Microsoft

Streaming Predictive Maintenance Demo

Last updated: 10/27/2017

# Introduction

This document will walk through the architecture of the Energy demo, providing details about the proposed scenario and the different components that are part of it.

Streaming Predictive Maintenance recreates a scenario that enables fast and easy analysis of the current situation in the transformers in the energy grid of an energy enterprise. Incoming data from the transformer sensors will be analyzed and pushed to a real-time dashboard, where it can be consulted to see how each transformer is performing, and whether human intervention is required.

Streaming Predictive Maintenance is powered by Azure. In order to build a business that handles and analyses hundreds of events per second, they not only rely on Azure’s proven scalability and availability but also its breadth of features to get to market quickly. By using services like HDInsight, Azure Automation or Operations Management Suite, the company can instead focus on empowering first responders instead of maintaining infrastructure.

# SETUP

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | 1. Open Powershell. | Before starting the demo, it is necessary to follow a couple of steps to deploy the Azure resources. |
| cd {PATH\_TO\_SOLUTION}\EnergyDemoARMDeploy | 1. Replace {PATH\_TO\_SOLUTION} with the path where the Demo solution is located. 2. Execute the command. |  |
| .\Deploy-Demo1.ps1 | 1. Execute the command |  |
|  | 1. Introduce your credentials and then click on Sign in. |  |
|  | 1. Introduce the option of your selected subscription 2. Press Enter | You will be displayed a list of available subscriptions for your account. Use whatever it fits best for you. |
|  | 1. Write a name for the deployment 2. Press Enter | Choose a name for the deployment. This name will be used to prefix all the deployed resources. **Make sure it meets the following requirements:**   * **Contains only letters or numbers (but not only numbers), as no other character is allowed**. * **It does not exceed 6 characters**.   Also make sure that is a unique name, not taken by any other resource. |
|  | 1. Write down the values of ClientId and TenantId | With that, the deployment of the resources will start.  It will start creating an Azure Active Directory application that is necessary to follow this demo. Write down the ClientId and TenantId values, as you will need them later. |
|  |  | Bear in mind that it could take up to 2 hours to be ready, since resources like HDInsight clusters take a long time to be created. |
|  | 1. Go to the Azure Portal 2. Login with your credentials 3. Click on Azure Active Directory | Once deployed, it is necessary to grant permissions for your user in the recently created Azure AD application. |
|  | 1. Click on App registrations |  |
|  | 1. In the search box, type the ClientId previously generated. 2. Click on the application found |  |
|  | 1. Click on Required permissions |  |
|  | 1. Click on Grant permissions |  |
|  | 1. Click on Yes | With this operation, your account has now permissions to access the PowerBI APIs. They are used later to send the processed information to PowerBI, so you can see a dashboard and take a look at what’s happening in real time. |
|  | 1. Open the [PowerBI webapp](https://app.powerbi.com/) 2. Sign in with your credentials | Now we need to create a workspace. This workspace will be used to store a dataset in which the data will be sent. |
|  | 1. Click on Workspaces |  |
|  | 1. In the opened blade, click on *Create app workspace* at the bottom. |  |
|  | 1. Enter a name for your workspace. Choose whatever you want. Write it down as you will use it later. 2. Leave the rest of options as they are. |  |
|  | 1. Click on Save |  |
|  | 1. Open Powershell. | Lastly, we are going to create a dataset for PowerBI. This dataset is where the information is sent. Since each user will own his own dataset, we need you to create one before going any further. |
| cd {PATH\_TO\_SOLUTION}\EnergyDemoARMDeploy | 1. Replace {PATH\_TO\_SOLUTION} with the path where the Demo solution is located. 2. Execute the command. |  |
| .\CreateStreamingDataset.ps1 /workspaceName {WORKSPACE\_NAME} | 1. Replace {WORKSPACE\_NAME} with the name of the workspace you previously created. 2. Execute the command. 3. Introduce your credentials. |  |
|  | 1. Click on Accept |  |
|  |  |  |
|  | 1. Open the Azure Portal 2. Login with your account 3. Click on Resource Groups | Now, let’s configure some alerts in OMS. When those alerts are triggered the Spark cluster will be automatically upscaled or downscaled |
|  | 1. Click on the deployed resource group 2. Click on the Log Analytics instance |  |
|  | 1. Click on OMS Portal |  |
|  | 1. Click on Log Search |  |
|  | 1. Enter the following query in the search box:   \* Type=Perf ObjectName=Processor InstanceName=\_Total CounterName=\"% Processor Time\" | measure avg(CounterValue) interval 30minutes   1. Click the Alert button | This query gets the Spark cluster average processor usage on the last 30 minutes. |
|  | 1. Write a name for the alert. | First alert will trigger when the processor usage is too high. |
|  | 1. Use a time window of 30 minutes |  |
|  | 1. Same for the alert frequency |  |
|  | 1. Set alert based on Metric measurement 2. Choose Aggregate Value “Greater than 80” 3. Set trigger alert based on “Total breaches Greater than 0” |  |
|  | 1. Disable Email notifications |  |
|  | 1. Enable runbook | A runbook has already been deployed by the setup scripts and is automatically selected once you click “Yes” |
|  |  | And that’s all!  Now you have everything you need to start the demo! |

