Microsoft

arelance
Building Digital Talent

AZ-204T00A

Learning Path 01: Implement Azure App Service web apps

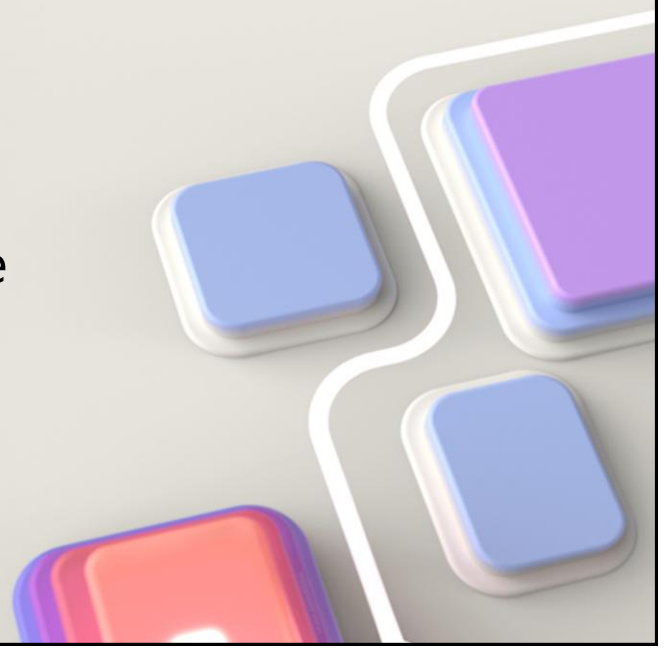
1

Agenda

- Explore Azure App Service
- Configure web app settings
- Scale apps in Azure App Service
- Explore Azure App Service deployment slots

2

Module 1: Explore Azure App Service



3

Learning objectives

- Describe Azure App Service key components and value.
- Explain how Azure App Service manages authentication and authorization.
- Identify methods to control inbound and outbound traffic to your web app.
- Deploy an app to App Service using Azure CLI commands.

4

Introduction

- Azure App Service is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends.
- Applications run and scale in both Windows and Linux-based environments

5

Examine Azure App Service

Built-in scale support

- Save costs and meet demand through scaling
- Scale out/in manually or automated based on metrics
- Scale up/down by changing the app service plan

Continuous integration/ deployment support

- Azure DevOps
- GitHub
- Bitbucket
- FTP
- Local Git repository
- Container registry
- And others

Deployment slots

- Target deployments to test or production environments
- Customize what settings are swapped between slots

6

Examine Azure App Service plans

App Service plans

- Define a set of compute resources
- Can run one or more apps in the same plan
- Can be scaled up or down at any time to meet compute or feature needs

Usage tiers

- Shared: shared compute resources targeting dev/test
- Basic: dedicated compute targeting dev/test
- Standard: run production workloads
- Premium: Enhanced performance and scale
- Isolated: high-performance, security and isolation

How does my app run and scale?

- Shared: apps receive CPU minutes on a shared VM instance and can't scale out
- Other tiers: apps run on all the VM instances configured in the App Service plan

7

Deploy to App Service

Every development team has unique requirements for their deployment pipeline. App Service supports both automated and manual deployment.

Automated deployment

- Azure DevOps
- GitHub
- Bitbucket

Manual deployment

- Git
- CLI
- Zipdeploy
- FTP/S

Deployment slots

- Target deployments to test or production environments
- Customize what settings are swapped between slots

8

Explore authentication and authorization in App Service

- Built-in authentication and authorization support
  - Implement with low to no code changes in your web app

- Identity providers available by default
  - Microsoft Identity Platform
  - Facebook
  - Google
  - Twitter
  - Apple
  - OpenID Connect

9

Discover App Service networking features

Multitenant App Service networking features

- Inbound features
  - App-assigned address
  - Access restrictions
  - Service endpoints
  - Private endpoints
- Outbound features
  - Hybrid Connections
  - Gateway-required VNet Integration
  - VNet Integration

Single-tenant networking

Azure App Service Environment hosts Isolated SKU plans directly in your Azure virtual network.

- **External:** Exposes the hosted apps by using an IP address that is accessible on the internet.
- **Internal load balancer:** Exposes the hosted apps on an IP address inside your virtual network.

10

Exercise: Create a static HTML web app by using Azure Cloud Shell

- In this exercise, you deploy a basic HTML+CSS site to Azure App Service by using the Azure CLI **az webapp up** command.
- You then update the code and redeploy it by using the same command.

Objectives

- Download the sample app
- Create the web app
- Update and redeploy the app

11

Summary and knowledge check

In this module, you learned how to:

- Describe Azure App Service key components and value.
- Explain how Azure App Service manages authentication and authorization.
- Identify methods to control inbound and outbound traffic to your web app.
- Deploy an app to App Service using Azure CLI commands.

1. Which App Service plan category provides the maximum scale-out capabilities?
2. What networking feature of App Service can be used to control outbound network traffic?

12

Module 2: Configure web app settings

13

Learning objectives

- Create application settings that are bound to deployment slots.
- Explain the options for installing TLS certificates for your app.
- Enable diagnostic logging for your app to aid in monitoring and debugging.
- Create virtual app to directory mappings.

14

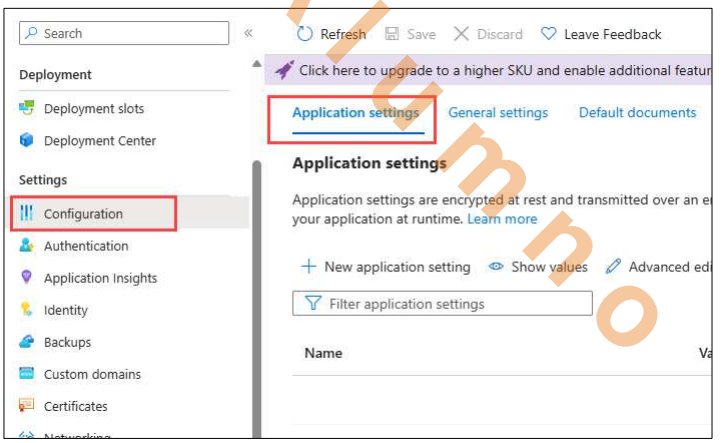
Introduction

- In App Service, app settings are passed as environment variables to the application code.
- For Linux apps and custom containers, App Service passes app settings to the container using the **--env** flag to set the environment variable in the container.

15

Configure application settings (1 of 2)

- Adding and editing settings
  - To add a new app setting, click **New application setting.**
  - To add or edit app settings in bulk, click the **Advanced** edit button.
- Configure connection strings
  - Adding and editing connection strings follow the same principles as other app settings and they can also be tied to deployment slots.



16

Configure application settings (2 of 2)

Editing settings in bulk

```
[
    {
      "name": "name-1",
      "value": "conn-string-1",
      "type": "SQLServer",
      "slotSetting": false
    },
    {
      "name": "name-2",
      "value": "conn-string-2",
      "type": "PostgreSQL",
      "slotSetting": false
    },
    ...
  ]
```


17

Configure general settings

Stack settings

The software stack to run the app, including the language and SDK versions.

Platform settings

Configure settings for the hosting platform, including:

- WebSocket protocol
- Always On
- Managed pipeline version
- HTTP version
- ARR affinity

Debugging

Enable remote debugging for ASP.NET , ASP.NET Core, or Node.js apps.

Incoming client certificates

Require client certificates in mutual authentication. TLS mutual authentication is used to restrict access to your app by enabling different types of authentication for it.

18

Configure path mappings

Linux and containerized apps

- Add custom storage for your containerized app.
- Containerized apps include all Linux apps and Windows and Linux custom containers running on App Service.

Windows apps (uncontainerized)

- Customize the IIS handler mappings and virtual applications and directories.
- Handler mappings enable adding custom script processors to handle requests for specific file extensions.

19

Enable diagnostic logging

Type	Platform	Description
Application logging	Windows, Linux	Logs messages generated by your application code. The messages are generated by the web framework you choose.
Web server logging	Windows	Raw HTTP request data in the W3C extended log file format.
Detailed error messages	Windows	Copies of the *.html* error pages that would have been sent to the client browser.
Failed request tracing	Windows	Detailed tracing information on failed requests
Deployment logging	Windows, Linux	Deployment logging happens automatically and there are no configurable settings for deployment logging.

20

Configure security certificates

- Options for adding certificates in App Service
  - Free App Service managed certificate
  - Purchase an App Service certificate
  - Import a certificate from Key Vault
  - Upload a private certificate
  - Upload a public certificate

21

Summary and knowledge check

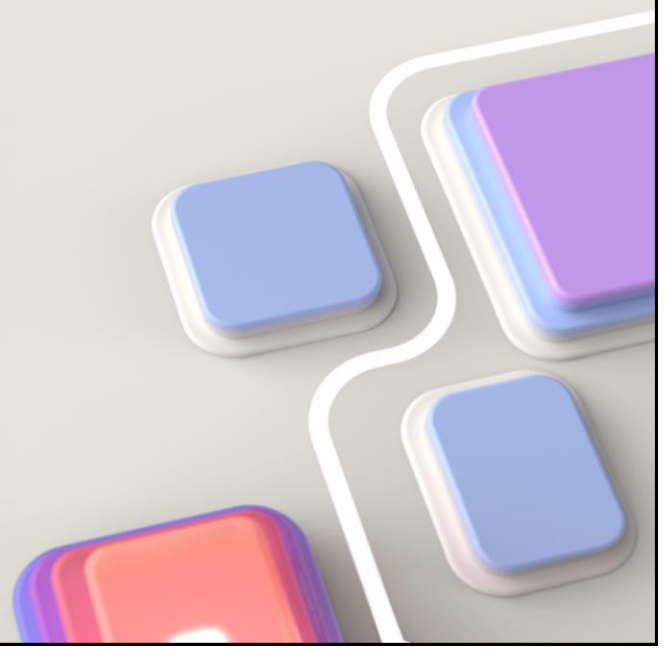
In this module, you learned how to:

- Create application settings that are bound to deployment slots.
- Explain the options for installing SSL/TLS certificates for your app.
- Enable diagnostic logging for your app to aid in monitoring and debugging.
- Create virtual app to directory mappings.

