File Transfer Utility from Linux to Azure



Use Microsoft Azure Xplat-CLI script for copy files to Azure Storage, Data Lake

Prepared for

SEAL

7-Nov-16

Version 1.0

Prepared by

**Manav Gupta**

Reviewers

**Andy Isley**

**Arshad Ali**

Table of Contents

[1 Overview 2](#_Toc466308514)

[2 Azure CLI Installation 3](#_Toc466308515)

[3 Running the script to upload to Azure 5](#_Toc466308516)

[4 Capabilities supported 6](#_Toc466308517)

[5 Sample Output 7](#_Toc466308518)

[6 Script to Load to Azure Storage Account 8](#_Toc466308519)

[7 Script to load to Azure Data Lake Storage Account 10](#_Toc466308520)

[8 Setting up Service Login in Azure Active Directory 12](#_Toc466308521)

[8.1 password authentication 12](#_Toc466308522)

[8.2 certificate authentication 16](#_Toc466308523)

[9 References 22](#_Toc466308524)

1. Overview

**Problem**

Copying data to Microsoft cloud storage services- Azure Blob Storage Account, Azure Data Lake Store is most fundamental step prior to any Azure based processing or ETL workflows. Well-known **AzCopy** command-line utility was designed for copying data to and from Microsoft Azure Blob, File, and Table storage using simple commands. AzCopy provides several options- you can copy data from one object to another within your storage account, or between storage accounts. But, AzCopy has OS supportability limitation- AzCopy works only on Windows, it is not available for Mac/Linux OSs.

In one of recent client engagements, project required migrating data from on-prem linux based source data store. To upload files on Azure, customer had to make uploading to Azure a 2-step process. First copy files manually to intermediary Windows machine and then use AzCopy on Windows machine to upload files to Azure. This was significantly slowing down the overall ETL workflow as well as project execution.

**Solution**

We needed a cross platform utility script to upload files onto Azure from non-Windows systems specifically Linux systems.

Microsoft provides Azure Cross-Platform Command-Line Interface (X-Plat CLI) which is a set of Open-Source, Cross-Platform commands for managing Microsoft Azure platform.

Microsoft Command Line Interface (CLI) provides commands to interact with Azure Storage as well as Azure Data Lake in a cross-platform manner. This blog presents simple CLI based scripts to transfer files from Linux to Azure.

1. Azure CLI Installation

X-Plat CLI command line tool is implemented in JavaScript (powered by Node.js). CLI is powered by Node.js application framework which makes it cross platform.

Running the script would require Azure CLI to be installed on your Linux machine.

Here are quick installation steps:

1. Pre-requisite is to first install npm (Nodejs package manager).

NPM is distributed with Node.js- which means that when you download Node.js, you automatically get npm installed on your computer.

Npmjs.org has all packages <https://www.npmjs.com/get-npm>.

There are several methods to install Azure CLI based on Unix distribution.

1. Make use of following commands to install node.js, npm on Ubuntu machine:

[me@linuxbox me]$ sudo apt-get install nodejs-legacy

[me@linuxbox me]$ sudo apt-get install nodejs

[me@linuxbox me]$ sudo apt-get install npm

1. Check and confirm both node.js and npm got installed on your terminal:

[me@linuxbox me]$ npm –v

[me@linuxbox me]$ node –v

[me@linuxbox me]$ nodejs –v

1. You can install the azure-cli npm package directly.

[me@linuxbox me]$ sudo npm install azure-cli –global

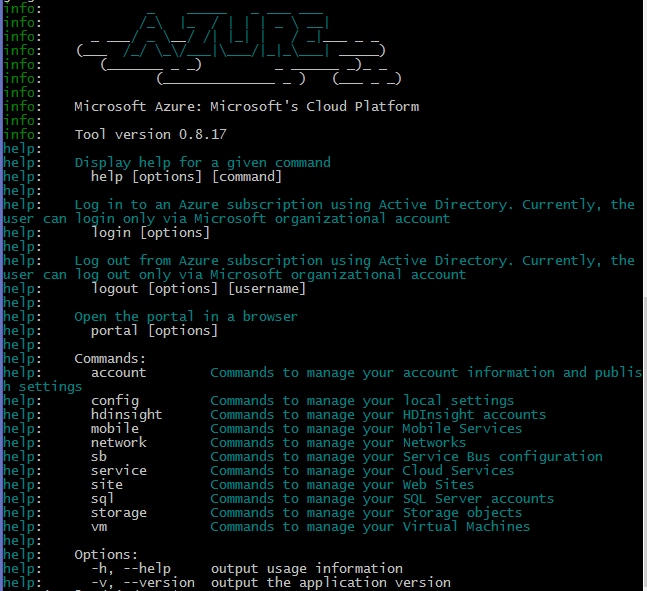
Azure-cli can depend on number of other npm packages which will get additionally pulled from apt-get repo; so installation can take time