# GENERATING LIVE DATA WITH KAFKA

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | 1. Open the Azure Portal 2. Login with your account 3. Click on Resource Groups | In order to start analyzing data in real time, it is necessary to obtain data from the transformers that currently exists in the grid. These transformers are grouped in substations, which are the elements that actually produce the data that the system is going to consume. This data provides information about voltage levels, temperature, and load of the transformers, and gives a clue about how the system’s behavior.  All the data generated by the substations is going to be sent to a Kafka cluster, where will be stored to be processed later.  So, let’s start configuring it. |
|  | 1. Click on the deployed resource group 2. Click on the Kafka cluster | We are going to start creating a topic in the Kafka cluster. A topic is where all the messages will be sent and stored.  To do so we first need to connect to the cluster via SSH. |
|  | 1. Click on Secure Shell SSH |  |
|  | 1. Copy the hostname |  |
|  | 1. Open your SSH client 2. Connect to the host previously copied | To connect using SSH, we need a SSH client like PuTTY or MobaXTerm. Make sure you have installed one of them. |
|  | 1. Use the following credentials:  * drwho * Patata.123456  1. If everything went correctly, you should see a welcome screen. |  |
| sudo apt install jq | 1. Execute the command | The command we need to execute to create the topic needs to know the hostnames of all the brokers in the cluster, since they will contain the partitions that are created for a particular topic.  Partitions allows to parallelize a topic by splitting the data in a topic across multiple brokers.  To know these brokers, we must execute a couple of commands first. |
| export KAFKAZKHOSTS=`curl --silent -u admin:'Patata.123456' G https://{CLUSTER\_NAME}.azurehdinsight.net/api/v1/clusters/{CLUSTER\_NAME}/services/ZOOKEEPER/components/ZOOKEEPER\_SERVER | jq -r '["\(.host\_components[].HostRoles.host\_name):2181"] | join(",")'`  export KAFKABROKERS=`curl --silent -u admin:'Patata.123456' -G https:// {CLUSTER\_NAME}.azurehdinsight.net/api/v1/clusters/{CLUSTER\_NAME}/services/KAFKA/components/KAFKA\_BROKER | jq -r '["\(.host\_components[].HostRoles.host\_name):9092"] | join(",")'`  echo '$KAFKAZKHOSTS='$KAFKAZKHOSTS  echo '$KAFKABROKERS='$KAFKABROKERS | 1. Replace {CLUSTER\_NAME} with the name of the cluster. 2. Execute the command. |  |
| /usr/hdp/current/kafka-broker/bin/kafka-topics.sh --create --replication-factor 1 --partitions 2 --topic timeseries --zookeeper $KAFKAZKHOSTS | 1. Execute the command | Now it’s time to run the command that creates the topic. Its name will be *timeseries*.  We will create two partitions for such topic. |
| /usr/hdp/current/kafka-broker/bin/kafka-console-consumer.sh --zookeeper $KAFKAZKHOSTS --topic timeseries --from-beginning | 1. Execute the command 2. Do not close the SSH client, leave it open in background. | Finally, we are going to leave a consumer running in the cluster.  A Kafka consumer is in charge of reading the messages stored in a topic. We are going to leave one running to test later that the data generated from the substations and sent to the topic is arriving correctly. |
|  | 1. Go to the Azure Portal 2. Go to the same resource group as before, called EnergyDemo. 3. Click on the Virtual Machine. | Now we are ready to start sending all the information generated by the substations.  There is an application created to simulate this real-time data. Since it needs to connect to the brokers in which the topic is located, it is necessary that both the machine where the program is executed and the cluster are in the same network. Otherwise, it would be impossible to send the data to Kafka.  Thus, we are going to use a Virtual Machine created in the resource group. |
|  | 1. Click on Connect 2. Open the .rdp file downloaded |  |
|  | 1. Click on More choices 2. Click on *User a different account* |  |
|  | 1. Use the following credentials  * drwho * Patata.123456  1. Click on Ok |  |
| cd C:\Demo\LiveDataGenerator | 1. Open a command window in the VM 2. Execute the command | You will find a program in the VM desktop called LiveDataGenerator  This program will send, every second, information about several substations. This information, as we said at the beginning, is related to temperature, voltage and load of the substations. A window with a lot of rows similar to the ones in the left will appear. |
| Microsoft.EnergyDemo.LiveDataGenerator timeseriesspearfish true | 1. Execute the command |  |
|  | 1. Go back to the SSH client | Now, if you take a look at the SSH client, you will see that new data is incoming each second to the topic.  That’s all! Now you have the data ingestion part ready, generating live data and being able to be consumed by the cluster which will process it. |

