different ways of looking at the data in the base table. For example, you can create a view with logic to hash certain fields and filter specific rows, and then apply table ACLs on the base table and views to restrict access to certain groups and users. This functionality is available for clusters running Python and SQL workloads.

Databricks will be releasing Unity Catalog, which will provide attribute-based access controls, auditing and lineage information.

For immediate RBAC and ABAC capabilities, we offer integration with Ranger via our partnership with Immuta and Privacera. Please consult your Databricks Solutions Architect for more information on this integration.

Please refer to the online documentation for more information on security in Databricks:

AWS AZURE

Please refer to the online documentation for more information about table access controls:

AWS AZURE

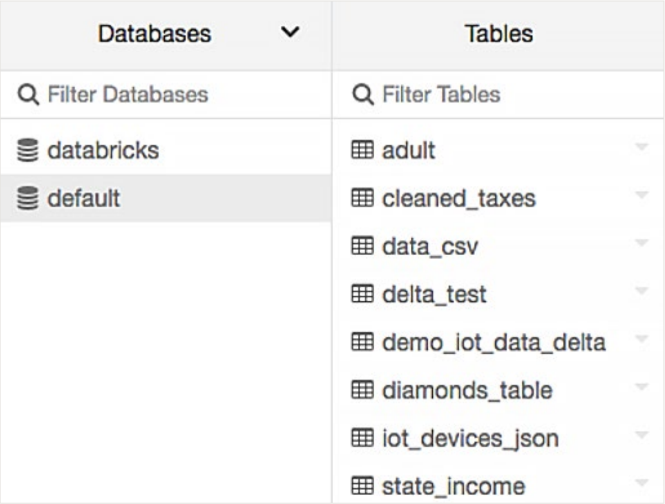
See the "Data sources" section in this guide for information on how to securely connect to your data sources.

CHAPTER 2: PLATFORM ADMINISTRATION

Data discovery and audit

Under Databricks, it's possible to explore your data catalog via the Hive metastore (either embedded or external). This lets you navigate through databases and, for each database, list available tables and, for each table, display the schema and a sample of data.

Under workspace



Databases	Tables
Filter Databases	Filter Tables
databricks	adult
default	cleaned_taxes
	data_csv
	delta_test
	demo_iot_data_delta
	diamonds_table
	iot_devices_json
	state_income

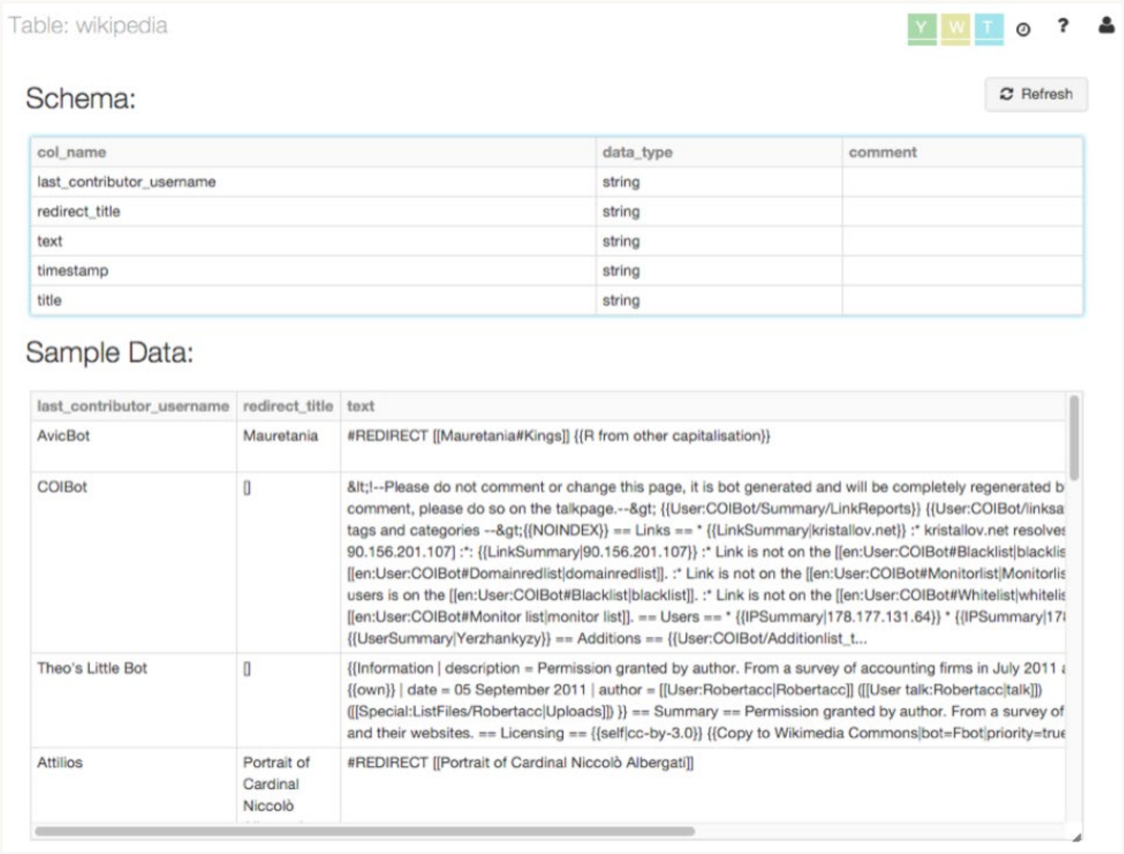


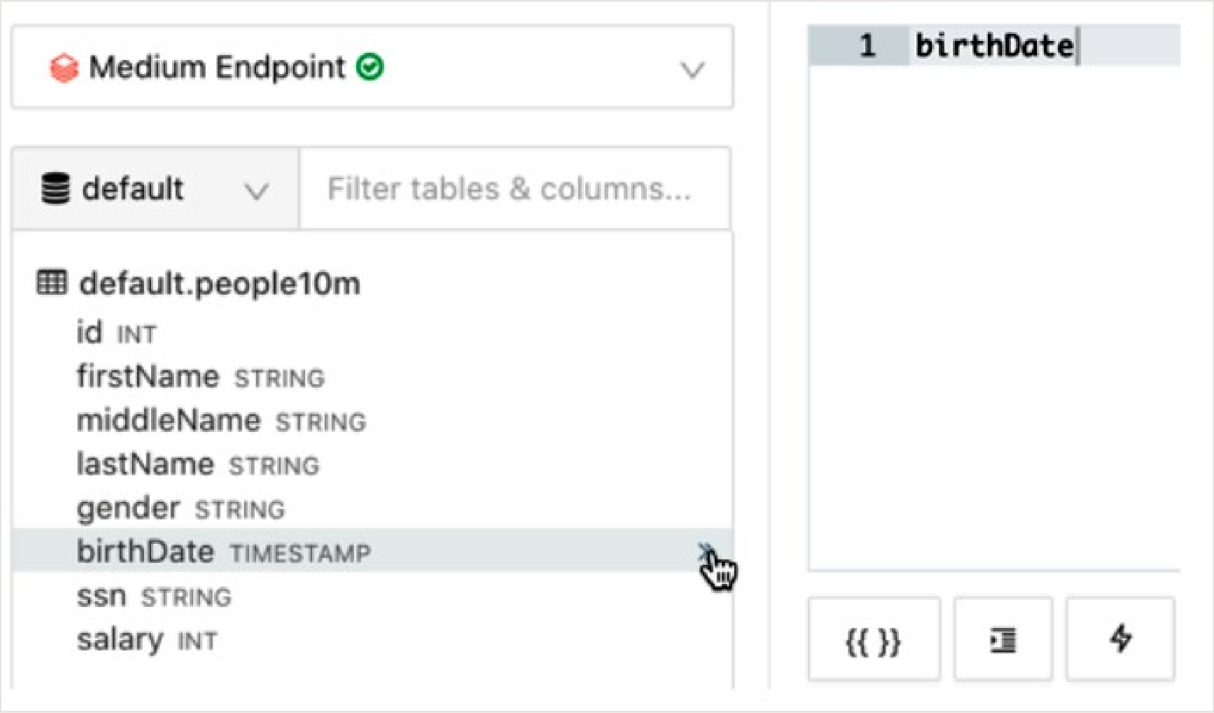
Table: wikipedia

Schema:

col_name	data_type	comment
last_contributor_username	string	
redirect_title	string	
text	string	
timestamp	string	
title	string	

Sample Data:

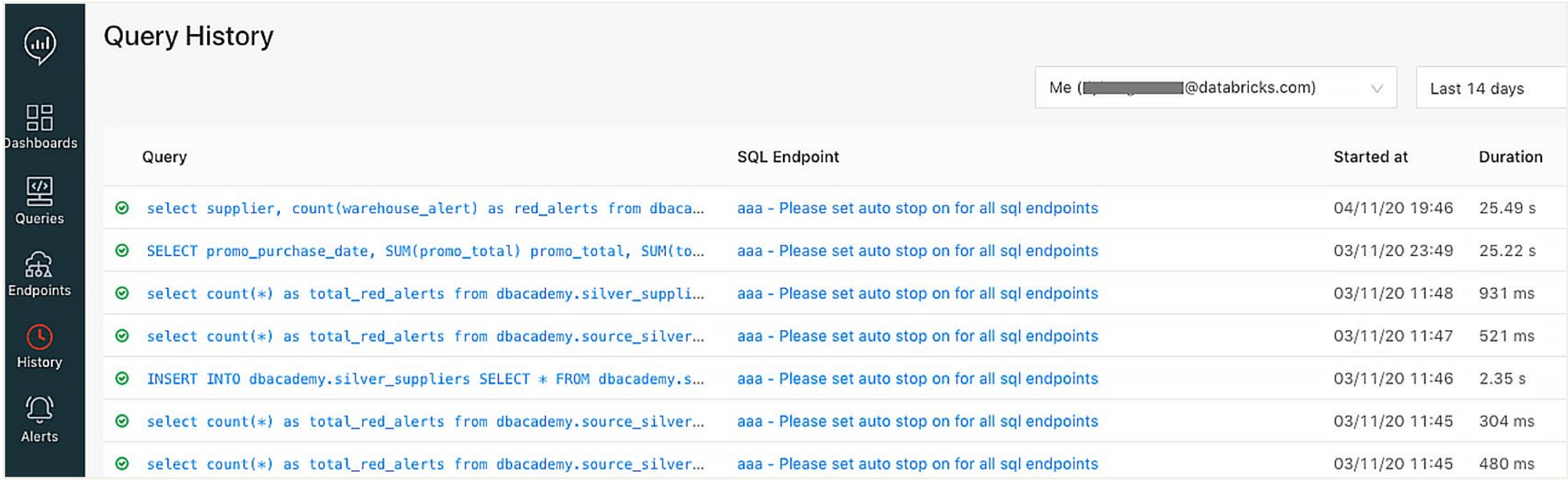
<table>
<tr><th>last_contributor_username</th><th>redirect_title</th><th>text</th></tr>
<tr><td>AvicBot</td><td>Mauretania</td><td>#REDIRECT [[Mauretania#Kings]] {{R from other capitalisation}}</td></tr>
<tr><td>COIBot</td><td>[]</td><td>&lt;!--Please do not comment or change this page, it is bot generated and will be completely regenerated b comment, please do so on the talkpage.--&gt; {{User:COIBot/Summary/LinkReports}} {{User:COIBot/linksa tags and categories --&gt;{{NOINDEX}} == Links == * {{LinkSummary|kristallov.net}} :* kristallov.net resolves 90.156.201.107] :*: {{LinkSummary|90.156.201.107}} :* Link is not on the [[en:User:COIBot#Blacklist|blacklis [[en:User:COIBot#Domainredlist|domainredlist]]. :* Link is not on the [[en:User:COIBot#Monitorlist|Monitorlis users is on the [[en:User:COIBot#Blacklist|blacklist]]. :* Link is not on the [[en:User:COIBot#Whitelist|whitelis [[en:User:COIBot#Monitor list|monitor list]]. == Users == * {{IPSummary|178.177.131.64}} * {{IPSummary|17 {{UserSummary|Yerzhankyzy}} == Additions == {{User:COIBot/Additionlist_t...</td></tr>
<tr><td>Theo's Little Bot</td><td>[]</td><td>{{Information | description = Permission granted by author. From a survey of accounting firms in July 2011 {{own}} | date = 05 September 2011 | author = [[User:Robertacc|Robertacc]] ([[User talk:Robertacc|talk]]) ([[Special:ListFiles/Robertacc|Uploads]]) }} == Summary == Permission granted by author. From a survey of and their websites. == Licensing == {{self|cc-by-3.0}} {{Copy to Wikimedia Commons|bot=Fbot|priority=true</td></tr>
<tr><td>Attilios</td><td>Portrait of Cardinal Niccolò</td><td>#REDIRECT [[Portrait of Cardinal Niccolò Albergati]]</td></tr>
</table>



default.people10m
id INT
firstName STRING
middleName STRING
lastName STRING
gender STRING
birthDate TIMESTAMP
ssn STRING
salary INT

```
1  birthDate
```

Under Databricks SQL, the history tab shows all run queries, the author for each query and the execution status. It's a seamless way to track access to data.



Query History

Me ([redacted]@databricks.com)

Last 14 days

Query	SQL Endpoint	Started at	Duration
select supplier, count(warehouse_alert) as red_alerts from dbaca...	aaa - Please set auto stop on for all sql endpoints	04/11/20 19:46	25.49 s
SELECT promo_purchase_date, SUM(promo_total) promo_total, SUM(to...	aaa - Please set auto stop on for all sql endpoints	03/11/20 23:49	25.22 s
select count(*) as total_red_alerts from dbacademy.silver_suppli...	aaa - Please set auto stop on for all sql endpoints	03/11/20 11:48	931 ms
select count(*) as total_red_alerts from dbacademy.source_silver...	aaa - Please set auto stop on for all sql endpoints	03/11/20 11:47	521 ms
INSERT INTO dbacademy.silver_suppliers SELECT * FROM dbacademy.s...	aaa - Please set auto stop on for all sql endpoints	03/11/20 11:46	2.35 s
select count(*) as total_red_alerts from dbacademy.source_silver...	aaa - Please set auto stop on for all sql endpoints	03/11/20 11:45	304 ms
select count(*) as total_red_alerts from dbacademy.source_silver...	aaa - Please set auto stop on for all sql endpoints	03/11/20 11:45	480 ms

CHAPTER

03

Application Development, Testing and Deployment

Data sources

Data migration

Hive metastore

HiveQL vs. Spark SQL

Delta Lake to optimize data pipelines

User-defined functions

Sqoop

Spark code development on Databricks

Notebook and IDE for code development

Source code management and CI/CD

Job scheduling and submission

CHAPTER 3: APPLICATION DEVELOPMENT, TESTING AND DEPLOYMENT

Data sources

Databricks is optimized for reading and writing to cloud storage: S3, Azure Blob, Azure Data Lake Storage Gen 1 and Azure Data Lake Storage Gen 2 (ADLS).

In AWS, Databricks is optimized for reading and writing to S3. In addition to S3 cloud storage, Databricks can also read/write to other storage end points, including relational databases (Oracle, Redshift, SQL Server, Teradata); HDFS, Apache Hive, NoSQL (HBase, Cassandra, MongoDB); in-memory cache (Redis, RocksDB); S3 SQS, message bus (Kafka, Kinesis); files (delimited text files, JSON, Parquet, ORC, Avro) and many others. Data sources can be in the cloud or on-premises.

Please refer to the online documentation to learn more about the data sources supported in Databricks.

In Azure, Databricks recommends using ADLS Gen 2 for optimal performance. In addition to Azure cloud storage, Azure Databricks can also read/write to other storage end points, including relational databases (Oracle, SQL Server, Teradata); HDFS, Apache Hive, NoSQL (HBase, Cassandra, MongoDB); in-memory cache (Redis, RocksDB); message bus (Kafka); files (delimited text files, JSON, Parquet, ORC, Avro) and many others. In addition, Azure Databricks has native integration with several Azure end points, including SQL Data Warehouse, Cosmos DB, Event Hub, IoT Hub and more. Data sources can be in the cloud or on-premises.

Please refer to the online documentation to learn more about the data sources supported in Azure Databricks.

Hadoop users are familiar with the Optimized Row Columnar (ORC) file format, which Databricks supports. However, Databricks is optimized for Parquet and Delta in cloud storage — and we recommend using Delta, an open source storage layer built atop Parquet that brings performance, reliability and consistency to cloud storage. You can easily convert your ORC files to Delta by reading the data into a Spark DataFrame and then saving it in the Delta format. To learn more, see the "Delta" section in this guide.

Databricks created a distributed file system built atop cloud storage called the Databricks File System, or DBFS. DBFS is an abstraction over cloud storage when used with Databricks Utilities (DBUtils) and offers the following functionality:

- Allows you to mount data from cloud storage so that you can access data without requiring credentials (similar to NFS mounts)
- Allows you to interact with cloud storage using UNIX directory and file commands instead of cloud-specific APIs
- Persists files to object storage, so you won't lose data after you terminate a cluster
- Once you create a mount point from any cluster in a Databricks workspace, then by default that mount point will be accessible to all clusters in that workspace

For more information, refer to these sections in this guide:

- Accessing AWS cloud storage
- Accessing Azure cloud storage

Accessing S3 cloud storage

ACCESS KEYS

- Easy to set up and use
- Needs to be used with caution — i.e., make sure notebook permissions are enabled so no one can read a notebook that contains credentials
- Allows use of the Secrets API with keys to keep the actual key values hidden
- How to use keys

IAM ROLES

- More secure
- Limited in that only one IAM role can be attached to a Databricks cluster
- How to use IAM roles

IAM CREDENTIAL PASS-THROUGH

- Each user syncs their credentials with their AWS account to authenticate which S3 buckets they have access to
- Requires integration with an identity provider: AWS identity federation with SAML Single Sign-On
- Using IAM credentials pass-through

Accessing Azure cloud storage

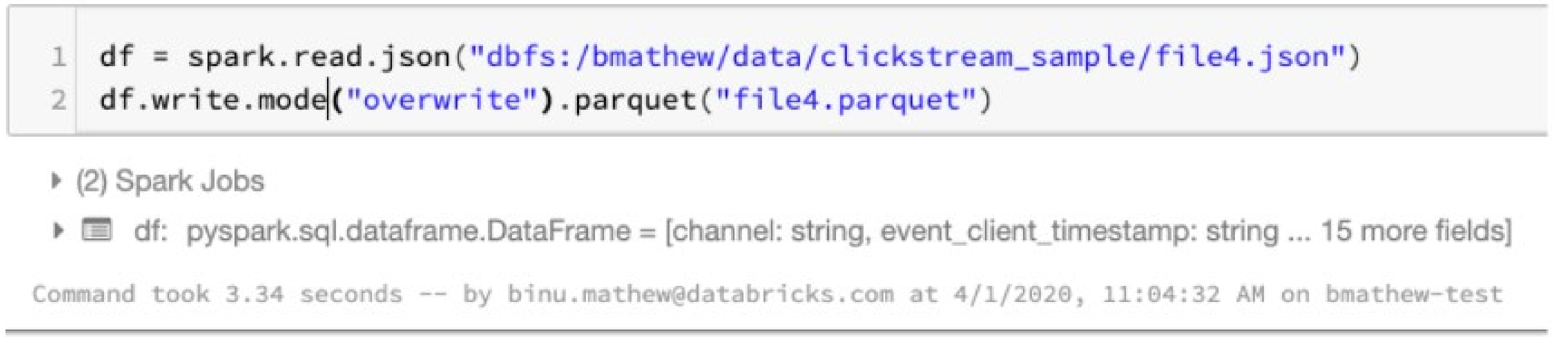
SHARED KEY OR SHARED ACCESS SIGNATURE

- Easy to set up and use
- Needs to be used with caution — i.e., make sure notebook permissions are enabled so no one can read a notebook that contains credentials
- Allows use of the Secrets API with keys to keep the actual key values hidden
- How to use keys with Azure Blob storage
- How to use keys with Azure Data Lake Storage

AZURE ACTIVE DIRECTORY (AAD) CREDENTIALS

- Access data directly from ADLS Gen 2 using your AAD credentials

Cloud storage end points are owned by you, and Databricks does not have direct access to them — with the exception of a shared cloud storage location called DBFS Root, which Databricks has read/write access to. DBFS Root is required and is used by Databricks to store metadata and logs and can also be used to store data. We recommend not storing your own data in DBFS Root. By default, if your code is writing data without specifying a location, then this data will get stored in the DBFS Root storage location. For this reason, we recommend that you always specify the location where you want the data to be stored. For example, the following code is writing out a file, but no location is specified. As a result, this data would get stored in DBFS Root.



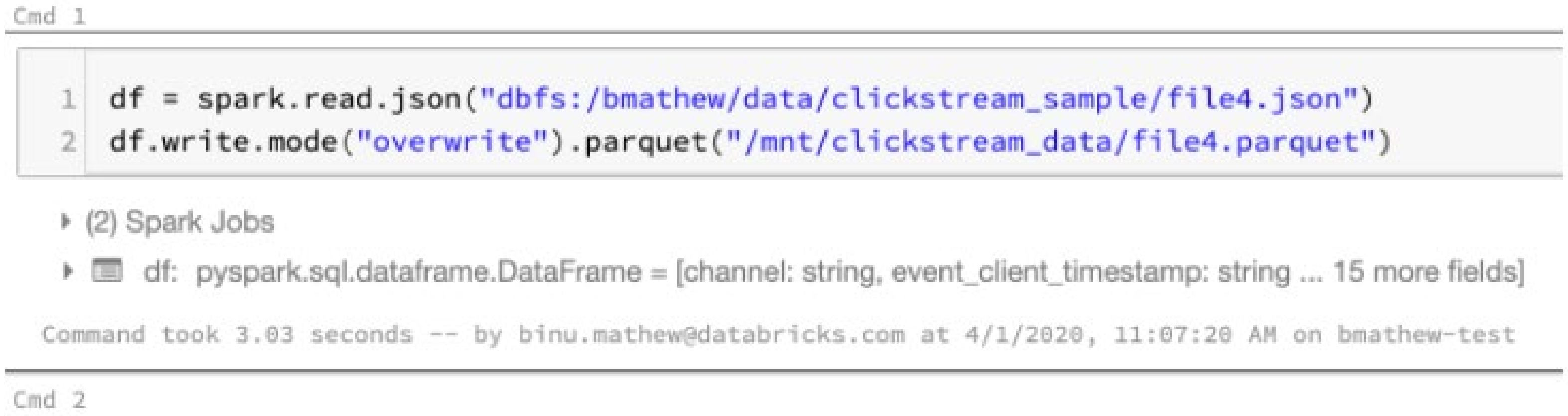
```
df = spark.read.json("dbfs:/bmathew/data/clickstream_sample/file4.json")
df.write.mode("overwrite").parquet("file4.parquet")
```

▸ (2) Spark Jobs

▸ df: pyspark.sql.dataframe.DataFrame = [channel: string, event_client_timestamp: string ... 15 more fields]

```
Command took 3.34 seconds -- by binu.mathew@databricks.com at 4/1/2020, 11:04:32 AM on bmathew-test
```

You should always specify a mount point that is not DBFS Root if you don't want the data stored there.