1. Confirm Azure CLI got installed on your terminal. Check the Azure CLI version:

[me@linuxbox me]$ azure –-version

0.10.4

1. X-Plat CLI has few top-level commands which correspond to different set of Microsoft Azure features. Typing “azure” will confirm complete installation of Azure CLI and lists each of the sub commands as following:



1. When you log in with the azure login command, your CLI profile and all logs are stored in .azure directory located in your user directory. Use azure <command name> --help to see description, usage, and options available with any command.
2. Running the script to upload to Azure

Running the script involves following steps:

1. Download attached bash scripts and configuration file.
2. Setting Azure properties in AzConfig.properties:
   1. AzConfig.properties is common configuration file for all Azure Configs used by LinuxAzCpy.sh and LinuxADLCpy.sh
   2. LinuxAzCpy.sh script would require user to add configs for storageAccountName, storageAccountKey, connectionString, containerName
   3. LinuxADLCpy.sh would require user to add configs for dataLakeStoreAccountName, azureLogin, azureSubscription
3. Setting Login properties in AzScriptlogin.properties:
   1. AzScriptlogin.properties is configuration file needed only if using service principal account based logging instead of user account logging.
   2. Supply values for azureTenantId, certificateFile, certificateThumbprint, servicePrincipal (Pls refer Section 8.2 how to generate these values)
4. Change Line Endings to Unix style (required if you copy scripts from windows):
   1. Script copied from Windows have Windows style line endings (\r\n) - you need to change them to unix style (\n).
   2. Install dos2Unix EOL converter utility from apt-get repository:

[me@linuxbox me]$ sudo apt-get install dos2unix

* 1. use dos2unix command on your scripts:

[me@linuxbox me]$ dos2unix LinuxAzCpy.sh

[me@linuxbox me]$ dos2unix LinuxADLCpy.sh

[me@linuxbox me]$ dos2unix AzConfig.properties

1. Enable auto-complete in Bash (optional step):

[me@linuxbox me]$ sudo apt-get install bash-completion

[me@linuxbox me]$ echo 'source ~/azure.completion.sh' >> ~/.bash\_profile

[me@linuxbox me]$ echo 'source ~/azure.completion.sh' >> ~/.bashrc # (for non-login shell)

1. Setting Permissions on script files:
   1. The next thing you need to do is give the shell permission to execute your script. This is done with the chmod command as follows:

[me@linuxbox me]$ chmod 755 LinuxAzCpy.sh

[me@linuxbox me]$ chmod 755 LinuxADLCpy.sh

* 1. The "755" will give you read, write, and execute permission. Everybody else will get only read and execute permission. If you want your script to be private (i.e., only you can read and execute), use "700" instead.
  2. You might also need to give permissions to your linux data folder:

[me@linuxbox me]$ sudo chmod 777 /<path to>/<source folder>

1. Command to run the script:

At this point, your script is ready run. Use following command to execute:

[me@linuxbox me]$ ./LinuxAzCpy –srcFolder <source> -destAzFolder <storage destination>

[me@linuxbox me]$ ./LinuxADLCpy –srcFolder <source> -destAzFolder <ADL destination>

1. Capabilities supported

As you noticed above, to run this script, simply type the script file name in the bash console and supply three command line parameters- sourceFolder and destAzFolder.

sourceFolder- Source folder on linux system

destAzFolder- Destination folder on Azure Storage or Data Lake storage

useServiceLogin- if set to ‘y’, script will its service principal account for log-in, instead of user account.

Following capabilities are supported:

1. Bulk upload multiple files to Azure in one invocation.
2. Supports wild-card and pattern matching for command line param- sourceFolder
3. Listing of all files copied in the destination Folder in that storage container, or data lake storage location at the end of copy operation.
4. You could use interactive user account or separate service principal account for logging into Azure for ADLSCpy.sh. script.