# DETECTING ANOMALIES WITH SPARK STREAMING

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | 1. Open the Azure Portal 2. Login with your account 3. Click on Resource Groups | It’s time to start analysing the arriving information. To do so, we are going to use [Spark Streaming](https://spark.apache.org/streaming/), which allow us to process a stream of data. Basically, there is one Spark Streaming job, which is submitted to the Spark cluster and it is running until the user wants to stop it.  So let’s start connecting to the Spark cluster. |
|  | 1. Click on EnergyDemo resource group 2. Click on the Spark cluster |  |
|  | 1. Click on Secure Shell SSH |  |
|  | 1. Copy the hostname |  |
|  | 1. Open MobaXterm 2. Connect to the SSH cluster using the hostname copied 3. Use the following credentials:  * drwho * Patata.123456 |  |
|  | 1. Click on *Upload to current folder* |  |
|  | 1. Go to the folder where the demo solution is located 2. Then, from there, go to Python > Streaming 3. Click on StreamingAnomalyDetection.py 4. Click open | Inside this folder it is located StreamingAnomalyDetection.py, which is the job in charge of everything. It works as follows: The job is submitted with a set of parameters it needs to work properly, like Azure Storage credentials, user’s credentials, and so on.  It will start downloading a Machine Learning model used to evaluate data. We provide you a pre-trained model in Azure Storage, so don’t worry about creating one. Just have in mind that it has been trained with data very similar to the one that the program is sending, so it can effectively predict anomalies.  Then it will be processing the data read from Kafka. It will classify if the data is an anomaly or not, and will send it both to the PowerBI dataset we created at the beginning and to the Azure Storage account, so the model can be retrained later (we will see how). |
|  | 1. Go to the Azure Portal 2. Go to the deployed resource group 3. Click on the Azure Storage account | Before submitting the job, we need several parameters to execute it.  Let’s see how we can take them. |
|  | 1. Click on Access keys | First, we need the Azure Storage credentials. |
|  | 1. Write down both the Storage account name and the key |  |
|  | 1. In the Azure Portal, go to the Kafka cluster. 2. Click on Dashboard | Next, we need the Kafka cluster Zookeeper host. |
|  | 1. You will be asked for credentials. Use:  * admin * Patata.123456 |  |
|  | 1. Click on Kafka |  |
|  | 1. Click on Configs |  |
|  | 1. In the Kafka Broker tab, write down the first Zookeeper hostname and port. |  |
| export PYSPARK\_PYTHON=python3  spark-submit --packages org.apache.spark:spark-streaming-kafka-0-8\_2.11:2.0.2 StreamingAnomalyDetection.py '{AZURE\_STORAGE\_ACCOUNT\_NAME}' '{AZURE\_STORAGE\_ACCOUNT\_PASSWORD}' '{ZOOKEEPER\_HOSTNAME}' '{TENANT\_ID}' '{CLIENT\_ID}' '{USERNAME}' '{PASSWORD}' '{WORKSPACE\_NAME}' | 1. Replace:  * {AZURE\_STORAGE\_ACCOUNT\_NAME} with the Storage account name * {AZURE\_STORAGE\_ACCOUNT\_PASSWORD} with the Storage account key * {ZOOKEEPER\_HOSTNAME} with the Zookeeper hostname * {TENANT\_ID} with the TenantId obtained during the deployment * {CLIENT\_ID} with the ClientId obtained during the deployment * {USERNAME} with your PowerBI account name * {PASSWORD} with your PowerBI account password * {WORKSPACE\_NAME} with the Workspace name you created during the setup  1. Press Enter to submit the job | Ok, now it’s time to submit the job.  Remember that [previously in the document](#_GENERATING_LIVE_DATA), we left the program in charge of generating data running, so now we should have our Kafka topic full of data. The Spark job will read the topic every 10 seconds. |
|  |  | Now, every 10 seconds, you will see the raw information from a substation. As we said, this information includes voltage, load and temperature of the device. |
