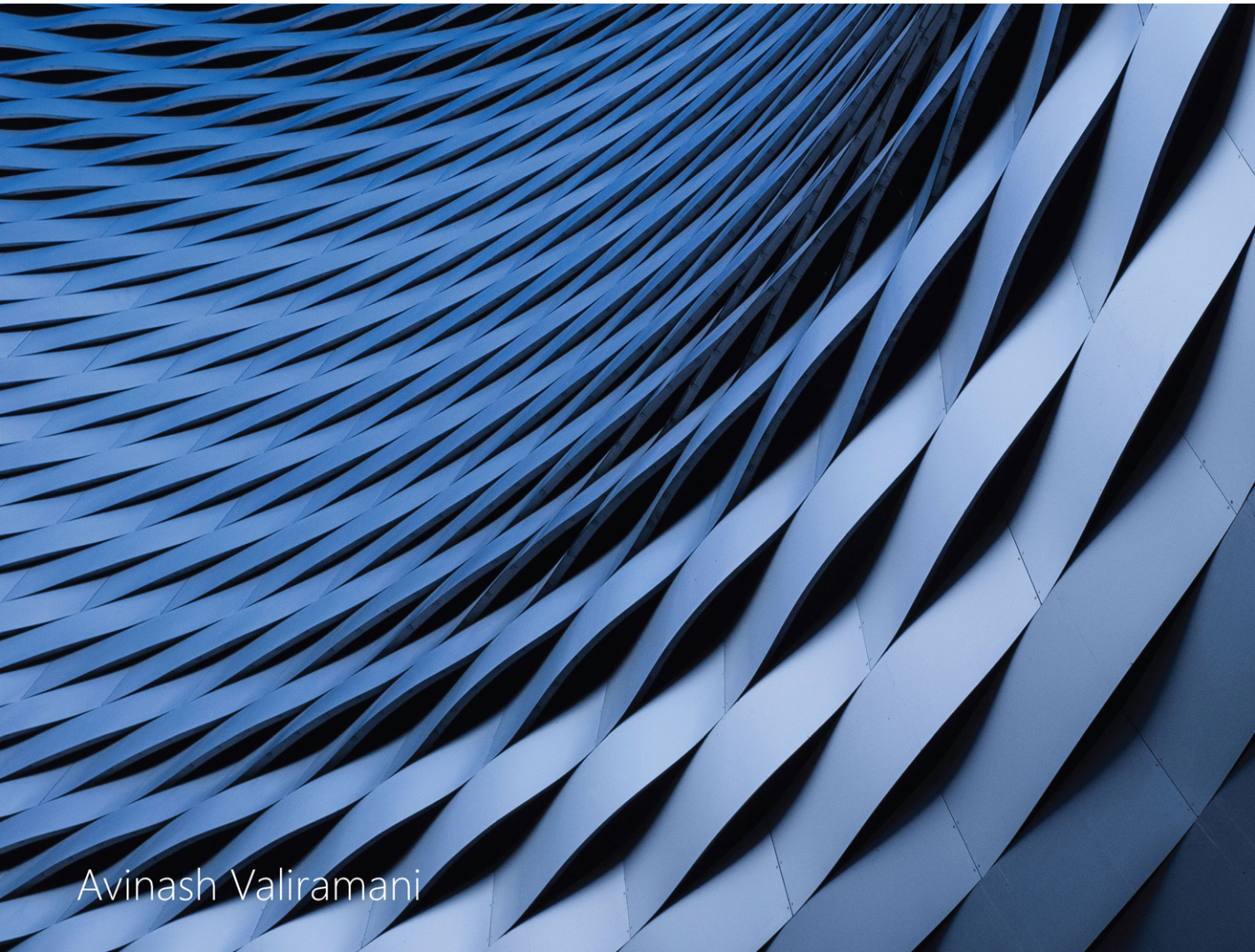
Microsoft

Microsoft Azure Storage

The Definitive Guide



Avinash Valiramani

Microsoft

Microsoft Azure Storage: The Definitive Guide

Avinash Valiramani

```
$bkpvaultstorage = New-AzDataProtectionBackupVaultStorageSettingObject -Type GeoRedundant
-DataStoreType VaultStore
New-AzDataProtectionBackupVault -ResourceGroupName $resourcegroup -vaultName $vaultname
-Location $region -storagesetting $bkpvaultstorage

#Create Backup policy
$policyDefinition = Get-AzDataProtectionPolicyTemplate -DatasourceType AzureBlob
$BkpPolicy=New-AzDataProtectionBackupPolicy `
                        -ResourceGroupName $resourcegroup `
                        -VaultName $vaultname `
                        -Name "DefaultBlobBackupPolicy" `
                        -Policy $policyDefinition

#Enable Backup
$bkp = Initialize-AzDataProtectionBackupInstance -DatasourceType AzureBlob -DatasourceLo-
cation $region -PolicyId $BkpPolicy[0].Id -DatasourceId $storageaccount.Id
New-AzDataProtectionBackupInstance -ResourceGroupName $resourcegroup -VaultName $Vault-
Name -BackupInstance $bkp
```