Cmd 1

```
df = spark.read.json("dbfs:/bmathew/data/clickstream_sample/file4.json")
df.write.mode("overwrite").parquet("/mnt/clickstream_data/file4.parquet")
```

▸ (2) Spark Jobs

▸ df: pyspark.sql.dataframe.DataFrame = [channel: string, event_client_timestamp: string ... 15 more fields]

```
Command took 3.03 seconds -- by binu.mathew@databricks.com at 4/1/2020, 11:07:20 AM on bmathew-test
```

Cmd 2

Please refer to the online documentation for more information:

DBFS and DBUtils

AWS AZURE

DBFS Root

AWS AZURE

Please refer to the notebook "Connecting to cloud storage" for examples of how to connect to cloud storage. The notebook will be submitted with these documents:

AWS AZURE

CHAPTER 3: APPLICATION DEVELOPMENT, TESTING AND DEPLOYMENT

Data migration

Databricks is optimized for cloud object storage: S3 in Amazon Web Services, Blob and Azure Data Lake Storage Generation 2 in Microsoft Azure, and Google Cloud Storage. In addition to cloud storage, Databricks can also read/write to other storage end points, including relational databases (Oracle, SQL Server, Teradata), HDFS, Apache Hive, NoSQL (HBase, Cassandra, Neo4j, MongoDB), in-memory cache (Redis, RocksDB), message bus (Kafka), files (delimited text files, JSON, Parquet, ORC, Avro) and many others. Data sources can be in the cloud or on-premises, but Databricks is optimized for the cloud.

To get started with data migration, first look at a dual ingestion strategy. You may already have a defined process to land data into Hadoop. This might be implemented via a third-party ingestion tool or perhaps an in-house built framework. A simple approach could be to fork the target such that data is landed to both HDFS and cloud storage. Getting an initial data feed provides an additional backup location of your data. It will also allow you to unlock new advanced analytics in the cloud with Databricks.

The next step is migration of historical data. This step may take some time based on the amount of data that exists in HDFS. If possible, try to align data sets with prioritized use cases that need to be migrated away from Hadoop. This will help identify the order in which you'll need to move data to the cloud.

There are two ways to move historical data from HDFS to the cloud: the push approach and the pull approach. The former is more commonly used than the latter, as data owners and information security teams have more control over how and when the data is sent to the cloud. Some customers may opt for the pull approach in scenarios where workflows need to be managed solely from the cloud.

Here are some options for migrating data from HDFS into the cloud.

For data volumes up to the hundreds of terabytes:

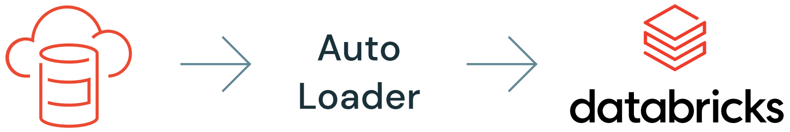
- Create a secure VPN connection between on-premises and cloud, or use a private dedicated connection offered by a cloud service provider (CSP)
- Stream IoT data into a message bus
- Use cloud service provider utilities and other tools (e.g., DistCp) to move files from Hadoop storage (HDFS) into cloud storage
- Use third-party tools such as WANdisco, which have the ability to synchronize HDFS with cloud storage

wanDISCO
syncsort
Knowledge Lens Accelerating Actionable Insights
StreamSets
Infoworks
ATTUNITY A Division of Qlik

For data volumes in the petabyte range or higher:

- Use a CSP service that allows you to mail your disks to them and which then loads them into cloud storage
- Use a CSP service that will transport your disks from your data centers to theirs and then load them into cloud storage
- Some CSPs may offer other data migration services — look for one that best meets your needs

Once your data has been loaded into cloud storage, Databricks provides a service, Auto Loader, to quickly and easily ingest your data into Databricks:



- Auto Loader in AWS
- Auto Loader in Azure

Migrating data to the cloud can be time consuming and challenging for your data teams. Databricks partners with vendors that provide tools to securely automate the migration of your data from on-premises storage to cloud storage. This can deliver significant economies of scale to assist with the migration to Databricks cloud — including:

- Reduction in costs
- At least a 2x–3x acceleration in migration timelines compared to a manual approach
- Overall reduction in end-to-end migration timelines

Please contact Databricks for more information.

For Azure, choose the right solution for data transfer.
For AWS, see the data migration services.

CHAPTER 3: APPLICATION DEVELOPMENT, TESTING AND DEPLOYMENT

Hive metastore

The metastore RDBMS is the central repository of Apache Hive metadata. It stores metadata for tables that include schema information such as column names, data types, location of data files, partitions and other metadata. Databricks uses a Hive metastore to store metadata for Spark SQL tables. The default configuration in Databricks is to create a managed Hive metastore that it maintains. This is administered by Databricks per workspace. Customers can choose to use an existing Hive metastore and integrate with Databricks as an external Hive metastore. Various versions of the Hive metastore are supported and a variety of back-end RDBMSs can be used, including MySQL, PostgreSQL, Oracle and SQL Server. Using SQL Server as the back-end Hive metastore is common with existing Hadoop customers, and Databricks supports using SQL Server with the latest version of Hive (3.1 at the time this guide was written).

The default Hive metastore with Databricks, which is the one Databricks hosts, currently has a limit of 250 connections from each workspace to our hosted Hive metastore. Each Databricks cluster opens at least two metastore connections. Thus, if there are 125 clusters running at the same time in the workspace, then the limit of 250 for the number of connections will be reached. For a customer using their own external Hive metastore, there is no known limit to the number of connections from a workspace to their own external Hive metastore. Since the customer manages the external Hive metastore, they can adjust the database server settings to control the number of connections.

For more information, see these sections within this guide:

AWS external Hive metastore integration
Azure external Hive metastore integration

Migrating the Hive metastore

If you have an existing Hive metastore from which you want to migrate some or all of your table definitions to Databricks, you can use RDBMS functionality and Hive commands to accomplish this. One way to do this is to migrate the entire existing Hive metastore to a new RDBMS.

1. On the current (old) RDBMS, take a dump of the Hive database and write this to a file. For example, if using MySQL and your Hive database is named "hive," then the command would be: mysqldump -u root hive >> my_dump_outputfile.sql

2. You will have to do some search and replace in the file for compatibility with Databricks

   - The location of the table data files needs to correspond to a location in Databricks. For the location path, you can use a mount point location, such as /path_to_my_directory (see the "Data sources" section to learn more about mount points), or use the cloud service provider's file system client when specifying the location, such as s3a://my_bucket/path_to_my_files, if using S3 in AWS. Write a search and replace function to adjust the location paths.
   - You can continue to use ORC files in Databricks, but keep in mind that Databricks is optimized for the Delta Lake file format, which uses open source Parquet, and we recommend using Delta Lake. Please refer to the "Delta Lake" section for more information on this format and the "Spark SQL" section for table creation commands.

- If you are using Parquet instead of Delta Lake and want to continue with the pure open source Parquet format, then the Hive style syntax “STORED AS” is not recommended in Databricks. The correct syntax to use in Databricks is “USING.” Write a search and replace function to make this change. Please refer to the “Spark SQL” section for table creation commands.

3. There may be some other search and replace you have to do, but the above examples are the most common issues you will face for Hive to Spark SQL compatibility.

4. After the search and replace is done, on the new RDBMS, create the Hive database. For example, if using MySQL, the command would be: create database hive;

5. Run the Hive schematool to initialize the schema to the exact version of the Hive schema on the old metastore. For example, if using MySQL, then this would be: $HIVE_HOME/bin/schematool -dbType mysql -initSchemaTo <hive_version> -userName <user_name> -passWord <password> -verbose

6. Import the Hive database from the file. For example, if using MySQL, this would be: mysql -u root hive < my_dump_outputfile.sql

7. Upgrade the schema to the latest schema version: $HIVE_HOME/bin/schematool -upgradeSchema -dbType mysql -userName <user_name> -passWord <password> -verbose

To migrate only schemas (databases and tables), you can generate a DDL file using the following Spark code — adapting data location and, if necessary, syntax — then export the DDL file into the new metastore.

```
dbs = spark.catalog.listDatabases()
for db in dbs:
  f = open(“your_file_name_{}.ddl”.format(db.
name), “w”)
 tables = spark.catalog.listTables(db.name)
  for t in tables:
    DDL = spark.sql(“SHOW CREATE TABLE {}.{}”.
format(db.name, t.name))
    f.write(DDL.first()[0])
    f.write(“\n”)
f.close()
```

For tables that will not be migrated to Delta tables, you will need to run MSCK REPAIR TABLE.

AWS external Hive metastore integration

If you are already using AWS, then you might also be using the AWS Glue Data Catalog. Databricks supports using the Glue Data Catalog as the metastore. Please refer to the online documentation for more information on how to configure this integration.

Please refer to the notebook "External MySQL Hive Metastore Integration With Databricks.dbc" for an example of how to use an existing Hive 3.1.0 metastore on MySQL with Databricks. The notebook will be submitted with this document.

Databricks also supports using SQL Server as the back end for the Hive metastore with the latest version of Hive. Please refer to the notebook "External SQL Server Hive Metastore Integration With Databricks.dbc" for an example of how to use an existing Hive 3.1.0 metastore on SQL Server with Databricks. The notebook will be submitted with this document.

Please refer to the online documentation for more information on using an external Hive metastore with Databricks.

Azure external Hive metastore integration

Please refer to the notebook "External SQL Server Hive Metastore Integration With Azure Databricks.dbc" for an example of how to use an existing Hive 3.1.0 metastore on SQL Server with Azure Databricks. The notebook will be submitted with this document.

Please refer to the notebook "External MySQL Hive Metastore Integration With Azure Databricks.dbc" for an example of how to use an existing Hive 3.1.0 metastore on MySQL with Azure Databricks. The notebook will be submitted with this document.

Please refer to the online documentation for more information on using an external Hive metastore with Azure Databricks.

CHAPTER 3: APPLICATION DEVELOPMENT, TESTING AND DEPLOYMENT

HiveQL vs. Spark SQL

Apache Hive is a data warehouse software project that was initially built for the Hadoop ecosystem. Hive can be used on-premises and in the cloud with a variety of storage mediums, including HDFS, Azure cloud storage, Amazon Web Services S3 object storage and Google Cloud Storage. Hive provides an abstraction layer that represents the data as tables with rows, columns and data types to query and analyze using a SQL interface called HiveQL. Hive uses an in-memory distributed engine called Apache Tez to process the data.

Apache Hive supports transactions (ACID) with Hive LLAP. Transactions guarantee consistent views of the data in an environment in which multiple users/processes are accessing the data at the same time for Create, Read, Update and Delete (CRUD) operations. Databricks offers Delta, which is similar to Hive LLAP in that it provides transaction (ACID) guarantees, but it offers several other benefits to help with performance and reliability when accessing the data. Delta is an open source project. More information about Delta can be found later in this guide.

Spark SQL is Apache Spark's module for interacting with structured data represented as tables with rows, columns and data types. Spark SQL is SQL 2003 compliant and uses Apache Spark as the distributed engine to process the data. In addition to the Spark SQL interface, a DataFrames API can be used to interact with the data using Java, Scala, Python and R.

Spark SQL is similar to HiveQL. Both use ANSI SQL syntax, and the majority of Hive functions will run on Databricks. This includes Hive functions for date/time conversions and parsing, collections, string manipulation, mathematical operations and conditional functions. There are some functions specific to Hive that would need to be converted to the Spark SQL equivalent or that don't exist in Spark SQL on Databricks. You can expect all HiveQL ANSI SQL syntax to work with Spark SQL on Databricks. This includes ANSI SQL aggregate and analytical functions.

Hive is optimized for the Optimized Row Columnar (ORC) file format and also supports Parquet. Databricks is optimized for Parquet and Delta. We always recommend using Delta, which uses open source Parquet as the file format.

Example of a HiveQL table creation using HDFS

```
CREATE EXTERNAL TABLE CUSTOMER_DB.CUSTOMER
(USER_ID INT, USER_NAME STRING) STORED AS
PARQUET
LOCATION ‘/data/customer’;
```

Spark SQL on Databricks table creation using object storage

```
CREATE TABLE CUSTOMER_DB.CUSTOMER (USER_ID INT,
USER_NAME STRING)
STORED AS PARQUET
LOCATION ‘/data/customer‘;
```

You don’t need to use the keyword EXTERNAL. Once you specify a location, the table automatically becomes an external table. For the location path, you can use a mount point location, such as /path_to_my_directory (see the “Data sources” section to learn more about mount points), or use the cloud service provider’s file system client when specifying the location, such as s3a://my_bucket/path_to_my_files, if using S3 in AWS.

Or

```
DROP TABLE IF EXISTS CUSTOMER_DB.MY_TABLE;
CREATE TABLE BMATHEW.MY_TABLE (USER_ID INT,
USER_NAME STRING)
USING DELTA
LOCATION ‘/data/customer‘;
```

Again, you don’t need to use the keyword EXTERNAL. The key difference between Hive and Spark SQL on Databricks when creating tables is that Hive syntax uses “stored as” whereas Databricks uses “using.” If you are using Hive LLAP today and migrating to Databricks, then we strongly recommend that you use Delta — “USING DELTA.” Delta provides transactions (ACID), is open source and will improve your data engineering, data science and BI workloads with improved performance, reliability and consistency when accessing the data.

There are also many options that you can set for table configuration. Here are a few common ones Hive users are familiar with that also work with Spark SQL on Databricks.

LOCATION — This is the cloud storage location where the data files will be stored. The default path will always be inside the default root blob storage account at /user/hive/warehouse/. Without specifying a location, the table will be created as a managed table — meaning that once you drop the table, all the data files will also be deleted. When you specify a location, the table becomes an unmanaged or external table — meaning that once you drop the table, the data remains in the directory. For the location path, you can use a mount point location, such as /path_to_my_directory (see the “Data sources” section to learn more about mount points), or use the cloud service provider’s file system client when specifying the location, such as s3a://my_bucket/path_to_my_files, if using S3 in AWS.

PARTITIONED BY — To partition the table by one or more columns. Always choose partition columns with a lower number of distinct values (low cardinality).

CLUSTERED BY — For columns with a high number of values (high cardinality), bucketing could help with performance.

TBL PROPERTIES — These are additional settings similar to those in Hive that let you specify certain configurations.

Here is an example using the above properties to create a table in Databricks using Parquet:

```
DROP TABLE IF EXISTS BMATHEW.MY_TABLE;
CREATE TABLE BMATHEW.MY_TABLE (
USER_ID INT,
USER_NAME STRING,
TRANSACTION_DATE DATE)
USING PARQUET
PARTITIONED BY (TRANSACTION_DATE)
CLUSTERED BY (USER_ID) SORTED BY (USER_ID) INTO
32 BUCKETS
LOCATION ‘/tmp/bmathew/test_hive_data’
TBLPROPERTIES (‘compression’=’snappy’,
‘owner’=’bmathew’);
```

It’s important to note that bucketing on Databricks is supported only when using Parquet, not Delta.

To view the properties of a table including the schema definition:

```
DESCRIBE FORMATTED BMATHEW.MY_TABLE;
```

The Hive style syntax will also work on Databricks:

```
CREATE TABLE my_table STORED AS PARQUET AS
(select 1 as user_id);
```

However, we recommend that you don’t use the Hive style syntax (i.e., **stored as parquet**). The syntax **using parquet** is specific to Spark SQL, and these tables will always use Databricks optimizations outside of open source for the Spark SQL Catalyst optimizer. By contrast, **stored as parquet** can be used for both Spark and Hive, but not all Databricks-specific Spark SQL optimizations may work as expected. Thus, we recommend using “USING PARQUET” if you don’t want to use Delta.

Please refer to the online documentation for more information:

Databricks databases and tables

AWS AZURE

Spark SQL

AWS AZURE

Hive compatibility

AWS AZURE

We recommend that you use Delta to store your data. Please read the next section, “Delta Lake to optimize data pipelines,” for more information and a notebook example.

CHAPTER 3: APPLICATION DEVELOPMENT, TESTING AND DEPLOYMENT

Delta Lake to optimize data pipelines

Hadoop provides several distributed programming frameworks to process your data. They include the legacy low-level MapReduce API, and higher-level frameworks such as Pig and Hive (with Tez). Hadoop also supports Spark. The Databricks Delta Engine makes data processing easy using Spark because the combination of Spark and Databricks delivers 6x faster performance improvement over open source Spark. Delta Engine is a 100% Apache Spark–compatible vectorized query engine that significantly accelerates query performance on Delta Lake and makes it easier for you to adopt and scale a lakehouse architecture.

Apache Hive supports transactions (ACID) with Hive LLAP. Transactions guarantee consistent views of the data in an environment in which multiple users and processes are accessing the data at the same time for Create, Read, Update and Delete (CRUD) operations. Databricks offers Delta Lake, which is similar to Hive LLAP in that it provides transaction (ACID) guarantees, but it offers several other benefits to help with performance and reliability when accessing the data. Delta is an open source project.

We recommend using Delta Lake when creating Spark SQL tables in Databricks. Delta is an open source storage format that creates an optimized Spark SQL table and offers these advantages:

- Uses open source Parquet as the underlying file format
- Creates a transaction log for the Parquet files
- Has ACID properties (that relational databases offer) to support transactions
- Guarantees consistency and reliability of the data by allowing multiple users and processes to access the data simultaneously for Create, Read, Update and Delete (CRUD) operations
- Employs data-skipping indexes to improve read performance
- Caches data on the local SSD drives of the VMs so that subsequent reads of the data will be fetched from disk on the VMs without connecting back to cloud storage. This feature will significantly speed up query performance and only works when launching the storage-optimized VMs.
- Provides versioning of the data to enable time travel (e.g., rollback)
- Has schema enforcement
- Has schema evolution
- Uses clustered indexes known as Z-Ordering
- Although it does not support bucketing, Delta's use of partitions, data skipping indexes and Z-Ordering help with performance

Please refer to the notebook "Delta on Databricks.dbc" for how to use Delta to optimize your data processing workloads. The notebook will be submitted with this document:

AWS AZURE

Please refer to the online documentation for more information about Delta on Databricks:

AWS AZURE

Please refer to the online documentation for more information on migrating existing data to Delta format:

AWS AZURE

We recently announced Delta Live Tables (DLT). This feature makes it easy to build and manage reliable data pipelines that deliver high-quality data on Delta Lake. DLT helps data engineering teams simplify ETL development and management with declarative pipeline development, automatic data testing and deep visibility for monitoring and recovery. This technology can drastically simplify existing Hadoop data pipelines, which lack built-in visibility, data quality and lineage information.

For more information, see the Delta Live Tables page.

CHAPTER 3: APPLICATION DEVELOPMENT, TESTING AND DEPLOYMENT

User-defined functions

Customers often have their own user-defined functions (UDFs) implemented in Hive to extend its built-in functionality. UDFs allow developers to enable new functions in higher-level languages, such as SQL, by abstracting the lower-level languages in which they were written (e.g., Java, Scala). Spark on Databricks has options for integrating UDFs with Spark SQL.

With minor changes, you can use the same Java UDFs from Hive in Databricks, if needed. To do so, be sure to import this in your Java source code:

```
import org.apache.hadoop.hive.ql.exec.UDF;
```

The above package will always be needed. You might also need to import:

```
import org.apache.hadoop.io.*;
```

You'll need to upload the JAR file to DBFS, launch a cluster with the JAR file attached to the cluster, add the JAR file path in your SQL cell, and then create a temporary function. Please refer to the UDF examples in the Azure or AWS archive files. The notebook will be submitted with this document.

You can also create UDFs in Databricks using Python and Scala, and call them via Spark SQL:

Python

AWS Azure

Scala

AWS Azure

When you create UDFs in Databricks, as described in the previous links in this guide, they'll only be accessible in your SparkSession. For example, on a shared high-concurrency cluster, other users will not be able to access your UDF directly since they'll have their own SparkSession. If you want to share a common UDF, everyone will have to share the same SparkSession. In order for other users to access your UDF, you'll need to share the SparkSession by using the following configuration setting for the cluster:

```
spark.databricks.session.share true
```

You could also write code using Java, Scala or Python to create libraries that perform the same function and then attach those libraries to the cluster. In this manner you would not need to share the SparkSession.

It's important to note that UDFs are not vectorized, so they operate on data one row at a time. UDFs written in Java and Scala will perform better than those written in Python. Functions written in Java and Scala are used within the Java Virtual Machine (JVM), whereas with Python, Spark has to first serialize the data from the JVM process to a format that Python understands. This serialization degrades performance. If the performance is not acceptable, we recommend writing functions in Java or Scala — you can still call the function from Python.

Vectorized UDFs are possible in Databricks using pandas with Apache Arrow. Please refer to the online documentation for more information on using pandas UDFs:

AWS AZURE

CHAPTER 3: APPLICATION DEVELOPMENT, TESTING AND DEPLOYMENT

Sqoop

Sqoop is running MapReduce under the hood, and hence is one of the main reasons MapReduce is still deployed. Many customers have moved off Sqoop and started using Spark to read data directly from relational systems. The syntax to read from databases in Spark is very simple, and you have flexibility with how the data is processed and persisted to a target destination.

You can replace the Sqoop calls in Spark code using the JDBC source. This spark code will reside in a Databricks notebook or packaged in a code artifact (JAR, python whl, etc.).

See the online documentation. Here is an example call:

```
val jdbcDF = spark.read
  .format(“jdbc”)
  .option(“url”, “jdbc:postgresql:dbserver”)
  .option(“dbtable”, “schema.tablename”)
  .option(“user”, “username”)
  .option(“password”, “password”)
  .load()
```

Note, with Databricks, you can leverage Secrets to ensure credentials are not exposed in the code.