|  |  | You will see the prediction as well for each row of data, indicating if it is an anomaly or not.  Besides, there are few other parameters like prediction, and sigma, which tells the user how reliable the anomaly predicted is. |
|  | 1. Go to the [PowerBI service](https://app.powerbi.com/) 2. Login using your credentials | However, there is no way a user can see anything easily here. How can anyone determine if a substation needs to be reviewed or not from the SSH client?  That’s where PowerBI helps. The data you have seen is also sent to PowerBI so we can build a dashboard and have an overview of all the substations that are sending data and see if there are outliers in real time. |
|  | 1. Click on Workspaces 2. Select the one you created in the setup | It’s not necessary that you have any knowledge of PowerBI, since we have built a dashboard for you. |
|  | 1. On dashboards, at the left, select NTEDemoDashboard |  |
| cid:image006.png@01D31693.B79698E0 |  | Much better, right?  Here you can see how the dashboard is updated in real time while the substations are sending their metrics. |
| cid:image001.png@01D31693.5E6EF550  cid:image002.png@01D31693.5E6EF550  cid:image004.png@01D31693.9414EFC0 |  | There are also some report for non real-time data. |

# DETECTING ANOMALIES WITH SPARK STREAMING in Spearfish

## Making Kafka cluster accessible for Spearfish clusters

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | 1. Open the Azure Portal 2. Login with your account 3. Click on Resource Groups 4. Click on the EnergyDemo resource group | Apart from the evaluation using the HDInisght cluster, we are going to show how to perform it using a platform called Spearfish, which simplifies the cluster management.  Spearfish is a unified approach that makes data analytics simpler, with a fully managed cloud infrastructure and a collaborative workspace to allow data scientists and data engineers work together.  In order to allow Spearfish clusters access to the Kafka cluster, we need to follow some steps first. |
|  | 1. Click on the Kafka cluster |  |
|  | 1. Click on the cluster URL |  |
|  | 1. Introduce the credentials when prompted:  * admin * Patata.123456 |  |
|  | 1. Click on Kafka |  |
|  | 1. Click on Configs |  |
|  | 1. In the filter, write *kafka-env* |  |
|  | 1. In the *kafka-env template* field, add the following text at the end:   # Configure Kafka to advertise IP addresses instead of FQDN  IP\_ADDRESS=$(hostname -i)  echo advertised.listeners=$IP\_ADDRESS  sed -i.bak -e '/advertised/{/advertised@/!d;}' /usr/hdp/current/kafka-broker/conf/server.properties  echo "advertised.listeners=PLAINTEXT://$IP\_ADDRESS:9092" >> /usr/hdp/current/kafka-broker/conf/server.properties |  |
|  | 1. Now, in the same filter field as before, write *listeners* |  |
|  | 1. Change the value to of the listeners parameter to:   PLAINTEXT://0.0.0.0:9092 |  |
|  | 1. Click Save to save the performed changes |  |
|  | 1. If prompted, click on Proceed anyway |  |
|  | 1. Click on Service Actions 2. Then click on Turn on Maintenance mode |  |
|  | 1. Click on restart, and then on restart all affected |  |
|  | 1. Click on Confirm |  |
|  | 1. Wait until it has completed and then click Ok |  |
|  | 1. Click on Service actions, ant then Turn off Maintenance mode | Now the cluster is configured to advertise the brokers IP addresses properly.  Before continuing the guide, write down the following parameters, as you will need them later:   * Cluster SSH address: Which is {CLUSTER\_NAME}-ssh.azurehdinsight.net * Cluster SSH user: drwho * Cluster SSH password: Patata.123456 |
|  | 1. Click on Summary | Finally, you will need to write down the IP addresses of the kafka brokers. |
|  | 1. Click on a Kafka broker |  |
|  | 1. Repeat the same steps for all the brokers. | In the summary window you will find the IP address of the broker. Write down the addresses of all the brokers. |