1. Which app configuration settings category is used to set the language and SDK version?
2. What types of diagnostic logging are supported on the Linux platform?

22

Module 3: Scale apps in Azure App Service



23

Learning objectives

- Identify scenarios for which autoscaling is an appropriate solution
- Create autoscaling rules for a web app
- Monitor the effects of autoscaling

24

Introduction

- Autoscaling adjusts the number of available instances to meet the varying demands on your application
- Create rules to specify the conditions where instances should be added or removed
- Control costs

25

Examine autoscale factors

- Autoscaling adjusts available resources based on the current demand.
- Autoscaling performs scaling in and out, as opposed to scaling up and down.
- Monitors the resource metrics of a web app as it runs and detects when additional resources are required based on the set conditions.
- When should you consider autoscaling?
  - Autoscaling provides elasticity for your services.
  - Autoscaling improves availability and fault tolerance.
  - Autoscaling isn't the best approach to handling long-term growth.

26

Identify autoscale factors

Autoscale conditions

- Scale based on a metric, such as CPU usage or the length of a disk queue.
- Scale based on a schedule, such as a time of day or day of the week.
- Can create multiple conditions to handle complex needs.

Autoscale metrics

- Metrics are based on all instances of an app
- CPU percentage
- Memory percentage
- Disk queue length – outstanding I/O requests
- HTTP queue length – number of client requests
- Data in – number of bytes received
- Data Out – number of bytes sent

27

Enable autoscale in App Service

- Enable autoscaling
  - Not all pricing tiers support autoscaling
  - Some are only single instance or limited to manual scaling
- Add scale conditions
  - A default scale condition is created
  - Edit the default or create additional conditions
- Create scale rules
  - Conditions can contain one or more scale rules
  - Rules can be set based on calendar, metric, or both
- Monitor autoscaling behavior
  - View autoscale changes on the **Run history** chart
  - Tracks number of instances and which condition triggered the change

28

Explore autoscale best practices

- Ensure the maximum and minimum values are different and have an adequate margin between them
- Choose the appropriate statistic for your diagnostics metric
- Choose the thresholds carefully for all metric types
- Check for conflicts when multiple rules are configured in a condition
- Always select a safe default instance count
- Configure autoscale notifications

29

Summary and knowledge check

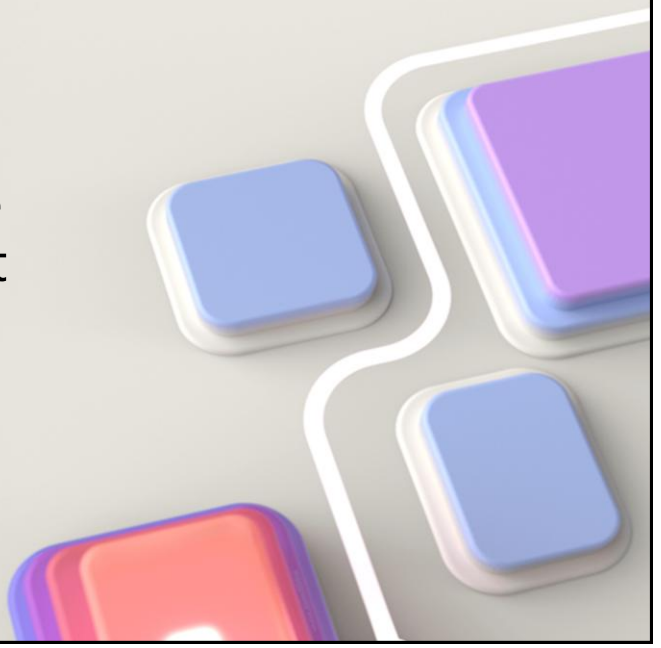
In this module, you learned how to:

- Identify scenarios for which autoscaling is an appropriate solution
- Create autoscaling rules for a web app
- Monitor the effects of autoscaling

1. How would you describe autoscaling?
2. Can you describe a scenario that is a suitable candidate for autoscaling?

30

Module 4: Explore Azure App Service deployment slots



31

Learning objectives

- Describe the benefits of using deployment slots
- Understand how slot swapping operates in App Service
- Perform manual swaps and enable auto swap
- Route traffic manually and automatically

32

Introduction

- Deployment slots enable you to preview, manage, test, and deploy different development environments.
- Deployment slots are live apps with their own host names.
- App content and configuration elements can be swapped between two slots.

33

Explore staging environments

- You can use a separate deployment slot instead of the default production slot when you're running in the **Standard**, **Premium**, or **Isolated** App Service plan tier.
- Deploying to a non-production slot has the following benefits:
  - You can validate app changes in a staging deployment slot before swapping it with the production slot.
  - Deploying an app to a slot first and swapping it into production makes sure that all instances of the slot are warmed up before being swapped into production.
  - After a swap, the slot with previously staged app now has the previous production app.

34

Examine slot swapping

When swapping slots, App Service does the following

1. Applies the following settings from the target slot to all instances of the source slot:
   - Slot-specific app settings and connection strings, if applicable.
   - Continuous deployment settings, if enabled.
   - App Service authentication settings, if enabled.
2. Wait for every instance in the source slot to complete its restart.
3. If local cache is enabled, trigger local cache initialization by making an HTTP request to the application root ("/") on each instance of the source slot.
4. If auto swap is enabled with custom warm-up, trigger Application Initiation by making an HTTP request to the application root ("/") on each instance of the source slot.
5. If all instances on the source slot are warmed up successfully, swap the two slots by switching the routing rules for the two slots.
6. Now that the source slot has the pre-swap app previously in the target slot, perform the same operation by applying all settings and restarting the instances.

35

Swap deployment slots

- Swap deployment slots on your app's Deployment slots page and the Overview page.
- Configure auto swap
- Swap with preview
- Specify a custom warm-up
- Roll back and monitor a swap

36

Route traffic in App Service

Route production traffic automatically

- Go to your app's resource page and select **Deployment slots**.
- In the **Traffic %** column of the slot you want to route to, specify a percentage (between 0 and 100) to represent the amount of total traffic you want to route.

Route production traffic manually

- In addition to automatic traffic routing, App Service can route requests to a specific slot.
- This is useful when you want your users to be able to opt in to or opt out of your beta app.
- To route production traffic manually, you use the **x-ms-routing-name** query parameter.

37

Summary and knowledge check

In this module, you learned how to:

- Describe the benefits of using deployment slots
- Understand how slot swapping operates in App Service
- Perform manual swaps and enable auto swap
- Route traffic manually and automatically

1. What is the default routing rule applied to new deployment slots?
2. Do all configuration elements follow the content across a swap?

38

Discussion and lab

39

Group discussion questions

- What are deployment slots and how can they be used?
- How does autoscale work in App Service? How would you scale-up an application?
- Can you name the different options to deploy apps to App Service both manually and automatically?

40

Lab 01: Build a web application on Azure platform as a service offerings

In this lab, you will explore how to create a web application on Azure by using the PaaS model. After the web application is created, you will learn how to upload existing web application files by using the Apache Kudu zip deployment option. You will then view and test the newly deployed web application.

http://aka.ms/az204labs

- Exercise 1: Build a backend API by using Azure Storage and the Web Apps feature of Azure App Service
- Exercise 2: Build a front-end web application by using Azure Web Apps

41

End of presentation

42

Microsoft

arelance
Building Digital Talent

AZ-204T00A

Learning Path 02: Implement Azure Functions

1

Agenda

- Explore Azure Functions
- Develop Azure Functions

2

Module 1: Explore Azure Functions

3

Learning objectives

- Explain functional differences between Azure Functions, Azure Logic Apps, and WebJobs
- Describe Azure Functions hosting plan options
- Describe how Azure Functions scale to meet business needs

4

Introduction

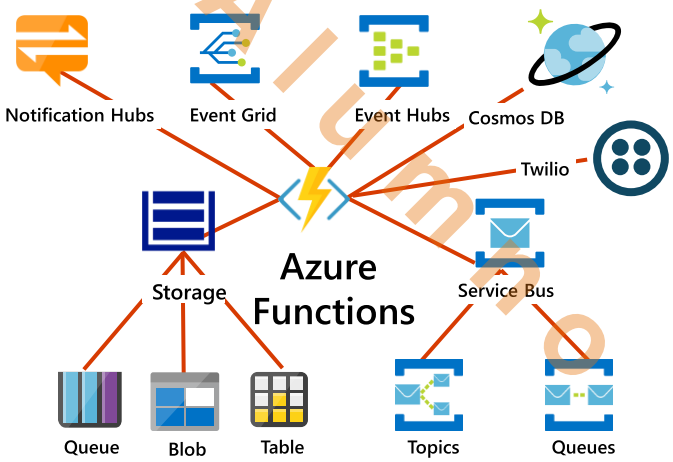
- Azure Functions is a serverless solution
- The cloud infrastructure provides all the up-to-date resources needed to keep your applications running
- Azure Functions scale with demand

5

Discover Azure Functions (1 of 3)

Overview

- Consider Functions for tasks like image or order processing, file maintenance, or for any tasks that you want to run on a schedule.
- Azure Functions supports *triggers*, which start execution of your code, and *bindings*, which simplify coding for input and output data.



