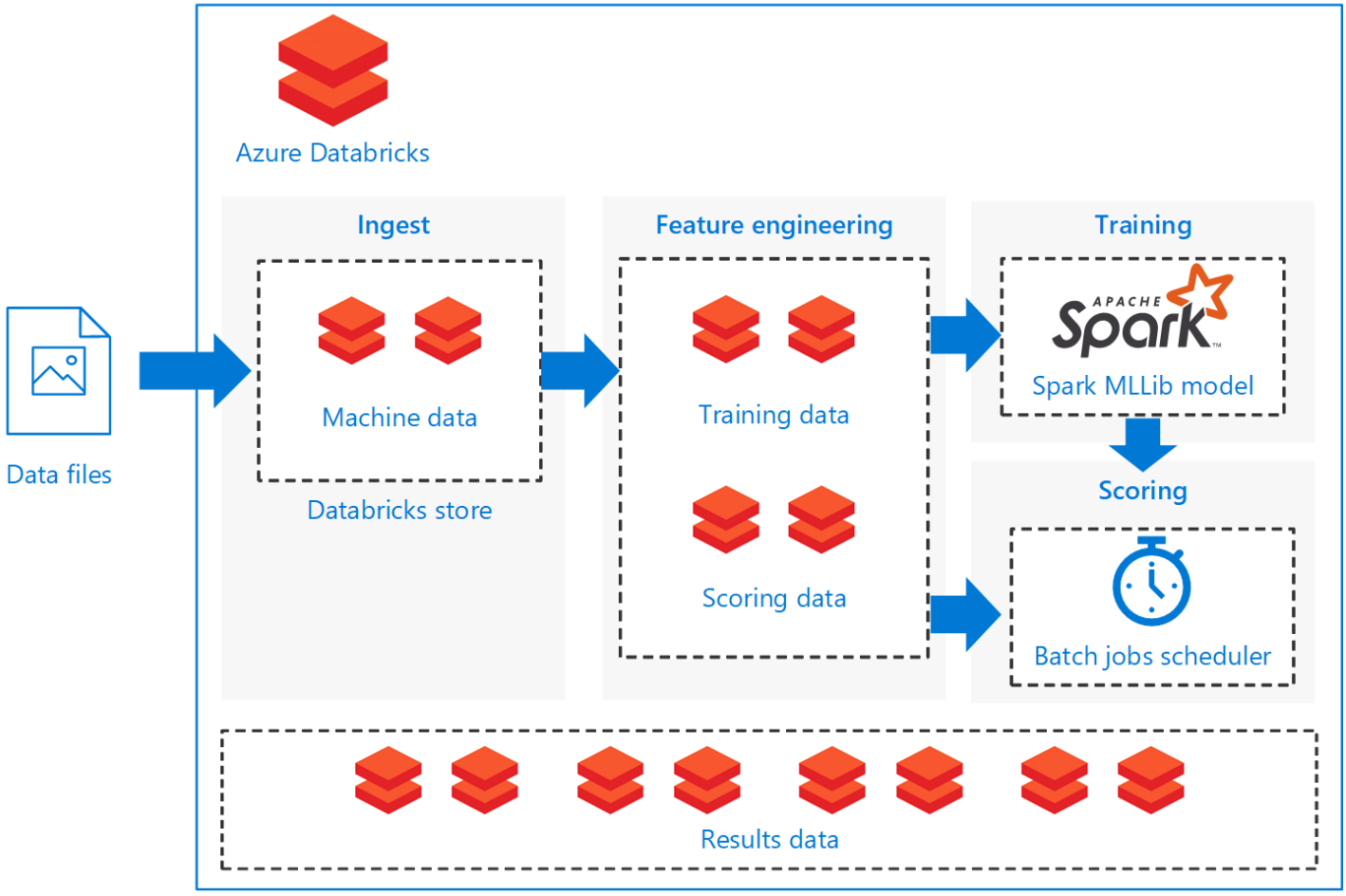
Batch scoring of Spark machine learning models on Azure Databricks

02/07/2019 • 5 minutes to read • Contributors

In this article

This reference architecture shows how to build a scalable solution for batch scoring an Apache Spark classification model on a schedule using Azure Databricks, an Apache Spark-based analytics platform optimized for Azure. The solution can be used as a template that can be generalized to other scenarios.

A reference implementation for this architecture is available on GitHub.



Scenario: A business in an asset-heavy industry wants to minimize the costs and downtime associated with unexpected mechanical failures. Using IoT data collected from their machines, they can create a predictive maintenance model. This model enables the business to maintain components proactively and repair them before they fail. By maximizing mechanical component use, they can control costs and reduce downtime.

A predictive maintenance model collects data from the machines and retains historical examples of component failures. The model can then be used to monitor the current state of the components and predict if a given component will fail in the near future. For common use cases and modeling approaches, see Azure AI guide for predictive maintenance solutions.