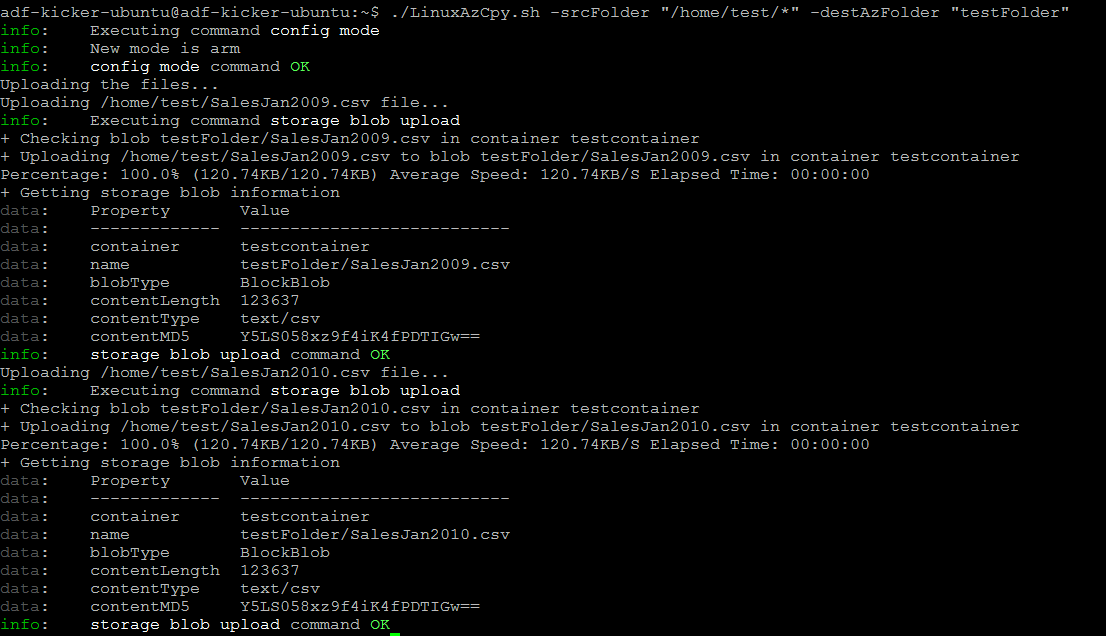
Sample supported copy operations:

[me@linuxbox me]$ ./ADLSCpy.sh -srcFolder "/home/test/\*.csv" -destAzFolder "testFolder"

[me@linuxbox me]$ ./ADLSCpy.sh -srcFolder "/home/test/\*" -destAzFolder "testFolder"

[me@linuxbox me]$ ./ADLSCpy.sh -srcFolder "/home/test/SalesJan2009.csv" -destAzFolder "testFolder"

1. Sample Output



1. Script to Load to Azure Storage Account

#!/bin/bash

. AzConfig.properties

################################################################################

# Filename:LinuxAzCpy.sh

# Description: This script has been created for uploading files to Azure Storage account

# It serves as a utility based on cross platform CLI script to upload files onto Azure from non-Windows systems specifically Mac OS X or Linux systems

#

# Usage: LinuxAzCpy.sh -srcFolder -destAzFolder

# Where

# -srcFolder -source folder in linux machine

# -destAzFolder -destination folder in Azure Storage Account container

#

# Authors: Manav Gupta

# Comments: tested on Ubuntu 16.04.1 VM

###############################################################################

for ARG in $\*

do

case $ARG in

-srcFolder) srcFolder=$2;;

-destAzFolder) destAzFolder=$4;;

esac

done

# Switch to Resource manager CLI command mode: commands would use the Azure Resource Manager API instead of legacy Service Management APIs,

# for working with Azure resources in the Resource Manager deployment model.

azure config mode arm

# Connect to an Azure subscription from the Azure Command-Line Interface (Azure CLI)

# This will output an url and a device code for you to use browser to login. Use azure login to authenticate interactively.

