Microsoft

AZ-204T00A

Learning Path 03: Develop solutions that use Blob storage

Agenda

- Explore Azure Blob storage
- Manage the Azure Blob storage lifecycle
- Work with Azure Blob storage

Module 1: Explore Azure Blob storage

Learning objectives

- Identify the different types of storage accounts and the resource hierarchy for blob storage.
- Explain how data is securely stored and protected through redundancy.
- Enable a storage account for static website hosting.

Introduction

- Azure Blob storage is Microsoft's **object** storage solution for the cloud.
- Blob storage is optimized for storing massive amounts of unstructured data.
- Blob storage is designed for:
  - Serving images or documents directly to a browser.
  - Storing files for distributed access.
  - Streaming video and audio.
  - Writing to log files.
  - Storing data for backup and restore, disaster recovery, and archiving.
  - Storing data for analysis by an on-premises or Azure-hosted service.

Explore Azure Blob storage (1 of 2)

Disks

Persistent disks for Azure IaaS VMs

Premium storage disk options

Storage Accounts

Files

Fully managed file shares in the cloud

SMB and REST access

"Lift and shift" legacy apps

Sync with on-premises

Blobs

Highly scalable, REST-based cloud object store

Block blobs: Sequential file I/O

Page blobs: Random-write pattern data

Append blobs

Tables

Massive auto-scaling NoSQL store

Dynamic scaling based on load

Queues

Reliable queues at scale for cloud services

Decouple and scale components

Message visibility

Built on a unified Distributed Storage System

Durability, Encryption at Rest, Strongly Consistent Replication, Fault Tolerance, Auto Load-Balancing

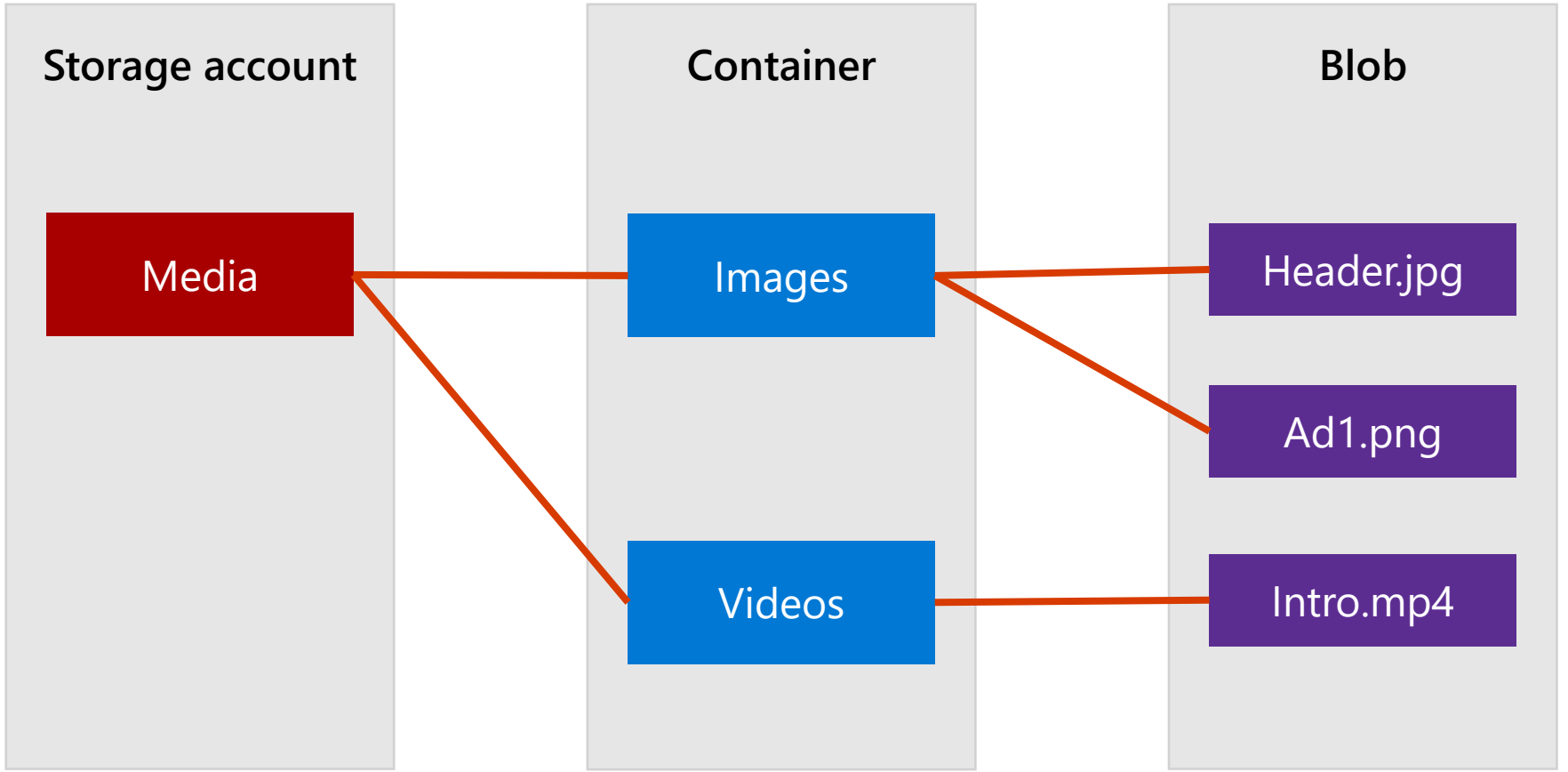
Explore Azure Blob storage (2 of 2)