6

Discover Azure Functions (2 of 3)

Compare Azure Functions and Azure Logic Apps

	Functions	Logic Apps
Development	Code-first (imperative)	Designer-first (declarative)
Connectivity	About a dozen built-in binding types, write code for custom bindings	Large collection of connectors, Enterprise Integration Pack for B2B scenarios, build custom connectors
Actions	Each activity is an Azure function; write code for activity functions	Large collection of ready-made actions
Monitoring	Azure Application Insights	Azure portal, Azure Monitor logs
Management	REST API, Visual Studio	Azure portal, REST API, PowerShell, Visual Studio
Execution context	Can run locally or in the cloud	Supports run-anywhere scenarios

7

Discover Azure Functions (3 of 3)

Compare Functions and WebJobs

	Functions	WebJobs with WebJobs SDK
Serverless app model with automatic scaling	Yes	No
Develop and test in browser	Yes	No
Pay-per-use pricing	Yes	No
Integration with Logic Apps	Yes	No
Trigger events	Timer Azure Storage queues and blobs Azure Service Bus queues and topics Azure Cosmos DB Azure Event Hubs HTTP/WebHook (GitHub Slack) Azure Event Grid	Timer Azure Storage queues and blobs Azure Service Bus queues and topics Azure Cosmos DB Azure Event Hubs File system

8

Compare Azure Functions hosting options (1 of 3)

Hosting plans:

- Consumption plan
- Flex Consumption plan (Preview)
- Premium plan
- Dedicated plan (App Service)
- Container Aps

Hosting plans dictate:

- How your function app is scaled.
- The resources available to each function app instance.
- Support for advanced functionality, such as Azure Virtual Network connectivity.

9

Compare Azure Functions hosting options (2 of 3)

Hosting plan	Description
Consumption	Default plan. Pay for compute resources only when your functions are running with automatic scale. Instances of the Functions host are dynamically scaled based on the number of incoming events.
Flex Consumption (Preview)	High scalability with compute choices, virtual networking, and pay-as-you-go billing. Automatic scaling, can specify pre-provisioned instances.
Premium	Automatically scales based on demand using prewarmed workers, which run applications with no delay after being idle, runs on more powerful instances, and connects to virtual networks.
Dedicated	Run your functions within an App Service plan, Best for long-running scenarios where Durable Functions can't be used.
Container Apps	Supports packaging custom libraries with your function code, provides high-end processing power provided by dedicated GPU compute.

10

Compare Azure Functions hosting options (3 of 3)

Timeout durations

The `functionTimeout` property in the *host.json* project file specifies the timeout duration.

Plan	Default (Minutes)	Maximum (Minutes)
Consumption	5	10
Flex Consumption (Preview)	30	Unlimited
Premium	30	Unlimited
Dedicated	30	Unlimited
Container Apps	30	Unlimited

11

Scale Azure Functions (1 of 2)

- The unit of scale for Azure Functions is the function app.
- A *scale controller* monitors the rate of events to determine whether to scale out or scale in.
- Instances may be "scaled in" to zero when no functions are running within a function app.
- The next request has the latency – a *cold start* – of scaling from zero to one

12

Scale Azure Functions (2 of 2)

Scaling behaviors

Plan	Scale out	Max instances
Consumption	Event driven. Scales out automatically based on the number of incoming trigger events.	Windows: 200 Linux: 100
Flex Consumption (Preview)	Per-function scaling. Event-driven scaling decisions are calculated on a per-function basis	Limited by memory usage across all instances in a given region.
Premium	Event driven. Scale out automatically based on the number of events that its functions are triggered on.	Windows: 100 Linux: 20-100
Dedicated	Manual/autoscale	10-30 100 (App Service Environment)
Container Apps	Event driven. Scale out automatically based on the number of events that its functions are triggered on.	10-300

13

Summary and knowledge check

In this module, you learned how to:

- Explain functional differences between Azure Functions, Azure Logic Apps, and WebJobs
- Describe Azure Functions hosting plan options
- Describe how Azure Functions scale to meet business needs

1. Which Azure Functions hosting plan is best when predictive scaling and costs are required?
2. Which serverless workflow would you choose if the solution requires a designer-first development model?

14

Module 2: Develop Azure Functions

15

Learning objectives

- Explain the key components of a function and how they are structured
- Create triggers and bindings to control when a function runs and where the output is directed
- Connect a function to services in Azure
- Create a function by using Visual Studio Code and the Azure Functions Core Tools

16

Introduction

- Functions make it easy to use your favorite code editor and development tools to create and test functions on your local computer.
- Your local functions can connect to live Azure services, and you can debug them on your local computer using the full Functions runtime.

17

Explore Azure Functions development (1 of 3)

A function app provides an execution context in Azure to run your functions.

- It is the unit of deployment and management for your functions.
- A function app is comprised of one or more individual functions that are managed, deployed, and scaled together.
- All functions in a function app share the same pricing plan, deployment method, and runtime version.

18

Explore Azure Functions development (2 of 3)

- A Functions project directory contains the following files in the project root folder, regardless of language:
  - `host.json` - contains configuration options that affect all functions in a function app instance
  - `local.settings.json` - stores app settings, and settings used by local development tools. Only used when running your project locally.
- Other function app configuration options are managed depending on where the function app runs:
  - **Deployed to Azure:** in your application settings
  - **On your local computer:** in the `local.settings.json` file

19

Explore Azure Functions development (3 of 3)

local.setting.json example

```
{
    "IsEncrypted": false,
    "Values": {
        "FUNCTIONS_WORKER_RUNTIME": "<language worker>",
        "AzureWebJobsStorage": "<connection-string>",
        "MyBindingConnection": "<binding-connection-string>",
        "AzureWebJobs.HttpExample.Disabled": "true"
    },
    "Host": {
        "LocalHttpPort": 7071,
        "CORS": "*",
        "CORSCredentials": false
    },
    "ConnectionStrings": {
        "SQLConnectionString": "<sqlclient-connection-string>"
    }
}
```


20

Create triggers and bindings (1 of 7)

Overview

- Triggers are what cause a function to run. A trigger defines how a function is invoked and a function must have exactly one trigger.
- Binding to a function is a way of declaratively connecting another resource to the function
- Bindings may be connected as input bindings, output bindings, or both.
- You can mix and match different bindings to suit your needs.
- Triggers and bindings let you avoid hardcoding access to other services.

21

Create triggers and bindings (2 of 7)

Trigger and binding definitions

- Triggers and bindings are defined differently depending on the development language.
- C# class library - decorating methods and parameters with C# attributes
- Java - decorating methods and parameters with Java annotations
- JavaScript/PowerShell/Python/TypeScript - updating *function.json* schema

```
{
  "dataType": "binary",
  "type": "httpTrigger",
  "name": "req",
  "direction": "in"
}
```


22

Create triggers and bindings (3 of 7)

Binding direction

- All triggers and bindings have a direction property in the *function.json* file
- For triggers, the direction is always `in`
- Input and output bindings use `in` and `out`
- Some bindings support a special direction `inout`. If you use `inout`, only the **Advanced editor** is available via the **Integrate** tab in the portal.
- When you use attributes in a class library to configure triggers and bindings, the direction is provided in an attribute constructor or inferred from the parameter type.

23

Create triggers and bindings (4 of 7)

Azure Functions trigger and binding example

Scenario: You want to write a new row to Azure Table storage whenever a new message appears in Azure Queue storage.

This scenario can be implemented using an Azure Queue storage trigger and an Azure Table storage output binding.

```
"bindings": [
    {
      "type": "queueTrigger",
      "direction": "in",
      "name": "order",
      "queueName": "myqueue-items",
      "connection": "STORAGE_ACCT_SETTING"
    },
    {
      "type": "table",
      "direction": "out",
      "name": "$return",
      "tableName": "outTable",
      "connection": "STORAGE_ACCT_SETTING"
    }
```


24

Create triggers and bindings (5 of 7)

C# script example

C# script code that works with the previous trigger and binding specified.

```
...
public static Person Run(JObject order, ILogger log)
{
    return new Person() {
        PartitionKey = "Orders",
        RowKey = Guid.NewGuid().ToString(),
        Name = order["Name"].ToString(),
        MobileNumber = order["MobileNumber"].ToString() };
}
public class Person
{
    public string PartitionKey { get; set; }
    public string RowKey { get; set; }
    public string Name { get; set; }
    public string MobileNumber { get; set; }
}
```


25

Create triggers and bindings (6 of 7)

JavaScript example

JavaScript code that works with the previous trigger and binding specified.

```
module.exports = async function (context, order) {
    order.PartitionKey = "Orders";
    order.RowKey = generateRandomId();
    context.bindings.order = order;
};
function generateRandomId() {
    return Math.random().toString(36).substring(2, 15) +
        Math.random().toString(36).substring(2, 15);
}
```


26

Create triggers and bindings (7 of 7)

Class library example

```
public static class QueueTriggerTableOutput
{
    [FunctionName("QueueTriggerTableOutput")]
    [return: Table("outTable", Connection = "CONNECTION")]
    public static Person Run(
        [QueueTrigger("myqueue-items", Connection = "CONNECTION")]JObject order,
        ILogger log)
    {
        return new Person() {
            PartitionKey = "Orders",
            RowKey = Guid.NewGuid().ToString(),
            Name = order["Name"].ToString(),
            MobileNumber = order["MobileNumber"].ToString() };
    }
...
```


27

Connect functions to Azure services (1 of 2)

Overview

- Your function project references connection information by name from its configuration provider.
- It does not directly accept the connection details, allowing them to be changed across environments.
- The default configuration provider uses environment variables. These might be set by Application Settings when running in the Azure Functions service, or from the local settings file when developing locally.

