Databricks DAB deployment using ADO Pipeline

# Steps for the Databricks Asset Bundle (DAB) deployment exercise:

### Azure Level

## Create Az Entra App for service principal (SP)

## Naming Standard: adv-devops-sp-*{{databricks name}}*

* 1. Provisioned following service principles for each environment.
  2. adv-devops-sp-adv-dnap-eus2-adb-01-i1-dev
  3. adv-devops-sp-adv-dnap-eus2-adb-01-i1-qa
  4. adv-devops-sp-adv-dnap-eus2-adb-01-i1-stage
  5. adv-devops-sp-adv-dnap-eus2-adb-01-i1-prod

## Grant contributor permission at subscription level where the Databricks workspace provisioned.

### Databricks level

We observed databricks SCIM limitation:

1. The **Azure Databricks SCIM Provisioning Connector** application does not support syncing Entra ID Service Principals.
2. Ref documentation:  <https://learn.microsoft.com/en-us/azure/databricks/admin/users-groups/scim/aad#microsoft-entra-id-service-principals-do-not-sync>
3. So, these service principals need to be added manually to respective databricks workspaces.

## Create Databricks service principal (SP)

* + 1. Microsoft Entra ID SP can be located in databricks via add SP dialog window

## Grant workspace admin role to SP.

* + 1. Short term plan it’s a manual grant.
    2. Process the same via automation when it’s ready.

## Grant catalog level access to the SP

* + 1. Grant USE\_CATALOG and CREATE\_SCHEMA

### Azure DevOps level

1. Created service connection at Azure DevOps for SP

2. Create Azure Virtual Machine Scale Set agents

An agent pool is a collection of agents. Instead of managing each agent individually, you organize agents into agent pools. When you configure an agent, it is registered with a single pool, and when you create a pipeline, you specify the pool in which the pipeline runs. When you run the pipeline, it runs on an agent from that pool that meets the demands of the pipelines. Our agent pool works on VM scale set principles.

Azure Virtual Machine Scale Set agents, hereafter referred to as scale set agents, are a form of self-hosted agents that can be auto-scale to meet your demands. This elasticity reduces your need to run dedicated agents all the time. Unlike Microsoft-hosted agents, you have flexibility over the size and image of machines on which agents run.

We have configured self-hosted agent pool for each environment as below-

for development environment - ADV-TF-DEV-EASTUS2-Data and Analytics Portfolio

for QA environment - ADV-TF-QA-EASTUS2-Data and Analytics Portfolio

for stage environment - ADV-TF-STAGE-EASTUS2-Data and Analytics Portfolio

for stage environment - ADV-TF-PROD-EASTUS2-Data and Analytics Portfolio

## 3. Create required variables in the Environment specific Library

A variable group in Azure DevOps is a collection of key-value pairs (variables) that are defined in the library section of a project. These variables can be shared and reused across multiple pipelines, making it easier to manage environment-specific configurations and secrets. They help centralize the management of variables, improve security for sensitive data, and reduce duplication.

Once we create a variable group, we need to provide access to the ADO pipeline to use that variable group.

Example of Variable group name for use case –

1. ADV-ADB- {use-case} -DEPLOY-DEV
2. ADV-ADB- {use-case} -DEPLOY-QA
3. ADV-ADB-{use-case} -DEPLOY-STAGE
4. ADV-ADB-{use-case} -DEPLOY-PROD

Variables in each group:

1. **DATABRICKS\_HOST** – for this variable we need to mention the databricks host value.
2. **env** – for this variable we need to provide the target environment value. This value will be used to validate and deploy DABs.
3. **pythonVersion** – Here we need to mention which python version we need to use.