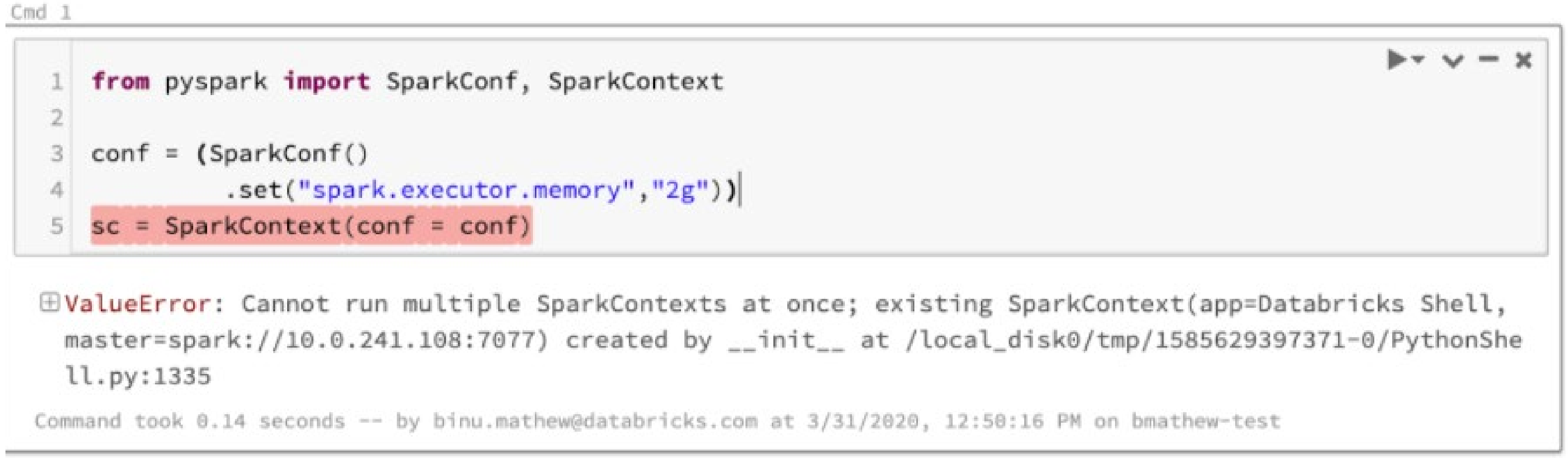
The Spark JDBC source also allows you to specify options similar to Sqoop, such as customized select query, fetch read and batch write sizes, and isolation settings.

Sqoop provides incremental loads via Sqoop Jobs, and internally it can track a field to determine new data. This field is typically a timestamp or may be an ever-increasing sequence ID. Sqoop will load new data into a target location and will persist the largest value for this field. This value is then used to retrieve new data from the source table. Spark does not support this functionality out of the box. You will need to track a “last modified timestamp” field or sequence ID in code and adjust the SQL query that is used to extract data from the JDBC source.

CHAPTER 3: APPLICATION DEVELOPMENT, TESTING AND DEPLOYMENT

Spark code development on Databricks

Hadoop users submitting Spark jobs to a Hadoop cluster via JAR files and scripts each get their own SparkContext, whereas Databricks shares a single SparkContext among all users on a Databricks Cluster. Both in Hadoop and on Databricks, each user gets their own SparkSession. When running a job on Databricks — either via Databricks notebooks or by uploading your own Java/Scala JAR files or Python scripts to DBFS (individual Python scripts or wheel or egg files) — the SparkContext is created for you. Since Databricks initializes the SparkContext, if you invoke a new SparkContext, your code will fail. For example, the following code will return an error:



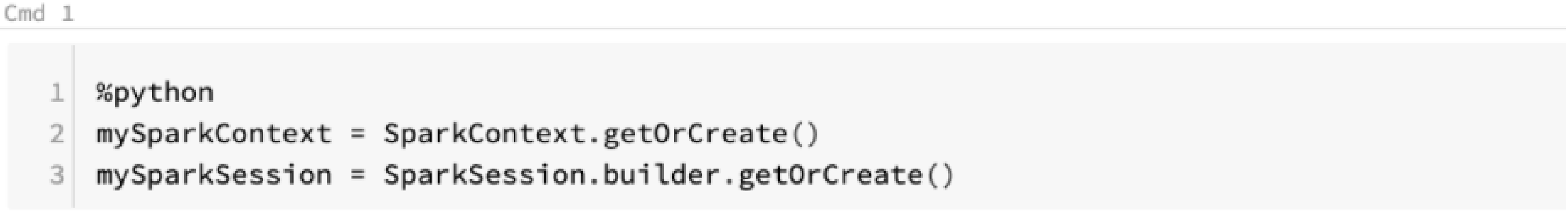
Cmd 1

```
from pyspark import SparkConf, SparkContext

conf = (SparkConf()
         .set("spark.executor.memory","2g"))
sc = SparkContext(conf = conf)
```

```
ValueError: Cannot run multiple SparkContexts at once; existing SparkContext(app=Databricks Shell, master=spark://10.0.241.108:7077) created by __init__ at /local_disk0/tmp/1585629397371-0/PythonShell.py:1335
```

Command took 0.14 seconds -- by binu.mathew@databricks.com at 3/31/2020, 12:50:16 PM on bmathew-test

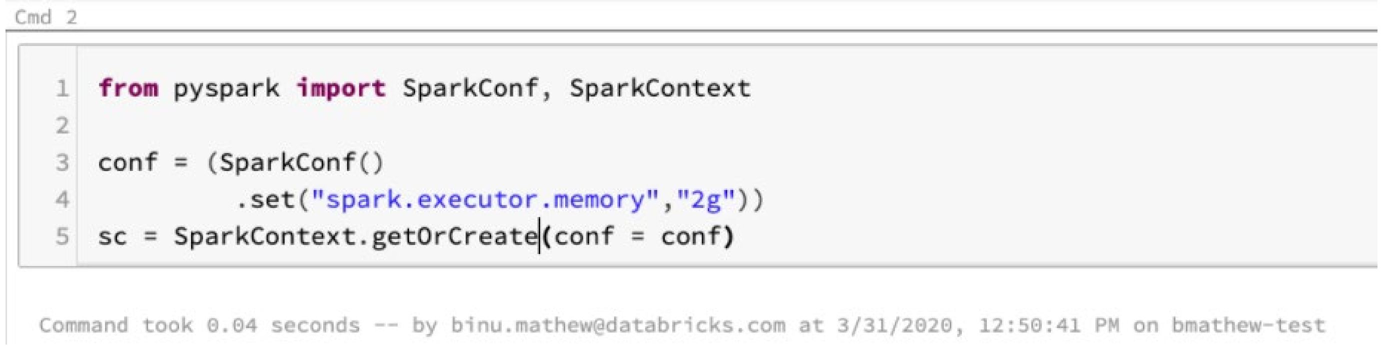
Use the shared SparkContext created by Databricks:



Cmd 1

```
%python
mySparkContext = SparkContext.getOrCreate()
mySparkSession = SparkSession.builder.getOrCreate()
```

Returning to our example, we would modify the code:



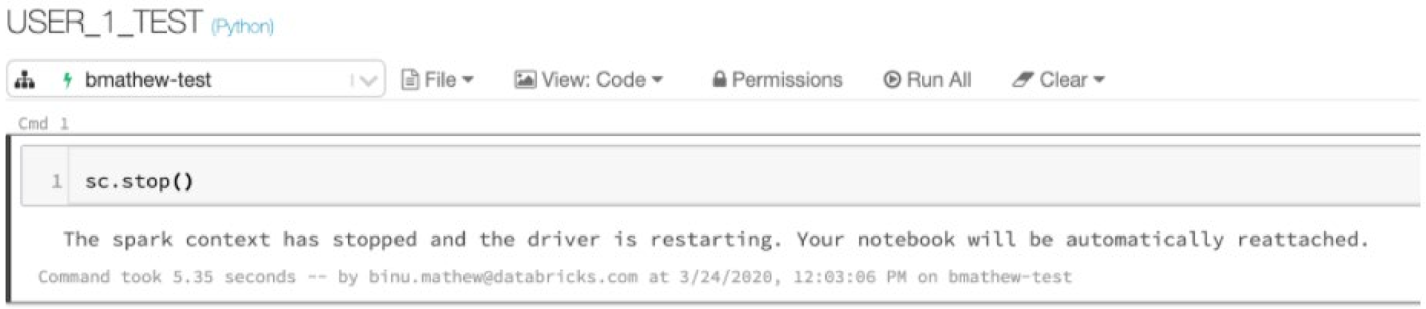
Cmd 2

```
from pyspark import SparkConf, SparkContext

conf = (SparkConf()
         .set("spark.executor.memory","2g"))
sc = SparkContext.getOrCreate(conf = conf)
```

Command took 0.04 seconds -- by binu.mathew@databricks.com at 3/31/2020, 12:50:41 PM on bmathew-test

Since Databricks creates a shared SparkContext for the cluster, you should not terminate the SparkContext, as this could impact other users who are running jobs on the same cluster. For example, let's look at two practitioners using the same cluster.

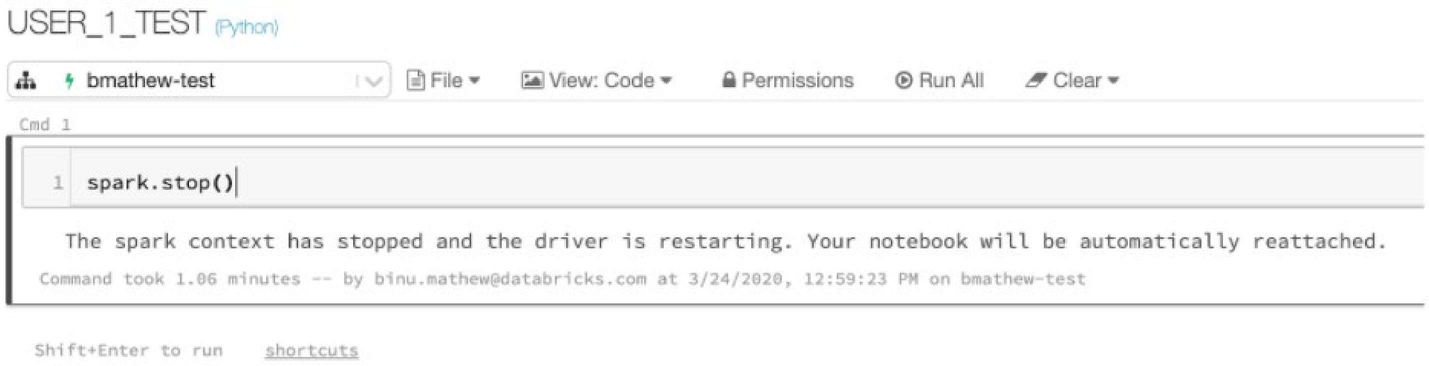
User 1 terminates the SparkContext by issuing these commands:



```
sc.stop()
```

```
The spark context has stopped and the driver is restarting. Your notebook will be automatically reattached.
Command took 5.35 seconds -- by binu.mathew@databricks.com at 3/24/2020, 12:03:06 PM on bmathew-test
```

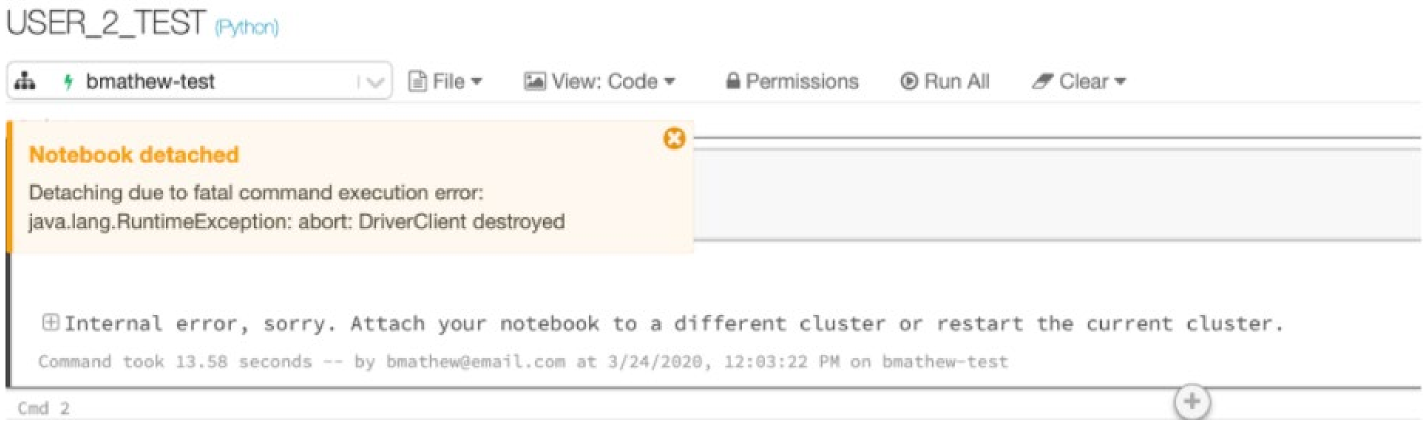
OR



```
spark.stop()
```

```
The spark context has stopped and the driver is restarting. Your notebook will be automatically reattached.
Command took 1.06 minutes -- by binu.mathew@databricks.com at 3/24/2020, 12:59:23 PM on bmathew-test
```

User 2 is trying to run a job but now receives an error message because the SparkContext has stopped:



Notebook detached

Detaching due to fatal command execution error:
java.lang.RuntimeException: abort: DriverClient destroyed

```
⊞Internal error, sorry. Attach your notebook to a different cluster or restart the current cluster.
Command took 13.58 seconds -- by bmathew@email.com at 3/24/2020, 12:03:22 PM on bmathew-test
```

Your job will run normally; however, it will end with the failure above.

For more information, please refer to the following documentation.

Let's look at few examples of how you might be creating a SparkContext in your code today:

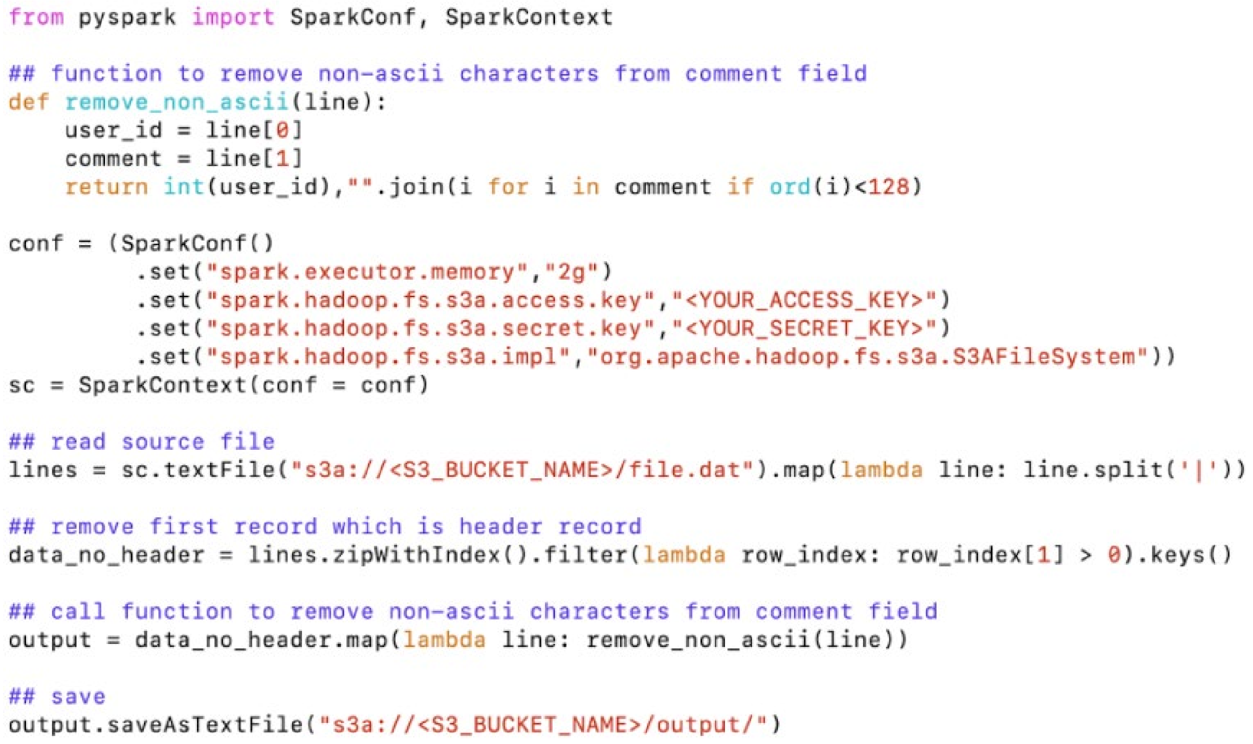
Example 1: Migrating Spark RDD code from Hadoop to Databricks

Example 2: Migrating Spark DataFrame code from Hadoop to Databricks

Example 1: Migrating Spark RDD code from Hadoop to Databricks

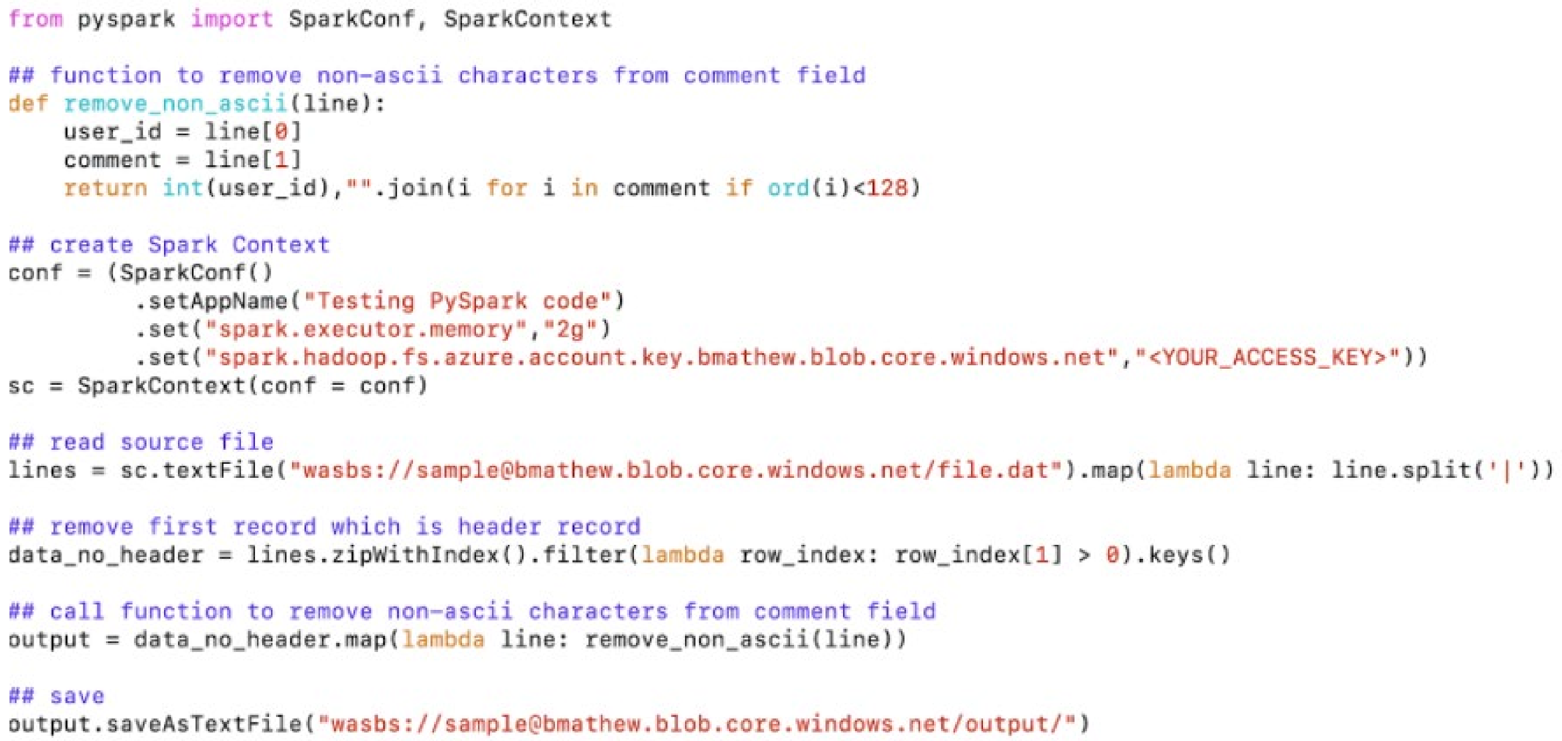
Existing Hadoop PySpark code using RDD API in a Python script needs to run on Databricks. The following is a working example of Hadoop PySpark code in a Python script using the RDD API to process data. Notice that in the code sample, a SparkContext is created. You might be doing this today in your code. In this example, we are accessing an Azure Blob storage account and need to set the credentials. There are different ways to do this, but here we are setting the credentials in the code. This could be when you are testing as a developer. Let's look at the same code in both AWS and Azure.