USING AZURE CLI

Use the following code to set up a storage backup from the Azure CLI:

```
#Define required variables
resourceGroup="RG01"
region="eastus"
storageaccname="mbspblobstorage01"
container="container"
directory="directory"
vnet="vNET01"
subnet="default"
endpointname="PrivateEndpoint"
vaultname="RecoveryServicesVault01"

#Setting up Azure Backup Integration
#Create Backup vault
az backup vault create –resource-group $resourceGroup \
    --name $vaultname \
    --location $region

#Set the storage redundancy and cross region restore config
az backup vault backup-properties set –name $vaultname  \
    --resource-group $resourceGroup \
    --backup-storage-redundancy "GeoRedundant" \
    --cross-region-restore-flag "True"
#Create Backup Policy
az dataprotection backup-policy get-default-policy-template –datasource-type AzureBlob
```

```
{
  “datasourceTypes”: [
    “Microsoft.Storage/storageAccounts/blobServices”
  ],
  “name”: “DefaultBlobPolicy”,
  “objectType”: “BackupPolicy”,
  “policyRules”: [
    {
      “isDefault”: true,
      “lifecycles”: [
        {
          “deleteAfter”: {
            “duration”: “P30D”,
            “objectType”: “AbsoluteDeleteOption”
          },
          “sourceDataStore”: {
            “dataStoreType”: “OperationalStore”,
            “objectType”: “DataStoreInfoBase”
          }
        }
      ],
      “name”: “Default”,
      “objectType”: “AzureRetentionRule”
    }
  ]
}

#Create Backup json – Replace all the resource groups and resource values in the command
before proceeding
az dataprotection backup-instance initialize –datasource-type AzureBlob  -location
$region –policy-id “/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/resourceGroups/
RG-01/providers/Microsoft.DataProtection/backupVaults/BackupVault01/backupPolicies/Default-
BlobBackupPolicy” –datasource-id “/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx/resourcegroups/
RG-01/providers/Microsoft.Storage/storageAccounts/mbspblobstorage01” > backup_instance.
json
#Enable Backup
az dataprotection backup-instance create –resource-group $resourceGroup –vault-name
$vaultname –backup-instance backup_instance.json
```

Blob snapshots

A blob snapshot is a point-in-time read-only copy of a blob that you create and store in the hot or cool tier. The snapshot is identical to the base blob, except it is a point-in-time copy, allowing you to access that version of the blob. A snapshot copies all the system properties and

metadata from the base blob (unless explicitly configured otherwise). You can have any number of snapshots of a blob, and each snapshot will persist until you either delete the snapshot or delete the base blob itself.

The URL for a snapshot is the same as the base blob, except the snapshot date and time value are appended to the end of the URL. For example, if the URL for a base blob is http://blockblobstorageaccount.blob.windows.net/blobcontainer/blob01, a snapshot for that blob taken on January 3, 2022, at 11 p.m. would have a URL similar to http://blockblobstorageaccount.blob.windows.net/blobcontainer/blob01?snapshot=2022-03-01T23:00:00.938291Z.

You can store snapshots on different storage tiers from base blobs. This can help reduce the cost of the snapshots. If you do not specify the storage tier to use, the snapshot is stored in the same tier as the base blob, and you will be charged for it at the same rate as for the base blob after the base blob changes and no longer matches the snapshot. If you store the snapshot in a different access tier, then you are charged for the entire blob based on the access tier rates. You should take this into account when planning your storage protection strategy.

NOTE **Snapshots of blobs in the archive tier are not supported at this time.**

Blob snapshots walkthrough

The following sections step you through the process of creating and managing blob snapshots using the Azure portal, Azure PowerShell, and Azure CLI.

USING AZURE PORTAL

To create and manage blob snapshots using the Azure portal, follow these steps:

1. Locate the page for the container for the blob for which you want to create a snapshot, select the check box for that blob and click **Create Snapshot**. (See Figure 1-64.)

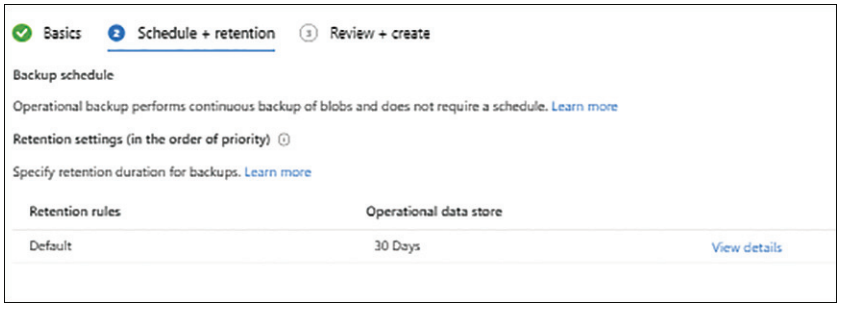


FIGURE 1-64 Create a blob snapshot.

2. To view the snapshot (and any other snapshots associated with the same blob), select the file and click **View Snapshots** (refer to Figure 1-64).

   The snapshot is listed in the Snapshots tab. (See Figure 1-65.)