## 4. Create required variables in the Environment specific Library

1. Point the git repository
2. Define required YAML for Azure DevOps pipeline

**main.yml**

The main.yml file is the main Azure DevOps pipeline which gets triggered once we create release tags in the GitHub. This ADO pipeline contains two or three target environments on which we would like to deploy the databricks asset bundles based upon our requirement.

trigger:

  tags:

    include:

      - v\*.\*.\*

pr: none

stages:

# DEV

- stage: DEV

  displayName: 'Deploy to DEV'

  variables:

    - group: ADV-ADB-MI-DEPLOY-DEV

  jobs:

    - job: Deploy\_DEV

      displayName: 'Deploy Databricks Assets to DEV'

      pool: $(poolName)

      steps:

      - template: /.ado/deploy-template.yml

        parameters:

          env: dev

#STAGE

- stage: STAGE

  displayName: 'Deploy to STAGE'

  dependsOn: DEV

  condition: succeeded()

  variables:

    - group: ADV-ADB-MI-DEPLOY-STAGE

  jobs:

    - deployment: Deploy\_STAGE

      displayName: 'Deploy Databricks Assets to STAGE'

      environment: ADV-ADB-STAGE

      pool: ADV-TF-STAGE-EASTUS2-Data and Analytics Portfolio

      strategy:

        runOnce:

          deploy:

            steps:

            - template: /.ado/deploy-template.yml

              parameters:

                env: stage

#PRODUCTION

- stage: PRODUCTION

  displayName: 'Deploy to PROD'

  dependsOn: STAGE

  condition: succeeded()

  variables:

    - group: ADV-ADB-MI-DEPLOY-PROD

  jobs:

    - deployment: Deploy\_PROD

      displayName: 'Deploy Databricks Assets to PRODUCTION'

      environment: ADV-ADB-PROD

      pool: ADV-TF-PROD-EASTUS2-Data and Analytics Portfolio

      strategy:

        runOnce:

          deploy:

            steps:

            - template: /.ado/deploy-template.yml

              parameters:

                env: prod

**deploy-template.yml**

The deploy-template.yml file is a ADO template that we are using which contains all the tasks required for deployment.

steps:

- checkout: self

  persistCredentials: true

  fetchDepth: 0

  clean: true

  displayName: "Checkout repository"

- task: Bash@3

  displayName: "Extract Path and Version from Tag"

  inputs:

    targetType: 'inline'

    script: |

      TAG=$(Build.SourceBranchName)

      if [[ "$TAG" =~ v([0-9]+\.[0-9]+\.[0-9]+)-(.\*) ]]; then

        VERSION=${BASH\_REMATCH[1]}

        ASSET\_PATH=${BASH\_REMATCH[2]}

        echo "##vso[task.setvariable variable=VERSION]$VERSION"

        echo "##vso[task.setvariable variable=ASSET\_PATH]$ASSET\_PATH"

        echo "Version: $VERSION"

        echo "Asset Path: $ASSET\_PATH"

      else

        echo "Error: Invalid tag format."

        exit 1

      fi

- script: env | sort

  displayName: 'Environment / Context'

- task: UsePythonVersion@0

  displayName: 'Use Python $(pythonVersion)'

  inputs:

    versionSpec: '$(pythonVersion)'

- task: ShellScript@2

  inputs:

    scriptPath: $(System.DefaultWorkingDirectory)/.ado/install-deps.bash

  displayName: 'Install additional dependencies'

- task: AzureCLI@2

  inputs:

    azureSubscription: adv-devops-sp-adv-dnap-eus2-adb-01-i1-${{ parameters.env }}

    scriptType: 'bash'

    scriptLocation: 'inlineScript'

    inlineScript: |

      echo "Getting access token..."