## Creating the Spearfish cluster and running the script

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | 1. Open the Azure Portal 2. Login with your account 3. Click on Resource Groups 4. Click on the EnergyDemo resource group | Let’s see how to create a Spearfish cluster using the Azure portal. |
|  | 1. Click on Add |  |
|  | 1. Search databricks 2. Click on Azure Databricks (preview) |  |
|  | 1. In the opening blade, click on Create |  |
|  | 1. Fill the field Workspace name with the name you want. 2. Leave the rest as it is. |  |
|  | 1. Click on create | Now wait until the resource is created. |
|  | 1. Access to <https://westeurope.azuredatabricks.net/> 2. Click on Sign in with Azure AD 3. Login with your account | Once created, let’s access to the portal. |
|  |  | Now we are inside the Spearfish portal. The core of Spearfish is Spark, so we are going to create a Spark cluster so we can run our evaluation script on it.  We could do it using the portal, but at the time of writing this document, Spark clusters that use Python 3 are only available through the API. We have prepared a script to do so. But first is necessary to create a token to access the Spearfish API. Let’s do it. |
|  | 1. On the top right corner, click on the user icon 2. Then, click on Admin Console. |  |
|  | 1. Click on Access Control |  |
|  | 1. In the Personal Access Tokens section, click on Enable | This will allow users to use personal access tokens instead of passwords to access the Spearfish REST API. |
|  | 1. On the top right corner, click on the user icon 2. Then, click on User Settings. | Now let’s create the access token. |
|  | 1. Click on Generate New Token |  |
|  | 1. Fill the comment field, and leave the Lifetime as it is 2. Click on Generate |  |
|  | 1. Make sure you copy the token generated, as it will not be available once you click on Done 2. Click on Done | Now it’s time to use the script and create the cluster. |
|  | 1. Go to EnergyDemo 🡪 AnomalyDetection 🡪 Databricks 2. Open CreatePython3Cluster.py |  |
|  | 1. Replace *{REPLACE\_WITH\_CLUSTER\_NAME}* with the cluster name you want. 2. Replace *{REPLACE\_WITH\_YOUR\_TOKEN}* with the generated token. | This python script basically creates a POST request against the Cluster Management API from Spearfish, using the Create cluster endpoint.  The body contains the configuration of the Spark cluster. We are going to use the version 3.2 of the cluster (that uses Scala 2.11).  Finally, we are setting the PYSPARK\_PYTHON environment variable, so Spark uses Python 3 when submitting jobs. |
|  | 1. Open the Command Prompt |  |
| cd {PATH\_TO\_ENERGY\_DEMO}\EnergyDemo\AnomalyDetection\Databricks | 1. Replace {PATH\_TO\_ENERGY\_DEMO} with the path to your EnergyDemo solution 2. Execute the command in the cmd |  |
| python CreatePython3Cluster.py | 1. Execute the command | If the request is done correctly, the response will be the id of the cluster. |
|  | 1. In the Spearfish portal, click on Clusters. | The cluster is now being created. It may take up to 20 minutes to be ready.  In the portal, going to the cluster section, you will see the state of the cluster. When it is ready to be used, the state will pass from Pending to Running. |
|  | 1. Click on workspaces | Now we need to attach some libraries to the cluster. Spearfish allows to add what are called Shared Libraries, which can be used across all the cluster that we create in the platform, and are deployed in all the nodes of the cluster. The platform automatically manages this, so we just need to point what libraries we want to use. |
|  | 1. Click on the arrow next to Shared 2. Click on Create 3. Click on Library | Spearfish allows to add packages that are available on [PyPI](https://pypi.python.org/pypi) (Python Package Index). |
|  | 1. Select Upload Python Egg or PyPI |  |
|  | 1. Copy and paste the list of libraries in the field:   future, Pillow, xmltodict, wheel, matplotlib, EasyDict, numpy, scikit-learn, scipy, scikit-image, Cython, pandas, statsmodels, urllib3, opencv, azure, opencv-python   1. Click on Install Library | It is easy to add multiple libraries at the same time. Just separate them with commas and you are done. |
