Creación de un Azure Blob Storage utilizando Java y Azure CLI.

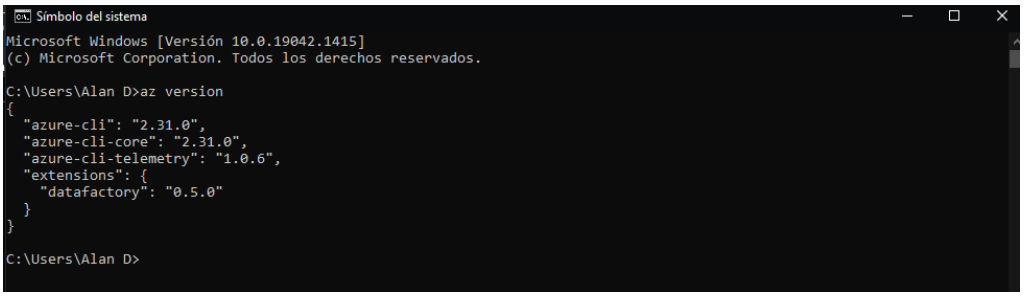
Requisitos Previos:

- IDE Eclipse
- Azure CLI
- Cuenta en Azure con una suscripción activa.
- Java Development Kit versión 8 u 11
- Apache Maven 3.0 o posterior

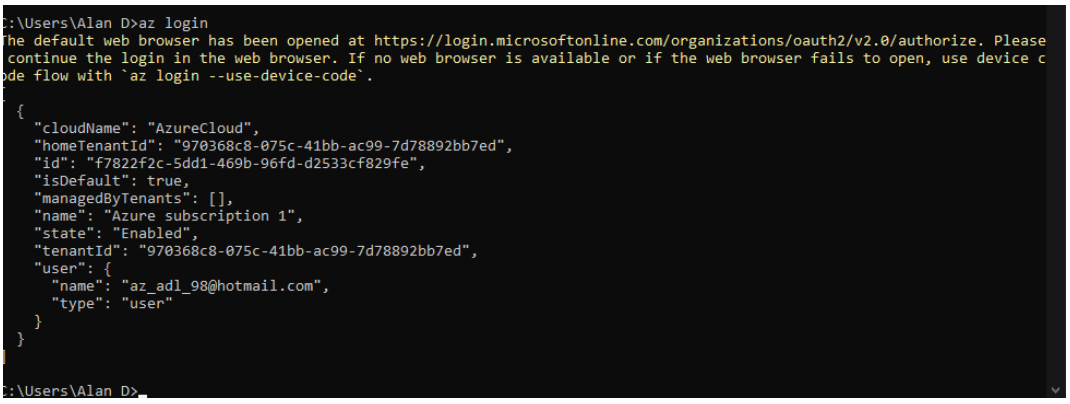
Instrucciones:

Empecemos utilizando la interfaz de la línea de comandos de Azure (CLI de Azure), una guía para su instalación puede ser consultada en [este vínculo.]

Una vez instalado, en una terminal de comandos de nuestro sistema operativo (Símbolos del Sistema de Windows para este ejemplo), verificaremos la correcta instalación de la herramienta Azure CLI escribiendo el comando **AZ VERSION.** Obtendremos un resultado como el siguiente:



Iniciaremos sesión en nuestra cuenta de Azure con el comando **AZ LOGIN.** Una ventana del navegador determinado se abrirá presentándonos instrucciones para iniciar sesión. Una vez terminado el proceso de iniciar sesión, en la terminal de comandos aparecerá lo siguiente:



Ya estamos conectados a Azure, ahora el primer paso para la creación de un Blob Storage es crear un “grupo de recursos”, que será el espacio dentro de nuestra suscripción en donde estarán alojados todos los recursos que utilizaremos para este ejemplo. Para su creación necesitaremos escribir el siguiente comando:

az group create --name MyResourceGroup --location eastus

Se puede sustituir “MyResourceGroup” por cualquier nombre.