      DATABRICKS\_TOKEN=$(az account get-access-token --resource 2ff814a6-3304-4ab8-85cb-cd0e6f879c1d --query "accessToken" -o tsv)

      echo "##vso[task.setvariable variable=DATABRICKS\_TOKEN]$DATABRICKS\_TOKEN"

  displayName: 'Fetch Databricks Token'

- script: |

    echo "Checking Databricks CLI installation..."

    if command -v databricks &> /dev/null; then

      echo "Databricks CLI is already installed. Removing existing version..."

      sudo rm -f /usr/local/bin/databricks

    fi

    echo "Installing Databricks CLI..."

    curl -fsSL https://raw.githubusercontent.com/databricks/setup-cli/main/install.sh | sudo sh

    echo "Databricks CLI version: $(databricks --version)"

  displayName: 'Install Databricks CLI'

- script: |

    echo "Configuring Databricks CLI..."

    export DATABRICKS\_CONFIG\_FILE="$(System.DefaultWorkingDirectory)/.databrickscfg"

    echo -e "[DEFAULT]\nhost=$(DATABRICKS\_HOST)\ntoken=$(DATABRICKS\_TOKEN)" > $DATABRICKS\_CONFIG\_FILE

  displayName: 'Configure Databricks CLI'

  env:

    DATABRICKS\_HOST: $(DATABRICKS\_HOST)

    DATABRICKS\_TOKEN: $(DATABRICKS\_TOKEN)

- script: |

    echo "Validating DAB of $(ASSET\_PATH) for $(env) environment..."

    databricks bundle validate -t $(env)

  displayName: 'Validate DAB'

  workingDirectory: $(System.DefaultWorkingDirectory)/bundles/$(ASSET\_PATH)

- script: |

    echo "Deploying DAB of $(ASSET\_PATH) to $(env) environment..."

    databricks bundle deploy -t $(env)

  displayName: 'Deploy DAB'

  workingDirectory: $(System.DefaultWorkingDirectory)/bundles/$(ASSET\_PATH)

High level information of each step:

Code checkout: In this step it pulls the repository code to the ADO agent

Environment context: In this step it will print all the details of the Azure DevOps agent environment.

Setting python version: Setting of required python version.

Install additional dependencies: In this step we are installing additional dependencies from the shell script present in .ado/install-deps.bash

Fetching Databricks token: In this step we are fetching the Databricks token and assigning the value to DATABRICKS\_TOKEN variable using Azure service principal, Databricks resource ID and Azure CLI.

Installing python dependencies: In this step we will be installing all the necessary python dependencies for running the jobs.

CopyFiles: In this task we are copying the content from repository local path of the DevOps agent to the BinaryDirectory folder.

Databricks CLI installation: in this step we are installing the latest databricks CLI to run bundle commands in the CLI.

Configuring Databricks CLI: In this step we are configuring the Databricks CLI by copying the Databricks host and Databricks token into the databrickscfg file.

Validate DAB: In this step we are validating the Databricks asset bundle files with the target environment.

Deploy DAB: In this step we are deploying the Databricks asset bundles to the target environment.

**Agent specific dependencies**

.ado/**install-deps.bash** contains all the necessary dependencies that are required to be installed on the agent to run the ADO pipeline.

curl https://packages.microsoft.com/keys/microsoft.asc | sudo tee /etc/apt/trusted.gpg.d/microsoft.asc

sudo curl https://packages.microsoft.com/config/ubuntu/20.04/prod.list > /tmp/prod.list

sudo cp /tmp/prod.list /etc/apt/sources.list.d/mssql-release.list

echo "installing ubuntu packages"

until sudo apt-get -y update && sudo ACCEPT\_EULA=Y apt-get install jq unzip mssql-tools18 unixodbc-dev -y

do

  echo "Waiting until apt lock is released"

  sleep 2

done

echo "finished installing ubuntu packages"

echo "installing az"

until curl -vsL https://aka.ms/InstallAzureCLIDeb | sudo bash

do

   echo "Waiting until apt lock is released"

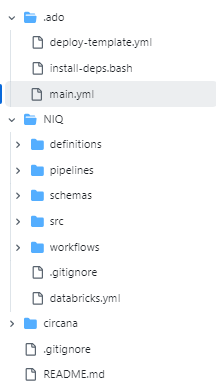
   sleep 2

done

echo "finished installing azure cli"