Connection values

- When the connection name resolves to a single exact value, the runtime identifies the value as a connection string, which typically includes a secret.
- The details of a connection string are defined by the service to which you wish to connect.
- A connection name can also refer to a collection of multiple configuration items.
- Environment variables can be treated as a collection by using a shared prefix that ends in double underscores __.

28

Connect functions to Azure services (2 of 2)

Configure an identity-based connection

- Some connections in Azure Functions are configured to use an identity instead of a secret. Support depends on the extension using the connection.
- In some cases, a connection string may still be required in Functions even though the service to which you are connecting supports identity-based connections.

Grant permission to the identity

- Whatever identity is being used must have permissions to perform the intended actions.
- This is typically done by assigning a role in Azure RBAC or specifying the identity in an access policy, depending on the service to which you are connecting.

29

Exercise: Create an Azure Function by using Visual Studio Code

In this exercise, you learn how to create a C# function that responds to HTTP requests. After creating and testing the code locally in Visual Studio Code, you'll deploy to Azure.

Objectives

- Create your local project
- Run the function locally
- Sign into Azure
- Publish the project to Azure
- Run the function in Azure
- Clean up resources

30

Summary and knowledge check

In this module, you learned how to:

- Explain the key components of a function and how they are structured
- Create triggers and bindings to control when a function runs and where the output is directed
- Connect a function to services in Azure
- Create a function by using Visual Studio Code and the Azure Functions Core Tools

1. What is required for a function to run. A trigger, binding, or both?
2. What supports the `in` and `out` direction settings. Triggers, bindings, or both?

31

Discussion and lab

32

Group discussion questions

- Contoso Inc built an Azure Function that is hosted using Consumption Plan. After analyzing the logs, you've noticed they are timing out after 5 minutes. What can you do to avoid this problem?
- Contoso Inc built an Azure Function to consume events from Event Hub. After some load testing, you've noticed they are not performing well. What can you do increase the performance?

33

Lab 02: Implement task processing logic by using Azure Functions

In this lab, you will demonstrate the ability to create a simple Azure function that echoes text that is entered and sent to the function by using HTTP POST commands. This will illustrate how the function can be triggered over HTTP.

http://aka.ms/az204labs

- Exercise 1: Create Azure resources
- Exercise 2: Configure a local Azure Functions project
- Exercise 3: Create a function that's triggered by an HTTP request
- Exercise 4: Create a function that triggers on a schedule
- Exercise 5: Create a function that integrates with other services
- Exercise 6: Deploy a local function project to an Azure Functions app

34

35

Microsoft

AZ-204T00A

Learning Path 03: Develop solutions that use Blob storage

1

Agenda

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

2

Module 1: Explore Azure Blob storage

3

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.

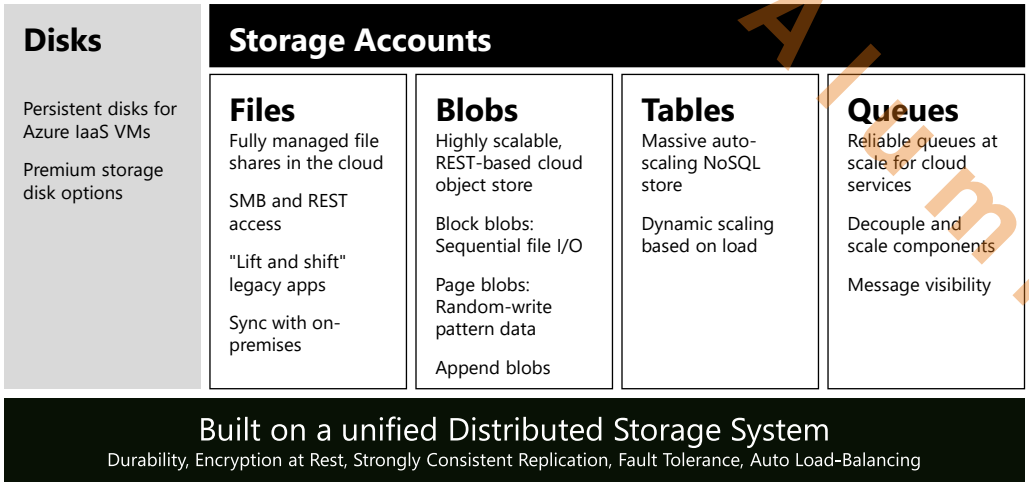
4

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.

5

Explore Azure Blob storage (1 of 2)



6

Explore Azure Blob storage (2 of 2)

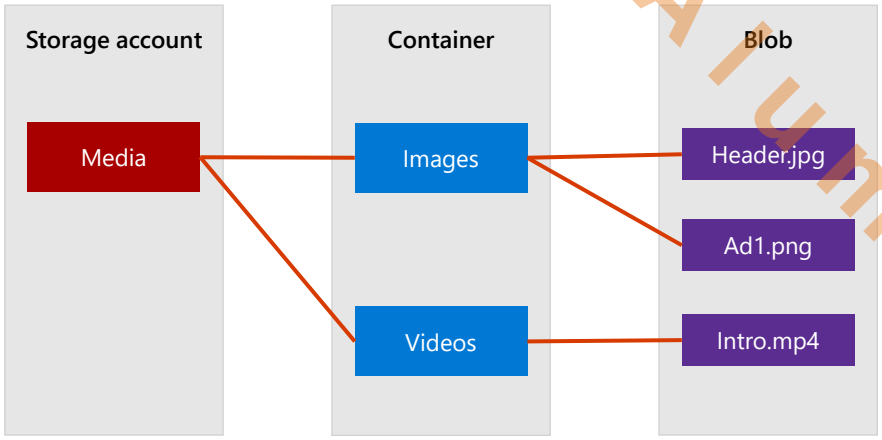
Performance level	Storage account type and supported storage services
Standard	**Standard general-purpose v2:** Blob, Queue, and Table storage, Azure Files
Premium	**Premium block blobs:** Blob storage **Premium page blobs:** Page blobs only **File shares:** Azure files

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.

7

Discover Azure Blob storage resource types



8

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.

9

Summary and knowledge check

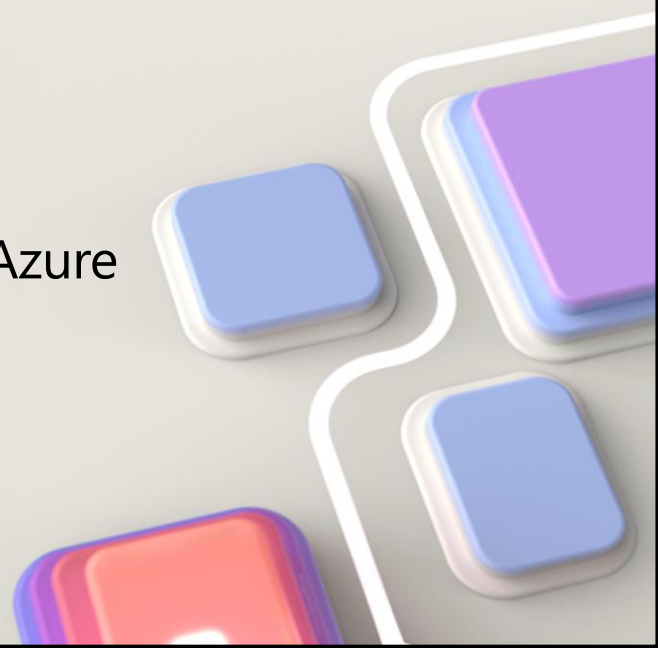
In this module, you learned how to:

- Identify the different types of storage accounts and the resource hierarchy for blob storage.
- Explain how data is securely stored.

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?

10

Module 2: Manage the Azure Blob storage lifecycle



11

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.

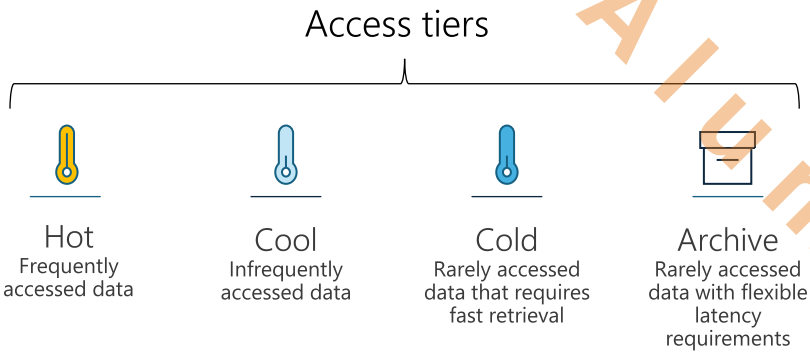
12

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.

13

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.

14

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)

15

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.

16

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

17

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>
```


18

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

19

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?

20

Module 3: Work with Azure Blob storage

21

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.

22

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.

23

Explore Azure Blob storage client library

Class	Description
`BlobClient`	Allows you to manipulate Azure Storage blobs.
`BlobClientOptions`	Provides the client configuration options for connecting to Azure Blob Storage.
`BlobContainerClient`	Allows you to manipulate Azure Storage containers and their blobs.
`BlobServiceClient`	Allows you to manipulate Azure Storage service resources and blob containers. The storage account provides the top-level namespace for the Blob service.
`BlobUriBuilder`	Provides a convenient way to modify the contents of a Uri instance to point to different Azure Storage resources like an account, container, or blob.

24

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;
}
```


25

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

26

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`

27

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`

28

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?

29

Discussion and lab

30

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?

