Azure Databricks

Last updated: December 1, 2023

Migration guide for init scripts from DBFS to ADLS Gen2

For Databricks Runtime 11.2 and below, move init scripts from DBFS to ADLS Gen2.

Note: In cases where your init scripts are "self-contained," i.e., DO NOT reference other files such as libraries, configuration files, or shell scripts. We highly recommend storing init scripts as workspace files instead, as setup is easier.

In general, you must:

1) Copy all init scripts and files referenced by the init scripts from DBFS to ADLS Gen2.
2) [If applicable] Update the init scripts to point to referenced files on ADLS Gen2.
3) Configure the clusters to authenticate to the ADLS Gen2.
4) Update cluster configuration and cluster policies to reference the init scripts on ADLS Gen2.

1) Copy all init scripts and referenced files from DBFS to ADLS Gen2

Identify all objects (clusters, jobs, DLT pipelines, ...) that use init scripts on DBFS. Use the script detection notebook to prepare a list of individual init scripts. For init scripts that reference files on DBFS, prepare the list of referenced files.
There are different ways of copying files from DBFS to the ADLS Gen2:

1. Outside of the Databricks workspace:
    a. Download identified files from DBFS to the local disk using the Databricks CLI. The CLI needs to be installed and configured with an authentication method of choice).
    b. Upload downloaded init scripts and referenced files using one of the following:
        - Azure CLI (the `az storage blob upload` command),
        - AzCopy (the `azcopy cp` command),
        - Azure Storage Explorer
        - PowerShell
        - Azure Portal

2. Use the Databricks notebook or a Databricks job to:
    a. Make sure that the cluster you use is configured correctly to access ADLS Gen2 (covered in this guide).
    b. In your Databricks Notebook, use the dbutils.fs.cp command to copy files from DBFS to ADLS Gen2

```
dbutils.fs.cp("dbfs:/<path>",
"abfss://<container>@<storage>.dfs.core.windows.net/<path>")
```

2) Update the init scripts to point to referenced files on ADLS Gen 2 cloud storage

To reference files from within your init scripts located on ADLS Gen2, the command inside the init script must be authenticated by one of the supported methods. Databricks recommends using the Service principal's Client ID / Secret (recommended), but customers are free to use Shared Access Signature or Storage Account Keys.

Configure the chosen authentication method as the environment variable defined in each cluster's configuration.

Below is an example of the init script that uses Azure CLI with service principal authentication to access files on ADLS Gen2. The service principal authentication configuration is passed using the environment variables ARM_CLIENT_ID, ARM_CLIENT_SECRET, and ARM_TENANT_ID set to the Databricks cluster. The init script installs the azure-cli package from the Microsoft Linux repository, sets authentication, and downloads a file from the given storage account:

```
#!/bin/bash

# Install azure-cli:
https://learn.microsoft.com/en-us/cli/azure/install-azure-cli-linux?pivots=
apt
apt-get update
apt-get install -y ca-certificates curl apt-transport-https lsb-release
gnupg jq
mkdir -p /etc/apt/keyrings
curl -sLS https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor
> /etc/apt/keyrings/microsoft.gpg
chmod go+r /etc/apt/keyrings/microsoft.gpg
AZ_DIST=$(lsb_release -cs)
echo "deb [arch=`dpkg --print-architecture`
signed-by=/etc/apt/keyrings/microsoft.gpg]
https://packages.microsoft.com/repos/azure-cli/ $AZ_DIST main" | tee
/etc/apt/sources.list.d/azure-cli.list
apt-get update
apt-get install -y azure-cli

# Login to Azure as SP
az login --service-principal -u "${ARM_CLIENT_ID}" -p
"${ARM_CLIENT_SECRET}" --tenant "${ARM_TENANT_ID}"
```

```
# Going to download file
az storage blob download -c "<container-name>" --account-name
"<storage-account-name>" --auth-mode login -n "<file-name>" -f /tmp/test.sh
# check the downloaded file
ls -ls /tmp/test.sh
```