4. Define DAB YAML and source in it dedicated context folder

Example



DAB file example

bundle:

  name: dnap\_temp\_devopstest\_bundle

workspace:

  root\_path: /Workspace/Users/${workspace.current\_user.userName}/.bundle/${bundle.name}

include:

  - pipelines/\*.yml

  - schemas/\*.yml

  - workflows/\*.yml

permissions:

  - level: CAN\_VIEW

    group\_name: ${var.view\_users\_group}

  - level: CAN\_MANAGE

    service\_principal\_name: ${workspace.current\_user.userName}

variables:

  cluster\_id:

    description: Cluster ID for Medium Shared Compute

    lookup:

      cluster: Medium Shared Compute Cluster

  env:

    description: 'Environment literal like dev, qa, stage, prod'

    default: dev

  DATABRICKS\_HOST:

    description: 'Workspace URL for Deploying the jobs or pipelines'

  catalog\_name:

    description: Test catalog to use

    default: dnap\_dev

  view\_users\_group:

    description: Environment specific name for the users group to add Can View permissions on the workflow

    default: az-adv-dnap-eus2-adb-01-i1-dev-users

  team\_tag:

    description: Advantage team name Tag

    default: dnap

  app\_tag:

    description: Application name Tag

    default: devops\_test

targets:

  dev:

    mode: development

    default: true

  $(env):

    mode: production

    workspace:

      host: $(DATABRICKS\_HOST)

    variables:

      DATABRICKS\_HOST: $(DATABRICKS\_HOST)

      catalog\_name: $(catalog\_name)

      view\_users\_group: $(view\_users\_group)

# ADO pipeline trigger procedure

### Tag Based Trigger

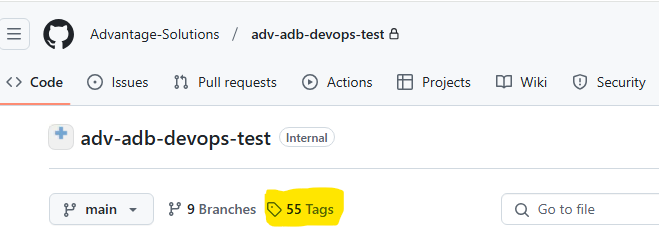
In this Azure DevOps pipeline, we are using GitHub tags to trigger the pipeline. Whenever we create a tag with a version and buildPath name in the below format then it will trigger ADO pipeline, and it will deploy the buildPath mentioned in the tag name.

Format of tag creation - v\*.\*.\*- buildPath

In place of **\*** we can provide any numeric value. For example, **v2.3.3-boost.**

### Follow the steps below to create the tags -

1. On GitHub, navigate to the main page of the repository. Under **Code** tab select **Tags.**



1. To the right of the list of files, click **Releases**. At the top of the page, click **Draft a new release**.

A screen shot of a computer

Description automatically generated

1. To choose a tag for the release, select the **Choose a tag** dropdown menu.

* To use an existing tag, click the tag.
* To create a new tag, type a version number for your release, then click **Create new tag**.

1. If you create a new tag, select the **Target** dropdown menu, then click the branch that contains the project you want to release.
2. Optionally, above the description field, select the **Previous tag** dropdown menu, then click the tag that identifies the previous release.

A screenshot of a computer

Description automatically generated

1. In the "Release title" field, type a title for your release.
2. In the "Describe this release" field, type a description for your release. If you @mention anyone in the description, the published release will include a **Contributors** section with an avatar list of all the mentioned users. Alternatively, you can automatically generate your release notes by clicking **Generate release notes**.