La localización por default es “eastus” pero puede ser cambiado según sea la necesidad, ver lista de ubicaciones con el comando **az account list-locations -o table**

```
C:\Users\Alan D>az group create --name MyResourceGroup  --location eastus
{
  "id": "/subscriptions/f7822f2c-5dd1-469b-96fd-d2533cf829fe/resourceGroups/MyResourceGroup",
  "location": "eastus",
  "managedBy": null,
  "name": "MyResourceGroup",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": "Microsoft.Resources/resourceGroups"
}

C:\Users\Alan D>
```

Después, crearemos una “cuenta de almacenamiento de Azure” con el comando:

az storage account create --resource-group MyResourceGroup --name padsastorage --location eastus

El nombre del la cuenta de almacenamiento debe ser único e irrepetible, se aceptan entre 3 y 24 caracteres y solo son válidas letras minúsculas y números.

```
C:\Users\Alan D>az storage account create --resource-group MyResourceGroup --name padsastorage --location eastus
{
  "accessTier": "Hot",
  "allowBlobPublicAccess": true,
  "allowCrossTenantReplication": null,
  "allowSharedKeyAccess": null,
  "azureFilesIdentityBasedAuthentication": null,
  "blobRestoreStatus": null,
  "creationTime": "2022-01-04T16:58:48.343063+00:00",
  "customDomain": null,
  "defaultToOAuthAuthentication": null,
  "enableHttpsTrafficOnly": true,
  "enableNfsV3": null,
  "encryption": {
    "encryptionIdentity": null,
    "keySource": "Microsoft.Storage",
    "keyVaultProperties": null,
    "requireInfrastructureEncryption": null,
    "services": {
      "blob": {
        "enabled": true,
        "keyType": "Account",
        "lastEnabledTime": "2022-01-04T16:58:48.452476+00:00"
      },
```

Para nuestro código Java, requeriremos de una “cadena de conexión” (connection-string) de nuestra cuenta de almacenamiento, la cual nos servirá para conectar nuestro entorno de desarrollo con los servicios de Azure. Esta se obtiene con el siguiente comando:

az storage account show-connection-string --name padsastorage

Una vez obtenida, configuraremos una “variable de entrono” la cual contrendrá esta cadena de conexión. Esto se realiza con el siguiente comando:

setx AZURE_STORAGE_CONNECTION_STRING “Cadena de conexión”

```
C:\Users\Alan D>setx AZURE_STORAGE_CONNECTION_STRING "DefaultEndpointsProtocol=https;EndpointSuffix=core.windows.net;Acc
ountName=padsastorage;AccountKey=moQNoRIuvgLCNBXQ+c1AdaByMC1DX5cTtzrJbfzTKjUiV6k/kzu8UMuCv5R/ppRnHvWv+BYTIX31YZil9/xocA=
="

CORRECTO: se guardó el valor especificado.

C:\Users\Alan D>
```

Ahora, entraremos a la codificación de nuestro proyecto Java.

Creamos un nuevo proyecto de Java utilizando el IDE Eclipse o cualquiera que admita codificación en Java.

Es importante realizar dentro del proyecto Java un archivo titulado **pom.xml** que será en donde radiquen las dependencias necesarias para la correcta utilización de la librería de Azure para Java. En algunas IDE como Spring Tools, este archivo se genera junto con el proyecto y solo es cuestión de colocar las dependencias de Azure. El archivo puede ser descargado desde GitHub y las dependencias de Azure necesarias para este proyecto son:

```
<dependencies>
    <dependency>
        <groupId>com.azure</groupId>
        <artifactId>azure-sdk-bom</artifactId>
        <version>1.0.5</version>
        <type>pom</type>
        <scope>import</scope>
    </dependency>
    <dependency>
        <groupId>com.microsoft.azure</groupId>
        <artifactId>azure-storage</artifactId>
        <version>8.6.6</version>
    </dependency>
    <dependency>
        <groupId>com.microsoft.azure</groupId>
        <artifactId>azure</artifactId>
        <version>1.16.0</version>
    </dependency>
    <dependency>
        <groupId>com.microsoft.azure.functions</groupId>
        <artifactId>azure-functions-java-library</artifactId>
        <version>1.3.1</version>
    </dependency>
    <dependency>
        <groupId>com.azure</groupId>
        <artifactId>azure-storage-blob</artifactId>
        <version>12.13.0</version>
    </dependency>
    <dependency>
        <groupId>com.azure.spring</groupId>
        <artifactId>azure-spring-boot-starter</artifactId>
    </dependency>
    <dependency>
        <groupId>com.azure.spring</groupId>
        <artifactId>azure-spring-boot-starter-storage</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
```

Creamos una clase principal y después de la declaración del main colocamos el siguiente código que será nuestra variable para hacer referencia a la variable de entorno de la conexión.

```
package all;

import java.io.File;

public class all {
    public static void main(String[] args) throws IOException {
        String storageConnectionString = System.getenv("AZURE_STORAGE_CONNECTION_STRING");
```