FIGURE 1-65 View snapshots.

3. Optionally, to change the snapshot's access tier, select the snapshot in the **Snapshots** tab and click **Change Tier**. Then open the **Access Tier** drop-down list in the Change Tier dialog box to choose a different tier and click **Save**. (See Figure 1-66.)

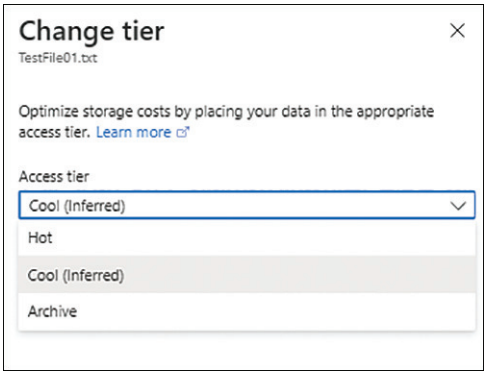


FIGURE 1-66 Changing the snapshot's access tier.

USING AZURE POWERSHELL

Use the following Azure PowerShell code to take a blob snapshot:

```
#Define required variables
$resourceGroup = "RG01"
$region = "eastus"
$storageaccname = "mbspblobstorage01"
```

```
$container = "container"
$vaultname = "RecoveryServicesVault01"
$blob = "TextFile01.txt"

#Generate Blob snapshot
$blob = Get-AzStorageBlob -Container $container -Blob $blob -Context $storageaccount.
context
$blob.BlobClient.CreateSnapshot()
```

USING AZURE CLI

Use the following code to take a blob snapshot from the Azure CLI:

```
#Define required variables
resourceGroup="RG01"
region="eastus"
storageaccname="mbspblobstorage01"
container="container"
directory="directory"
#Generate Blob snapshot
az storage blob snapshot \
    --container-name $container \
    --name TextFile01.txt \
    --account-name $storageAccount
```

Disaster recovery

Disaster recovery is a critical component of any application architecture. The higher the criticality of the application, the more redundancy is required to ensure minimal to no downtime or data loss.

While setting up the data redundancy for the storage account and taking regular backups does ensure that you are able to recover from an outage, you must take into account other application components in your disaster recovery planning, too. This includes components such as the web application firewall or application gateway, web applications or application servers, connected API services, and so on. This ensures that in a disaster scenario, once the blob storage is online, all other related components can also be recovered and can read the storage with minimal interruption and data loss.

We've covered earlier how storage redundancy options such as GRS, GZRS, RA-GRS, and RA-GZRS can replicate your data asynchronously to a secondary Azure region.

One caveat to note: If the primary region becomes unavailable, you can set up your applications to automatically switch to the secondary region to perform read operations in case you are using either RA-GRS or RA-GZRS storage accounts. This will ensure that the application is online in some form while a full storage failover is performed, either by you or by Microsoft.

Once the storage has failed over completely, write operations to the storage account in the secondary region are also allowed, and your application can then start to commit changes to the storage as earlier. In case of GRS or GZRS storage, the read and write operations can only be performed once the storage has been failed over to the secondary region.

You saw earlier that the primary blob storage endpoint points to *https://<storage-account-name>.blob.core.windows.net/<container-name>*. Similarly, the secondary storage endpoint would be reachable at *https://<storage-account-name>-secondary.blob.core.windows.net/<container-name>*. (The *-secondary* suffix is appended automatically by the secondary endpoint.) You can use this endpoint to connect to the secondary storage. The storage account access keys would remain the same in both the primary and secondary endpoints.

Storage account failover

There are two ways you can fail over a storage account to the secondary region:

- **Microsoft-managed failover** In the event of a region-wide outage, Microsoft performs a full region failover to the secondary region. In such cases, you need not perform manual failover operations on your storage accounts. You would only have to ensure that when the DNS entries for the storage are updated, your applications are ready to resume normal operations.
- **Customer-managed failover** In addition, you are able to perform a manual failover on your own, in case of an outage. When an outage occurs, Microsoft will initially actively work toward restoring the data and operations in the primary region, if possible. If they are unable to do so, they will declare that region as unrecoverable and initiate the failover to the secondary region. In case you are unable to wait until such time, you can perform a manual failover from your storage account properties to bring your storage account online and make it accessible to your applications.
- In either scenario, DNS entries must be updated automatically or manually before write operations to the storage can begin. Also take into account any private endpoints you may have created in the primary region; you need to make sure the same endpoints are set up in the secondary region, too.

Last Sync Time

Data synchronized using GRS is often behind data in the primary region. The data sync is asynchronous to avoid affecting write operations and storage performance in the primary region. This allows write operations to be committed on the primary storage without waiting for the same operations to be written and acknowledged by the secondary storage. However, at the time of a disaster, some data written and committed to the primary storage might not yet have been committed to the secondary storage—in which case, that data would be lost.

You can determine whether this has happened by checking the Last Sync Time property for your storage account. This value is a GMT date/time value that you can query using Azure PowerShell, Azure CLI, or one of the Azure Storage client libraries. Any write operations performed after this Last Sync Time property value are most likely missing in the secondary region