A screenshot of a computer

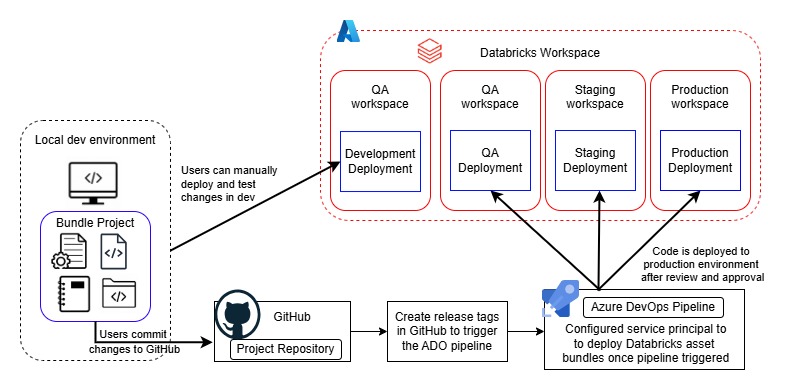
Description automatically generated

1. Optionally, to include binary files such as compiled programs in your release, drag and drop or manually select files in the binaries box.
2. Optionally, to notify users that the release is not ready for production and may be unstable, select **This is a pre-release**.

A screenshot of a computer

Description automatically generated

1. Optionally, select **Set as latest release**. If you do not select this option, the latest release label will automatically be assigned based on semantic versioning.
2. Optionally, if GitHub Discussions is enabled for the repository, create a discussion for the release.
   1. Select **Create a discussion for this release**.
   2. Select the **Category** dropdown menu, then click a category for the release discussion.
3. If you're ready to publicize your release, click **Publish release**. To work on the release later, click **Save draft**. You can then view your published or draft releases in the releases feed for your repository.



# Naming Conventions

1. Variable Group

ADV-ADB-{{use case name}}-DEPLOY-{{environment}}

Here in place of use case we need to provide value – AIAS, MI, NGHQ

And in place of environment, we need to provide value – DEV, QA, STAGE, PROD

Example - ADV-ADB-AIAS-DEPLOY-QA

1. Service Principal name

adv-devops-sp-{{databricks name}}

**adv-devops-sp** would be prefix for all the service principal names

Here the databricks workspace name is **adv-dnap-eus2-adb-01-i1-qa**

For example, in QA we have the following naming: adv-devops-sp-adv-dnap-eus2-adb-01-i1-qa.

DEV - adv-devops-sp-adv-dnap-eus2-adb-01-i1-dev.

QA - adv-devops-sp-adv-dnap-eus2-adb-01-i1-qa.  
STAGE - adv-devops-sp-adv-dnap-eus2-adb-01-i1-stage.

PROD - adv-devops-sp-adv-dnap-eus2-adb-01-i1-prod.

# Databricks Asset Bundles

1. Bundle name is defined in this databricks.yml file.
2. Workspace.root\_path is the path were the Databricks asset bundles files will be stored.
3. Include is defined as, we may have many DLT pipelines and many DLT jobs, so maintaining their respective definition YAML files into respective jobs and dlt\_pipelines folder or in a resource folder would be easy.
4. Permissions section is for managing permissions like CAN\_MANAGE, CAN\_VIEW, CAN\_RUN for a particular group or a service principal.
5. Variables defined in the section are used in the target section defined below. As a result, the values passed to those variables get passed to the jobs and DLT pipeline definition.
6. Target section generally has targets defined for all the environments, in our case we would have 4 different environments (*dev, qa, stage, prod)* according to the use case.
7. Mode used for deployment has two types, production mode and development mode.
   1. Development mode deploys the artifact content into user folder of a user who executes the bundle deployment commands. Also, the DLT pipelines/jobs are deployed with a prefix like '[dev my\_user\_name]'.mentioned under targets: so that they can be differentiated.

Setting this mode also disables any schedules and automatic triggers for jobs and enables the 'development' mode for Delta Live Tables pipelines.