AWS:



```
from pyspark import SparkConf, SparkContext

## function to remove non-ascii characters from comment field
def remove_non_ascii(line):
    user_id = line[0]
    comment = line[1]
    return int(user_id),"".join(i for i in comment if ord(i)<128)

conf = (SparkConf()
         .set("spark.executor.memory","2g")
         .set("spark.hadoop.fs.s3a.access.key","<YOUR_ACCESS_KEY>")
         .set("spark.hadoop.fs.s3a.secret.key","<YOUR_SECRET_KEY>")
         .set("spark.hadoop.fs.s3a.impl","org.apache.hadoop.fs.s3a.S3AFileSystem"))
sc = SparkContext(conf = conf)

## read source file
lines = sc.textFile("s3a://<S3_BUCKET_NAME>/file.dat").map(lambda line: line.split('|'))

## remove first record which is header record
data_no_header = lines.zipWithIndex().filter(lambda row_index: row_index[1] > 0).keys()

## call function to remove non-ascii characters from comment field
output = data_no_header.map(lambda line: remove_non_ascii(line))

## save
output.saveAsTextFile("s3a://<S3_BUCKET_NAME>/output/")
```

Azure:



```
from pyspark import SparkConf, SparkContext

## function to remove non-ascii characters from comment field
def remove_non_ascii(line):
    user_id = line[0]
    comment = line[1]
    return int(user_id),"".join(i for i in comment if ord(i)<128)

## create Spark Context
conf = (SparkConf()
         .setAppName("Testing PySpark code")
         .set("spark.executor.memory","2g")
         .set("spark.hadoop.fs.azure.account.key.bmathew.blob.core.windows.net","<YOUR_ACCESS_KEY>"))
sc = SparkContext(conf = conf)

## read source file
lines = sc.textFile("wasbs://sample@bmathew.blob.core.windows.net/file.dat").map(lambda line: line.split('|'))

## remove first record which is header record
data_no_header = lines.zipWithIndex().filter(lambda row_index: row_index[1] > 0).keys()

## call function to remove non-ascii characters from comment field
output = data_no_header.map(lambda line: remove_non_ascii(line))

## save
output.saveAsTextFile("wasbs://sample@bmathew.blob.core.windows.net/output/")
```

For this code to run on Databricks, the changes required will depend on how you run the code:

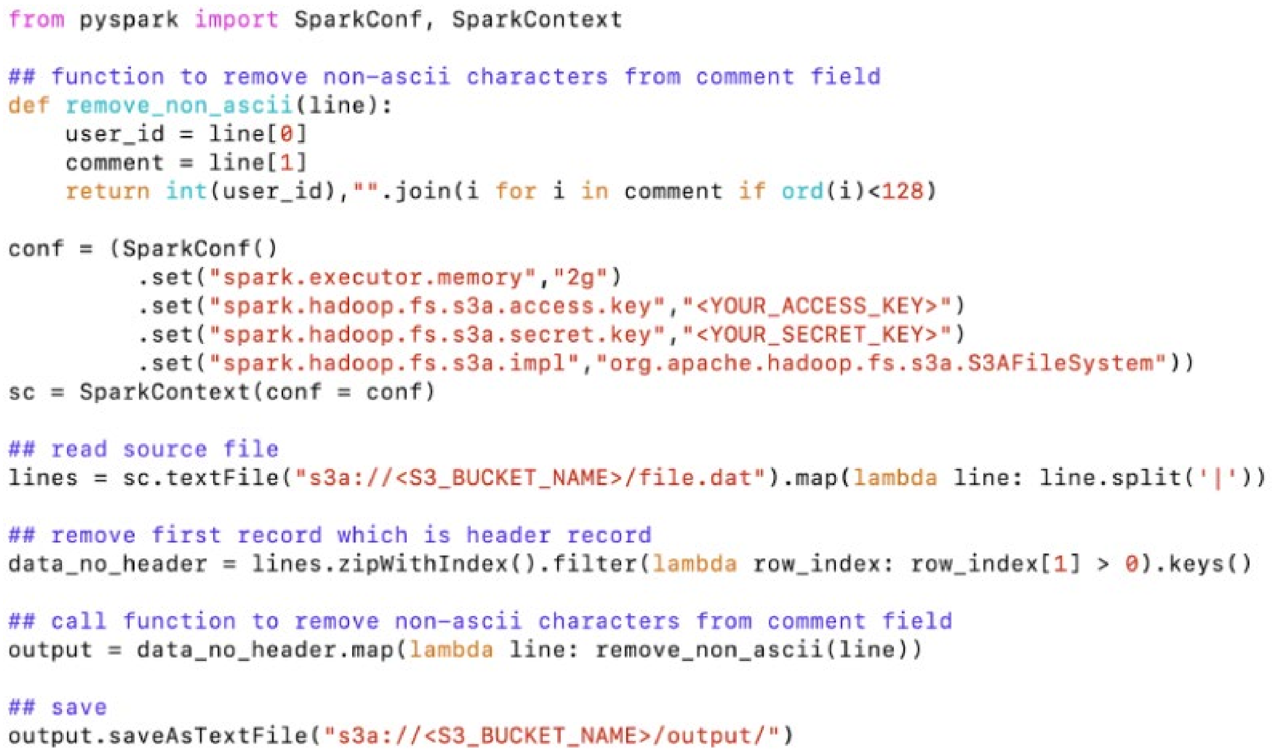
- Running on an existing cluster as a Databricks job
- Running on a new cluster as a Databricks job

Running on an existing cluster as a Databricks job

To run this same Python script on an existing Databricks cluster as a Databricks job using spark-submit, we need to edit the Python script and use the existing SparkContext that Databricks creates — i.e., SparkContext.getOrCreate. Also, you cannot set the application name in the same way that you would if running on Hadoop with Spark on YARN. Setting the application name will have no effect.

Here's a working example of the same code modified to run on Databricks, both for AWS and Azure. You would also make the same change for Java and Scala code. The rest of your code will remain the same when using any of the Spark APIs.

AWS:

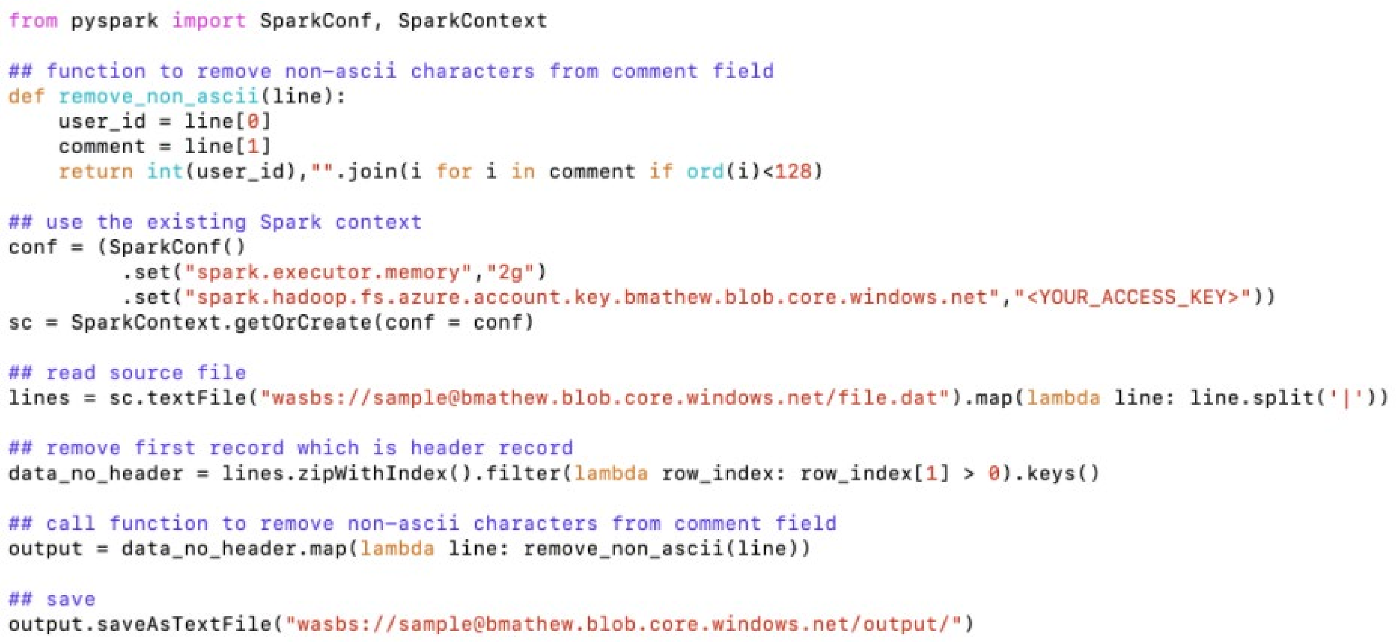


```
from pyspark import SparkConf, SparkContext

## function to remove non-ascii characters from comment field
def remove_non_ascii(line):
    user_id = line[0]
    comment = line[1]
    return int(user_id),"".join(i for i in comment if ord(i)<128)

conf = (SparkConf()
         .set("spark.executor.memory","2g")
         .set("spark.hadoop.fs.s3a.access.key","<YOUR_ACCESS_KEY>")
         .set("spark.hadoop.fs.s3a.secret.key","<YOUR_SECRET_KEY>")
         .set("spark.hadoop.fs.s3a.impl","org.apache.hadoop.fs.s3a.S3AFileSystem"))
sc = SparkContext(conf = conf)

## read source file
lines = sc.textFile("s3a://<S3_BUCKET_NAME>/file.dat").map(lambda line: line.split('|'))

## remove first record which is header record
data_no_header = lines.zipWithIndex().filter(lambda row_index: row_index[1] > 0).keys()

## call function to remove non-ascii characters from comment field
output = data_no_header.map(lambda line: remove_non_ascii(line))

## save
output.saveAsTextFile("s3a://<S3_BUCKET_NAME>/output/")
```

There are other ways to access S3 data sources from Databricks without using keys. Please refer to the "Data sources" section for more information. Notice in the previous example that we are entering the key. You can use the Secrets API to prevent the actual key value from being shown. Please refer to the online documentation for more information on the Secrets API.

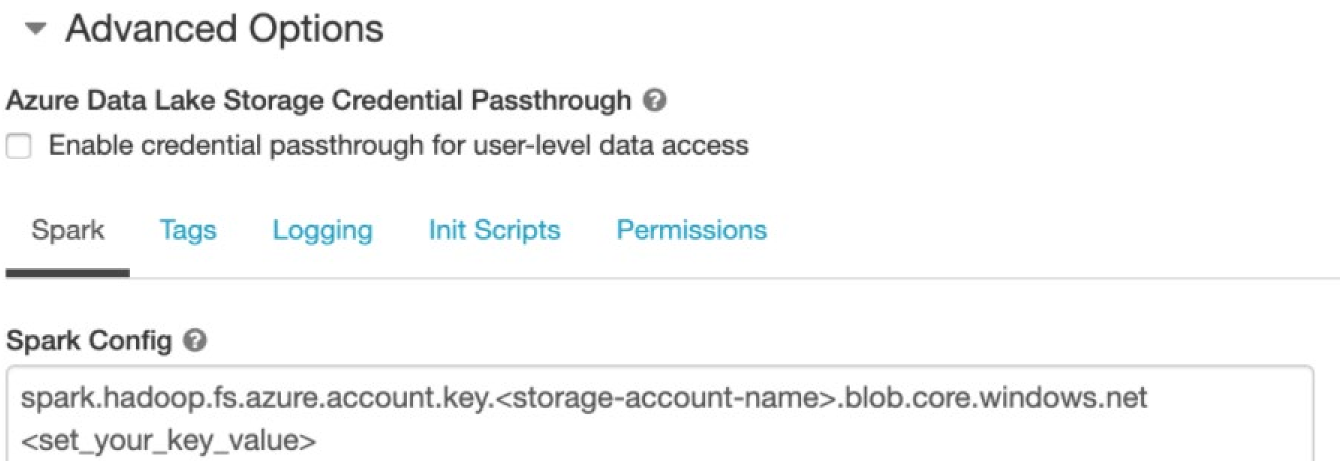
Please refer to the online documentation for more information on accessing S3 storage.

Azure:



```
from pyspark import SparkConf, SparkContext

## function to remove non-ascii characters from comment field
def remove_non_ascii(line):
    user_id = line[0]
    comment = line[1]
    return int(user_id),"".join(i for i in comment if ord(i)<128)

## use the existing Spark context
conf = (SparkConf()
         .set("spark.executor.memory","2g")
         .set("spark.hadoop.fs.azure.account.key.bmathew.blob.core.windows.net","<YOUR_ACCESS_KEY>"))
sc = SparkContext.getOrCreate(conf = conf)

## read source file
lines = sc.textFile("wasbs://sample@bmathew.blob.core.windows.net/file.dat").map(lambda line: line.split('|'))

## remove first record which is header record
data_no_header = lines.zipWithIndex().filter(lambda row_index: row_index[1] > 0).keys()

## call function to remove non-ascii characters from comment field
output = data_no_header.map(lambda line: remove_non_ascii(line))

## save
output.saveAsTextFile("wasbs://sample@bmathew.blob.core.windows.net/output/")
```

If you are running a Java/Scala JAR or Python script using the RDD API as a spark-submit job on a Databricks cluster, you can set the credentials in your code as shown in the previous example (similar to what you might be doing currently on Hadoop). However, this only works when using spark-submit to run the job. If you are using a Spark Python task to submit your Python RDD code as a job on Databricks or if you are running notebook code, then you cannot set the credentials in the code. For these use cases, you must define the Azure storage credentials as a Spark Config setting for the Databricks cluster as shown here:



□ Enable credential passthrough for user-level data access

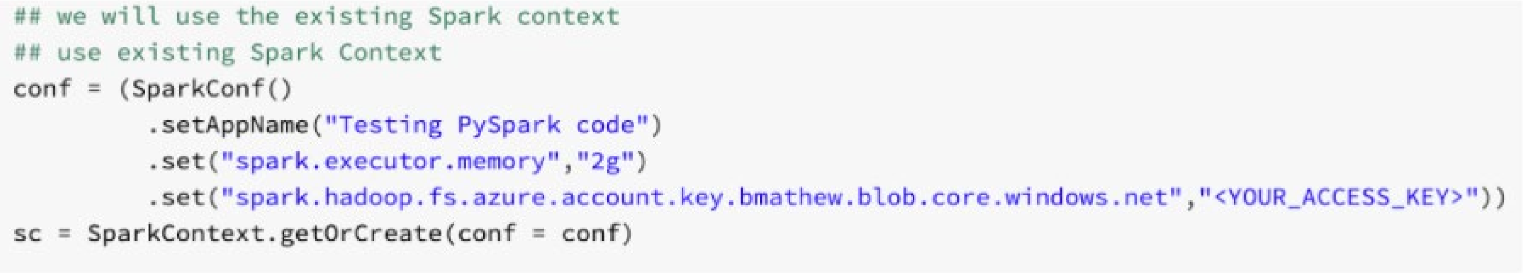
Spark Config

spark.hadoop.fs.azure.account.key.<storage-account-name>.blob.core.windows.net
<set_your_key_value>

Notice in this example that we are entering the key. You can use the Secrets API to prevent the actual key value from being shown. Please refer to the online documentation for more information on the Secrets API on Azure.

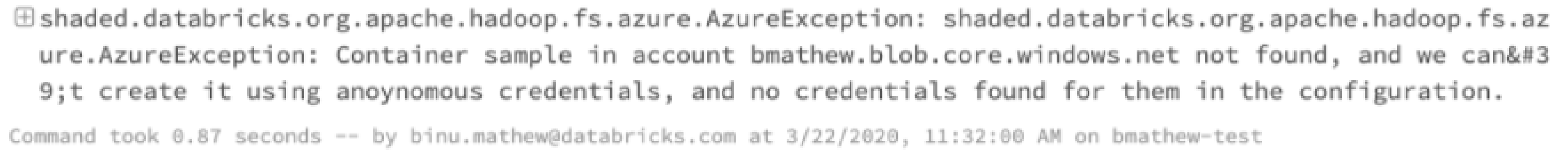
Let's look at what happens when we try to set the storage credentials directly in a notebook when using Python with the RDD API. The settings will be ignored, and you will get the following error.

Our notebook code setting the storage credentials:



```
## we will use the existing Spark context
## use existing Spark Context
conf = (SparkConf()
         .setAppName("Testing PySpark code")
         .set("spark.executor.memory","2g")
         .set("spark.hadoop.fs.azure.account.key.bmathew.blob.core.windows.net","<YOUR_ACCESS_KEY>"))
sc = SparkContext.getOrCreate(conf = conf)
```

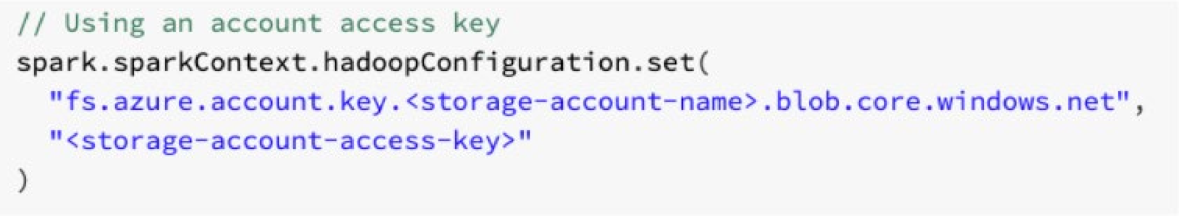
Error received:



```
⊞shaded.databricks.org.apache.hadoop.fs.azure.AzureException: shaded.databricks.org.apache.hadoop.fs.az
 ure.AzureException: Container sample in account bmathew.blob.core.windows.net not found, and we can&#3
 9;t create it using anoynomous credentials, and no credentials found for them in the configuration.
Command took 0.87 seconds -- by binu.mathew@databricks.com at 3/22/2020, 11:32:00 AM on bmathew-test
```

You will also get the same error if you try to run Python RDD code uploaded to DBFS and submit it as a Spark Python Task. If you are using Python with the DataFrames API, you can set the credentials both in your code and in notebooks. The next example will show how this is done.

If you are using Scala in a Databricks notebook with the RDD API, you can set the credentials for the storage account both in your code and from the notebooks. For example:



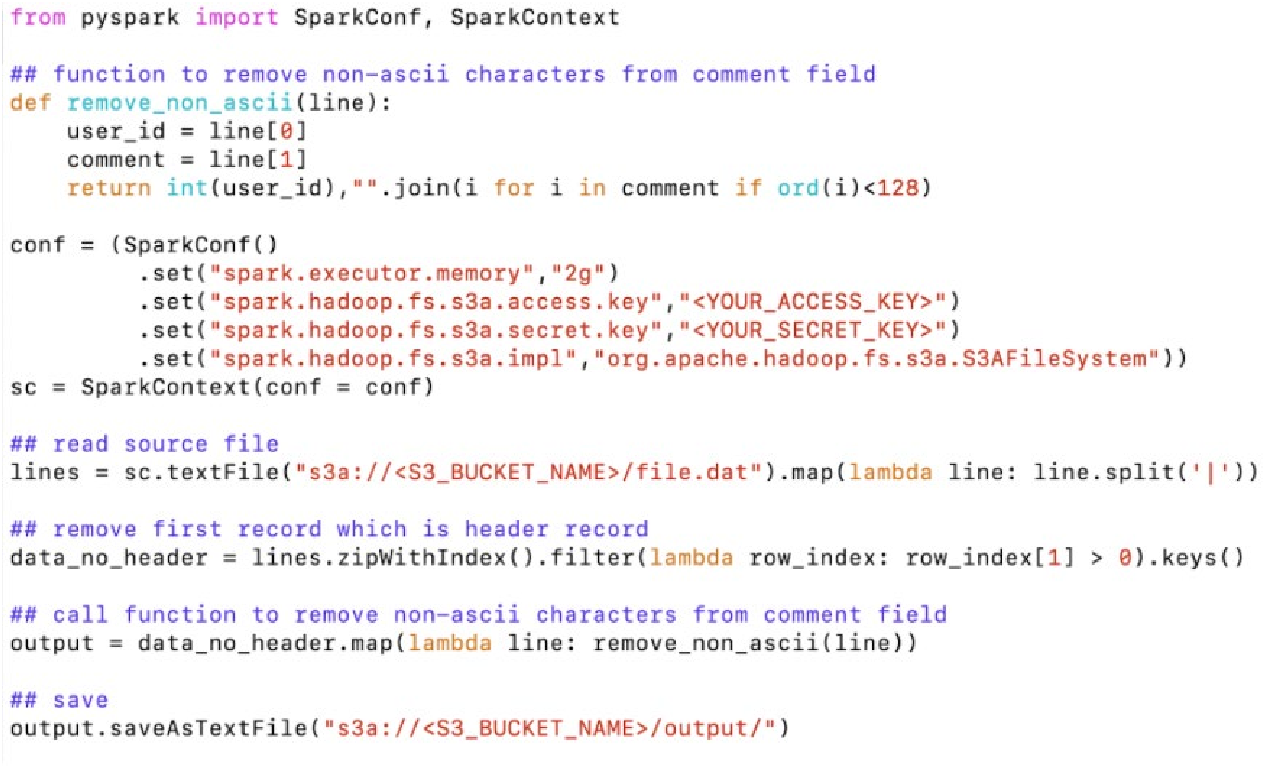
```
// Using an account access key
spark.sparkContext.hadoopConfiguration.set(
  "fs.azure.account.key.<storage-account-name>.blob.core.windows.net",
  "<storage-account-access-key>"
)
```

Please refer to the online documentation for more information on accessing cloud storage using the RDD API.

Running on a new cluster as a Databricks job

If we're running our Hadoop code on a new Databricks cluster that will be used once and only for this job before terminating — *and* if we're using spark-submit on Databricks — we do not need to change how we interact with SparkContext and can create a new SparkContext. This means our original code, which created the new SparkContext, will work — but only when using spark-submit on Databricks. If you are not using spark-submit on Databricks, you'll have to modify your code to use the existing SparkContext. For example, if you run the code as a job using Spark Python task (and not spark-submit), you will need to modify the code to use the existing SparkContext.

In AWS, we could change the code to the following — and in Azure, it will be similar:



```
from pyspark import SparkConf, SparkContext

## function to remove non-ascii characters from comment field
def remove_non_ascii(line):
    user_id = line[0]
    comment = line[1]
    return int(user_id),"".join(i for i in comment if ord(i)<128)

conf = (SparkConf()
         .set("spark.executor.memory","2g")
         .set("spark.hadoop.fs.s3a.access.key","<YOUR_ACCESS_KEY>")
         .set("spark.hadoop.fs.s3a.secret.key","<YOUR_SECRET_KEY>")
         .set("spark.hadoop.fs.s3a.impl","org.apache.hadoop.fs.s3a.S3AFileSystem"))
sc = SparkContext(conf = conf)

## read source file
lines = sc.textFile("s3a://<S3_BUCKET_NAME>/file.dat").map(lambda line: line.split('|'))

## remove first record which is header record
data_no_header = lines.zipWithIndex().filter(lambda row_index: row_index[1] > 0).keys()

## call function to remove non-ascii characters from comment field
output = data_no_header.map(lambda line: remove_non_ascii(line))

## save
output.saveAsTextFile("s3a://<S3_BUCKET_NAME>/output/")
```

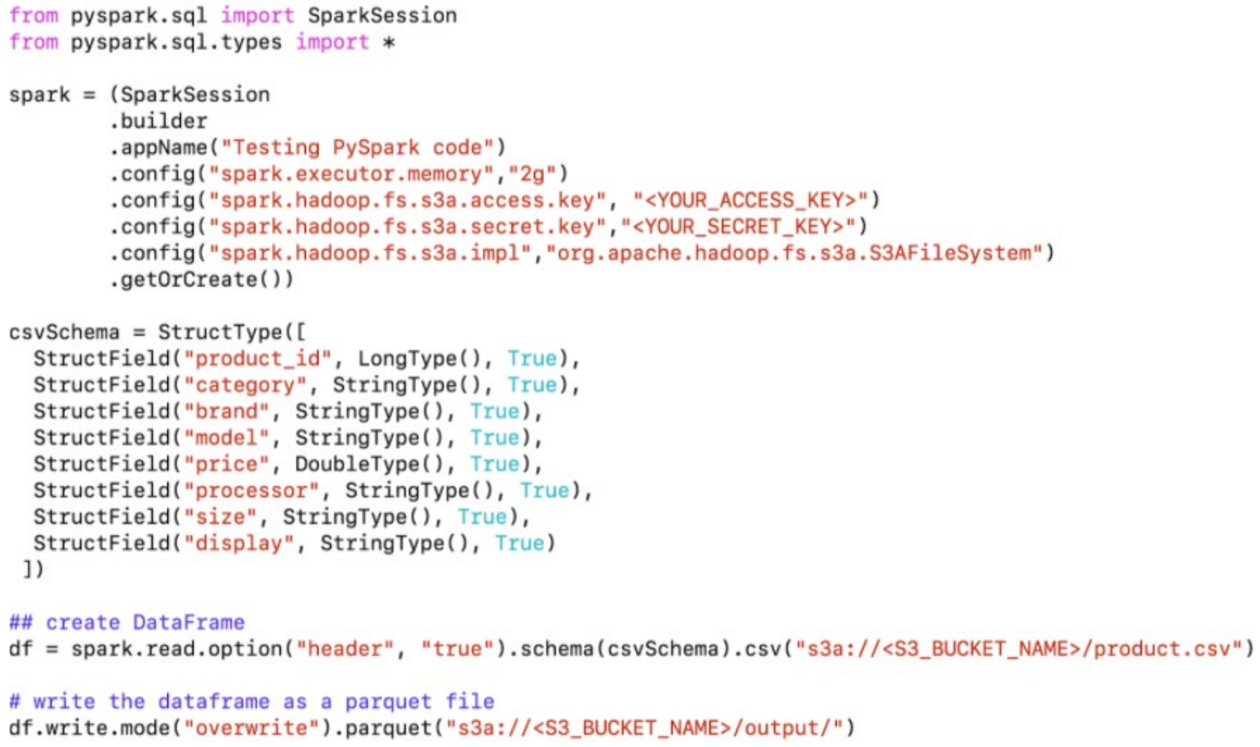
Example 2: Migrating Spark DataFrame code from Hadoop to Databricks

Existing Hadoop PySpark code written using the DataFrame API in a Python script needs to run on Databricks. The following are working examples of Hadoop PySpark code in a Python script using the DataFrame API to process data.