This reference architecture is designed for workloads that are triggered by the presence of new data from the component machines. Processing involves the following steps:

1. Ingest the data from the external data store onto an Azure Databricks data store.
2. Train a machine learning model by transforming the data into a training data set, then building a Spark MLlib model. MLlib consists of most common machine learning algorithms and utilities optimized to take advantage of Spark data scalability capabilities.
3. Apply the trained model to predict (classify) component failures by transforming the data into a scoring data set. Score the data with the Spark MLLib model.
4. Store results on the Databricks data store for post-processing consumption.

Notebooks are provided on GitHub to perform each of these tasks.

Architecture

The architecture defines a data flow that is entirely contained within Azure Databricks based on a set of sequentially executed notebooks. It consists of the following components:

Data files. The reference implementation uses a simulated data set contained in five static data files.

Ingestion. The data ingestion notebook downloads the input data files into a collection of Databricks data sets. In a real-world scenario, data from IoT devices would stream onto Databricks-accessible storage such as Azure SQL Server or Azure Blob storage. Databricks supports multiple data sources.

Training pipeline. This notebook executes the feature engineering notebook to create an analysis data set from the ingested data. It then executes a model building notebook that trains the machine learning model using the Apache Spark MLlib scalable machine learning library.

Scoring pipeline. This notebook executes the feature engineering notebook to create scoring data set from the ingested data and executes the scoring notebook. The scoring notebook uses the trained Spark MLlib model to generate predictions for the observations in the scoring data set. The predictions are stored in the results store, a new data set on the Databricks data store.

Scheduler. A scheduled Databricks job handles batch scoring with the Spark model. The job executes the scoring pipeline notebook, passing variable arguments through notebook parameters to specify the details for constructing the scoring data set and where to store the results data set.

The scenario is constructed as a pipeline flow. Each notebook is optimized to perform in a batch setting for each of the operations: ingestion, feature engineering, model building, and model scorings. To accomplish this, the feature engineering notebook is designed to generate a general data set for any of the training, calibration, testing, or scoring operations. In this scenario, we use a temporal split strategy for these operations, so the notebook parameters are used to set date-range filtering.

Because the scenario creates a batch pipeline, we provide a set of optional examination notebooks to explore the output of the pipeline notebooks. You can find these in the GitHub repository:

- `1a_raw-data_exploring`
- `2a_feature_exploration`
- `2b_model_testing`

- `3b_model_scoring_evaluation`

Recommendations

Databricks is set up so you can load and deploy your trained models to make predictions with new data. We used Databricks for this scenario because it provides these additional advantages:

- Single sign-on support using Azure Active Directory credentials.
- Job scheduler to execute jobs for production pipelines.
- Fully interactive notebook with collaboration, dashboards, REST APIs.
- Unlimited clusters that can scale to any size.
- Advanced security, role-based access controls, and audit logs.

To interact with the Azure Databricks service, use the Databricks Workspace interface in a web browser or the command-line interface (CLI). Access the Databricks CLI from any platform that supports Python 2.7.9 to 3.6.

The reference implementation uses notebooks to execute tasks in sequence. Each notebook stores intermediate data artifacts (training, test, scoring, or results data sets) to the same data store as the input data. The goal is to make it easy for you to use it as needed in your particular use case. In practice, you would connect your data source to your Azure Databricks instance for the notebooks to read and write directly back into your storage.

You can monitor job execution through the Databricks user interface, the data store, or the Databricks CLI as necessary. Monitor the cluster using the event log and other metrics that Databricks provides.

Performance considerations

An Azure Databricks cluster enables autoscaling by default so that during runtime, Databricks dynamically reallocates workers to account for the characteristics of your job. Certain parts of your pipeline may be more computationally demanding than others. Databricks adds additional workers during these phases of your job (and removes them when they're no longer needed). Autoscaling makes it easier to achieve high cluster utilization, because you don't need to provision the cluster to match a workload.

Additionally, more complex scheduled pipelines can be developed by using Azure Data Factory with Azure Databricks.

Storage considerations

In this reference implementation, the data is stored directly within Databricks storage for simplicity. In a production setting, however, the data can be stored on cloud data storage such as Azure Blob Storage. Databricks also supports Azure Data Lake Store, Azure SQL Data Warehouse, Azure Cosmos DB, Apache Kafka, and Hadoop.

Cost considerations

Azure Databricks is a premium Spark offering with an associated cost. In addition, there are standard and premium Databricks pricing tiers.

For this scenario, the standard pricing tier is sufficient. However, if your specific application requires automatically scaling clusters to handle larger workloads or interactive Databricks dashboards, the premium level could increase costs further.

The solution notebooks can run on any Spark-based platform with minimal edits to remove the Databricks-specific packages. See the following similar solutions for various Azure platforms:

- Python on Azure Machine Learning Studio
- SQL Server R services

- PySpark on an Azure Data Science Virtual Machine

Deploy the solution

To deploy this reference architecture, follow the steps described in the GitHub repository to build a scalable solution for scoring Spark models in batch on Azure Databricks.

Related architectures

We have also built a reference architecture that uses Spark for building real-time recommendation systems with offline, pre-computed scores. These recommendation systems are common scenarios where scores are batch-processed.