A continuación, escribiremos dentro de un “try” una serie de códigos que nos permitirán hacer referencia a la cuenta de almacenamiento con la que estaremos trabajando, así como invocar al manejador de blobs con las que cuanta la API de Azure y con ella realizar acciones sobre la cuenta de almacenamiento.

```
package all;

import java.io.File;

public class all {
    public static void main(String[] args) throws IOException {
        String storageConnectionString = System.getenv("AZURE_STORAGE_CONNECTION_STRING");
        try {
            // Referencia a la cadena de conexión
            CloudStorageAccount storageAccount = CloudStorageAccount.parse(storageConnectionString);

            // Crear el cliente del blob
            CloudBlobClient blobClient = storageAccount.createCloudBlobClient();
```

A partir de este punto estaremos trabajando con funciones de la librería de Azure, por lo tanto, es necesario importar las librerías requeridas para que el código pueda funcionar correctamente. En algunos IDE se pueden importar las librerías posicionándose sobre la función con error y presionado ctrl + barra espaciadora. Si no es el caso, presentamos las librerías importadas para este proyecto:

```
package all;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;


import com.microsoft.azure.functions.ExecutionContext;
import com.microsoft.azure.functions.annotation.BindingName;
import com.microsoft.azure.functions.annotation.BlobTrigger;
import com.microsoft.azure.functions.annotation.FunctionName;
import com.microsoft.azure.functions.annotation.StorageAccount;
import com.microsoft.azure.storage.CloudStorageAccount;
import com.microsoft.azure.storage.blob.CloudBlob;
import com.microsoft.azure.storage.blob.CloudBlobClient;
import com.microsoft.azure.storage.blob.CloudBlobContainer;
import com.microsoft.azure.storage.blob.CloudBlockBlob;
import com.microsoft.azure.storage.blob.ListBlobItem;
```

El siguiente código nos será muy útil para la creación de contenedores. Un contenedor es el espacio dentro de la cuenta de almacenamiento en donde estarán alojados los BLOBS (Binary Large Objects, objetos binarios grandes) que estaremos enviando.

```
package all;

import java.io.File;

public class all {
    public static void main(String[] args) throws IOException {
        String storageConnectionString = System.getenv("AZURE_STORAGE_CONNECTION_STRING");
        try {
            // Referencia a la cadena de conexión
            CloudStorageAccount storageAccount = CloudStorageAccount.parse(storageConnectionString);

            // Crear el cliente del blob
            CloudBlobClient blobClient = storageAccount.createCloudBlobClient();

            // Referencia al contenedor y lo crea si no existe
            CloudBlobContainer container = blobClient.getContainerReference("new");
            container.createIfNotExists();
```

Ahora estaremos subiendo nuestro primer blob a nuestro contenedor de Blob Storage. Para ello necesitamos tener dentro de la misma carpeta del proyecto y fuera de cualquier folder (a la misma altura que el archivo pom.xml) nuestro objeto que queramos almacenar en el servicio Blob Storage. Para este ejemplo, usaremos un archivo csv llamado Alumnos.

```
package all;

import java.io.File;

public class all {
    public static void main(String[] args) throws IOException {
        String storageConnectionString = System.getenv("AZURE_STORAGE_CONNECTION_STRING");
        try {
            // Referencia a la cadena de conexión
            CloudStorageAccount storageAccount = CloudStorageAccount.parse(storageConnectionString);

            // Crear el cliente del blob
            CloudBlobClient blobClient = storageAccount.createCloudBlobClient();

            // Referencia al contenedor y lo crea si no existe
            CloudBlobContainer container = blobClient.getContainerReference("new");
            container.createIfNotExists();


            //Subir un blob a Azure Blob Storage

            String csv = "Alumnos.csv";
            CloudBlockBlob blob1 = container.getBlockBlobReference(csv);
            File source = new File(csv);
            blob1.upload(new FileInputStream(source), source.length());
```

Dentro de la carpeta del proyecto crearemos una nueva carpeta llamada “data” que será el destino de los archivos descargados desde Blob Storage.

