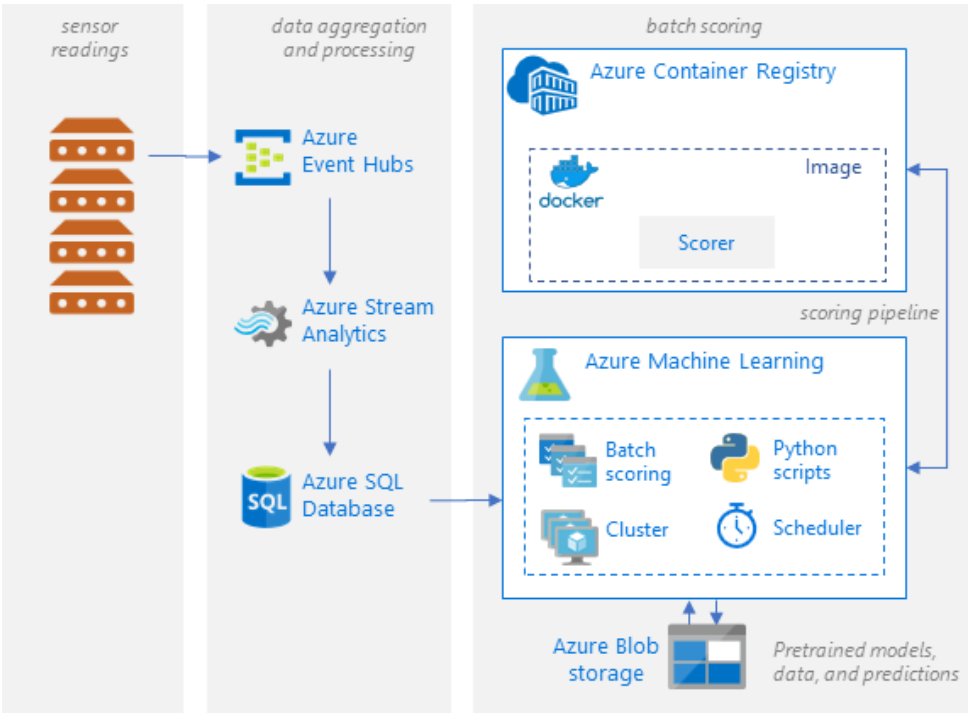
Batch scoring of Python machine learning models on Azure

01/30/2019 • 5 minutes to read • Contributors all

In this article

This reference architecture shows how to build a scalable solution for batch scoring many models on a schedule in parallel using Azure Machine Learning Service. The solution can be used as a template and can generalize to different problems.

A reference implementation for this architecture is available on GitHub.



Scenario: This solution monitors the operation of a large number of devices in an IoT setting where each device sends sensor readings continuously. Each device is assumed to be associated with pretrained anomaly detection models that need to be used to predict whether a series of measurements, that are aggregated over a predefined time interval, correspond to an anomaly or not. In real-world scenarios, this could be a stream of sensor readings that need to be filtered and aggregated before being used in training or real-time scoring. For simplicity, this solution uses the same data file when executing scoring jobs.

This reference architecture is designed for workloads that are triggered on a schedule. Processing involves the following steps:

1. Send sensor readings for ingestion to Azure Event Hubs.
2. Perform stream processing and store the raw data.
3. Send the data to a Machine Learning cluster that is ready to start taking work. Each node in the cluster runs a scoring job for a specific sensor.

4. Execute the scoring pipeline, which runs the scoring jobs in parallel using Machine Learning Python scripts. The pipeline is created, published, and scheduled to run on a predefined interval of time.
5. Generate predictions and store them in Blob storage for later consumption.

Architecture

This architecture consists of the following components:

Azure Event Hubs. This message ingestion service can ingest millions of event messages per second. In this architecture, sensors send a stream of data to the event hub.

Azure Stream Analytics. An event-processing engine. A Stream Analytics job reads the data streams from the event hub and performs stream processing.

Azure SQL Database. Data from the sensor readings is loaded into SQL Database. SQL is a familiar way to store the processed, streamed data (which is tabular and structured), but other data stores can be used.

Azure Machine Learning Service. Machine Learning is a cloud service for training, scoring, deploying, and managing machine learning models at scale. In the context of batch scoring, Machine Learning creates a cluster of virtual machines on demand with an automatic scaling option, where each node in the cluster runs a scoring job for a specific sensor. The scoring jobs are executed in parallel as Python-script steps that are queued and managed by Machine Learning. These steps are part of a Machine Learning pipeline that is created, published, and scheduled to run on a predefined interval of time.

Azure Blob Storage. Blob containers are used to store the pretrained models, the data, and the output predictions. The models are uploaded to Blob storage in the 01_create_resources.ipynb notebook. These one-class SVM models are trained on data that represents values of different sensors for different devices. This solution assumes that the data values are aggregated over a fixed interval of time.

Azure Container Registry. The scoring Python script runs in Docker containers that are created on each node of the cluster, where it reads the relevant sensor data, generates predictions and stores them in Blob storage.

Performance considerations

For standard Python models, it's generally accepted that CPUs are sufficient to handle the workload. This architecture uses CPUs. However, for deep learning workloads, GPUs generally outperform CPUs by a considerable amount — a sizeable cluster of CPUs is usually needed to get comparable performance.

Parallelizing across VMs versus cores

When running scoring processes of many models in batch mode, the jobs need to be parallelized across VMs. Two approaches are possible:

- Create a larger cluster using low-cost VMs.
- Create a smaller cluster using high performing VMs with more cores available on each.

In general, scoring of standard Python models is not as demanding as scoring of deep learning models, and a small cluster should be able to handle a large number of queued models efficiently. You can increase the number of cluster nodes as the dataset sizes increase.

For convenience in this scenario, one scoring task is submitted within a single Machine Learning pipeline step. However, it can be more efficient to score multiple data chunks within the same pipeline step. In those cases, write custom code to read in multiple datasets and execute the scoring script for those during a single-step execution.

Management considerations

- **Monitor jobs**. It's important to monitor the progress of running jobs, but it can be a challenge to monitor across a cluster of active nodes. To inspect the state of the nodes in the cluster, use the Azure Portal to manage the machine learning workspace. If a node is inactive or a job has failed, the error logs are saved to blob storage, and are also accessible in the Pipelines section. For richer monitoring, connect logs to Application Insights, or run separate processes to poll for the state of the cluster and its jobs.
- **Logging**. Machine Learning Service logs all stdout/stderr to the associated Azure Storage account. To easily view the log files, use a storage navigation tool such as Azure Storage Explorer.

Cost considerations

The most expensive components used in this reference architecture are the compute resources. The compute cluster size scales up and down depending on the jobs in the queue. Enable automatic scaling programmatically through the Python SDK by modifying the compute's provisioning configuration. Or use the Azure CLI to set the automatic scaling parameters of the cluster.

For work that doesn't require immediate processing, configure the automatic scaling formula so the default state (minimum) is a cluster of zero nodes. With this configuration, the cluster starts with zero nodes and only scales up when it detects jobs in the queue. If the batch scoring process happens only a few times a day or less, this setting enables significant cost savings.

Automatic scaling may not be appropriate for batch jobs that happen too close to each other. The time that it takes for a cluster to spin up and spin down also incurs a cost, so if a batch workload begins only a few minutes after the previous job ends, it might be more cost effective to keep the cluster running between jobs. That depends on whether scoring processes are scheduled to run at a high frequency (every hour, for example), or less frequently (once a month, for example).

Deployment

To deploy this reference architecture, follow the steps described in the GitHub repo.