Using SparkSession

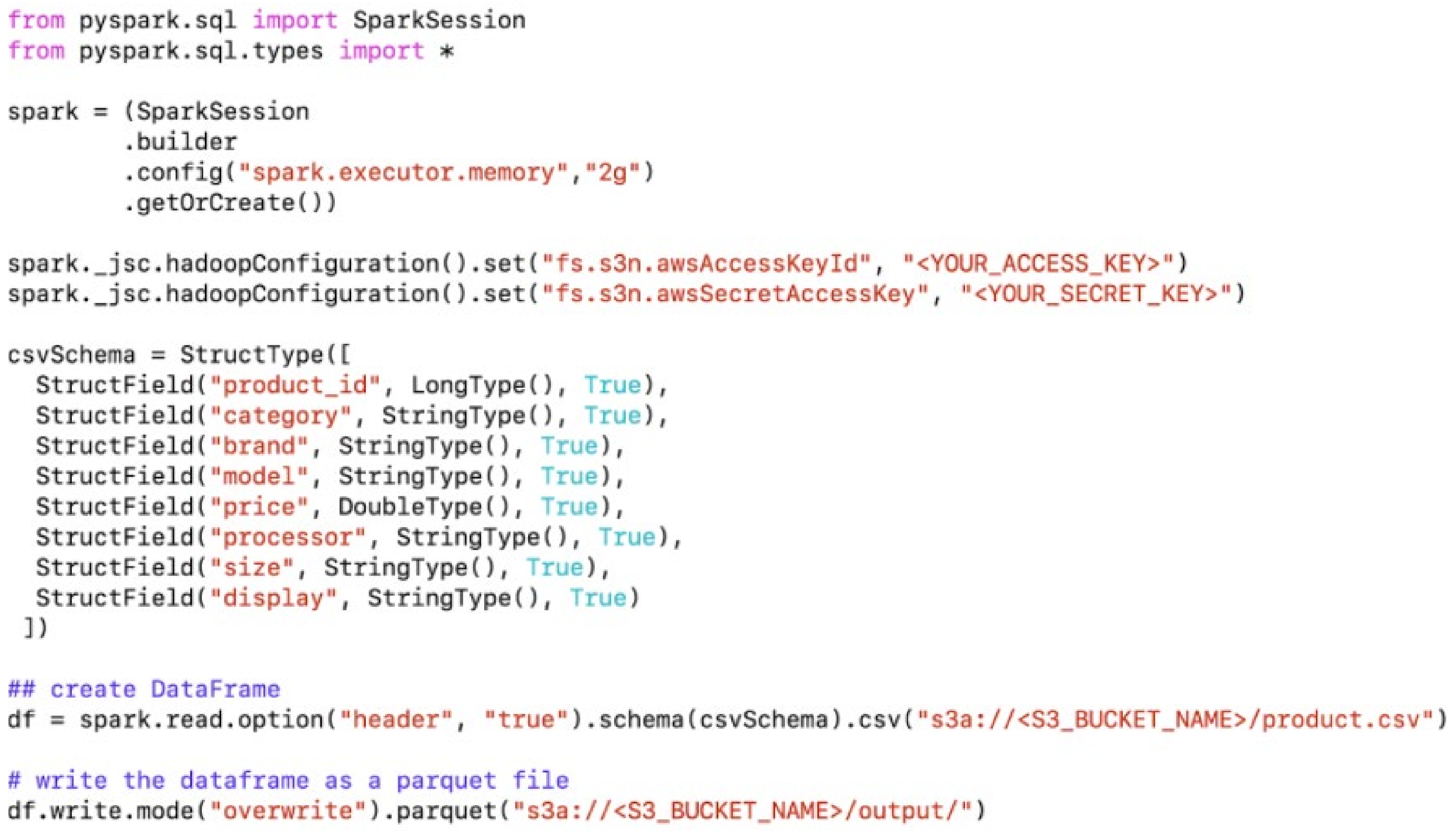
If you are already using the SparkSession in your Hadoop code, similar to the following code sample, you might not need to make any changes to your code. The SparkSession is a single entry point that lets you interact with Spark using the DataFrame and data set APIs. If you use the SparkSession, then you don't need to explicitly create SparkConf, SparkContext or SQLContext, as they are all encapsulated within the SparkSession. Here are two code examples, one in AWS and one in Azure.

AWS:



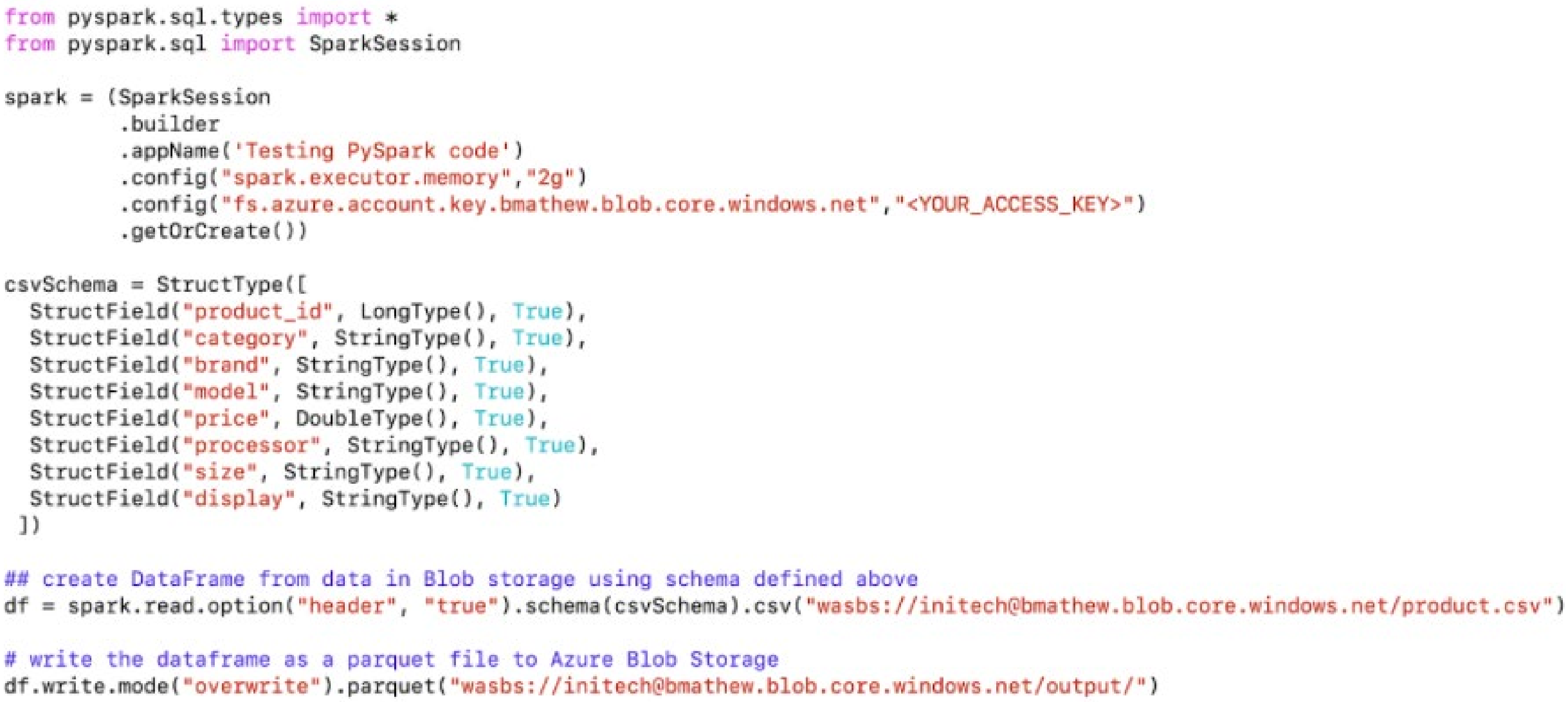
```python
from pyspark.sql import SparkSession
from pyspark.sql.types import *

spark = (SparkSession
        .builder
        .appName("Testing PySpark code")
        .config("spark.executor.memory","2g")
        .config("spark.hadoop.fs.s3a.access.key", "<YOUR_ACCESS_KEY>")
        .config("spark.hadoop.fs.s3a.secret.key","<YOUR_SECRET_KEY>")
        .config("spark.hadoop.fs.s3a.impl","org.apache.hadoop.fs.s3a.S3AFileSystem")
        .getOrCreate())

csvSchema = StructType([
  StructField("product_id", LongType(), True),
  StructField("category", StringType(), True),
  StructField("brand", StringType(), True),
  StructField("model", StringType(), True),
  StructField("price", DoubleType(), True),
  StructField("processor", StringType(), True),
  StructField("size", StringType(), True),
  StructField("display", StringType(), True)
 ])

## create DataFrame
df = spark.read.option("header", "true").schema(csvSchema).csv("s3a://<S3_BUCKET_NAME>/product.csv")

# write the dataframe as a parquet file
df.write.mode("overwrite").parquet("s3a://<S3_BUCKET_NAME>/output/")
```

If your Hadoop code is creating a SparkSession like the one shown here, you'll need to change how you set the S3 credentials. The same is true for Java and Scala code, although Java code has to be submitted as a JAR file to run as a job on Databricks since Java is not supported in the notebook. There are other ways to access S3 data sources from Databricks without using keys. Please refer to the "Data sources" section for more information. Remember, the application name that you set will be ignored in Databricks. The rest of your code will remain the same when using any of the Spark APIs.



```
from pyspark.sql import SparkSession
from pyspark.sql.types import *

spark = (SparkSession
        .builder
        .config("spark.executor.memory","2g")
        .getOrCreate())

spark._jsc.hadoopConfiguration().set("fs.s3n.awsAccessKeyId", "<YOUR_ACCESS_KEY>")
spark._jsc.hadoopConfiguration().set("fs.s3n.awsSecretAccessKey", "<YOUR_SECRET_KEY>")

csvSchema = StructType([
  StructField("product_id", LongType(), True),
  StructField("category", StringType(), True),
  StructField("brand", StringType(), True),
  StructField("model", StringType(), True),
  StructField("price", DoubleType(), True),
  StructField("processor", StringType(), True),
  StructField("size", StringType(), True),
  StructField("display", StringType(), True)
 ])

## create DataFrame
df = spark.read.option("header", "true").schema(csvSchema).csv("s3a://<S3_BUCKET_NAME>/product.csv")

# write the dataframe as a parquet file
df.write.mode("overwrite").parquet("s3a://<S3_BUCKET_NAME>/output/")
```

Azure:



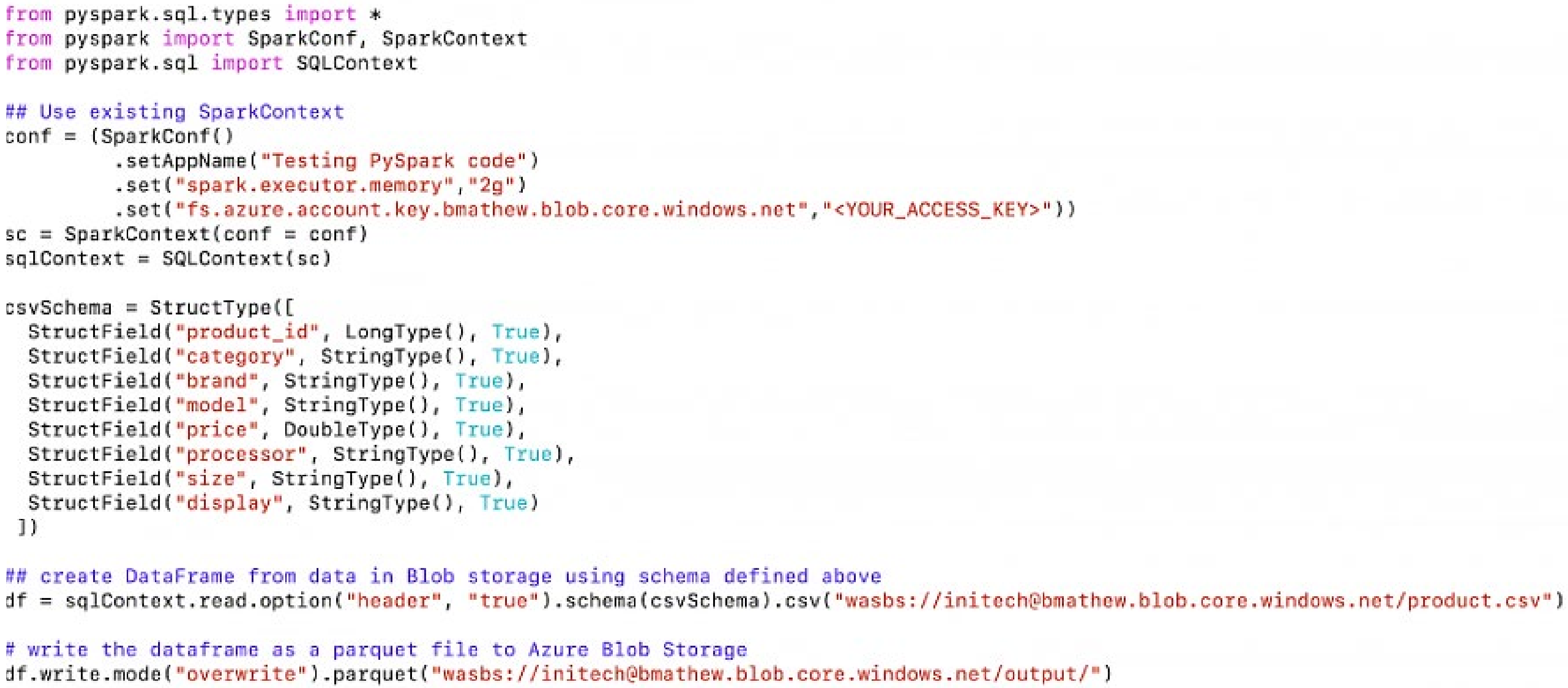
```
from pyspark.sql.types import *
from pyspark.sql import SparkSession

spark = (SparkSession
         .builder
         .appName('Testing PySpark code')
         .config("spark.executor.memory","2g")
         .config("fs.azure.account.key.bmathew.blob.core.windows.net","<YOUR_ACCESS_KEY>")
         .getOrCreate())

csvSchema = StructType([
  StructField("product_id", LongType(), True),
  StructField("category", StringType(), True),
  StructField("brand", StringType(), True),
  StructField("model", StringType(), True),
  StructField("price", DoubleType(), True),
  StructField("processor", StringType(), True),
  StructField("size", StringType(), True),
  StructField("display", StringType(), True)
 ])

## create DataFrame from data in Blob storage using schema defined above
df = spark.read.option("header", "true").schema(csvSchema).csv("wasbs://initech@bmathew.blob.core.windows.net/product.csv")

# write the dataframe as a parquet file to Azure Blob Storage
df.write.mode("overwrite").parquet("wasbs://initech@bmathew.blob.core.windows.net/output/")
```

If your Hadoop code is creating a SparkSession like the one shown here, this code will run exactly as is on Databricks — both as a Python script and as code executed from within a Databricks notebook. The same is true for Java and Scala code, although Java code has to be submitted as a JAR file to run as a job on Databricks since Java is not supported in the notebook. The only thing to note is that setting the application name is not supported on Databricks and will have no impact.

Let's look at an example in which we create a SparkContext in our Hadoop Spark code.

Using SparkContext

Here's a working example of our Hadoop PySpark code in which a SparkContext is explicitly created. The following example was run on Azure, but it also applies to other clouds.



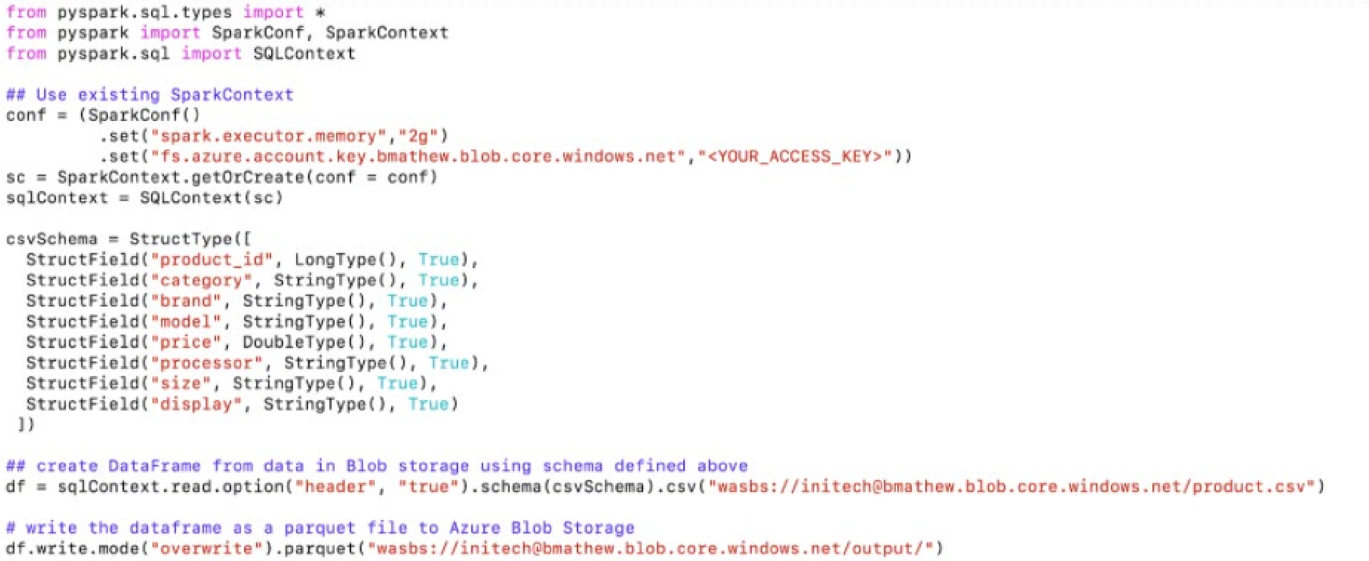
```
from pyspark.sql.types import *
from pyspark import SparkConf, SparkContext
from pyspark.sql import SQLContext

## Use existing SparkContext
conf = (SparkConf()
         .setAppName("Testing PySpark code")
         .set("spark.executor.memory","2g")
         .set("fs.azure.account.key.bmathew.blob.core.windows.net","<YOUR_ACCESS_KEY>"))
sc = SparkContext(conf = conf)
sqlContext = SQLContext(sc)

csvSchema = StructType([
  StructField("product_id", LongType(), True),
  StructField("category", StringType(), True),
  StructField("brand", StringType(), True),
  StructField("model", StringType(), True),
  StructField("price", DoubleType(), True),
  StructField("processor", StringType(), True),
  StructField("size", StringType(), True),
  StructField("display", StringType(), True)
 ])

## create DataFrame from data in Blob storage using schema defined above
df = sqlContext.read.option("header", "true").schema(csvSchema).csv("wasbs://initech@bmathew.blob.core.windows.net/product.csv")

# write the dataframe as a parquet file to Azure Blob Storage
df.write.mode("overwrite").parquet("wasbs://initech@bmathew.blob.core.windows.net/output/")
```

For this code to run on Databricks, the changes you'll need to make will depend on how you run the code:

- Running on an existing cluster as a Databricks job
- Running on a new cluster as a Databricks job

Running on an existing cluster as a Databricks job

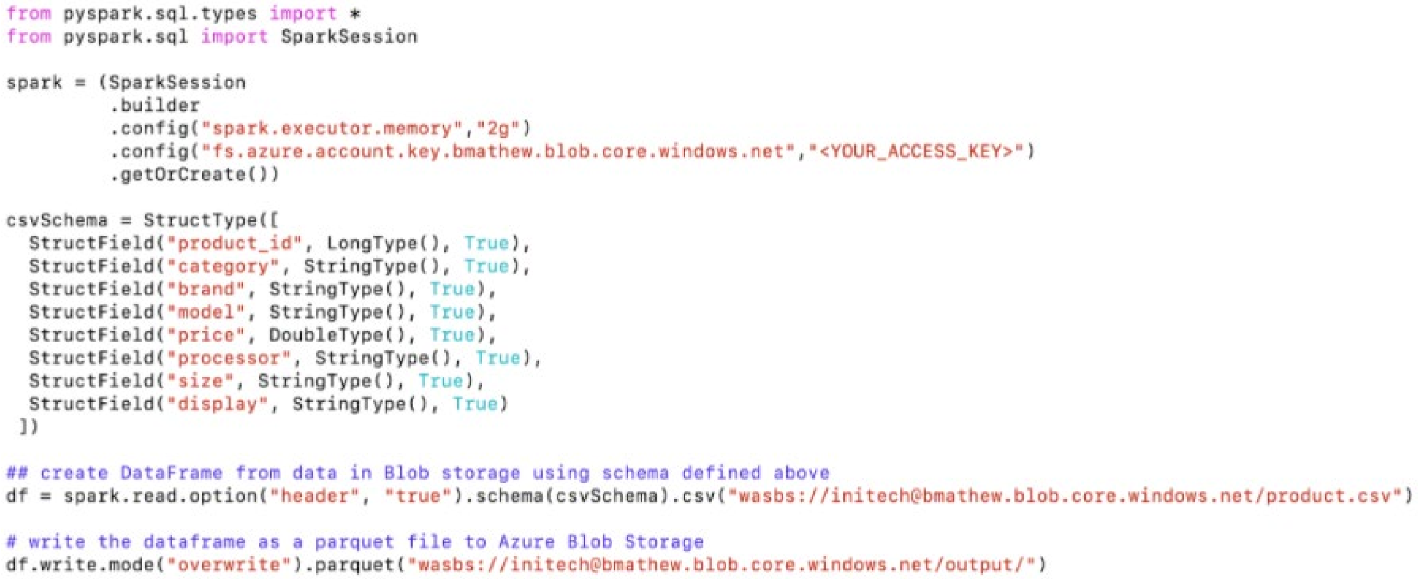
For this code to run on an existing Databricks cluster as a job, you'll need to edit and use the existing SparkContext (i.e., SparkContext.getOrCreate). You can set the application name, but this does not have any impact in Databricks and it does not get recorded in the Spark history server UI.



```
from pyspark.sql.types import *
from pyspark import SparkConf, SparkContext
from pyspark.sql import SQLContext

## Use existing SparkContext
conf = (SparkConf()
         .set("spark.executor.memory","2g")
         .set("fs.azure.account.key.bmathew.blob.core.windows.net","<YOUR_ACCESS_KEY>"))
sc = SparkContext.getOrCreate(conf = conf)
sqlContext = SQLContext(sc)

csvSchema = StructType([
  StructField("product_id", LongType(), True),
  StructField("category", StringType(), True),
  StructField("brand", StringType(), True),
  StructField("model", StringType(), True),
  StructField("price", DoubleType(), True),
  StructField("processor", StringType(), True),
  StructField("size", StringType(), True),
  StructField("display", StringType(), True)
 ])

## create DataFrame from data in Blob storage using schema defined above
df = sqlContext.read.option("header", "true").schema(csvSchema).csv("wasbs://initech@bmathew.blob.core.windows.net/product.csv")

# write the dataframe as a parquet file to Azure Blob Storage
df.write.mode("overwrite").parquet("wasbs://initech@bmathew.blob.core.windows.net/output/")
```

After we make this change, this code can run as a Python script or directly in a Databricks notebook cell. The same change needs to be made for Java and Scala code. The rest of your code will remain the same when using any of the Spark APIs.