#azure login #-u $user

# To change the default subscription, use the azure account set command, and specify the subscription that you wish to be the default

#azure account set "$subName"

# set a default Azure storage account in environment variables for all the storage commands in the same session.

# this enables you to run the Azure CLI storage commands without specifying the storage account and key explicitly.

export AZURE\_STORAGE\_ACCOUNT=$storageAccountName

export AZURE\_STORAGE\_ACCESS\_KEY=$storageAccountKey

export container\_name=$containerName

export blob\_name=$blobName

# set a default storage account using connection string. get the connection string by command:

#azure storage account connectionstring show <storage\_account\_name>

export AZURE\_STORAGE\_CONNECTION\_STRING=$connectionString

# Azure CLI commands to access Azure Storage

echo "Uploading the files..."

# Upload local files to a block blob by default. To specify the type for the blob, you can use the --blobtype parameter.

#azure storage blob upload $srcFilePath $containerName $blobName

#find \*.gz -exec azure storage blob upload {} $containerName /{} \;

#find $srcFolder -exec azure storage blob upload {} $containerName testFolder/basename{} \;

for f in $srcFolder

do

echo "Uploading $f file..."

azure storage blob upload $f $container\_name $destAzFolder/$(basename $f)

#cat $f

done

#echo "Listing the blobs after uploading..."

#azure storage blob list $container\_name

#azure storage delete $container\_name file.txt

#echo "Done"

1. Script to load to Azure Data Lake Storage Account

#!/bin/bash

. AzConfig.properties

. AzScriptLogin.properties

################################################################################

# Filename:LinuxADLCpy.sh

# Description: This script has been created for uploading files to Azure Data Lake store

# It serves as a utility based on cross platform CLI script to upload files onto Azure from non-Windows systems specifically Mac OS X or Linux systems

#

# Usage: LinuxADLCpy.sh -srcFolder -$destAzFolder

# Where

# -srcFolder -source folder in linux machine

# -destAzFolder -destination folder in Azure Data Lake Store

# -useServiceLogin -use service account login with cert auth. Enter 'y' to activate.

#

# Authors: Manav Gupta

# Comments: tested on Ubuntu 16.04.1 VM

###############################################################################

for ARG in $\*

do

case $ARG in

-srcFolder) srcFolder=$2;;

-destAzFolder) destAzFolder=$4;

-useServiceLogin) useServiceLogin=$6;

esac

done

# Switch to Resource manager CLI command mode: commands would use the Azure Resource Manager API instead of legacy Service Management APIs

azure config mode arm

if [ "$useServiceLogin" = "y" ] ; then

then

echo "script logging with service principal account and cert auth"

azure login --service-principal --tenant "$azureTenantId" --username "$servicePrincipal" --certificate-file "$certificateFile" --thumbprint "$certificateThumbprint"

else

azure login

fi

azure account set "$azureSubscription"

# To change the default subscription, use the azure account set command, and specify the subscription that you wish to be the default

#azure account set "$subName"

echo "Uploading the files..."

#azure datalake store filesystem import "$dataLakeStoreAccountName" "$srcFolder" "$destAzFolder"

# Upload local files to a block blob by default. To specify the type for the blob, you can use the --blobtype parameter.

#azure storage blob upload $srcFilePath $containerName $blobName

#find \*.gz -exec azure storage blob upload {} $containerName /{} \;

#find $srcFolder -exec azure storage blob upload {} $containerName testFolder/basename{} \;

for f in $srcFolder

do

echo "Uploading $f file..."

# azure storage blob upload $f $container\_name $destAzFolder/$(basename $f)

azure datalake store filesystem import "$dataLakeStoreAccountName" "$f" "$destAzFolder/$(basename $f)"

#cat $f

done

#echo "Listing the files after uploading..."

azure datalake store filesystem list "$dataLakeStoreAccountName" "$destAzFolder"

#azure datalake store account delete $dataLakeStoreAccountName

#echo "Done"

1. Setting up Service Login in Azure Active Directory

**Problem**

Your script is ready but you may not want to run the script and underlying process under your own user account credentials.

You want to assign different permissions for the script.

You may require automated login instead of interactive login with each run of Script.

**Solution**

With Azure CLI, you can go a step further, and create a Service Principal account for your script and give it just the set of permissions that it needs.