Performance level	Storage account type	Supported storage services
Standard	Standard general-purpose v2	Blob, Queue, and Table storage, Azure Files
Premium	Premium block blobs	Blob storage
Premium	Premium page blobs	Page blobs only

Access tiers for block blob data

- **Hot:** for frequent access of objects in the storage account
- **Cool:** for storing large amounts of data that is infrequently accessed and stored for at least 30 days.
- **Cold:** for storing data that is infrequently accessed and stored for a minimum of 90 days.
- **Archive**: available only for individual block blobs and is optimized for data that can tolerate several hours of retrieval latency and will remain in the Archive tier for at least 180 days.

Discover Azure Blob storage resource types



Explore Azure Storage security features

Azure Storage provides a comprehensive set of security capabilities:

- Microsoft Entra ID and Role-Based Access Control (RBAC) are supported for Azure Storage
- Data can be secured in transit between an application and Azure
- Delegated access to the data objects in Azure Storage can be granted using a shared access signature

Azure Storage encryption for data at rest

- Azure Storage encryption is enabled for all new and existing storage accounts, cannot be disabled, and does not affect Azure Storage performance.
- Can use either a *Microsoft-managed* key for encryption or your own keys.
- Two options for using your own keys:
  - A *customer-managed key* is used for encrypting all data in the storage account.
  - A *customer-provided key* is used during read/write operations and allows granular control over how blob data is encrypted/decrypted.

Discover static website hosting in Azure Storage

Features

- Serve static content directly from a storage container name *$web*.
- Enables you to use serverless architectures that include Azure Functions.
- Can be made available via a custom domain.
- Enabled in the **Static website** section of the storage account nav bar.

Limitations

- No ability to configure headers as part of the static website feature.
- AuthN and AuthZ are not supported.

Summary and knowledge check

In this module, you learned how to:

- Identify the different types of storage accounts and the resource hierarchy for blob storage.
- Explain how data is securely stored.
- Enable a storage account for static website hosting.

