Guía: Automatizando Azure Blob Storage con Bicep y probándolo con una aplicación .NET CLI

Introducción

Usar Bicep en Azure es como tener un martillo en tu caja de herramientas.
Un martillo no construye la casa por sí solo, pero sin él sería difícil armarla rápido y con precisión. Con Bicep pasa lo mismo: no ejecuta tu aplicación, pero te permite "armar" toda la infraestructura que tu proyecto necesita, de forma clara, ordenada y repetible.

Cuando trabajás con Bicep, dejás de improvisar y empezás a trabajar con planos. Así, cada vez que necesites levantar un nuevo entorno en la nube, lo hacés en minutos y sin errores.

En esta guía vamos a recorrer, paso a paso, cómo:

- Crear un **Storage Account** en Azure usando un archivo Bicep.
- Obtener su **Connection String** para que tu aplicación pueda conectarse.
- Probar todo con una **aplicación de consola en .NET 8** que sube archivos a Blob Storage usando un **SAS Token**.

La idea es que al final no solo tengas un recurso en Azure funcionando, sino que también cuentes con una base para automatizar futuros despliegues y probarlos de inmediato con código real.

1. Crear el Storage Account con Bicep

Definí tu infraestructura en un archivo `storageAccount.bicep`:

```
param accountName string
param location string = resourceGroup().location
```

```
param skuName string = 'Standard_LRS'
param kind string = 'StorageV2'

resource storageAccount 'Microsoft.Storage/storageAccounts@2023-01-01'
= {
  name: toLower(accountName)
  location: location
  sku: { name: skuName }
  kind: kind
}
```

Luego desplegalo con Azure CLI en **una sola línea**:

```
az deployment group create -g demoBicep --template-file
storageAccount.bicep --parameters accountName=stg2025demo
```

2. Obtener el Connection String

El **Connection String** es la clave que tu aplicación usará para conectarse al Storage Account.
Podés obtenerlo así:

```
az storage account show-connection-string --name stg2025demo --
resource-group demoBicep --query
"{AzureStorage:{ConnectionString:connectionString}}" -o json
```

3. Preparar la aplicación .NET para la prueba

Cloná el repositorio del proyecto desde GitHub:

```
git clone https://github.com/joseantcloud/dotnet-blob-storage-cli.git
cd dotnet-blob-storage-cli
```

Instalá las dependencias necesarias:

```
dotnet add package Microsoft.Extensions.Configuration
dotnet add package Microsoft.Extensions.Configuration.Json
dotnet add package Azure.Storage.Blobs
```

4. Configurar el acceso al contenedor

1. En el portal de Azure, seleccioná tu **Storage Account**.
2. Cambiá el **Access level** del contenedor a **Container**.
3. Creá un **SAS Token** con permisos **READ, ADD, CREATE y WRITE**.
4. Copiá la **SAS URL del contenedor** para usarla en la app.

5. Ejecutar la aplicación y subir archivos

En la terminal, dentro del proyecto:

```
dotnet clean
dotnet build
dotnet run
```

La aplicación te pedirá:

1. La **SAS URL** del contenedor.
2. La **ruta de la carpeta local** con los archivos.
3. Si querés subir **todos** o seleccionar por número.

Una vez completado, verás en Azure que tus archivos fueron subidos con éxito.

Conclusión

Con **Bicep** para automatizar y una pequeña app en **.NET 8** para probar, podés crear un flujo rápido y repetible para trabajar con Azure Blob Storage.
Esto no solo sirve como ejercicio de aprendizaje, sino que también puede integrarse en **pipelines CI/CD** para despliegues más grandes y complejos.