Setting up your script’s service identity in Azure Active Directory involves following broad steps:

1. First, create a Service Principal which is internally tied to an application.
2. Then use a Service Principal account to give the script the access to proper resources.
3. Then login using service principal and one of following authentication methods:
   1. password credentials (script will prompt to enter password before running)
   2. certificate credentials (script will automatically use stored certificate- suitable for unmanned scheduled usage of script)
   3. password authentication

Steps to create service principal and login using service principal using sample parameters:

**STEP 0 (Install jq- program for parsing json)**

***[me@linuxbox me]$ sudo apt-get install jq***

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following packages were automatically installed and are no longer required:

linux-headers-4.4.0-36 linux-headers-4.4.0-36-generic linux-headers-4.4.0-38 linux-headers-4.4.0-38-generic linux-image-4.4.0-36-generic linux-image-4.4.0-38-generic

linux-image-extra-4.4.0-36-generic linux-image-extra-4.4.0-38-generic

Use 'sudo apt autoremove' to remove them.

The following additional packages will be installed:

libonig2

The following NEW packages will be installed:

jq libonig2

0 upgraded, 2 newly installed, 0 to remove and 57 not upgraded.

Need to get 232 kB of archives.

After this operation, 829 kB of additional disk space will be used.

Do you want to continue? [Y/n] y

Get:1 http://azure.archive.ubuntu.com/ubuntu xenial/universe amd64 libonig2 amd64 5.9.6-1 [88.1 kB]

Get:2 http://azure.archive.ubuntu.com/ubuntu xenial/universe amd64 jq amd64 1.5+dfsg-1 [144 kB]

Fetched 232 kB in 0s (4,348 kB/s)

Selecting previously unselected package libonig2:amd64.

(Reading database ... 168463 files and directories currently installed.)

Preparing to unpack .../libonig2\_5.9.6-1\_amd64.deb ...

Unpacking libonig2:amd64 (5.9.6-1) ...

Selecting previously unselected package jq.

Preparing to unpack .../jq\_1.5+dfsg-1\_amd64.deb ...

Unpacking jq (1.5+dfsg-1) ...

Processing triggers for man-db (2.7.5-1) ...

Setting up libonig2:amd64 (5.9.6-1) ...

Setting up jq (1.5+dfsg-1) ...

Processing triggers for libc-bin (2.23-0ubuntu3) ...

**STEP 1 (Login to Azure)**

To set up service principal, first you should get logged into Azure your user account.

***[me@linuxbox me]$ azure login***

info: Executing command login

|info: To sign in, use a web browser to open the page <https://aka.ms/devicelogin>. Enter the code BB3RJ2N5P to authenticate.

/info: Added subscription subscription-007

info: login command OK

**STEP 2 (Initialize your script variables)**

*[me@linuxbox me]$ $subscriptionId=“26178ce2-a798-4265-ab7d-d9ac09cc7371”*

*[me@linuxbox me]$ $script-name=“myscriptApp”*

**STEP 3 (Create AD application & Service Principal for the application)**

In order to create the Service Principal, an Application must first be created within the Azure Active Directory (AAD) with your script’s own service principal name and password.

*[****me@linuxbox me]$ azure ad sp create --name “myscriptApp” --password “myscriptPw”***

info: Executing command ad sp create

+ Creating application myscriptApp

+ Creating service principal for application 3ed40759-f3b8-4546-ae53-bf73fa9072f5

data: Object Id: a85f576e-7c2a-4209-acd5-0708139da613 (The Object Id is needed when granting permissions)

data: Display Name: myscriptApp

data: Service Principal Names:

data: 3ed40759-f3b8-4546-ae53-bf73fa9072f5

data: http://myscriptApp

info: ad sp create command OK

**STEP 4 (Retrieve AppId from Azure AD for your script, retrieve ObjectId to grant permissions to)**

***[me@linuxbox me]$ appId=$(azure ad app show --search “$script-name" --json | jq -r '.[0].appId')***

***[me@linuxbox me]$ echo $appId***

3ed40759-f3b8-4546-ae53-bf73fa9072f5

***[me@linuxbox me]$ objectId=$(azure ad app show --applicationId $appId --json | jq -r '.[0].objectId')***

***[me@linuxbox me]$ echo objectId***

a85f576e-7c2a-4209-acd5-0708139da613

**STEP 5 (Grant permissions to the Service)**

***[me@linuxbox me]$ azure role assignment create --objectId “$objectId” --roleName "Contributor" --scope "/subscriptions/$subscriptionId/"***