|  | 1. Click on the Attach checkbox | Now we need to select which clusters are going to use that libraries. We need to attach them to the clusters.  In our case, we only have one, but we could attach the library at multiple running clusters.  It may take a couple of minutes to attach the libraries. Be sure that the status is changed to Attached before continuing. |
|  | 1. Go to the workspaces section 2. Click on the arrow next to *Workspace* 3. Click on Import | Now we need to configure the access to Kafka from the clusters. We have to do so using a bash script.  Spearfish has a method to execute custom scripts on the cluster called Init Scripts. These are Shell scripts that are executed when the cluster is being initialized. These scripts are located in a special directory in the Spearfish File System (DBFS), which is a distributed file system installed on all Spark clusters in this platform.  Let’s see how to create a Init script. |
|  | 1. Click on the box to upload a file | We have a notebook prepared that setups a Init script. Let’s import it. |
|  | 1. The notebook is in EnergyDemo\AnomalyDetection\Databricks\Notebooks 2. Double click on ConfigureKafkaAccessNotebook.ipynb |  |
|  | 1. Click on Import |  |
|  | Use here the parameters you wrote down in the [section 5.1](#_Making_Kafka_cluster).   1. Replace {CLUSTER\_SSH\_ADDRESS} with the SSH address of the cluster. 2. Replace {CLUSTER\_SSH\_USER} with the SSH user of the cluster. 3. Replace {CLUSTER\_SSH\_PASSWORD} with the SSH password of the cluster. 4. Replace each {BROKER\_X\_IP\_ADDRESS} with the IP address of each broker. 5. Replace {REPLACE\_WITH\_CLUSTER\_NAME} with the name of the cluster. | The imported notebook will be opened automatically.  The shell script is stored in a variable that will be later dumped in a file.  The biggest advantage of using Init scripts is that you can do whatever you would do in an actual cluster. So, if you seem limited by the Spearfish platform, you can always use the potential of the bash to execute any tasks that otherwise would be impossible.  All the notebooks have access to the DBFS API. There is a list of functions to easily operate with it, like list files, create directories, remove files/directories, and so on.  We are using the DBFS API here to put a shell script in a folder called **init**, which store all the initialization scripts for the cluster. Inside the init folder, there is a directory for each cluster created in the platform. Be sure put your cluster name correctly. Otherwise, the script won’t be executed. |
|  | 1. Click on Detached 2. Select the created cluster to attach the notebook to it | We need to attach this notebook to a running cluster in order to execute it. |
|  | 1. Run the notebook |  |
|  |  | Ok, now the Init script is installed. But we need to execute it. To do so, we need to restart the cluster. |
|  | 1. Go to Clusters |  |
|  | 1. Click on the restart button of your cluster. | The restart may take 20 minutes or even something more. |
|  | 1. Go to the workspace 2. Import the Notebook StreamingEvaluation.ipynb, which is in the same directory as the ConfigureKafkaAccessNotebook.pynb one. | We are almost done. We have the access configured and the necessary libraries installed, so we just need to import the notebook that is in charge of the evaluation. |
|  | 1. Replace:  * {AZURE\_STORAGE\_ACCOUNT\_NAME} with the Storage account name * {AZURE\_STORAGE\_ACCOUNT\_PASSWORD} with the Storage account key * {TENANT\_ID} with the TenantId obtained during the deployment * {CLIENT\_ID} with the ClientId obtained during the deployment * {USERNAME} with your PowerBI account name * {PASSWORD} with your PowerBI account password * {WORKSPACE\_NAME} with the Workspace name you created during the setup * Each {BROKER\_X\_IP\_ADDRESS} with each broker IP address. | The evaluation is done using the available nodes in the cluster.  The notebook is divided in several blocks.  Before anything else, replace the parameters from the second block. |
|  | 1. Attach the notebook to the cluster | Ok, now it’s time to submit the job.  To do so, let’s run all the notebook blocks. |
|  | 1. Click on Run All |  |
|  | 1. Give it a name | Now the cluster is listening to Kafka and processing all the data arriving.  Now we need to run the LiveDataSimulator in order to send information about substations. |