* 1. Production mode deploys the artifact into the workspace based on the path mentioned under root\_path clause. Jobs and DLT pipelines get deployed with their original names. DLT pipelines needs to have production mode only when deploying using production mode.

1. In targets we are passing the values of target environment, databricks host and view user group from the variable group.

# Databricks Asset Bundle files issues

1. Usage of run\_as

The run\_as setting can be configured as a top-level mapping to apply to resources, or within a target deployment mapping in a bundle configuration file. It can be set to a user name or a service principal name.

* 1. User name

**run\_as**:

**user\_name**: "john.doe@youradv.com"

* 1. service principal name

**run\_as**:

**service\_principal\_name**: "5cf3z04b-a73c-4x46-9f3d-52da7999069e"

We used both user name and service principal name but the DAB deployment were getting failed because we got the error pipelines don’t support a bundle file configuration with setting with run\_as user that is different form the owner, in our case the owner is the service principal name that we are using. So, once we removed the run\_as parameter after that deployments were successful as it was picking up the SP automatically as owner.

*Error: pipelines do not support a setting a run\_as user that is different from the owner.*

*Current identity: \*\*\*. Run as identity: john.doe@advantagesolutions.net.*

*See* [*https://docs.databricks.com/dev-tools/bundles/run-as.html*](https://docs.databricks.com/dev-tools/bundles/run-as.html) *to learn more about the run\_as property.*

*in bronze-pipeline-resource.yml:3:5*

1. Permissions

The following permissions in a bundle configuration file define the permission levels for a user, group, and service principal, which are applied to all jobs, pipelines, experiments, and models defined in resources in the bundle. For more information click here [permissions for DAB](https://docs.databricks.com/en/dev-tools/bundles/permissions.html).

* 1. CAN\_VIEW

If we include the CAN\_VIEW permission for a particular group in our bundle configuration, then only that group members can view the workflows, jobs and DLT pipelines. We are passing the user group name as a variable in ADO pipeline who needs CAN\_VIEW permission.

* 1. CAN\_MANAGE

If we include the CAN\_MANAGE permission for a particular group in our bundle configuration, then that particular group members member can manage the workflows, jobs and DLT pipelines. Here we have given this permission to the admin group.

If we don’t use the CAN\_MANAGE permission in our DAB file, then we will get the below warning message -

*Warning: permissions section should include \*\*\* or one of their groups with CAN\_MANAGE permissions*

*in databricks.yml:16:3*

In the current bundle file we are providing user\_group with CAN\_VIEW permission and for service\_principal\_name with CAN\_MANAGE permissions.

When we used only CAN\_VIEW permission for user group we got warnings in the validate DAB step and error in the deploy DAB step. We can see the details in the following link [error1](https://dev.azure.com/adv-dev/Data%20and%20Analytics%20Portfolio/_build/results?buildId=140244&view=logs&j=2ad04f02-2745-599b-b781-103ebff8b319&t=0f1e80b0-336b-5121-ca6d-d65c211937f5).

1. Redeployment

If we are creating any resources/schema in a deployment and after that we are trying to do a redeployment of the resources to the same environment and in that case first we need to remove that particular resource/schema from the databricks workspace and then only we can deploy it. Else we can get a error as below –

*Error: cannot create schema: Schema 'dev\_adv\_devops\_sp\_adv\_dnap\_eus2\_adb\_01\_i1\_qa\_test-devops-schema' already exists*

1. Workspace

In the databricks asset bundle files we have a workspace parameter, where we are mentioning the root\_path to store the bundle files in the databricks workspace. Here we are using ${workspace.current\_user.userName} this pattern help us to update the bundle file under user name in our case the user name is service principal we are using.

workspace:

*root\_path: /Workspace/Users/${workspace.current\_user.userName}/.bundle/${bundle.name}*

1. Mode: development

If mode is production and pipeline is using development:true then it will fail during validation step and will throw below error.