31

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

32

33

Microsoft

arelance
Building Digital Talent

AZ-204T00A

Learning Path 04: Develop solutions that use Azure Cosmos DB

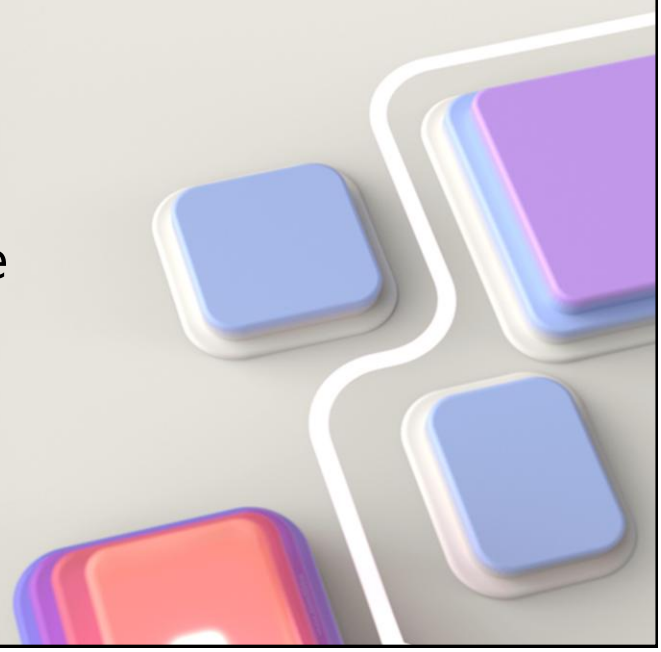
1

Agenda

- Explore Azure Cosmos DB
- Work with Azure Cosmos DB

2

Module 1: Explore Azure Cosmos DB



3

Learning objectives

- Identify the key benefits provided by Azure Cosmos DB.
- Describe the elements in an Azure Cosmos DB account and how they are organized.
- Explain the different consistency levels and choose the correct one for your project.
- Explore the APIs supported in Azure Cosmos DB and choose the appropriate API for your solution.
- Describe how request units impact costs.
- Create Azure Cosmos DB resources by using the Azure portal.

4

Introduction

- Azure Cosmos DB is a fully managed NoSQL database
- Designed to provide low latency, elastic scalability of throughput
- Well-defined semantics for data consistency, and high availability

5

Identify key benefits of Azure Cosmos DB

- **Global replication:** Automatic and synchronous multi-region replication, supports automatic and manual failover
- **Varied consistency levels:** Offers five consistency models. Provides control over performance-consistency tradeoffs, backed by comprehensive SLAs
- **Low latency:** Serve <10 ms read and <10 ms write requests at the 99th percentile
- **Elastic scale-out:** Elastically scale throughput from 10 to 100s of millions of requests/sec across multiple regions

6

Explore the resource hierarchy (1 of 2)

Elements in an Azure Cosmos DB account

- The account is the fundamental unit of global distribution and high availability.
- Azure Cosmos DB account contains a unique DNS name

Azure Cosmos DB databases

- You can create one or multiple Azure Cosmos DB databases under your account.
- A database is analogous to a namespace.
- A database is the unit of management for a set of Azure Cosmos DB containers.

Azure Cosmos DB containers

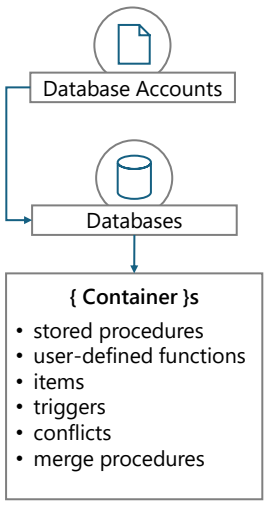
- An Azure Cosmos DB container is the unit of scalability both for provisioned throughput and storage.
- A container is horizontally partitioned and then replicated across multiple regions.

Azure Cosmos DB items

- Depending on which API you use, an Azure Cosmos DB item can represent either a document in a collection, a row in a table, or a node or edge in a graph.

7

Explore the resource hierarchy (2 of 2)



Depending on the Cosmos API, a container is realized as:

- collection
- table
- graph
- ...

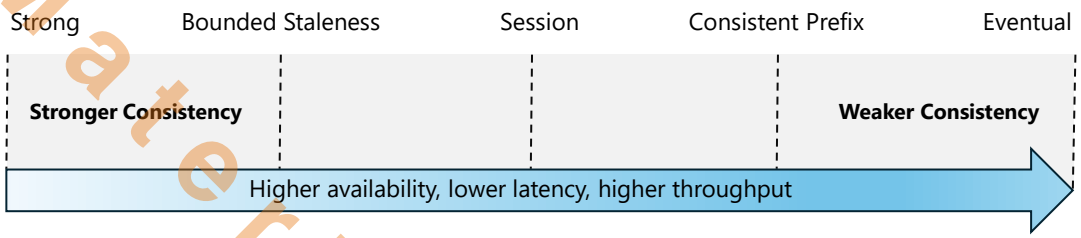
Depending on the Cosmos API, an item is realized as:

- document
- row
- node
- edge
- ...

8

Explore consistency levels (1 of 2)

Azure Cosmos DB approaches data consistency as a spectrum of choices instead of two extremes.



9

Explore consistency levels (2 of 2)

Consistency Level	Description
Strong	When writes are performed on your primary database, the operation is replicated to the replica instances. The write operation is committed (and visible) on the primary only after it has been committed and confirmed by all replicas.
Bounded Staleness	Like the Strong level with the difference that you can configure how stale documents can be within replicas. Staleness is the quantity of time (or the version count) a replica document can be behind the primary document.
Session	This level guarantees that all read and write operations are consistent within a user session. Within the user session, all reads and writes are monotonic and guaranteed to be consistent across primary and replica instances.
Consistent Prefix	This level has loose consistency but guarantees that when updates show up in replicas, they will show up in the correct order (that is, as prefixes of other updates) without any gaps.
Eventual	This level has the loosest consistency and essentially commits any write operation against the primary immediately. Replica transactions are asynchronously handled and will eventually (over time) be consistent with the primary. This tier has the best performance, because the primary database does not need to wait for replicas to commit to finalize its transactions.

10

Choose the right consistency level

Azure Cosmos DB for NoSQL Azure Cosmos DB for Table	Azure Cosmos DB for Cassandra Azure Cosmos DB for MongoDB Azure Cosmos DB for Apache Gremlin	Consistency guarantees in practice
For many real-world scenarios, session consistency is optimal and it's the recommended option.	Azure Cosmos DB provides native support for wire protocol-compatible APIs for popular databases.	Consistency guarantees for a read operation correspond to the freshness and ordering of the database state that you request.
If your application requires strong consistency, it is recommended that you use bounded staleness consistency level.	These include API for MongoDB, API for Apache Cassandra, and API for Apache Gremlin.	Read-consistency is tied to the ordering and propagation of the write/update operations.
If you need the highest availability and the lowest latency, then use eventual consistency level.	When using API for Apache Gremlin the default consistency level configured on the Azure Cosmos account is used.	The Probabilistically Bounded Staleness metric provides an insight into how often you can get a stronger consistency than the configured level.

11

Explore supported APIs

- **API for NoSQL:** This API stores data in document format. It offers the best end-to-end experience as we have full control over the interface, service, and the SDK client libraries.
- **API for MongoDB:** This API stores data in a document structure, via BSON format. It is compatible with MongoDB wire protocol.
- **API for Apache Cassandra:** This API stores data in column-oriented schema and is wire protocol compatible with Apache Cassandra.
- **API for Table:** This API stores data in key/value format.
- **API for Apache Gremlin:** This API allows users to make graph queries and stores data as edges and vertices.
- **API for PostgreSQL:** Stores data either on a single node, or distributed in a multi-node configuration

12

Discover request units

Account types and Request Units

The type of Azure Cosmos DB account created determines the way RUs are charged, there are three modes of account creation:

- **Provisioned throughput:** You manage the number of RUs for your app in 100 RUs per second increments.
- **Serverless:** Billed for the number of RUs consumed by your database operations.
- **Autoscale:** Automatically and instantly scale RUs based on usage.

13

Exercise: Create Azure Cosmos DB resources by using the Azure portal

In this exercise you learn how to create an Azure Cosmos DB account and add data.

Objectives

- Create an Azure Cosmos DB account
- Add a database and a container
- Add data to your database
- Clean up resources

14

Summary and knowledge check

In this module, you learned how to:

- Identify the key benefits provided by Azure Cosmos DB.
- Describe the elements in an Azure Cosmos DB account and how they are organized.
- Explain the different consistency levels and choose the correct one for your project.
- Explore the APIs supported in Azure Cosmos DB and choose the appropriate API for your solution.
- Describe how request units impact costs.
- Create Azure Cosmos DB resources by using the Azure portal.

1. What consistency level offers the greatest throughput?
2. What are request units (RUs) in Azure Cosmos DB?

15

Module 2: Work with Azure Cosmos DB

16

Learning objectives

- Identify classes and methods used to create resources.
- Create resources by using the Azure Cosmos DB .NET v3 SDK.
- Write stored procedures, triggers, and user-defined functions by using JavaScript.
- Implement change feed notifications

17

Introduction

- This module focuses on Azure Cosmos DB .NET SDK v3 for API for NoSQL.
- Because Azure Cosmos DB supports multiple API models, version 3 of the .NET SDK uses the generic terms "container" and "item".
- This module also covers JavaScript when discussing stored procedures, triggers, and user-defined functions.