Escribimos el código que nos permitirá listar los blobs que existan dentro del contenedor, verifique de que en realidad se traten de objetos en vez de directorios locales y procede a descargar los archivos en la carpeta creada dentro del proyecto.

```
//Subir un blob a Azure Blob Storage

String csv = "Alumnos.csv";
CloudBlockBlob blob1 = container.getBlockBlobReference(csv);
File source = new File(csv);
blob1.upload(new FileInputStream(source), source.length());



//Obtienen los Blobs del contenedor y los descarga en una carpeta
   for (ListBlobItem blobItem : container.listBlobs()) {
       // Verifica si el objeto es en realidad un BLOB
       if (blobItem instanceof CloudBlob) {
           //Descarga el archivo y lo guarda con el mismo nombre
            CloudBlob blob = (CloudBlob) blobItem;
            blob.download(new FileOutputStream("./data/" + blob.getName()));
    }
```

Finalmente, agregamos la sentencia catch para manejar los errores que se puedan encontrar.

El código completo será el siguiente:

```
21 public class BlobStorage {
22     public static void main(String[] args) throws IOException {
23         String storageConnectionString = System.getenv("AZURE_STORAGE_CONNECTION_STRING");
24         try {
25             // Referencia a la cadena de conexión
26             CloudStorageAccount storageAccount = CloudStorageAccount.parse(storageConnectionString);
27 
28             // Crear el cliente del blob
29             CloudBlobClient blobClient = storageAccount.createCloudBlobClient();
30 
31             // Referencia al contenedor y lo crea si no existe
32             CloudBlobContainer container = blobClient.getContainerReference("nuevo");
33             container.createIfNotExists();
34 
35 
36             //Subir un blob a Azure Blob Storage
37 
38             String csv = "Alumnos.csv";
39             CloudBlockBlob blob1 = container.getBlockBlobReference(csv);
40             File source = new File(csv);
41             blob1.upload(new FileInputStream(source), source.length());
42 
43 
44 
45             //Obtienen los Blobs del contenedor y los descarga en una carpeta
46                for (ListBlobItem blobItem : container.listBlobs()) {
47                    // Verifica si el objeto es en realidad un BLOB
48                    if (blobItem instanceof CloudBlob) {
49                        //Descarga el archivo y lo guarda con el mismo nombre
50                         CloudBlob blob = (CloudBlob) blobItem;
51                         blob.download(new FileOutputStream("./data/" + blob.getName()));
52                     }
53                    }
54     } catch (Exception e) {
55             e.printStackTrace();
56     }
57 
58 }
59 }
```

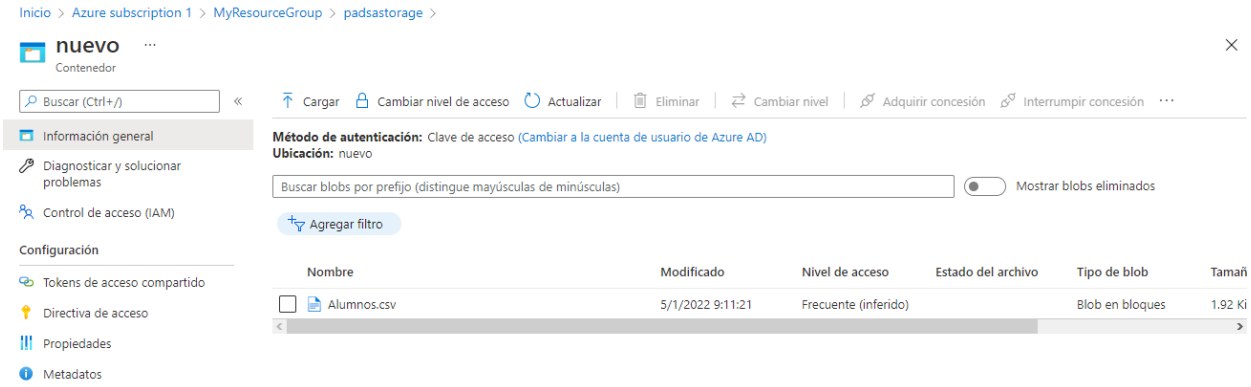
Guardamos todos nuestros cambios y ejecutamos el programa.

Después de la ejecución, tendremos un nuevo contenedor con el archivo Alumnos.csv y en la carpeta “data” estará también el archivo, pero descargado desde el contenedor de Azure.

Para comprobar la creación del contenedor, podemos listar los contenedores creados en nuestra cuenta de almacenamiento con el comando

az container list

O podemos acceder a nuestro portal de Azure en donde tendremos de forma visual todos los procesos que realizamos desde Azure CLI como la creación del grupo de recursos, la cuenta de almacenamiento y el contenedor con el archivo cargado.



Repositorio GitHub:

El proyecto generado en este ejemplo puede ser descargado en el siguiente repositorio de GitHub:

https://github.com/PADSA-github/Cloud/tree/main/Azure/Blob-Storage-Java