Since we are using the DataFrame API to read from our source file, we could only use the SparkSession since it also encapsulates SparkConf, SparkContext and SQLContext. This is more in line with current Spark programming practices.

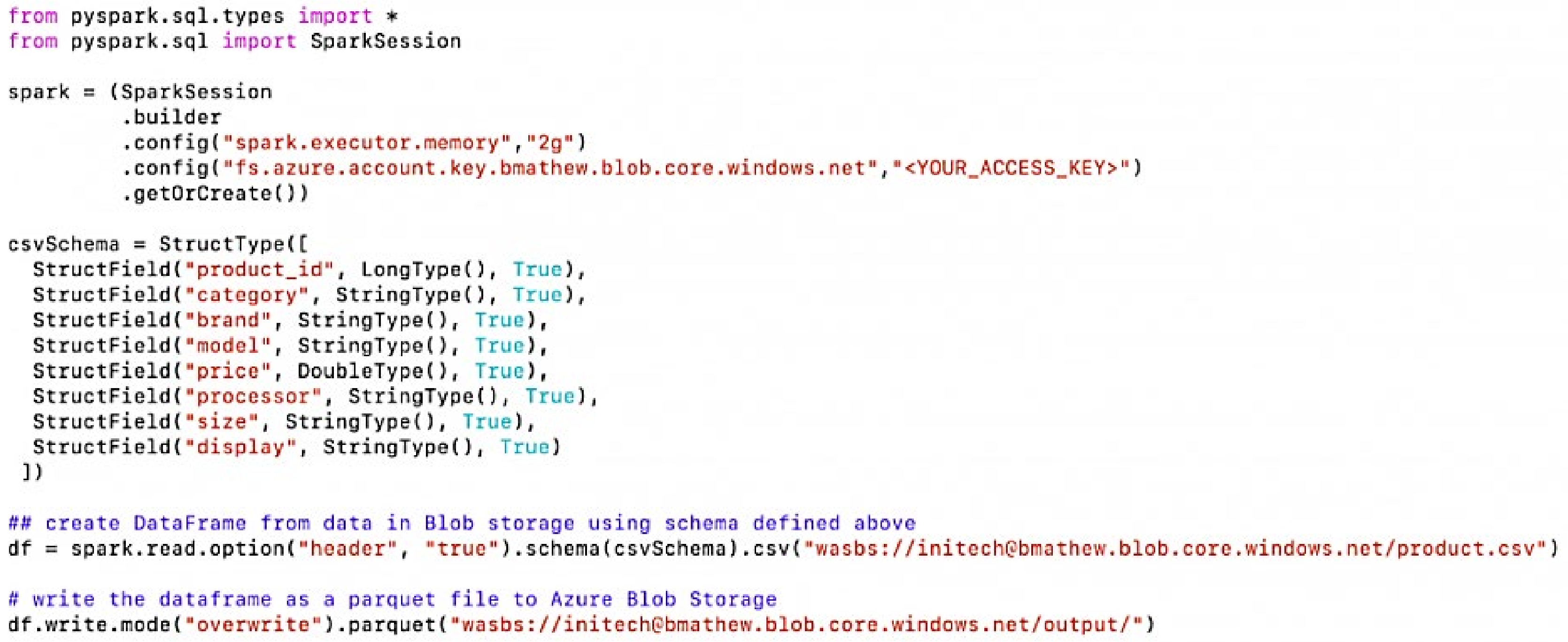


```
from pyspark.sql.types import *
from pyspark.sql import SparkSession

spark = (SparkSession
          .builder
          .config("spark.executor.memory","2g")
          .config("fs.azure.account.key.bmathew.blob.core.windows.net","<YOUR_ACCESS_KEY>")
          .getOrCreate())

csvSchema = StructType([
  StructField("product_id", LongType(), True),
  StructField("category", StringType(), True),
  StructField("brand", StringType(), True),
  StructField("model", StringType(), True),
  StructField("price", DoubleType(), True),
  StructField("processor", StringType(), True),
  StructField("size", StringType(), True),
  StructField("display", StringType(), True)
 ])

## create DataFrame from data in Blob storage using schema defined above
df = spark.read.option("header", "true").schema(csvSchema).csv("wasbs://initech@bmathew.blob.core.windows.net/product.csv")

# write the dataframe as a parquet file to Azure Blob Storage
df.write.mode("overwrite").parquet("wasbs://initech@bmathew.blob.core.windows.net/output/")
```

Running on a new cluster as a Databricks job

If we're running our Hadoop code on a new Databricks cluster that will be used once and only for this job before terminating — *and* if we're using spark-submit on Databricks — we do not need to change how we interact with SparkContext and can create a new SparkContext. This means our original code, which created the new SparkContext, will work — but only when using spark-submit on Databricks to submit as a job. If you are not using spark-submit on Databricks, you will have to modify your code to use the existing SparkContext. For example, if you run the code as a job using Spark Python Task (and not spark-submit), you will need to modify the code to use the existing SparkContext.

A better way to write this code would be to use the SparkSession, which encapsulates SparkConf, SparkContext and SQLContext. Using SparkSession is more in line with current Spark programming practices. Our code can be simplified as shown here:



```
from pyspark.sql.types import *
from pyspark.sql import SparkSession

spark = (SparkSession
         .builder
         .config("spark.executor.memory","2g")
         .config("fs.azure.account.key.bmathew.blob.core.windows.net","<YOUR_ACCESS_KEY>")
         .getOrCreate())

csvSchema = StructType([
  StructField("product_id", LongType(), True),
  StructField("category", StringType(), True),
  StructField("brand", StringType(), True),
  StructField("model", StringType(), True),
  StructField("price", DoubleType(), True),
  StructField("processor", StringType(), True),
  StructField("size", StringType(), True),
  StructField("display", StringType(), True)
 ])

## create DataFrame from data in Blob storage using schema defined above
df = spark.read.option("header", "true").schema(csvSchema).csv("wasbs://initech@bmathew.blob.core.windows.net/product.csv")

# write the dataframe as a parquet file to Azure Blob Storage
df.write.mode("overwrite").parquet("wasbs://initech@bmathew.blob.core.windows.net/output/")
```

CHAPTER 3: APPLICATION DEVELOPMENT, TESTING AND DEPLOYMENT

Notebook and IDE for code development

Apache Zeppelin is a popular notebook development environment used with Hadoop to develop and test code and to query data. Databricks notebooks are similar, but they offer more features:

Data Access

Quickly access available data sets or connect to any data source, on-premises or in the cloud.

Multi-Language Support

Explore data using interactive notebooks with support for multiple programming languages within the same notebook, including R, Python, Scala and SQL.

Automatic Versioning

Tracking changes and versioning automatically happen so that you can continue where you left off or revert changes.

Real-Time Coauthoring

Work on the same notebook in real time while tracking changes with detailed revision history.

Dashboards and Visualizations

Create rich dashboards and visualize insights using point-and-click visualizations, or use powerful scriptable options like matplotlib, ggplot and D3.

Data Science

Automatically log experiments, parameters and results from notebooks directly to MLflow as runs, and quickly see and load previous runs and code versions from the sidebar.

Notebook Workflows and Job Scheduling

Create multi-stage pipelines and execute notebooks as jobs for production pipelines on a specific schedule.

Security

Quickly manage access to each individual notebook — or a collection of notebooks — and experiments, with one common security model.

Integrations

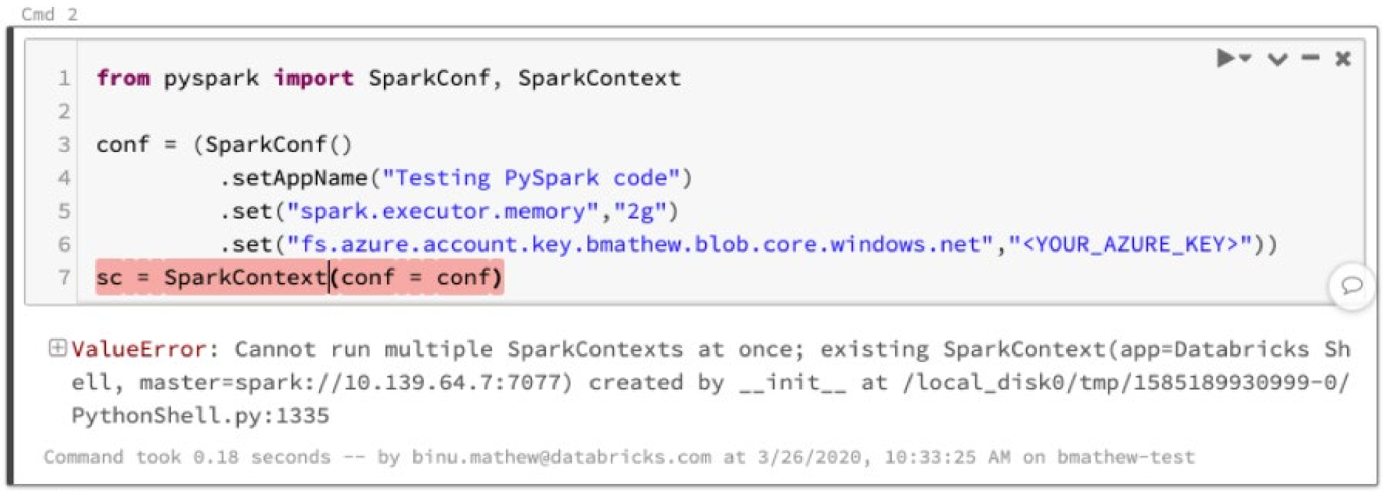
Connect to Tableau, Looker, Power BI, RStudio, Snowflake, etc., allowing data scientists and engineers to use familiar tools.

Autoscaling and On-Demand Clusters

Quickly attach notebooks to auto-manage clusters to efficiently and cost-effectively scale up compute at unprecedented scale.

Code written in Apache Zeppelin notebooks that are used with Hadoop typically don't need you to create a SparkContext, as one is already created for you. This is also true for Databricks notebooks, which don't need you to explicitly instantiate a SparkContext or a SparkSession, as this is already done for you. Therefore, the code from Zeppelin notebooks specific to the Spark APIs may not need changes to run on Azure Databricks.

Creating a new SparkContext will fail on Databricks:

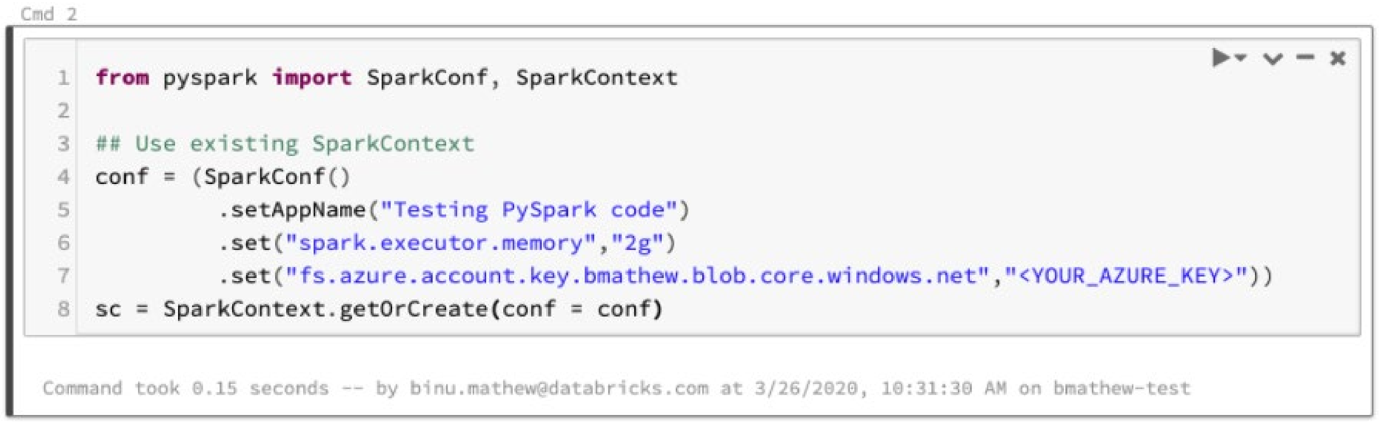


```
from pyspark import SparkConf, SparkContext

conf = (SparkConf()
         .setAppName("Testing PySpark code")
         .set("spark.executor.memory","2g")
         .set("fs.azure.account.key.bmathew.blob.core.windows.net","<YOUR_AZURE_KEY>"))
sc = SparkContext(conf = conf)
```

```
ValueError: Cannot run multiple SparkContexts at once; existing SparkContext(app=Databricks Sh
ell, master=spark://10.139.64.7:7077) created by __init__ at /local_disk0/tmp/1585189930999-0/
PythonShell.py:1335
```

```
Command took 0.18 seconds -- by binu.mathew@databricks.com at 3/26/2020, 10:33:25 AM on bmathew-test
```

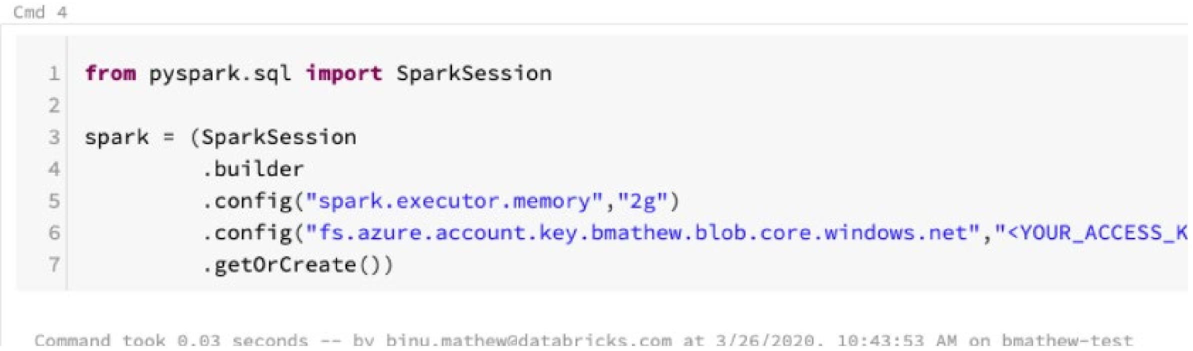
You have to use the existing SparkContext:



```
from pyspark import SparkConf, SparkContext

## Use existing SparkContext
conf = (SparkConf()
         .setAppName("Testing PySpark code")
         .set("spark.executor.memory","2g")
         .set("fs.azure.account.key.bmathew.blob.core.windows.net","<YOUR_AZURE_KEY>"))
sc = SparkContext.getOrCreate(conf = conf)
```

```
Command took 0.15 seconds -- by binu.mathew@databricks.com at 3/26/2020, 10:31:30 AM on bmathew-test
```

With Databricks notebook code development, the SparkSession is already created for you, and the SparkSession itself encapsulates SparkConf, SparkContext and SQLContext. For example, the following will work in a Databricks notebook, but is not necessary to do so:

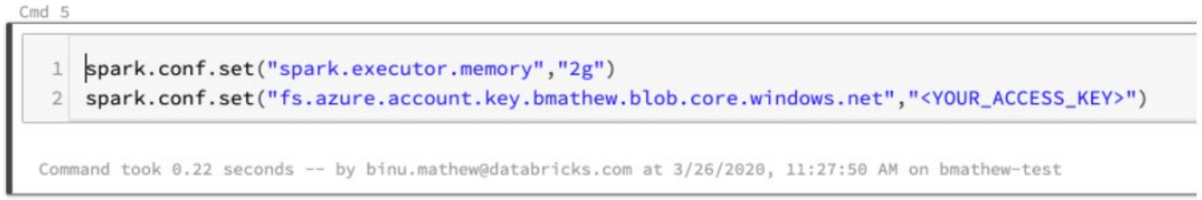


Cmd 4

```
from pyspark.sql import SparkSession

spark = (SparkSession
         .builder
         .config("spark.executor.memory","2g")
         .config("fs.azure.account.key.bmathew.blob.core.windows.net","<YOUR_ACCESS_K
         .getOrCreate())
```

```
Command took 0.03 seconds -- by binu.mathew@databricks.com at 3/26/2020, 10:43:53 AM on bmathew-test
```

Instead, you could just do the following since the SparkSession is already instantiated for you:



Cmd 5

```
spark.conf.set("spark.executor.memory","2g")
spark.conf.set("fs.azure.account.key.bmathew.blob.core.windows.net","<YOUR_ACCESS_KEY>")
```

```
Command took 0.22 seconds -- by binu.mathew@databricks.com at 3/26/2020, 11:27:50 AM on bmathew-test
```

Databricks also offers the option of using other notebooks and IDEs to interact with the platform. This includes using Zeppelin, Jupyter, Visual Studio, PyCharm, IntelliJ, Eclipse, RStudio and more. Please refer to the online documentation for more information on using other notebooks and IDEs:

AWS AZURE

Please refer to the online documentation for the full list of features and more information about using Databricks notebooks:

AWS AZURE

CHAPTER 3: APPLICATION DEVELOPMENT, TESTING AND DEPLOYMENT

Source code management and CI/CD

Databricks notebooks have basic version control built into them. Please review the online documentation to learn more about this feature:

AWS

AZURE

Notebooks can also be linked to the following source code management (SCM) systems:

GitHub

Please see the online documentation for more information:
AWS Azure

Bitbucket

Please see the online documentation for more information:
AWS Azure

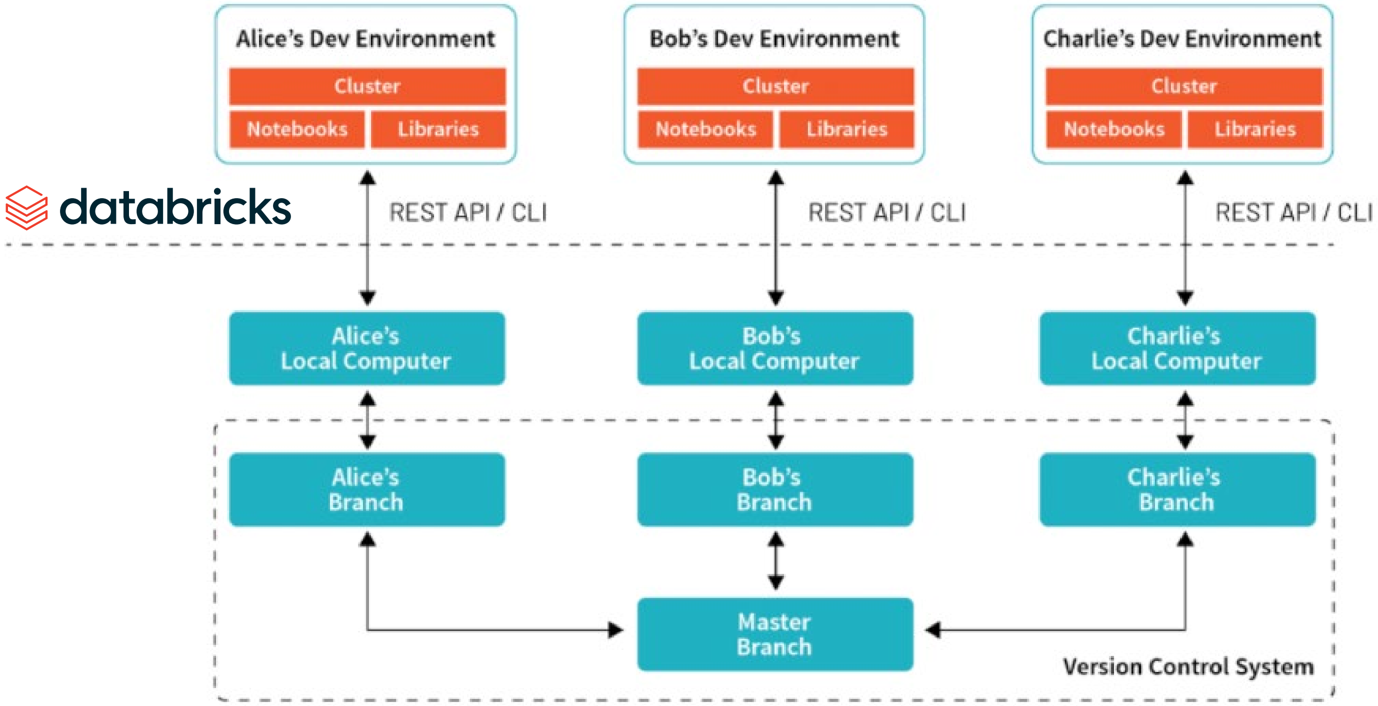
GitLab

Please see the online documentation for more information:
AWS Azure

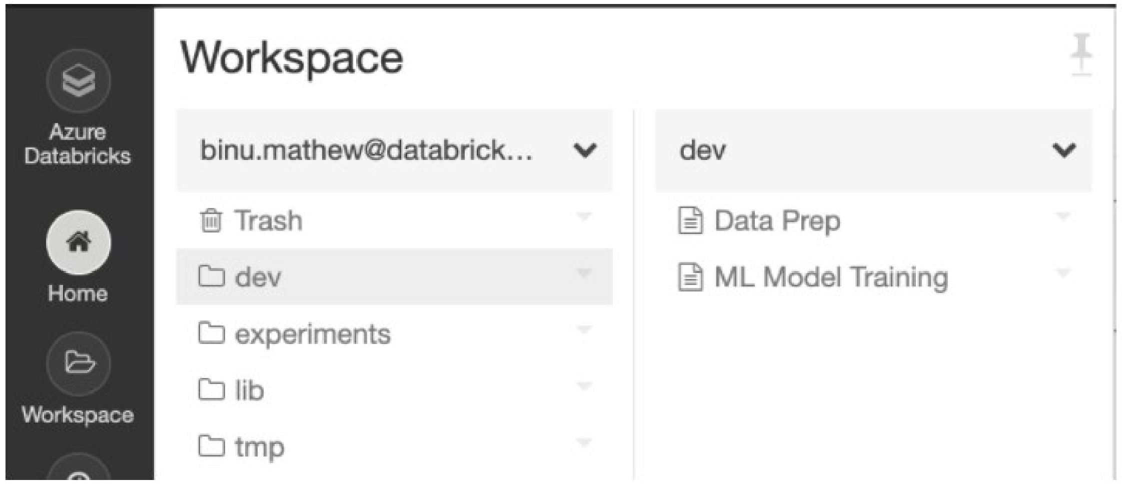
Azure DevOps

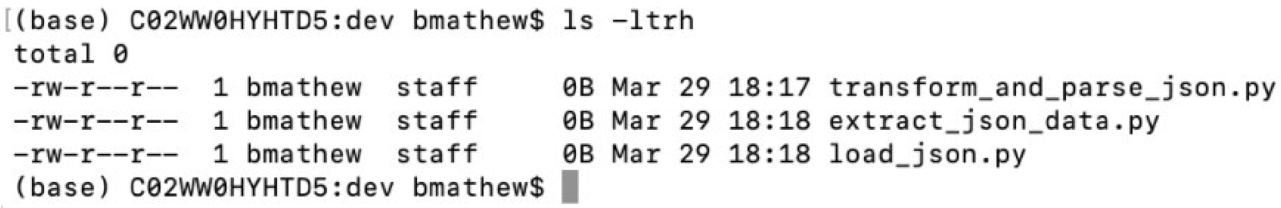
Please see the online documentation for more information.

You can also use the command line interface to sync notebooks and non-notebook code (Java, Scala and Python source code) with any version control system.



Let’s look at an example. A Databricks user has developed Python scripts and notebooks that need to be checked into the company’s GitHub repository for the project being worked on. The notebooks were developed in Databricks and the Python scripts were developed locally on a laptop.



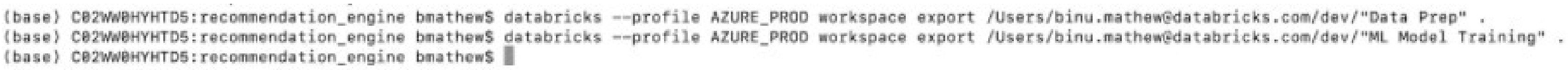


```
(base) C02WW0HYHTD5:dev bmathew$ ls -ltrh
total 0
-rw-r--r--  1 bmathew  staff     0B Mar 29 18:17 transform_and_parse_json.py
-rw-r--r--  1 bmathew  staff     0B Mar 29 18:18 extract_json_data.py
-rw-r--r--  1 bmathew  staff     0B Mar 29 18:18 load_json.py
(base) C02WW0HYHTD5:dev bmathew$ 
```

The user is working on a local branch from their laptop and needs to check in the Python scripts and notebook code and push to the remote SCM server. This can easily be done using the command line interface (CLI) and your SCM tool commands.

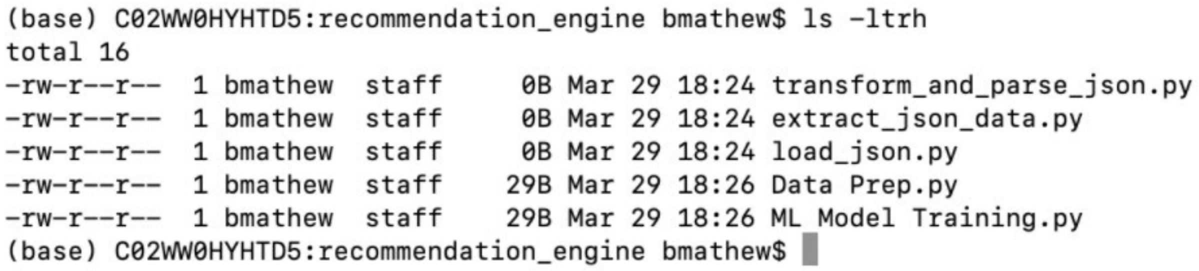
1. From the user’s laptop, we will export the notebooks from the Databricks Workspace and onto the local laptop using the Databricks CLI.

```
databricks --profile AZURE_PROD workspace export /Users/binu.mathew@databricks.com/dev/”Data Prep” .
databricks --profile AZURE_PROD workspace export /Users/binu.mathew@databricks.com/dev/”ML Model
Training” .
```



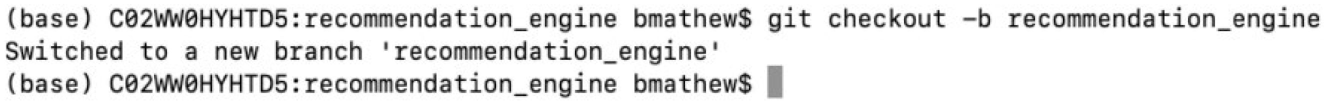
```
(base) C02WW0HYHTD5:recommendation_engine bmathew$ databricks --profile AZURE_PROD workspace export /Users/binu.mathew@databricks.com/dev/"Data Prep" .
(base) C02WW0HYHTD5:recommendation_engine bmathew$ databricks --profile AZURE_PROD workspace export /Users/binu.mathew@databricks.com/dev/"ML Model Training" .
(base) C02WW0HYHTD5:recommendation_engine bmathew$
```

Now let’s look at our directory on the local laptop. Notice the notebooks were exported:



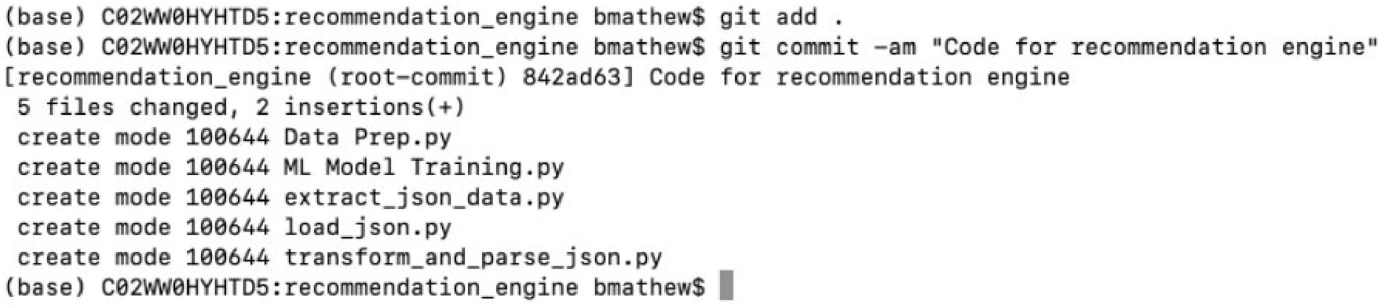
```
(base) C02WW0HYHTD5:recommendation_engine bmathew$ ls -ltrh
total 16
-rw-r--r--  1 bmathew  staff     0B Mar 29 18:24 transform_and_parse_json.py
-rw-r--r--  1 bmathew  staff     0B Mar 29 18:24 extract_json_data.py
-rw-r--r--  1 bmathew  staff     0B Mar 29 18:24 load_json.py
-rw-r--r--  1 bmathew  staff    29B Mar 29 18:26 Data Prep.py
-rw-r--r--  1 bmathew  staff    29B Mar 29 18:26 ML Model Training.py
(base) C02WW0HYHTD5:recommendation_engine bmathew$
```

2. From the user’s laptop, we create a local branch using the SCM tool commands.



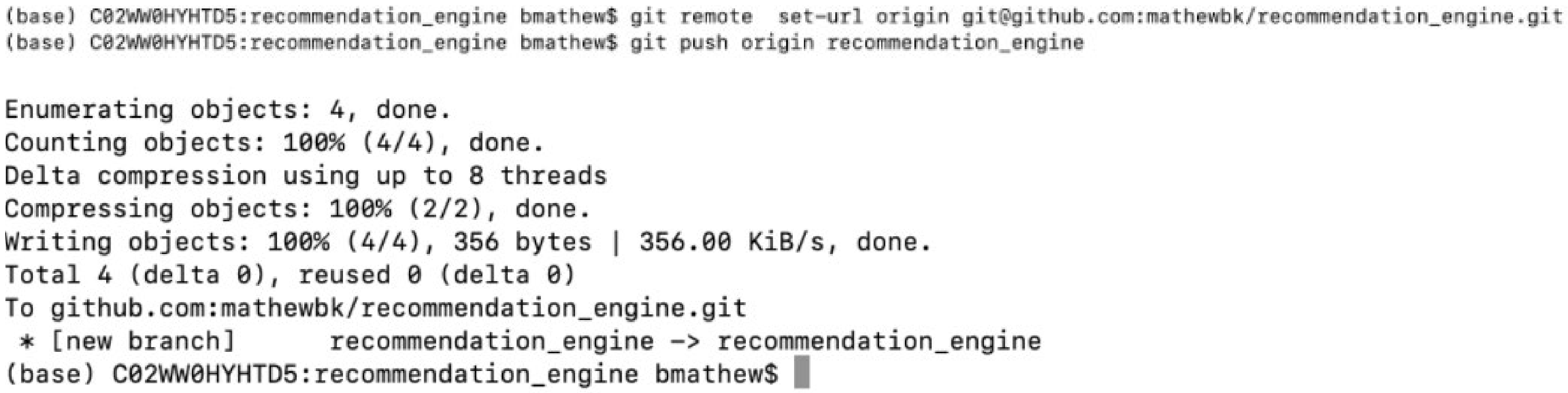
```
(base) C02WW0HYHTD5:recommendation_engine bmathew$ git checkout -b recommendation_engine
Switched to a new branch 'recommendation_engine'
(base) C02WW0HYHTD5:recommendation_engine bmathew$
```

3. We add all our code in this local branch. This includes the Python scripts and the notebooks.



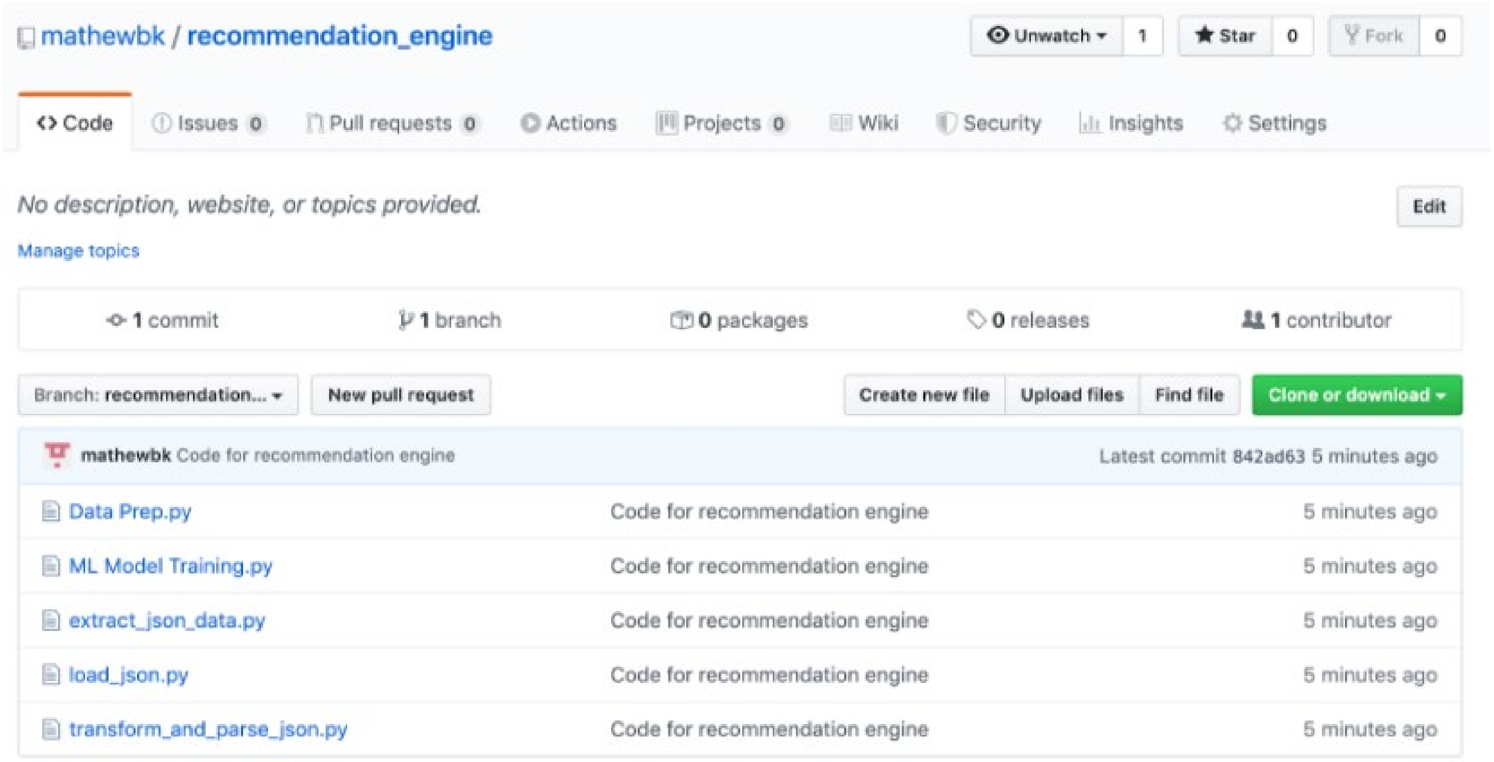
```
(base) C02WW0HYHTD5:recommendation_engine bmathew$ git add .
(base) C02WW0HYHTD5:recommendation_engine bmathew$ git commit -am "Code for recommendation engine"
[recommendation_engine (root-commit) 842ad63] Code for recommendation engine
 5 files changed, 2 insertions(+)
 create mode 100644 Data Prep.py
 create mode 100644 ML Model Training.py
 create mode 100644 extract_json_data.py
 create mode 100644 load_json.py
 create mode 100644 transform_and_parse_json.py
(base) C02WW0HYHTD5:recommendation_engine bmathew$
```

4. Push the local branch to remove the SCM server using the SCM tool commands.



```
(base) C02WW0HYHTD5:recommendation_engine bmathew$ git remote  set-url origin git@github.com:mathewbk/recommendation_engine.git
(base) C02WW0HYHTD5:recommendation_engine bmathew$ git push origin recommendation_engine

Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 356 bytes | 356.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To github.com:mathewbk/recommendation_engine.git
 * [new branch]      recommendation_engine -> recommendation_engine
(base) C02WW0HYHTD5:recommendation_engine bmathew$ 
```

5. We can see that the local branch was pushed to the SCM server.



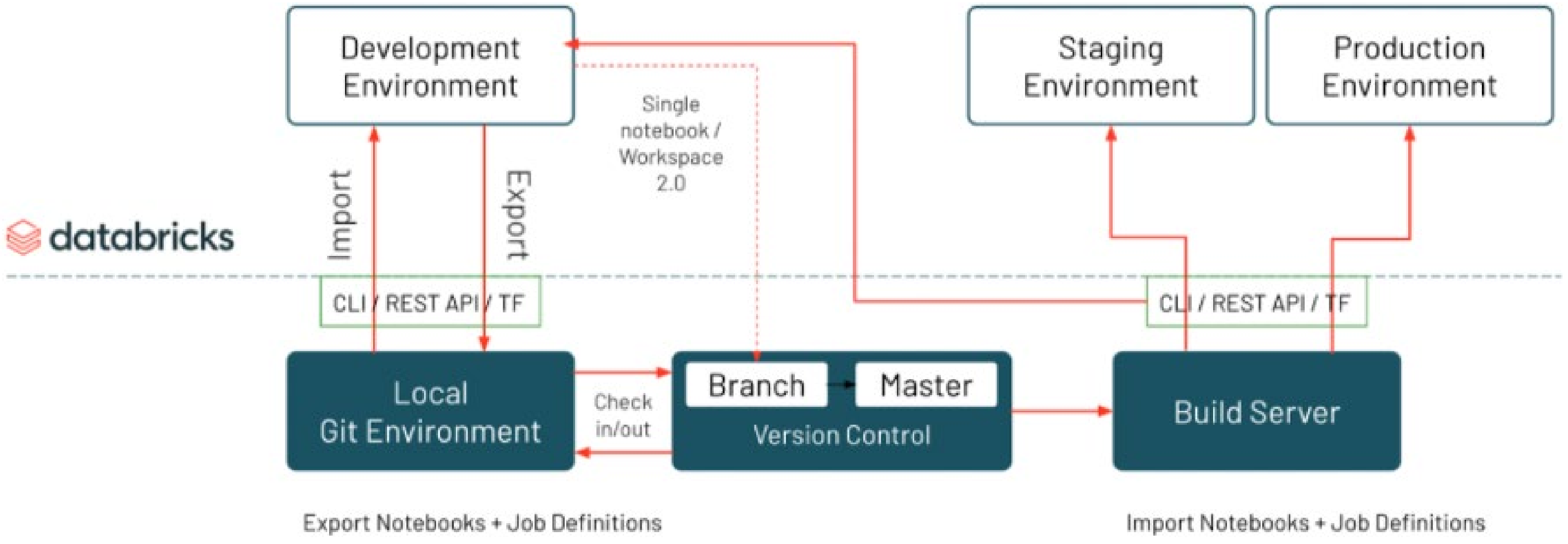
No description, website, or topics provided.

1 commit · 1 branch · 0 packages · 0 releases · 1 contributor

mathewbk Code for recommendation engine — Latest commit 842ad63 5 minutes ago

Data Prep.py	Code for recommendation engine	5 minutes ago
ML Model Training.py	Code for recommendation engine	5 minutes ago
extract_json_data.py	Code for recommendation engine	5 minutes ago
load_json.py	Code for recommendation engine	5 minutes ago
transform_and_parse_json.py	Code for recommendation engine	5 minutes ago

After code has been checked into the source code management system, it can then be sent to the build server and to different environments for testing and final deployment.



Please refer to the online documentation for more information on CI/CD integration with Databricks:

AWS

AZURE

CHAPTER 3: APPLICATION DEVELOPMENT, TESTING AND DEPLOYMENT

Job scheduling and submission

Databricks allows you to submit code as Java/Scala JAR files, Python scripts and wheel/egg files, or notebook code to run as scheduled or immediate jobs on a Databricks cluster. A basic cron-style job scheduler is provided with the platform to schedule and launch jobs. The Databricks job scheduler can be accessed through the UI, REST API or command line interface. You can also use third-party external job schedulers to create more complex workflows/DAG by utilizing the REST API interface for Databricks. You can create and launch a workflow from an external job scheduler and have each task in the workflow execute code on a Databricks cluster (Java/Scala JAR files, Python scripts and wheel/egg, or notebooks) via REST calls, and then get the return code (exit code) via REST to determine how to proceed in your workflow — i.e., continue processing or fail the entire workflow.

Azure Data Factory (ADF) and Apache Airflow are popular tools to schedule and launch tasks. Databricks has native integration with both ADF and Apache Airflow for job scheduling. Please refer to the online documentation for more information on this integration:

AWS

AZURE

Important considerations for job scheduling on Databricks:

- A workspace may have up to 1,000 jobs that appear in the UI
- The number of jobs a workspace can create in an hour is limited to 5,000 (includes "run now" and "runs submit"). This limit also affects jobs created by the REST API and notebook workflows.
- The number of actively concurrent runs a workspace can create is limited to 150
- Multiple workspaces can be used if you exceed the above limits

Spark JAR Jobs vs. Spark Submit Jobs

Jobs in Databricks are based on different task types. The task type dictates the type of code to be executed. The traditional spark-submit syntax is also supported for JAR files. These are the supported task types:

- Notebook
- Spark JAR
- Spark Submit
- Python Script

When dealing with JAR files, we recommend using the Spark JAR task type. The Spark Submit task does not support autoscaling or the use of the Databricks utilities JAR. The latter allows you to leverage the dbutils.* APIs to manage DBFS, Notebooks, Secrets, etc.