18

Explore Microsoft .NET SDK v3 for Azure Cosmos DB (1 of 3)

Creating items

```
// Get container reference
CosmosClient client = new CosmosClient(endpoint, key);
Container container = client.GetContainer(databaseName, collectionName);

// create anonymous type in .NET
Product orangeSoda = new Product {
    id = "7cc3212d-0e2c-4a13-b348-f2d879c43342",
    name = "Orange Soda", group = "Beverages",
    diet = false, price = 1.50m, quantity = 2000
};

// Upload item
Product item = await container.CreateItemAsync(orangeSoda);
Product item = await container.UpsertItemAsync(orangeSoda);
```


19

Explore Microsoft .NET SDK v3 for Azure Cosmos DB (2 of 3)

Reading items

```
// Get container reference
CosmosClient client = new CosmosClient(endpoint, key);
Container container = client.GetContainer(databaseName, collectionName);


// Get unique fields
string id = "7cc3212d-0e2c-4a13-b348-f2d879c43342";
PartitionKey partitionKey = new PartitionKey("Beverages");


// Read item using unique id
ItemResponse<Product> response = await container.ReadItemAsync<Product>(
          id,
          partitionKey );


// Serialize response
Product item = response.Resource;
```


20

Explore Microsoft .NET SDK v3 for Azure Cosmos DB (3 of 3)

Query items

```
// Get container reference
CosmosClient client = new CosmosClient(endpoint, key);
Container container = client.GetContainer(databaseName, collectionName);

// Use SQL query language
FeedIterator<Product> iterator = container.GetItemQueryIterator<Product>(
    "SELECT * FROM products p WHERE p.diet = false"
);

// Iterate over results
while (iterator.HasMoreResults)
{
    FeedResponse<Product> batch = await iterator.ReadNextAsync();
    foreach(Product item in batch)  { }
}
```


21

Exercise : Create resources by using the Microsoft .NET SDK v3

In this exercise you create an Azure Cosmos DB account and then perform operations on the account through a console app.

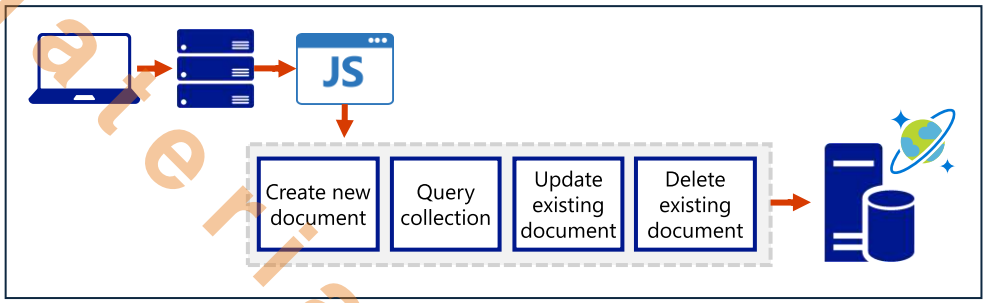
Objectives

- Create resources in Azure
- Set up and build the console application
- Add code to connect to an Azure Cosmos DB account
- Create a database
- Create a container

22

Create stored procedures (1 of 4)

- In Azure Cosmos DB, JavaScript is hosted in the same memory space as the database
- Requests made within stored procedures and triggers run in the same scope of a database session



23

Create stored procedures (2 of 4)

Example code

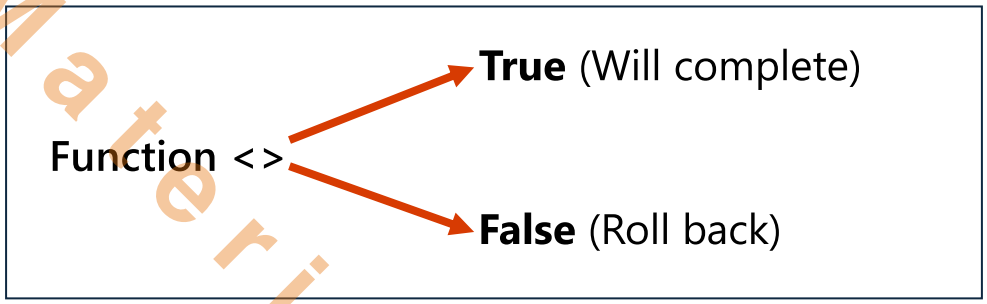
```
function createSampleDocument(documentToCreate) {
    var context = getContext();
    var collection = context.getCollection();
    var accepted = collection.createDocument(
        collection.getSelfLink(),
        documentToCreate,
        function (error, documentCreated) {
            context.getResponse().setBody(documentCreated.id)
        }
    );
    if (!accepted) return;
}
```


24

Create stored procedures (3 of 4)

Bounded execution

- All Azure Cosmos DB operations must complete within a limited amount of time

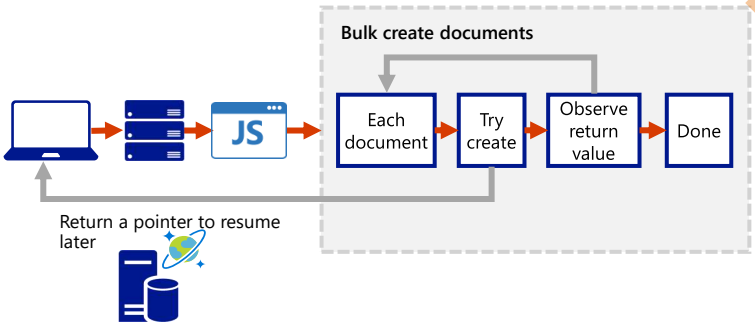


25

Create stored procedures (4 of 4)

Transaction continuation

- JavaScript functions can implement a continuation-based model to batch or resume execution
  - The continuation value can be any value of your choice
  - Your applications can then use this value to resume a transaction from a new starting point



26

Create triggers and user-defined functions (1 of 2)

- **Pre-triggers:** Executed before modifying, or creating, a database item.
  - Can be used to validate the properties of an Azure Cosmos item that is being created, for example.
  - Pre-triggers cannot have any input parameters. The request object in the trigger is used to manipulate the request message associated with the operation.
- **Post-triggers:** Executed after modifying a database item
  - Can be used to query for the metadata item and updates it with details about the newly created item.
- **User-defined function:** User-defined functions extend the Azure Cosmos DB SQL API's query language grammar and implement custom business logic.
  - Can only be called from inside queries. They don't have access to the context object and are meant to be used as compute-only code.

27

Create triggers and user-defined functions (2 of 2)

Pre-trigger example code

```
function validateToDoItemTimestamp() {
    var context = getContext();
    var request = context.getRequest();

    // item to be created in the current operation
    var itemToCreate = request.getBody();

    // validate properties
    if (!("timestamp" in itemToCreate)) {
        var ts = new Date();
        itemToCreate["timestamp"] = ts.getTime();
    }

    // update the item that will be created
    request.setBody(itemToCreate);
}
```


28

Explore change feed in Azure Cosmos DB (1 of 3)

- Change feed in Azure Cosmos DB is a persistent record of changes to a container in the order they occur.
- Work with Azure Cosmos DB change feed using either a push model or a pull model.
- Recommended to use the push model – no need to worry about polling the change feed for future changes.
- Two ways to read from the change feed with a push model:
  - Azure Functions Azure Cosmos DB triggers
  - The change feed processor library

29

Explore change feed in Azure Cosmos DB (2 of 3)

- Azure Functions
  - Automatically triggered on each new event in your Azure Cosmos DB container's change feed.
  - With the *Azure Functions trigger for Azure Cosmos DB*, you can use the Change Feed Processor's scaling and reliable event detection functionality without the need to maintain any worker infrastructure.
- Change feed processor
  - Part of the Azure Cosmos DB .NET V3 and Java V4 SDKs.
  - Simplifies the process of reading the change feed and distributes the event processing across multiple consumers effectively.
  - Has four main components: the monitored container, the lease container, the compute instance, and the delegate.

30

Explore change feed in Azure Cosmos DB (3 of 3)

Change feed processor example code

```
private static async Task<ChangeFeedProcessor> StartChangeFeedProcessorAsync(
    CosmosClient cosmosClient,
    IConfiguration configuration)
{
    string databaseName = configuration["SourceDatabaseName"];
    string sourceContainerName = configuration["SourceContainerName"];
    string leaseContainerName = configuration["LeasesContainerName"];

    Container leaseContainer = cosmosClient.GetContainer(databaseName, leaseContainerName);
    ChangeFeedProcessor changeFeedProcessor = cosmosClient.GetContainer(databaseName, sourceContainerName)
        .GetChangeFeedProcessorBuilder<ToDoItem>(processorName: "changeFeedSample", onChangesDelegate:
         HandleChangesAsync)
            .WithInstanceName("consoleHost")
            .WithLeaseContainer(leaseContainer)
            .Build();

    Console.WriteLine("Starting Change Feed Processor...");
    await changeFeedProcessor.StartAsync();
    Console.WriteLine("Change Feed Processor started.");
    return changeFeedProcessor;
}
```


31

Summary and knowledge check

In this module, you learned how to:

- Identify classes and methods used to create resources
- Create resources by using the Azure Cosmos DB .NET v3 SDK
- Write stored procedures, triggers, and user-defined functions by using JavaScript
- Implement change feed notifications

1. When defining a stored procedure in the Azure portal input parameters are always sent as what type to the stored procedure?
2. What would one use to validate properties of an item being created?

32

33

Group discussion questions

- What are the benefits of using Azure Cosmos DB? What types of apps would benefit most?
- Describe the five consistency levels. Can you give an example of an application that matches the characteristics of each consistency level?

34

Lab 04: Construct a polyglot data solution

In this lab, you will create an Azure Cosmos DB resource and a storage account resource. Using C# and .NET, you will access the Cosmos DB resource and upload data into it.