info: Executing command role assignment create

+ Finding role with specified name

/data: RoleAssignmentId : /subscriptions/26178ce2-a798-4265-ab7d-d9ac09cc7371/providers/Microsoft.Authorization/roleAssignments/6773a5a3-f07f-4c92-89ba-cda15cda74bc

data: RoleDefinitionName : Contributor

data: RoleDefinitionId : acdd72a7-3385-48ef-bd42-f606fba81ae7

data: Scope : /subscriptions/26178ce2-a798-4265-ab7d-d9ac09cc7371

data: Display Name : myscriptApp

data: SignInName : undefined

data: ObjectId : a85f576e-7c2a-4209-acd5-0708139da613

data: ObjectType : ServicePrincipal

info: role assignment create command OK

**STEP 6 (Get Tenant ID for your AD)**

Tenant is an instance of Active Directory. To retrieve the tenant id for your currently authenticated subscription, use:

***[me@linuxbox me]$ echo tenantId=$(azure account show --subscriptionNameOrId “$subscriptionId” --json | jq -r '.[0].tenantId')***

*tenantId=72f988bf-86f1-41af-91ab-2d7cd011db74*

**STEP 7 (Log in as Service Account)**

Use below command to perform script service login with password auth. You would be prompted for entering script password.

***[me@linuxbox me]$ azure login --username “$appId” --service-principal --tenant $tenantId***

info: Executing command login

Password: \*\*\*\*\*\*\*\*

\info: Added subscription subscription-007

info: Setting subscription " subscription-007" as default

+

info: login command OK

* 1. certificate authentication

Use following steps to authenticate with Azure Active Directory using an application Service Principal and certificate credential:

**STEP 0 (Installations)**

**Install OpenSSL- to create self-signed certs and use cryptographic libraries**

***[me@linuxbox me]$ sudo apt-get install openssl***

Reading package lists... Done

Building dependency tree

Reading state information... Done

openssl is already the newest version (1.0.2g-1ubuntu4.5).

The following packages were automatically installed and are no longer required:

linux-headers-4.4.0-36 linux-headers-4.4.0-36-generic linux-headers-4.4.0-38 linux-headers-4.4.0-38-generic linux-image-4.4.0-36-generic linux-image-4.4.0-38-generic linux-image-extra-4.4.0-36-generic

linux-image-extra-4.4.0-38-generic

Use 'sudo apt autoremove' to remove them.

0 upgraded, 0 newly installed, 0 to remove and 56 not upgraded.

**Install sed for text transformations and filter text in the pipeline**

***[me@linuxbox me]$ sudo apt-get install sed***

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following packages were automatically installed and are no longer required:

linux-headers-4.4.0-36 linux-headers-4.4.0-36-generic linux-headers-4.4.0-38

linux-headers-4.4.0-38-generic linux-image-4.4.0-36-generic

linux-image-4.4.0-38-generic linux-image-extra-4.4.0-36-generic

linux-image-extra-4.4.0-38-generic

Use 'sudo apt autoremove' to remove them.

The following NEW packages will be installed:

sed

0 upgraded, 1 newly installed, 0 to remove and 57 not upgraded.

1 not fully installed or removed.

Need to get 139 kB of archives.

After this operation, 311 kB of additional disk space will be used.

Get:1 http://azure.archive.ubuntu.com/ubuntu xenial/main amd64 sed amd64 4.2.2-7 [139 kB]

Fetched 139 kB in 0s (985 kB/s)

Selecting previously unselected package sed.

(Reading database ... 168449 files and directories currently installed.)

Preparing to unpack .../archives/sed\_4.2.2-7\_amd64.deb ...

Unpacking sed (4.2.2-7) ...

Processing triggers for install-info (6.1.0.dfsg.1-5) ...

Processing triggers for man-db (2.7.5-1) ...

Setting up sed (4.2.2-7) ...

Setting up initramfs-tools (0.122ubuntu8.1) ...

update-initramfs: deferring update (trigger activated)

Processing triggers for initramfs-tools (0.122ubuntu8.1) ...

update-initramfs: Generating /boot/initrd.img-4.4.0-45-generic

W: mdadm: /etc/mdadm/mdadm.conf defines no arrays.

**Install jq for parsing json**

***[me@linuxbox me]$ sudo apt-get install jq***

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following packages were automatically installed and are no longer required:

linux-headers-4.4.0-36 linux-headers-4.4.0-36-generic linux-headers-4.4.0-38 linux-headers-4.4.0-38-generic linux-image-4.4.0-36-generic linux-image-4.4.0-38-generic

linux-image-extra-4.4.0-36-generic linux-image-extra-4.4.0-38-generic

Use 'sudo apt autoremove' to remove them.