## Executing the LiveDataGenerator

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | 1. Open the Azure Portal 2. Login with your account 3. Click on Resource Groups | Let’s create a separate topic for the data generated for Spearfish, since it is aggregated a bit different.  The steps are the same as in the [section 3](#_GENERATING_LIVE_DATA). |
|  | 1. Click on the deployed resource group 2. Click on the Kafka cluster |  |
|  | 1. Click on Secure Shell SSH |  |
|  | 1. Copy the hostname |  |
|  | 1. Open your SSH client 2. Connect to the host previously copied | To connect using SSH, we need a SSH client like PuTTY or MobaXTerm. Make sure you have installed one of them. |
|  | 1. Use the following credentials:  * drwho * Patata.123456  1. If everything went correctly, you should see a welcome screen. |  |
| /usr/hdp/current/kafka-broker/bin/kafka-topics.sh --create --replication-factor 1 --partitions 2 --topic timeseriesspearfish --zookeeper $KAFKAZKHOSTS | 1. Execute the command. | Run the command that creates the topic. Its name will be *timeseriesspearfish*.  We will create two partitions for such topic. |
|  | 1. Go to the Azure Portal 2. Go to the same resource group as before, called EnergyDemo. 3. Click on the Virtual Machine. | Now we are ready to start sending all the information generated by the substations.  Let’s connect to the same Virtual Machine as before. |
|  | 1. Click on Connect 2. Open the .rdp file downloaded |  |
|  | 1. Click on More choices 2. Click on *User a different account* |  |
|  | 1. Use the following credentials  * drwho * Patata.123456  1. Click on Ok |  |
| cd C:\Demo\LiveDataGenerator | 1. Open a command window in the VM 2. Execute the command |  |
| Microsoft.EnergyDemo.LiveDataGenerator timeseriesspearfish true | 1. Execute the command |  |
|  |  | Now all the data generated from substations will be sent to Kafka, and, this time, the Spearfish cluster is going to process it. |
|  |  | Go to the PowerBI dashboard, and you will see the dashboard updated in real time while the substations are sending their metrics. |