Additionally, as Contoso may want to access the data in Cosmos DB through a user-friendly interface, you will implement a .NET solution that accesses and displays the data from Cosmos DB in a web browser.

Finally, you will set the consistency level for your Cosmos DB instance and implement an Azure function for change feed notifications.

http://aka.ms/az204labs

- Exercise 1: Creating data store resources in Azure
- Exercise 2: Review and upload data
- Exercise 3: Configure a .NET web application

35

End of presentation

36

Microsoft

arelance
Building Digital Talent

AZ-204T00A

Learning Path 05: Implement containerized solutions

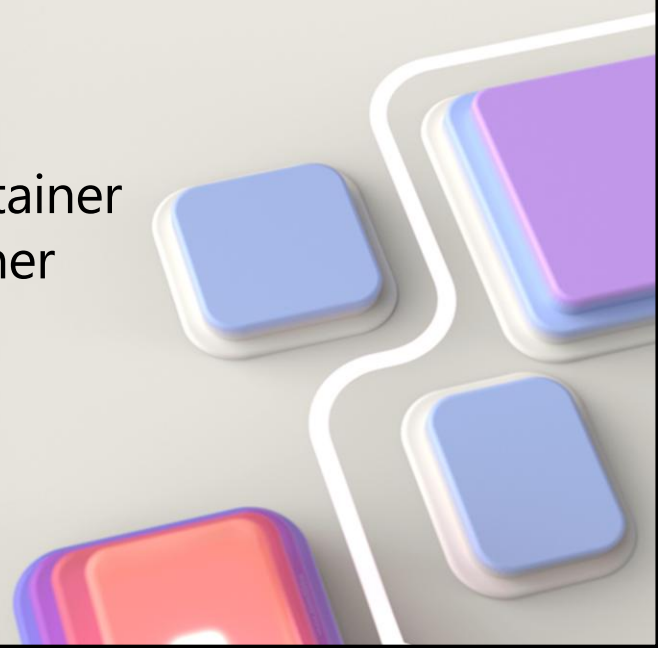
1

Agenda

- Manage container images in Azure Container Registry
- Run container images in Azure Container Instances
- Implement Azure Container Apps

2

Module 1: Manage Container Images in Azure Container Registry



3

Learning objectives

- Explain the features and benefits Azure Container Registry offers
- Describe how to use ACR Tasks to automate builds and deployments
- Explain the elements in a Dockerfile
- Build and run an image in the ACR by using Azure CLI

4

Discover the Azure Container Registry (1 of 2)

Use the Azure Container Registry (ACR) service with your existing container development and deployment pipelines or use Azure Container Registry Tasks to build container images in Azure.

Use cases

Pull images from an Azure container registry to various deployment targets:

- *Scalable orchestration systems* that manage containerized applications across clusters of hosts.
- *Azure services* that support building and running applications at scale.

Azure Container Registry service tiers

Azure Container Registry is available in multiple service tiers.

- Basic
- Standard
- Premium

5

Discover the Azure Container Registry (2 of 2)

Supported images and artifacts

- Grouped in a repository, each image is a read-only snapshot of a Docker-compatible container.
- Azure container registries can include both Windows and Linux images.
- Azure Container Registry also stores Helm charts and images built to the Open Container Initiative (OCI) Image Format Specification.

Azure Container Registry Tasks

- Use Azure Container Registry Tasks (ACR Tasks) to streamline building, testing, pushing, and deploying images in Azure.

6

Explore storage capabilities

Every Basic, Standard, and Premium Azure container registry benefits from advanced Azure storage features.

- **Encryption-at-rest:** All container images in your registry are encrypted at rest.
- **Geo-redundant storage:** Azure uses a geo-redundant storage scheme to guard against loss of your container images.
- **Geo-replication:** For scenarios requiring even more high-availability assurance, consider using the geo-replication feature of Premium registries.
- **Zone redundancy:** Premium service tier, uses Azure availability zones to replicate your registry to a minimum of three separate zones in each enabled region.
- **Scalable storage:** Create as many repositories, images, layers, or tags as you need, up to the registry limit.

7

Build and manage containers with tasks

ACR Tasks provides cloud-based container image building for platforms including Linux, Windows, and ARM. It can automate OS and framework patching for your Docker containers.

Task scenarios

ACR Tasks supports several scenarios to build and maintain container images and other artifacts:

- Quick task
- Automatically triggered tasks
- Multi-step task

8

Explore elements of a Dockerfile (1 of 2)

```
# Step 1: Specify the parent image for the new image
FROM ubuntu:18.04

# Step 2: Update OS packages and install additional software
RUN apt -y update &&  apt install -y wget nginx software-properties-common apt-transport-https \
    && wget -q <URL>/ubuntu/18.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb \
    && dpkg -i packages-microsoft-prod.deb \
    && add-apt-repository universe \
    && apt -y update \
    && apt install -y dotnet-sdk-3.0

# Step 3: Configure Nginx environment
CMD service nginx start

# Step 4: Configure Nginx environment
COPY ./default /etc/nginx/sites-available/default
```


9

Explore elements of a Dockerfile (2 of 2)

```
# STEP 5: Configure work directory
WORKDIR /app

# STEP 6: Copy website code to container
COPY ./website/. .

# STEP 7: Configure network requirements
EXPOSE 80:8080

# STEP 8: Define the entry point of the process that runs in the container
ENTRYPOINT ["dotnet", "website.dll"]
```


10

Exercise: Build and run a container image by using Azure Container Registry Tasks

In this exercise you learn how to ACR Tasks to create a registry and build, push, and run an image in the ACR.

Objectives

- Create an Azure Container Registry
- Build and push image from a Dockerfile
- Verify the results
- Run the image in the ACR
- Clean up resources

11

Summary and knowledge check

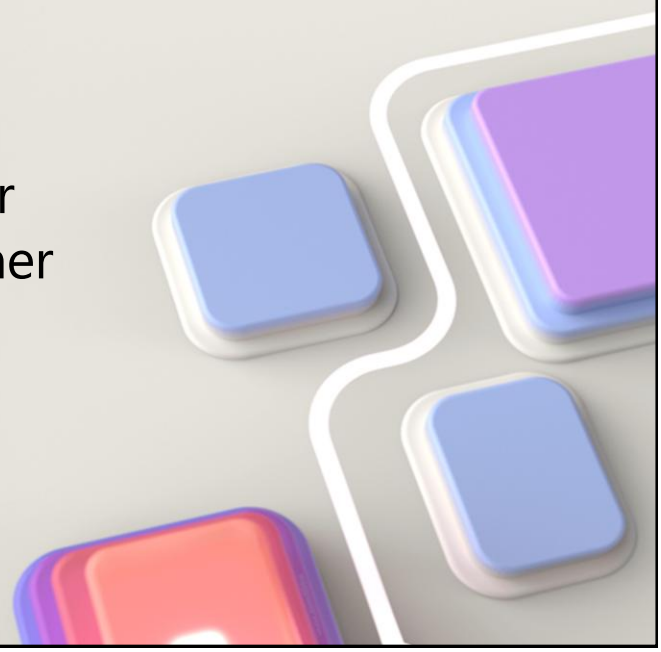
In this module, you learned how to:

- Explain the features and benefits Azure Container Registry offers
- Describe how to use ACR Tasks to automate builds and deployments
- Explain the elements in a Dockerfile
- Build and run an image in the ACR by using Azure CLI

1. Which Azure Container Registry option supports geo-replication to manage a single registry across multiple regions?
2. Which Azure container registry tiers benefit from encryption-at-rest?

12

Module 2: Run container images in Azure Container Instances



13

Learning objectives

- Describe the benefits of Azure Container Instances and how resources are grouped.
- Deploy a container instance in Azure by using the Azure CLI.
- Start and stop containers using policies.
- Set environment variables in your container instances.
- Mount file shares in your container instances.

14

Introduction

- Azure Container Instances (ACI) offers the fastest and simplest way to run a container in Azure.
- No requirement to manage any virtual machines and without having to adopt a higher-level service.

15

Explore Azure Container Instances (1 of 3)

Feature	Description
Fast startup times	Containers can start in seconds without the need to provision and manage VMs
Public IP connectivity and DNS name	Containers can be directly exposed to the internet with an IP address and a fully qualified domain name (FQDN)
Hypervisor-level security	Container applications are as isolated in a container as they would be in a VM
Custom sizes	ACI provides optimum utilization by allowing exact specifications of CPU cores and memory
Persistent storage	Containers support direct mounting of Azure Files shares
Linux and Windows containers	The same API is used to schedule both Linux and Windows containers
Co-scheduled groups	Container Instances supports scheduling of multicontainer groups that share host machine resources
Virtual network deployment	Container Instances can be deployed into an Azure virtual network

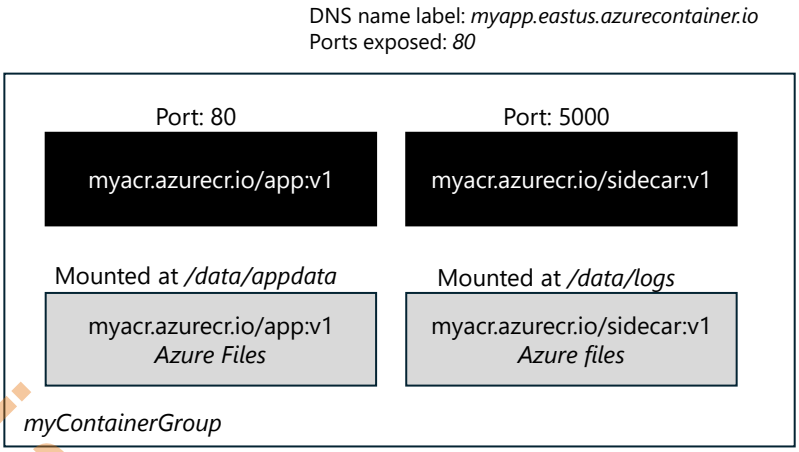
16

Explore Azure Container Instances (2 of 3)

Container groups

The top-level resource in Azure Container Instances is the container group.