If you would like to convert an existing spark-submit job to an equivalent Spark JAR version, you'll need to make the following changes:

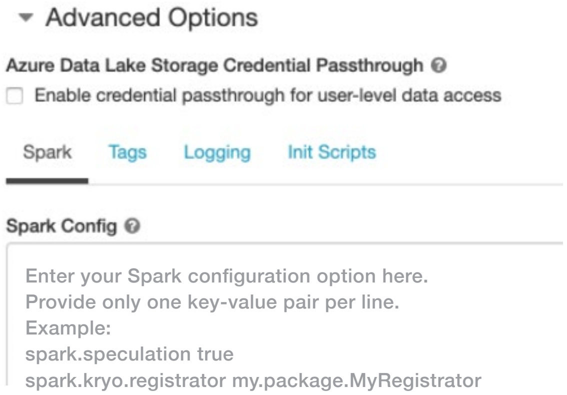
- Your code will need to retrieve the SparkContext using SparkContext.getOrCreate()
- Parameters will be passed to the job via task parameter variables in JSON format
  AWS Azure
- Any files passed via —FILES must be moved to cloud storage. The corresponding code that accesses the files will need to reference cloud storage.
- Num-executors will be specified in the cluster configuration for the job as the Max Workers in the cluster configuration
  AWS Azure
- Any Spark configuration parameters will be specified in the cluster configuration for the job

Spark Submit jobs

Hadoop users familiar with Spark may already be using spark-submit via YARN to run their Spark applications. Databricks allows you to execute jobs using spark-submit, similar to how you might be doing this today on Hadoop.

There are some differences between how Spark works on YARN and on Databricks:

- The default functionality on a Databricks cluster is to launch one executor per worker VM and use all the cores on that worker VM. Much of the RAM will also be used for that single executor minus the RAM used for OS and other system processes.
- To launch multiple executors on a VM, you need to configure `spark.executor.cores` and `spark.executor.memory` for the cluster settings under Spark Config:



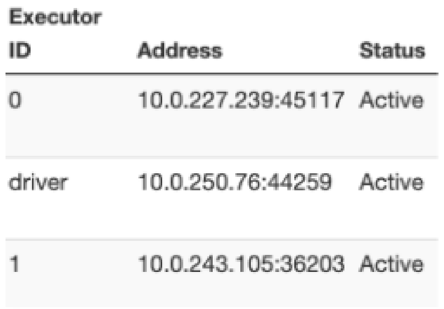
Azure Data Lake Storage Credential Passthrough

□ Enable credential passthrough for user-level data access

Spark Config

Enter your Spark configuration option here.
Provide only one key-value pair per line.
Example:
spark.speculation true
spark.kryo.registrator my.package.MyRegistrator

- For example, let's say you launch a cluster with two worker VMs, with each having 61GB RAM and eight cores. By default, Databricks will launch a total of two executors:

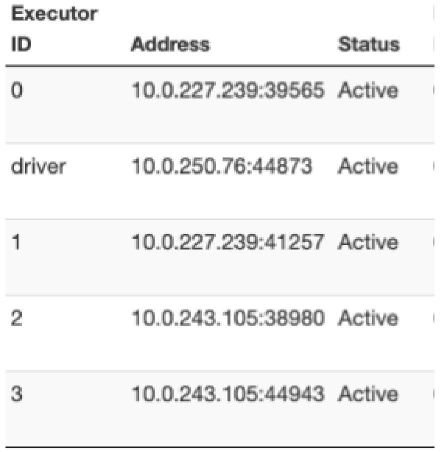


Executor ID	Address	Status
0	10.0.227.239:45117	Active
driver	10.0.250.76:44259	Active
1	10.0.243.105:36203	Active

- If you want two executors per VM, you can configure Spark Config like this:

```
spark.executor.cores 4
spark.executor.memory 25g
```

- It's important to point out for this example that setting the memory to 30g (approximately half) did not launch multiple executors. A lower value of 25g worked.
- You will now get two executors per worker VM, with each executor using the same IP but a different port:



Executor ID	Address	Status
0	10.0.227.239:39565	Active
driver	10.0.250.76:44873	Active
1	10.0.227.239:41257	Active
2	10.0.243.105:38980	Active
3	10.0.243.105:44943	Active

YARN shares the resources on a Hadoop cluster, allowing for multiple production jobs to be submitted to the same cluster. The Hadoop cluster is an always-on cluster, whereas Databricks clusters are ephemeral and autoscaling. A Databricks cluster remains active for the duration of the job and then terminates. For Databricks, we recommend that production jobs each run on their own autoscaling cluster. After job completion, that cluster will terminate. This provides better isolation — jobs are run independently, there's no resource contention with multiple jobs sharing and competing for resources, and there are stronger job completion guarantees.

Please refer to the online documentation for more information:

Creating and submitting jobs on Databricks

AWS Azure

Migrating production workloads from Apache Spark on Hadoop to Databricks

AWS Azure

Spark job submissions to YARN can be done via spark-submit and you might be doing this today. For example, the following is a spark-submit command on YARN to run a Python script:

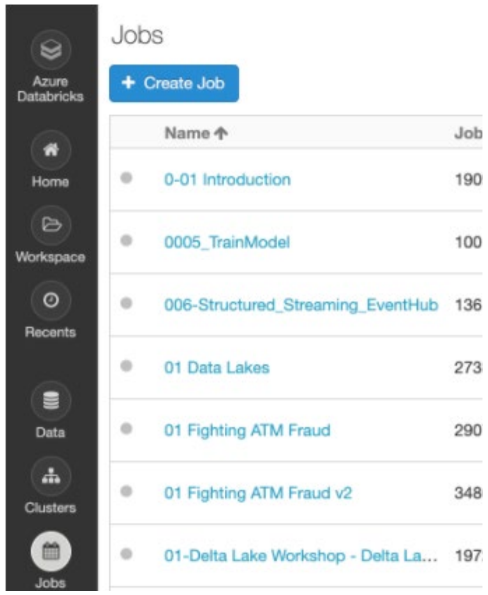
```
spark-submit --master=yarn --executor-memory=25g parse_clicksteam.py
```

Job submission on Databricks can be done via the UI, REST API and command line interface (CLI). Let's look at some examples:

- Job submission via UI
- Job submission via API and CLI
- Job submission to existing cluster

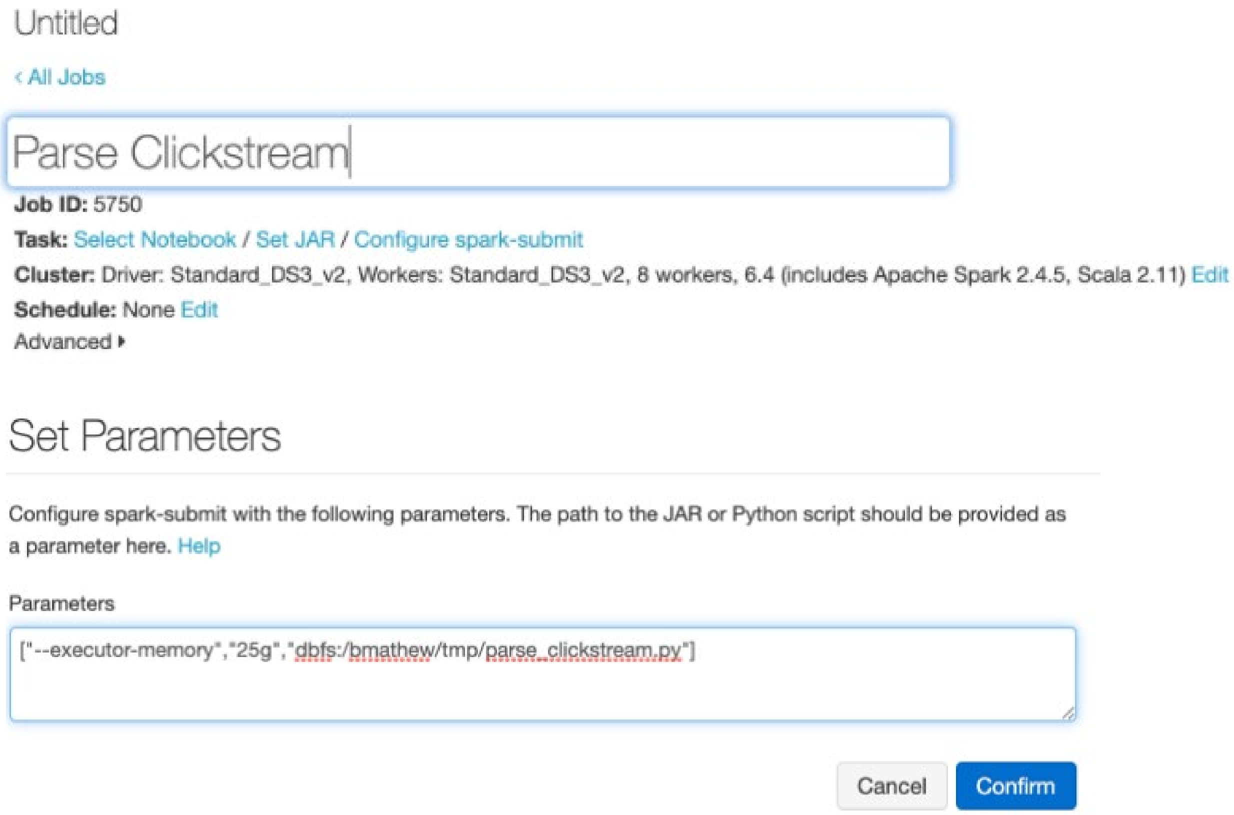
1. Job submission via UI

Create a new job, configure spark-submit, configure the cluster, schedule the job and set advanced properties if needed:



Jobs

Name ↑	Job
0-01 Introduction	190
0005_TrainModel	100
006-Structured_Streaming_EventHub	136
01 Data Lakes	273
01 Fighting ATM Fraud	290
01 Fighting ATM Fraud v2	348
01-Delta Lake Workshop - Delta La...	197



Untitled

‹ All Jobs

Parse Clickstream

Job ID: 5750

Task: Select Notebook / Set JAR / Configure spark-submit

Cluster: Driver: Standard_DS3_v2, Workers: Standard_DS3_v2, 8 workers, 6.4 (includes Apache Spark 2.4.5, Scala 2.11) Edit

Schedule: None Edit

Advanced ▸

Set Parameters

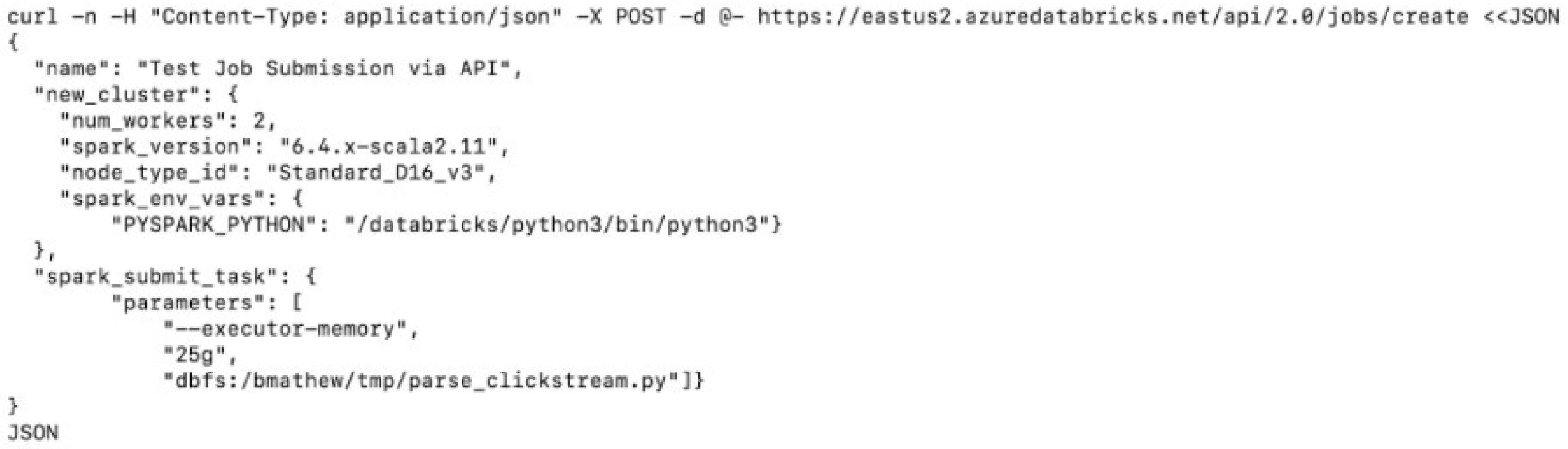
Configure spark-submit with the following parameters. The path to the JAR or Python script should be provided as a parameter here. Help

Parameters

```
["--executor-memory","25g","dbfs:/bmathew/tmp/parse_clickstream.py"]
```

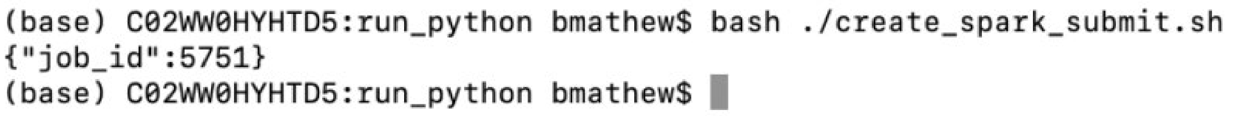
2. Job submission via API and CLI

You can use curl to call the REST API and create the job:



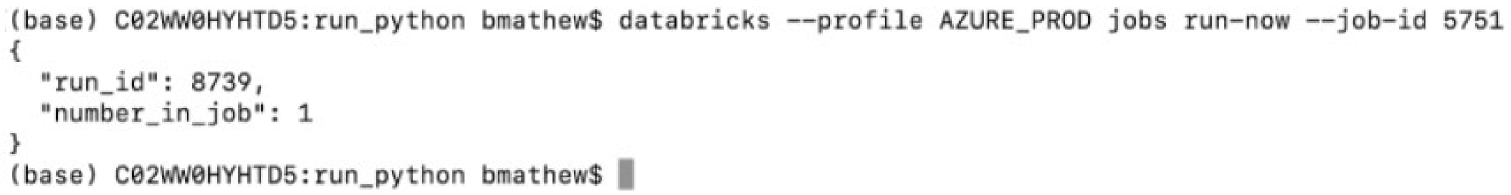
```
curl -n -H "Content-Type: application/json" -X POST -d @- https://eastus2.azuredatabricks.net/api/2.0/jobs/create <<JSON
{
  "name": "Test Job Submission via API",
  "new_cluster": {
    "num_workers": 2,
    "spark_version": "6.4.x-scala2.11",
    "node_type_id": "Standard_D16_v3",
    "spark_env_vars": {
        "PYSPARK_PYTHON": "/databricks/python3/bin/python3"}
  },
  "spark_submit_task": {
        "parameters": [
            "--executor-memory",
            "25g",
            "dbfs:/bmathew/tmp/parse_clickstream.py"]}
}
JSON
```

Execute the command so that the job gets created. In this example, we created a file containing the command that will execute as a bash script. This will return a job ID:



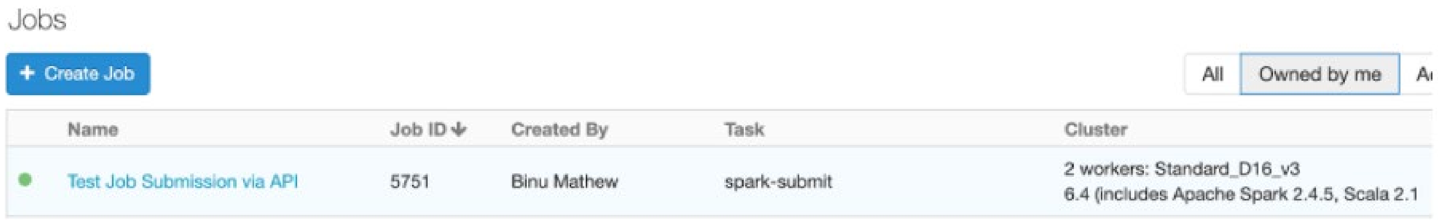
```
(base) C02WW0HYHTD5:run_python bmathew$ bash ./create_spark_submit.sh
{"job_id":5751}
(base) C02WW0HYHTD5:run_python bmathew$ █
```

Run the job from the CLI using the job ID. This will return a run ID:



```
(base) C02WW0HYHTD5:run_python bmathew$ databricks --profile AZURE_PROD jobs run-now --job-id 5751
{
  "run_id": 8739,
  "number_in_job": 1
}
(base) C02WW0HYHTD5:run_python bmathew$ █
```

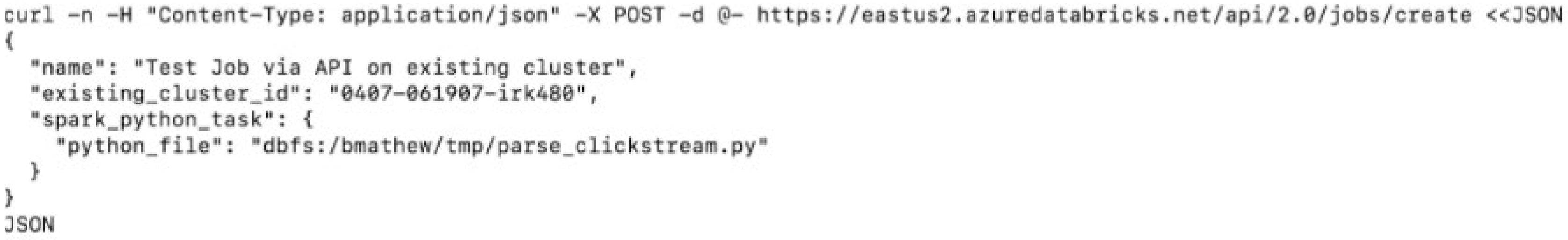
We can see from the UI that the job is running:



	Name	Job ID ↓	Created By	Task	Cluster
●	Test Job Submission via API	5751	Binu Mathew	spark-submit	2 workers: Standard_D16_v3 6.4 (includes Apache Spark 2.4.5, Scala 2.1

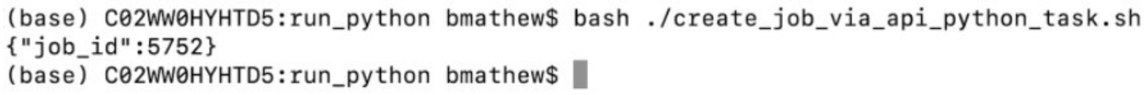
3. Job submission to existing cluster

Job submission via spark-submit on Databricks can only be executed on new clusters and not on existing clusters. To run a job on an existing cluster, we can create the job as a spark_python_task:



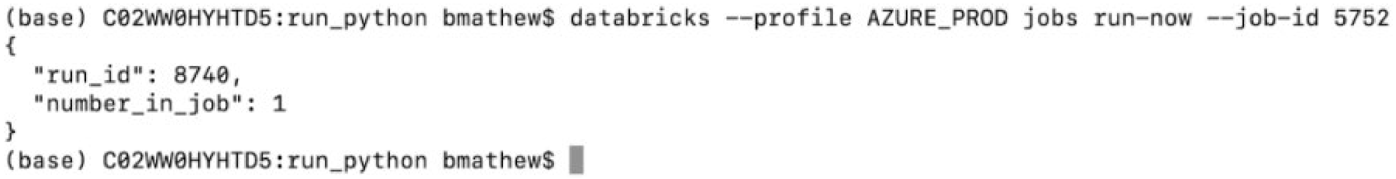
```
curl -n -H "Content-Type: application/json" -X POST -d @- https://eastus2.azuredatabricks.net/api/2.0/jobs/create <<JSON
{
  "name": "Test Job via API on existing cluster",
  "existing_cluster_id": "0407-061907-irk480",
  "spark_python_task": {
    "python_file": "dbfs:/bmathew/tmp/parse_clickstream.py"
  }
}
JSON
```

Execute the command so that the job gets created. In this example, we created a file containing the command. This will return a job ID:



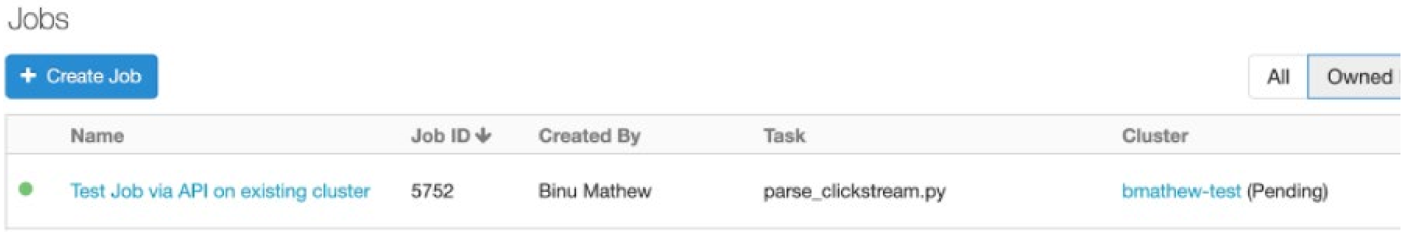
```
(base) C02WW0HYHTD5:run_python bmathew$ bash ./create_job_via_api_python_task.sh
{"job_id":5752}
(base) C02WW0HYHTD5:run_python bmathew$ 
```

Run the job from the CLI using the job ID:



```
(base) C02WW0HYHTD5:run_python bmathew$ databricks --profile AZURE_PROD jobs run-now --job-id 5752
{
  "run_id": 8740,
  "number_in_job": 1
}
(base) C02WW0HYHTD5:run_python bmathew$ 
```

We can see from the UI that the job is running:



	Name	Job ID ↓	Created By	Task	Cluster
●	Test Job via API on existing cluster	5752	Binu Mathew	parse_clickstream.py	bmathew-test (Pending)

CHAPTER

04

The Path Forward

Next steps

CHAPTER 4: THE PATH FORWARD

Next steps

Migration of your Hadoop environment to Databricks delivers significant business benefits, including:

- Reduction of operational cost
- Increased productivity of your data teams
- Unlocking of advanced AI and BI capabilities that drive top-line growth

Databricks, along with their preferred community of migration partners, is available to assist with your initiative by providing the following services:

- Inventorying your existing Hadoop landscape
- Developing a detailed future state reference architecture
- Quantifying the business benefits of migration
- Creating a joint implementation plan with your team
- Co-delivering a migration project
- Retiring your existing Hadoop environment

Please reach out to us at sales@databricks.com if you are interested in exploring Hadoop to Databricks migration.

MORE RESOURCES AVAILABLE AT DATABRICKS.COM/MIGRATION.

About Databricks

Databricks is the data and AI company. More than 5,000 organizations worldwide — including Comcast, Condé Nast, H&M and over 40% of the Fortune 500 — rely on the Databricks Lakehouse Platform to unify their data, analytics and AI. Databricks is headquartered in San Francisco, with offices around the globe. Founded by the original creators of Apache Spark™, Delta Lake and MLflow, Databricks is on a mission to help data teams solve the world's toughest problems. To learn more, follow Databricks on Twitter, LinkedIn and Facebook.