The following additional packages will be installed:

libonig2

The following NEW packages will be installed:

jq libonig2

0 upgraded, 2 newly installed, 0 to remove and 57 not upgraded.

Need to get 232 kB of archives.

After this operation, 829 kB of additional disk space will be used.

Do you want to continue? [Y/n] y

Get:1 http://azure.archive.ubuntu.com/ubuntu xenial/universe amd64 libonig2 amd64 5.9.6-1 [88.1 kB]

Get:2 http://azure.archive.ubuntu.com/ubuntu xenial/universe amd64 jq amd64 1.5+dfsg-1 [144 kB]

Fetched 232 kB in 0s (4,348 kB/s)

Selecting previously unselected package libonig2:amd64.

(Reading database ... 168463 files and directories currently installed.)

Preparing to unpack .../libonig2\_5.9.6-1\_amd64.deb ...

Unpacking libonig2:amd64 (5.9.6-1) ...

Selecting previously unselected package jq.

Preparing to unpack .../jq\_1.5+dfsg-1\_amd64.deb ...

Unpacking jq (1.5+dfsg-1) ...

Processing triggers for man-db (2.7.5-1) ...

Setting up libonig2:amd64 (5.9.6-1) ...

Setting up jq (1.5+dfsg-1) ...

Processing triggers for libc-bin (2.23-0ubuntu3) ...

**STEP 1 (Login to Azure)**

To set up service principal, first you should get logged into Azure your user account.

***[me@linuxbox me]$ azure login***

info: Executing command login

|info: To sign in, use a web browser to open the page <https://aka.ms/devicelogin>. Enter the code BB3RJ2N5P to authenticate.

/info: Added subscription subscription-007

info: login command OK

**STEP 2 (Initialize your script variables)**

*[me@linuxbox me]$ $subscriptionId=“26178ce2-a798-4265-ab7d-d9ac09cc7371”*

*[me@linuxbox me]$ $script-name=“myscriptApp”*

**STEP 3 (Create self-signed Certificate)**

***[me@linuxbox me]$ openssl req -x509 -days 3650 -newkey rsa:2048 -out cert.pem -nodes -subj "/CN=$script-name***

You should see two files created in the current directly: **cert.pem** contains the public key **key.pem** contains the private key.

Combine public and private key to **myscriptappcert.pem.**

***[me@linuxbox me]$ cat privkey.pem cert.pem > myscriptappcert.pem***

**STEP 4 (Create AD application)**

In order to create the Service Principal, an Application must first be created within the Azure Active Directory (AAD) with your information. To do so using the CLI, you will need to come up with three “properties”, an **Application Name**, **Application Home Page**, and **Application URI**. For purposes of creating a Service Principal to manage your subscription, you can use whatever values make sense for you. **cert.pem** file contains the public key value of the self-signed certificate we generated. We will grab cert value from cert.pem skipping the first line (BEGIN CERTIFICATE) and the last line (END CERTIFICATE).

***[me@linuxbox me]$ azure ad app create --name "$script-name" --home-page "http://* *$script-name/" --identifier-uris "***[***http://$script-name/***](http://$script-name/)***" --cert-value ""***

info: Executing command ad app create

+ Creating application myscriptApp

data: AppId: 48842eb4-117e-48d3-939d-9ca4cbb22270

data: ObjectId: 1677ecb1-97c0-441b-b3c8-d9277ff84cbf

data: DisplayName: myscriptApp

data: IdentifierUris: 0=http://myscriptApp/

data: ReplyUrls:

data: AvailableToOtherTenants: False

data: HomePage: [http://myscriptApp/](http://manavcertapp/)

info: ad app create command OK

**STEP 5 (Retrieve AppId from Azure AD for your script)**

***[me@linuxbox me]$ appId=$(azure ad app show --search “$script-name" --json | jq -r '.[0].appId')***

***[me@linuxbox me]$ echo $appId***

*48842eb4-117e-48d3-939d-9ca4cbb22270*

**STEP 6 (Create Service Principal for the application)**

To create the Service Principal, you need to supply **appId** generated from previous step, and use the **azure ad sp create** command. Do so as follows:

***[me@linuxbox me]$ azure ad sp create --applicationId "$appId"***

info: Executing command ad sp create

+ Creating service principal for application 48842eb4-117e-48d3-939d-9ca4cbb22270

data: Object Id: b4ba43d6-db86-43a4-83bd-6485d321814e

data: Display Name: myscriptApp

data: Service Principal Names:

data: 48842eb4-117e-48d3-939d-9ca4cbb22270

data: [http://myscriptApp/](http://manavcertapp/)

info: ad sp create command OK

**STEP 7 (Retrieve ObjectId to grant permissions)**

***[me@linuxbox me]$ objectId=$(azure ad app show --applicationId $appId --json | jq -r '.[0].objectId')***

***[me@linuxbox me]$ echo objectId***

*1677ecb1-97c0-441b-b3c8-d9277ff84cbf*

**STEP 8 (Grant permissions to Service)**

Now you have a service principal account, you need to grant this account access proper privileges. To grant permissions to the newly created Service Principal we need to use its **Object Id** listed above and the supply **Subscription Id** that privileges will be granted to. For the Service Principal to make changes to the subscription, we will use the **Contributor** role. To give the Service Principal permission, use the command **azure role assignment create** as follows:

***[me@linuxbox me]$ azure role assignment create --objectId "$objectId" --roleName Contributor --scope "/subscriptions/$subscriptionId/"***

info: Executing command role assignment create

+ Finding role with specified name

|data: RoleAssignmentId : /subscriptions/26178ce2-a798-4265-ab7d-d9ac09cc7371/providers/Microsoft.Authorization/roleAssignments/f719698a-edf3-414d-9a6a-22ee03fdfbb3

data: RoleDefinitionName : Owner

data: RoleDefinitionId : 8e3af657-a8ff-443c-a75c-2fe8c4bcb635

data: Scope : /subscriptions/26178ce2-a798-4265-ab7d-d9ac09cc7371

data: Display Name : myscriptApp

data: SignInName : undefined

data: ObjectId : b4ba43d6-db86-43a4-83bd-6485d321814e

data: ObjectType : ServicePrincipal

data:

+

info: role assignment create command OK

**STEP 9 (Get Tenant ID for your AD)**

To obtain Azure Active Directory Tenant Id, use the command azure account:

***[me@linuxbox me]$ echo tenantId=$(azure account show --subscriptionNameOrId “$subscriptionId” --json | jq -r '.[0].tenantId')***

***[me@linuxbox me]$ echo $tenantId***

*72f988bf-86f1-41af-91ab-2d7cd011db74*

**STEP 10 (Get certificate thumbprint)**

To be able to uniquely identify the certificate, we get its hexadecimal thumbprint representation and then convert to base64. We can use **sed** to replace the colons and remove “SHA1 Fingerprint=” substring, **xxd** to convert to bytes, and **base64** to encode.

***[me@linuxbox me]$ cert-thumbprint=$(openssl x509 -in "myscriptappcert.pem" -fingerprint -noout | sed 's/SHA1 Fingerprint=//g' | sed 's/://g')***

***[me@linuxbox me]$ echo $ cert-thumbprint***

*81BC2250EA0B3A1875A0A9B3E165BFC725533862*

**STEP 11 (Log in as Service Account)**

Use below command to perform script service login with certificate auth.

***[me@linuxbox me]$ azure login --service-principal --username "$appId" --certificate-file "myscriptappcert.pem" --thumbprint "$cert-thumbprint" --tenant "$tenantId"***

1. References
2. Steps to Install the Azure CLI- <https://azure.microsoft.com/en-us/documentation/articles/xplat-cli-install/>
3. Azure CLI Github project- <https://github.com/azure/azure-xplat-cli>
4. Transfer data with the AzCopy Command-Line Utility- <https://azure.microsoft.com/en-us/documentation/articles/storage-use-azcopy/>
5. Using Azure CLI with Azure Storage- <https://azure.microsoft.com/en-us/documentation/articles/storage-azure-cli/>
6. Login in to Azure from Azure CLI- <https://azure.microsoft.com/en-us/documentation/articles/xplat-cli-connect/>
7. Get started with Azure Data Lake Store using Azure Command Line- <https://azure.microsoft.com/en-us/documentation/articles/data-lake-store-get-started-cli/>