Below are the some of the ways to do authentication of azure blob storage

* **Connect to services with connection strings**

Azure Storage uses a storage key to authorize the application

Example:

final String storageConnection = "DefaultEndpointsProtocol=https;"

+ "AccountName=" + storageName

+ ";AccountKey=" + storageKey

+ ";EndpointSuffix=core.windows.net";

* **Azure management libraries for Java**

We can use Azure management libraries for Java to query resources to build connection strings in code.

For example, this code uses the management libraries to create a storage account connection string

// create a new storage account

StorageAccount storage = azure.storageAccounts().getByResourceGroup("myResourceGroup","myStorageAccount");

// create a storage container to hold the file

List<StorageAccountKey> keys = storage.getKeys();

final String storageConnection = "DefaultEndpointsProtocol=https;"

+ "AccountName=" + storage.name()

+ ";AccountKey=" + keys.get(0).value()

+ ";EndpointSuffix=core.windows.net";

* **Authenticate with the Azure management libraries for Java**

Two options are available to authenticate your application with Azure when using the Java management libraries to create and manage resources.

1. Authenticate with an ApplicationTokenCredentials object
2. File based authentication
3. **Authenticate with an ApplicationTokenCredentials object**

Create an instance of ApplicationTokenCredentials to supply the service principal credentials to the top-level Azure object from inside your code.

Example:

import com.microsoft.azure.credentials.ApplicationTokenCredentials;

import com.microsoft.azure.AzureEnvironment;

// ...

ApplicationTokenCredentials credentials = new ApplicationTokenCredentials(client,

tenant,

key,

AzureEnvironment.AZURE);

Azure azure = Azure

.configure()

.withLogLevel(LogLevel.NONE)

.authenticate(credentials)

.withDefaultSubscription();

The client, tenant and key are the same service principal values used with file-based authentication. The AzureEnvironment.AZURE value creates credentials against the Azure public cloud. Change this to a different value if you need to access another cloud (for example, AzureEnvironment.AZURE\_GERMANY).

Read the service principal values from environment variables or a secret management store like Key Vault. Avoid setting these values as cleartext strings in your code to prevent accidentally exposing credentials in your version control history.

1. **File based authentication**

The simplest way to authenticate is to create a properties file that contains credentials for an Azure service principal using the following format

Example:

# sample management library properties file

subscription=########-####-####-####-############

client=########-####-####-####-############

key=XXXXXXXXXXXXXXXX

tenant=########-####-####-####-############

managementURI=https\://management.core.windows.net/

baseURL=https\://management.azure.com/

authURL=https\://login.windows.net/

graphURL=https\://graph.windows.net/

**Subscription**: use the id value from az account show in the Azure CLI 2.0 or from azure portal.

**Client**: use the appId value from the output taken from a service principal created to run the application. If you don't have a service principal for your app, create one with the Azure CLI 2.0.

**Key**: use the password value from the service principal create CLI output

**Tenant**: use the tenant value from the service principal create CLI output

Save this file in a secure location on your system where your code can read it. Set an environment variable with the full path to the file in your shell

Set Path of above created file in environment

export AZURE\_AUTH\_LOCATION=/Users/raisa/azureauth.properties

Create the entry point Azure object to start working with the libraries. Read the location of the properties file through the environment variable.

// pull in the location of the authentication properties file from the environment

final File credFile = new File(System.getenv("AZURE\_AUTH\_LOCATION"));

Azure azure = Azure

.configure()

.withLogLevel(LogLevel.NONE)

.authenticate(credFile)

.withDefaultSubscription();

* **Create an Azure Credential File**

Steps to create credential file in Spring boot app

1. Open a command prompt.
2. Navigate to the resources directory of your Spring Boot app; for example

cd C:\SpringBootApp\storage\src\main\resources

1. Sign in to your Azure account (For the to work you need to install [Azure CLI](https://docs.microsoft.com/en-us/cli/azure/index))

az login

1. List all subscriptions

az account list

Azure will return a list of your subscriptions, and you will need to copy the GUID for the subscription that you want to use; for example:

[

{

"cloudName": "AzureCloud",

"id": "11111111-1111-1111-1111-111111111111", (Copy this)

"isDefault": true,

"name": "Converted Windows Azure MSDN - Visual Studio Ultimate",

"state": "Enabled",

"tenantId": "22222222-2222-2222-2222-222222222222",

"user": {

"name": "gena.soto@wingtiptoys.com",

"type": "user"

}

}

]

1. Specify the GUID for the subscription you want to use with Azure; for example:

az account set -s 11111111-1111-1111-1111-111111111111

1. Create your Azure Credential file

az ad sp create-for-rbac --sdk-auth > fileName.azureauth

This command will create a fileName.azureauth file in your resources directory with contents that resemble the following example:

{

"clientId": "33333333-3333-3333-3333-333333333333",

"clientSecret": "44444444-4444-4444-4444-444444444444",

"subscriptionId": "11111111-1111-1111-1111-111111111111",

"tenantId": "22222222-2222-2222-2222-222222222222",

"activeDirectoryEndpointUrl": "https://login.microsoftonline.com",

"resourceManagerEndpointUrl": "https://management.azure.com/",

"activeDirectoryGraphResourceId": "https://graph.windows.net/",

"sqlManagementEndpointUrl": "https://management.core.windows.net:8443/",

"galleryEndpointUrl": "https://gallery.azure.com/",

"managementEndpointUrl": "https://management.core.windows.net/"

}

1. Configure your Spring Boot app to use your Azure Storage account
2. Locate the application.properties in the resources directory of your app; for example

C:\SpringBoot\storage\src\main\resources\application.properties

1. Open the application.properties file in a text editor, add the following lines, and then replace the sample values with the appropriate properties for your storage account

spring.cloud.azure.credential-file-path=fileName.azureauth

spring.cloud.azure.resource-group=wingtiptoysresources

spring.cloud.azure.region=westUS

spring.cloud.azure.storage.account=wingtiptoysstorage

blob=azure-blob://containerName/blobName

1. Save and close the application.properties file
2. In controller class use defined property as below

@Value("${blob}")

private Resource blobFile;

@GetMapping

public String readBlobFile() throws IOException {

return StreamUtils.copyToString(

this.blobFile.getInputStream(),

Charset.defaultCharset());

}

@PostMapping

public String writeBlobFile(@RequestBody String data) throws IOException {

try (OutputStream os = ((WritableResource) this.blobFile).getOutputStream()) {

os.write(data.getBytes());

}

return "file was updated";

}