1. What type of blobs are used to store virtual hard drive files?

2. What type of storage accounts is recommended for most scenarios using Azure Storage?

Module 2: Manage the Azure Blob storage lifecycle

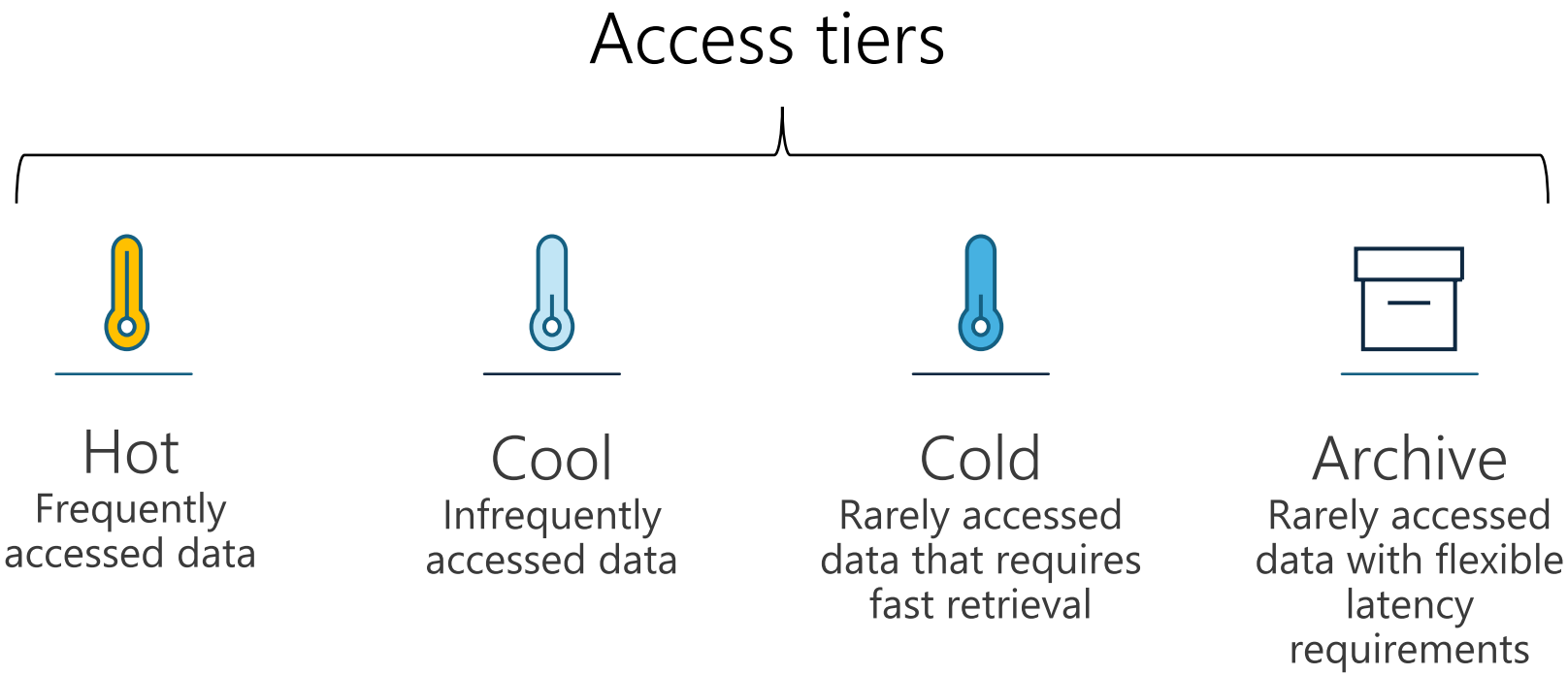
Learning objectives

- Describe how each of the access tiers are optimized.
- Create and implement a lifecycle policy.
- Rehydrate blob data stored in an archive tier.

Introduction

- Data stored in the cloud grows at an exponential pace.
- The need for access to data can drop drastically as the data ages.
- It can be helpful to organize your data based on how frequently it will be accessed and how long it will be retained to manage.

Explore the Azure Blob storage lifecycle (1 of 2)



- The Hot access tier has the lowest access cost, but the highest storage cost.
- Moving through the tiers from Hot to Archive the storage costs decrease, but the access costs increase.

Explore the Azure Blob storage lifecycle (2 of 2)

- Azure Blob storage lifecycle management offers a rich, rule-based policy for General Purpose v2 and Blob storage accounts.
- Use the policy to transition your data to the appropriate access tiers or expire at the end of the data's lifecycle.
- The lifecycle management policy lets you:
  - Transition blobs to a cooler storage tier to optimize for performance and cost
  - Delete blobs at the end of their lifecycles
  - Define rules to be run once per day at the storage account level
  - Apply rules to containers or a subset of blobs (using prefixes as filters)

Discover Blob storage lifecycle policies (1 of 2)

Policies

- A *policy* is a collection of *rules*
- Each rule within the policy has several parameters
  - name
  - enabled
  - type
  - definition

Rules

- Each rule definition includes a filter set and an action set.
- The filter set limits rule actions to a certain set of objects within a container or objects names.
- The action set applies the tier or delete actions to the filtered set of objects.

Discover Blob storage lifecycle policies (2 of 2)

Policy example

```
{
  "rules": [
    {
      "name": "rule1",
      "enabled": true,
      "type": "Lifecycle",
      "definition": {...}
    },
    {
      "name": "rule2",
      "type": "Lifecycle",
      "definition": {...}
    }
  ]
}
```

Parameter name	Parameter type	Required
name	String	`True`
enabled	Boolean	`False`
type	An enum value	`True`
definition	An object that defines the lifecycle rule	`True`

Implement Blob storage lifecycle policies

- Azure portal
  - Azure portal List view
  - Azure portal Code view
- Command line
  - PowerShell
  - Azure CLI
- REST APIs

```
az storage account management-policy create \
    --account-name <storage-account> \
    --policy @policy.json \
    --resource-group <resource-group>
```

Rehydrate blob data from the archive tier

Two options for rehydrating a blob in the archive tier:

- Copy an archived blob to an online tier
- Change a blob's access tier to an online tier

Rehydration priority

- Standard priority
- High priority

Summary and knowledge check

In this module, you learned how to:

- Describe how each of the access tiers are optimized.
- Create and implement a lifecycle policy.
- Rehydrate blob data stored in an archive tier.