# Creating real-time dashboard

| Screen | Click Steps | Demo Script |
| --- | --- | --- |
|  | 1. Go to the [PowerBI service](https://app.powerbi.com/) 2. Login using your credentials |  |
|  | 1. Click on Workspaces 2. Select the one you created in the setup |  |
|  | 1. Select Reports |  |
|  | 1. Expand Create and click Streaming dataset |  |
|  | 1. Create streaming dataset (API) |  |
|  | 1. In the next step, check “Historic data analysis” and fill the column names and data types as seen in the image. Then, click “create” |  |
|  | 1. Copy the Push URL and use it when it comes to set up the script to push data into the dashboard. |  |
|  | 1. Now, select create report based on the created dataset |  |
|  | 1. Save the report as soon as possible |  |
|  | 1. Name it and start generating it |  |
|  | 1. Start by adding cards for different values:  * Avg temperature, load and voltage. * Max temperature, load and voltage * Anomalies avg and count * Row count |  |
|  | 1. Next, add 3 clustered column charts, one for each measure (temperature, load and voltage), with ValueTime in axis and anomaly as legend |  |
|  | 1. Add 3 line charts, one for each measure, with ValueTime in axis and DeviceId as legend |  |
|  | 1. Add 3 scatter plot charts, one for each measure, with ValueTime in details, the average of the measure in X axis, the average of the next measure in Y axis, maximum of measure both in Size and Color saturation |  |
|  | 1. Now is time to add some custom controls, click import from store |  |
|  | 1. Look for bowtie chart and add it, then bullet chart by okviz and add it. Add Tornado chart and Waffle chart as well |  |
|  | 1. Add 3 bullet charts, one for each measure, with DeviceId in category, maximum of measure in value and average of measure in comparison value |  |
|  | 1. Now is time to make the report area bigger, click on the background and select Format. Then, change type to Custom in Page Size, and set Width to 2560 and Height to 1080 |  |
|  | 1. Add now 3 treemap charts, one for each measure, with DeviceId in group, average of measure in values and maximum of measure in color saturation |  |
|  | 1. Add 3 tornado charts, one for each measure, with DeviceId in group and average of measure and anomaly on values |  |
|  | 1. Add 3 bowtie charts, one for each measure, with average of measure as value and DeviceId as destination |  |
|  | 1. Add a Waffle chart, with DeviceId as category data and anomaly as values |  |
|  | 1. Now modify the format of the different elements created in previous steps, color, titles, etc. |  |
|  | 1. Select one control and click pin visual |  |
|  | 1. Select New dashboard in the dialog, name it as seen in the image and click pin |  |
|  | 1. Pin all the controls that you want to see in the dashboard and then, select the dashboard in the lateral menu to edit it |  |
|  | 1. Move and resize the controls to set the desired layout and you are finished |  |

# re-training the model to improve predictions

|  |  |  |
| --- | --- | --- |
|  | 1. Open the Azure Portal 2. Login with your account 3. Click on Resource Groups | From time to time, it is recommended to train the model again with new data to improve predictions, detecting anomalies with more accuracy. This way we can minimize false negatives or false positives (for example, a temperature value that is totally normal for a given substation could be detected as an anomaly, and we don’t want that).  In a real scenario, this could be performed in a daily basis. At the end of the day, the data outputted from the Spark cluster to the Azure Storage would be used to train the model and save it in Azure Storage.  Let’s see how we can re-train the model. |
|  | 1. Click on EnergyDemo resource group 2. Click on the Spark cluster |  |
|  | 1. Click on Secure Shell SSH |  |
|  | 1. Copy the hostname |  |
|  | 1. Open a new session in MobaXterm 2. Connect to the SSH cluster using the hostname copied 3. Use the following credentials:  * drwho * Patata.123456 |  |
|  | 1. Click on *Upload to current folder* |  |
|  | 1. Go to the folder where the demo solution is located 2. Then, from there, go to Python > Batch 3. Click on TrainAnomalyDetection.py 4. Click open |  |
|  | 1. Go to the Azure Portal 2. Go to the deployed resource group 3. Click on the Azure Storage account |  |
|  | 1. Click on Access keys |  |
|  | 1. Write down both the Storage account name and the key |  |
| export PYSPARK\_PYTHON=python3  spark-submit --packages org.apache.spark:spark-streaming-kafka-0-8\_2.11:2.0.2 TrainAnomalyDetection.py '{AZURE\_STORAGE\_ACCOUNT\_NAME}' '{AZURE\_STORAGE\_ACCOUNT\_PASSWORD}' | 1. Replace:  * {AZURE\_STORAGE\_ACCOUNT\_NAME} with the Storage account name * {AZURE\_STORAGE\_ACCOUNT\_PASSWORD} with the Storage account key  1. Execute the command |  |
|  |  | It may take a few minutes to have the new model ready.  After that, it will be stored in Azure Storage, and the Spark job will automatically download it to start (or continue) evaluating data against it. |