Here is an example of the init script that uses the AzCopy tool to download files from the storage account. As in the previous example, it uses service principal authentication and requires the same set of environment variables.

```
#!/bin/bash

# Install from
https://learn.microsoft.com/en-us/azure/storage/common/storage-use-azcopy-v
10
curl -L -s -o azcopy.tar.gz https://aka.ms/downloadazcopy-v10-linux
tar zxf azcopy.tar.gz
cp azcopy_linux_*/azcopy /usr/bin/
chmod a+x /usr/bin/azcopy
rm -rf azcopy_linux_* azcopy.tar.gz

#
https://learn.microsoft.com/en-us/azure/storage/common/storage-use-azcopy-a
uthorize-azure-active-directory#authorize-a-service-principal
export AZCOPY_AUTO_LOGIN_TYPE=SPN
export AZCOPY_SPA_APPLICATION_ID="${ARM_CLIENT_ID}"
export AZCOPY_SPA_CLIENT_SECRET="${ARM_CLIENT_SECRET}"
export AZCOPY_TENANT_ID="${ARM_TENANT_ID}"

azcopy cp
"https://<storage-account-name>.blob.core.windows.net/<container-name>/<fil
e-name>" /tmp/test2.sh
ls -ls /tmp/test2.sh
```

Here is an example of the Terraform code that uploads the init script from the disk to the configured's storage account and attaches it to the cluster configured with all necessary Spark configuration properties and environment variables:

```
locals {
  storage_account = "..."
  container       = "..."
  source_file     = "adls-init-script.sh"
  tenant_id       = "..."
  sp_secret_path  = "{{secrets/scope/sp-secret}}"
  sp_id_path      = "{{secrets/scope/sp-id}}"
}

resource "azurerm_storage_blob" "init_script" {
  name                   = local.source_file
  storage_account_name   = local.storage_account
  storage_container_name = local.container
  type                   = "Block"
  source                 = local.source_file
  content_md5            = md5(file(local.source_file))
}

resource "databricks_cluster" "with_init_script" {
  cluster_name          = "Test ABFSS init script"
  spark_version         = "10.4.x-scala2.12"
  node_type_id          = data.databricks_node_type.smallest.id
  autotermination_minutes = 20
  autoscale {
      min_workers = 1
      max_workers = 10
  }

  spark_conf = {

"spark.hadoop.fs.azure.account.auth.type.${local.storage_account}.dfs.core.
windows.net" : "OAuth"

"spark.hadoop.fs.azure.account.oauth.provider.type.${local.storage_account}
.dfs.core.windows.net" :
"org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider"

"spark.hadoop.fs.azure.account.oauth2.client.endpoint.${local.storage_accou
nt}.dfs.core.windows.net" :
"https://login.microsoftonline.com/${local.tenant_id}/oauth2/token",

"spark.hadoop.fs.azure.account.oauth2.client.id.${local.storage_account}.df
s.core.windows.net" : local.sp_id_path
```

```
"spark.hadoop.fs.azure.account.oauth2.client.secret.${local.storage_account
}.dfs.core.windows.net" : local.sp_secret_path
  }

  spark_env_vars = {
      "ARM_CLIENT_ID" : local.sp_id_path
      "ARM_CLIENT_SECRET" : local.sp_secret_path
      "ARM_TENANT_ID" : local.tenant_id
  }

  init_scripts {
      abfss {
      destination =
"abfss://${local.container}@${local.storage_account}.dfs.core.windows.net/$
{azurerm_storage_blob.init_script.name}"
      }
  }
}
```

3) Configure your clusters to have access to the cloud storage

Configure your cluster to connect to ADLS Gen2. To access an init script on a cloud storage location, the cluster needs to have the following Spark conf properties configured to authenticate using Azure service principal authentication:

- `spark.hadoop.fs.azure.account.auth.type.<storage-account>.dfs.core.windows.net` - should have fixed value `OAuth`.
- `spark.hadoop.fs.azure.account.oauth.provider.type.<storage-account>.dfs.core.windows.net` - should have fixed value `org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider`.
- `spark.hadoop.fs.azure.account.oauth2.client.endpoint.<storage-account>.dfs.core.windows.net` - should be set to the URL of the AAD(Entra) authentication endpoint, like this: `https://login.microsoftonline.com/<Azure-tenant-id>/oauth2/token`
- `spark.hadoop.fs.azure.account.oauth2.client.id.<storage-account>.dfs.core.windows.net` - should be set to the service principal's application ID (NOT the principal ID!). The reference to the secrets could be used there as well: `{{secrets/scope/sp-id}}`.