1. Which access tier is considered to be offline and can't be read or modified?

2. What storage account type supports lifecycle policies?

Module 3: Work with Azure Blob storage

Learning objectives

- Create an application to create and manipulate data by using the Azure Storage client library for Blob storage.
- Manage container properties and metadata by using .NET and REST.

Introduction

- The Azure Storage client libraries for .NET offer a convenient interface for making calls to Azure Storage.
- The latest version of the Azure Storage client library is version 12.x.
- Microsoft recommends using version 12.x for new applications.

Explore Azure Blob storage client library

Class	Description
`BlobClient`	Represents a specific blob and provides general operations to work with the blob.
`BlobContainerClient`	Represents a specific blob container and provides operations to work with the container and the blobs within.
`BlobServiceClient`	Represents the storage account, and provides operations to retrieve and configure account properties, and to work with blob containers in the storage account.
`AppendBlobClient`	Represents an append blob, and provides operations specific to append blobs, such as appending log data.
`BlockBlobClient`	Represents a block blob, and provides operations specific to block blobs, such as staging and then committing blocks of data.

Create a client object

- When an app creates a client object an endpoint URI is passed.
- The endpoint string can be constructed manually, as shown in the example, or
- It can be queried at runtime using the Azure Storage management library.
- Example code is using the `DefaultAzureCredential` for authentication.

```
using Azure.Identity;
using Azure.Storage.Blobs;

public BlobServiceClient GetBlobServiceClient(string accountName)
{
    BlobServiceClient client = new(
        new Uri($"https://{accountName}.blob.core.windows.net"),
        new DefaultAzureCredential());

    return client;
}
```

Exercise: Create Blob storage resources by using the .NET client library

In this exercise you learn how to use the Azure Blob storage client library to operations in Azure Blob storage in a console app.

Objectives

- Create a container
- Upload blobs to a container
- List the blobs in a container
- Download blobs
- Delete a container

Manage container properties and metadata by using .NET

- Blob containers support system properties and user-defined metadata, in addition to the data they contain.
- Retrieve container properties
  - `GetProperties`
  - `GetPropertiesAsync`
- Set metadata
  - `SetMetadata`
  - `SetMetadataAsync`

Set and retrieve properties and metadata for blob resources by using REST

- Metadata header format: `x-ms-meta-name:string-value`
- URI syntax to retrieve properties and metadata from containers and blobs
  - `GET/HEAD https://myaccount.blob.core.windows.net/mycontainer?restype=container`
  - `GET/HEAD https://myaccount.blob.core.windows.net/mycontainer/myblob?comp=metadata`
- URI syntax to set properties and metadata from containers and blobs
  - `PUT https://myaccount.blob.core.windows.net/mycontainer?restype=container`
  - `PUT https://myaccount.blob.core.windows.net/mycontainer/myblob?comp=metadata`

Summary and knowledge check

In this module, you learned how to:

- Create an application to create and manipulate data by using the Azure Storage client library for Blob storage.
- Manage container properties and metadata by using .NET and REST

1. What standard HTTP headers are supported for both containers and blobs when setting properties by using REST?

2. What class of the Azure Storage client library for .NET allows you to manipulate both Azure Storage containers and their blobs?

Discussion and lab

Group discussion questions

- How would/could you apply access tiers and lifecycle management policies to data your apps are currently using?
- Contoso Inc wants to manage the data encryption in their storage account with their own keys. Can you describe a scenario where they would want to use a customer-managed key over a customer-provided key?

Lab 03: Retrieve Azure Storage resources and metadata by using the Azure Storage SDK for .NET

In this lab, you will learn how to use the Azure Storage SDK to access Azure Storage containers within a C# application. You will also learn how to access metadata and expose URI information to gain access to the contents of the containers in the storage account.

http://aka.ms/az204labs

- Exercise 1: Create Azure resources
- Exercise 2: Upload a blob into a container
- Exercise 3: Access containers by using the .NET SDK
- Exercise 4: Retrieve blob Uniform Resource Identifiers (URIs) by using the .NET SDK

End of presentation