*Error: target with 'mode: production' cannot include a pipeline with 'development: true'*

If we are using mode:development then it should have current user root path or else we would get below error.

*Error: root\_path must start with '~/' or contain the current username to ensure uniqueness when using 'mode: development'*

When we are using development mode in targets, we should always have the path name same as current username.

*Error: prefix should contain the current username or ${workspace.current\_user.short\_name} to ensure uniqueness when using 'mode: development'*

*in databricks.yml:71:20*

1. If a catalog doesn’t have *USE CATALOG* permissions for the service principal/user/group, then it we will be not able to deploy the asset bundle files and it will throw below error while deploying -

*Error: terraform apply: exit status 1*

*Error: cannot create pipeline: User does not have USE CATALOG on Catalog 'dnap\_hr\_ai\_staffing\_qa'.*

*with databricks\_pipeline.pipeline\_aias\_bronze\_to\_gold,*

*on bundle.tf.json line 268, in resource.databricks\_pipeline.pipeline\_aias\_bronze\_to\_gold:*

*268: },*

*Error: cannot create pipeline: User does not have USE CATALOG on Catalog 'dnap\_hr\_ai\_staffing\_qa'.*

*with databricks\_pipeline.pipeline\_aiautostaffing\_bronze,*

*on bundle.tf.json line 288, in resource.databricks\_pipeline.pipeline\_aiautostaffing\_bronze:*

*288: }*

1. If a DLT pipeline is already present in a particular environment and when we try to redeploy it, the deployment will get failed, and it will throw an error in DAB deploy step as –

*Error: cannot create pipeline: The pipeline name 'AIAS - Bronze to Gold' is already used by another pipeline. This check can be skipped by setting `allow\_duplicate\_names = true` in the request.*

If we add *`allow\_duplicate\_names = true`*in the pipeline, then it will create a new pipeline with duplicate name.

1. Databricks Publish issue *Error: cannot create pipeline: Direct Publishing Mode is not supported in your workspace. Please contact Databricks support to enable this feature for your workspace.*

*with databricks\_pipeline.pipeline\_aias\_bronze\_to\_gold,*

*on bundle.tf.json line 257, in resource.databricks\_pipeline.pipeline\_aias\_bronze\_to\_gold: 257: },*

1. Targets: we need to define the targets properly if not we might get below error.

*Validating DAB for stage environment...*

*Error: stage: no such target. Available targets: qa, prod, dev*

*Name: dnap\_aiautostaffing\_bundle*

*Workspace:*

*Path: /Workspace/Users/${workspace.current\_user.userName}/.bundle/${bundle.name}*

*Found 1 error*

# ADO pipeline issues

1. Installation of Databricks CLI

When we are trying to install the Databricks CLI directly in the pipeline using the command *curl -fsSL https://raw.githubusercontent.com/databricks/setup-cli/main/install.sh | sudo sh* it’s installed for the first time for a runner but when we are trying to run the ADO pipeline with same runner then it is failing.

To avoid this, we are using a script to check which version of databricks CLI is installed if it is 0.18 then we are installing the latest one.

1. Configure Databricks CLI

At first, we used the step below to configure the Databricks CLI but the ADO pipeline was timing out after 1 hour.

- script: |

echo Configure Databricks CLI...

databricks configure --token

echo -e "[DATABRICKS\_HOST]\n[DATABRICKS\_TOKEN]" | databricks configure --token

env:

DATABRICKS\_HOST: $(DATABRICKS\_HOST)

DATABRICKS\_TOKEN: $(DATABRICKS\_TOKEN)

To mitigate this issue, we included another step which copies the values of databricks\_host and databricks\_token to the databrickscfg file.

1. Python Version

The python version can be passed as a parameter in the ADO pipeline.

# Issues observed in new ADO pipeline

1. Need to have uniform repo structure for all the repos. Because we need to change the ADO template for each use case.