- `spark.hadoop.fs.azure.account.oauth2.client.secret.<storage-account>.dfs.core.windows.net` - should be set to the service principal's secret. For security reasons, it's best to use a secret reference like this: `{{secrets/scope/sp-secret}}`.

Example of using Terraform to configure these properties:

```
resource "databricks_cluster" "with_init_script" {
  cluster_name          = "Test ABFSS init script"
  spark_version         = data.databricks_spark_version.latest_lts.id
  node_type_id          = data.databricks_node_type.smallest.id
  autotermination_minutes = 20
  autoscale {
      min_workers = 1
      max_workers = 10
  }

  spark_conf = {

"spark.hadoop.fs.azure.account.auth.type.${local.storage_account}.dfs.core.windows.net": "OAuth",

"spark.hadoop.fs.azure.account.oauth.provider.type.${local.storage_account}.dfs.core.windows.net":
"org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",

"spark.hadoop.fs.azure.account.oauth2.client.endpoint.${local.storage_account}.dfs.core.windows.net":
"https://login.microsoftonline.com/<azure-tenant-id>/oauth2/token",

"spark.hadoop.fs.azure.account.oauth2.client.id.${local.storage_account}.dfs.core.windows.net": "{{secrets/scope/sp-id}}",

"spark.hadoop.fs.azure.account.oauth2.client.secret.${local.storage_account}.dfs.core.windows.net": "{{secrets/scope/sp-secret}}"
  }

  init_scripts {
      abfss {
      destination =
"abfss://test@storage.dfs.core.windows.net/empty_init_script.sh"
      }
  }
```

```
}
```

4) Update cluster configuration and cluster policies to reference the init scripts on ADLS Gen2

Change the init script paths in affected clusters, jobs, Delta Live Tables pipelines, and cluster policies to point to ADLS Gen2 path instead of DBFS. If you use the init scripts detection notebook, click links in the generated HTML tables:

- **Clusters**: edit your existing clusters' configuration: remove init scripts that use files on DBFS, and add init scripts with source "ABFSS", providing the path to the file on ADLS Gen2, e.g. `abfss://<container>@<account>.dfs.core.windows.net/path/to/init-script` (more information).
  Note: If you are using cluster policies, you must update both the cluster policy and the configurations of any other clusters using the policy. Cluster policy changes do not propagate to other clusters using that policy.
- **Jobs**: Edit cluster configuration for each task that uses a dedicated job cluster and each shared job cluster. Update init script location from DBFS to ADLS, as described above.
- **If using Cluster policies**: Open cluster policy, click Edit, and in the policy definition, search for blocks like the following, where N is the item number

```
{
  "init_scripts.N.dbfs.destination": {
    "type": "fixed",
    "value": "dbfs:/FileStore/init-scripts/empty_init_script.sh"
  }
}
```

and replace them with the following, adjusting the file path:

```
{
  "init_scripts.N.abfss.destination": {
      "type": "fixed",
      "value":
"abfss://<container>@<account>.dfs.core.windows.net/empty_init_script.sh"
  }
}
```

- **Delta Live Tables pipelines**: Open pipeline settings, select the "JSON" tab, and in the cluster definition(s), change entries in the `init_scripts` array from

```
{
 "dbfs": {
   "destination": "dbfs:/FileStore/init-scripts/empty_init_script.sh"
 }
}
```

to

```
{
 "abfss": {
   "destination":
"abfss://<container>@<account>.dfs.core.windows.net/empty_init_script.sh"
 }
}
```