The containers in a container group share a lifecycle, resources, local network, and storage volumes.



17

Explore Azure Container Instances (3 of 3)

Deployment

- There are two common ways to deploy a multi-container group: ARM template or a YAML file.

Resource allocation

- ACI allocates resources such as CPUs, memory, and optionally GPUs (preview) to a container group by adding the resource requests of the instances in the group.

Networking

- Container groups share an IP address and a port namespace on that IP address.

Storage

- Specify external volumes to mount within a container group.
- Map those volumes into specific paths within the individual containers in a group.

Common scenarios

- Multi-container groups are useful in cases where you want to divide a single functional task into a small number of container images.

18

Exercise: Deploy a container instance by using the Azure CLI

In this exercise you learn how to use the Azure CLI in the Azure Cloud Shell to create and run a container in Azure Container Instances.

Objectives

- Create the resource group
- Create a container
- Verify the container is running
- Clean up resources

19

Run containerized tasks with restart policies (1 of 2)

Overview

With a configurable restart policy, you can specify that your containers are stopped when their processes have completed.

Container restart policy

When you create a container group in Azure Container Instances, you can specify one of three restart policy settings:

- `Always`
- `Never`
- `OnFailure`

20

Set environment variables in container instances (1 of 2)

YAML example

- Set a secure environment variable by specifying the `secureValue` property instead of the regular value for the variable's type.
- The two variables defined in the YAML demonstrate the two variable types.

```
az container create \
    --resource-group myResourceGroup \
    --name mycontainer2 \
    --image mcr.microsoft.com/azuredocs/aci-wordcount:latest \
    --restart-policy OnFailure \
    --environment-variables 'NumWords'='5' 'MinLength'='8'
```


21

Set environment variables in container instances (2 of 2)

- Provides dynamic configuration of the application or script run by the container.
- ACI supports both Windows and Linux containers to pass secrets as environment variables
- In the example two variables are passed to the container when it is created.

```
az container create \
    --resource-group myResourceGroup \
    --name mycontainer2 \
    --image mcr.microsoft.com/azuredocs/aci-wordcount:latest \
    --restart-policy OnFailure \
    --environment-variables 'NumWords'='5' 'MinLength'='8'
```


22

Mount an Azure file share in Azure Container Instances (1 of 2)

Overview

- By default, Azure Container Instances are stateless. If the container crashes or stops, all of its state is lost.
- To persist state beyond the lifetime of the container, you must mount a volume from an external store.

Limitations

- You can only mount Azure Files shares to Linux containers.
- Azure file share volume mount requires the Linux container run as root.
- Azure File share volume mounts are limited to CIFS support.

23

Mount an Azure file share in Azure Container Instances (2 of 2)

Deploy container and mount volume - YAML

- You can also deploy a container group and mount a volume in a container with the Azure CLI and a YAML template.

Mount multiple volumes

- To mount multiple volumes in a container instance, you must deploy using an Azure Resource Manager template or a YAML file.
- To use a template or YAML file, provide the share details and define the volumes by populating the volumes array in the properties section of the template.

24

Summary and knowledge check

In this module, you learned how to:

- Describe the benefits of Azure Container Instances and how resources are grouped
- Deploy a container instance in Azure by using the Azure CLI
- Start and stop containers using policies
- Set environment variables in your container instances
- Mount file shares in your container instances

1. What method is recommended when deploying a multi-container group that includes only containers?
2. What is the purpose of a restart policy in Azure Container Instances?

25

Module 3: Implement Azure Container Apps

26

Learning objectives

- Describe the features benefits of Azure Container Apps
- Deploy container app in Azure by using the Azure CLI
- Utilize Azure Container Apps built-in authentication and authorization
- Create revisions and implement app secrets

27

Introduction

Azure Container Apps provides the flexibility you need with a serverless container service built for microservice applications and robust autoscaling capabilities without the overhead of managing complex infrastructure.

28

Explore Azure Container Apps

Azure Container Apps enables you to run microservices and containerized applications on a serverless platform that runs on top of Azure Kubernetes Service.

- Supports dynamic scaling based on any KEDA-supported scaler
- Container apps are deployed to a single Container Apps environment, which acts as a secure boundary around groups of container apps.
- Independently develop, upgrade, version, and scale core areas of functionality in an overall system.
- Native Distributed Application Runtime (Dapr) integration

29

Exercise: Deploy a container app

In this exercise you create a secure Container Apps environment and deploy container app.

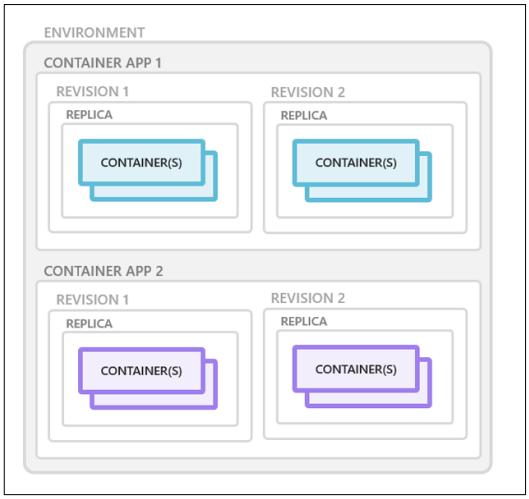
Objectives

- Prepare your environment
- Create an Azure Container Apps environment
- Create a container app
- Verify deployment
- Clean up resources

30

Explore containers in Azure Container Apps

- Containers for an Azure Container App are grouped together in pods inside revision snapshots.
- Can define multiple containers in a single container app to implement the sidecar pattern.
- Deploy images hosted on private registries by providing credentials in the Container Apps configuration.



31

Implement authentication and authorization in Azure Container Apps

- Container Apps uses federated identity, in which a third-party identity provider manages the user identities and authentication flow for you.
  - Available identity providers include: Microsoft Identity Platform, Facebook, GitHub, Google, Twitter, and any OpenID Connect provider.
- The authentication flow is the same for all providers, but differs depending on whether you want to sign in with the provider's SDK.
  - **Without provider SDK** (server-directed flow or server flow): The application delegates federated sign-in to Container Apps. Delegation is typically the case with browser apps, which presents the provider's sign-in page to the user.
  - **With provider SDK** (client-directed flow or client flow): The application signs users in to the provider manually and then submits the authentication token to Container Apps for validation.

32

Manage revisions and secrets in Azure Container Apps

Revisions

- Azure Container Apps implements container app versioning by creating revisions.
- Control which revisions are active, and the external traffic that is routed to each active revision.
- The `az containerapp update` command can modify environment variables, compute resources, scale parameters, and deploy a different image.
- If the update includes revision-scope changes, a new revision is generated.

Secrets

- Secrets are defined at the application level, secured values are available to container apps.
- Each application revision can reference one or more secrets.
- When you create a container app, secrets are defined using the `--secrets` parameter.

33

Explore Dapr integration with Azure Container Apps

The Distributed Application Runtime (Dapr) provides capabilities for enabling application intercommunication, whether through messaging via pub/sub or reliable and secure service-to-service calls.

Dapr API	Description
Service-to-service invocation	Discover services and perform reliable, direct service-to-service calls with automatic mTLS authentication and encryption.
State management	Provides state management capabilities for transactions and CRUD operations.
Pub/sub	Allows publisher and subscriber container apps to intercommunicate via an intermediary message broker.
Bindings	Trigger your applications based on events.
Actors	Dapr actors are message-driven, single-threaded, units of work designed to quickly scale.
Observability	Send tracing information to an Application Insights backend.
Secrets	Access secrets from your application code or reference secure values in your Dapr components.
Configuration	Access app configuration and be notified of updates.

34

Summary and knowledge check

In this module, you learned how to:

- Describe the features benefits of Azure Container Apps
- Deploy container app in Azure by using the Azure CLI
- Utilize Azure Container Apps built-in authentication and authorization
- Create revisions and implement app secrets

1 What is a revision in Azure Container Apps?

35

Discussion and lab

36

Group discussion questions

- What factors should you consider when deciding between Azure Container Instances and Azure Container Apps as a deployment target?
- Describe the architecture of an Azure Container App environment. How could some of your apps work within that architecture?
- Describe at least two elements of a Dockerfile. What tools would you use to create a container image?

37

Lab 05: Deploy compute workloads by using images and containers

In this lab, you will explore how to create and deploy containers to the Azure Container Registry using a .NET application and docker files. And also deploy a containerized solution to Azure Container Apps.

http://aka.ms/az204labs

- Exercise 1: Create a Docker container image and deploy it to Azure Container Registry
- Exercise 2: Deploy an Azure container instance
- Exercise 3: Create a secure Container Apps environment and deploy container app

38

39

Microsoft

arelance
Building Digital Talent

AZ-204T00A

Learning Path 06: Implement user authentication and authorization

1

Agenda

- Explore the Microsoft identity platform
- Implement authentication by using the Microsoft Authentication Library
- Implement shared access signatures
- Explore Microsoft Graph

2

Module 1: Explore the Microsoft identity platform

3

Learning objectives

- Identify the components of the Microsoft identity platform
- Describe the three types of service principals and how they relate to application objects
- Explain how permissions and user consent operate, and how conditional access impacts your application

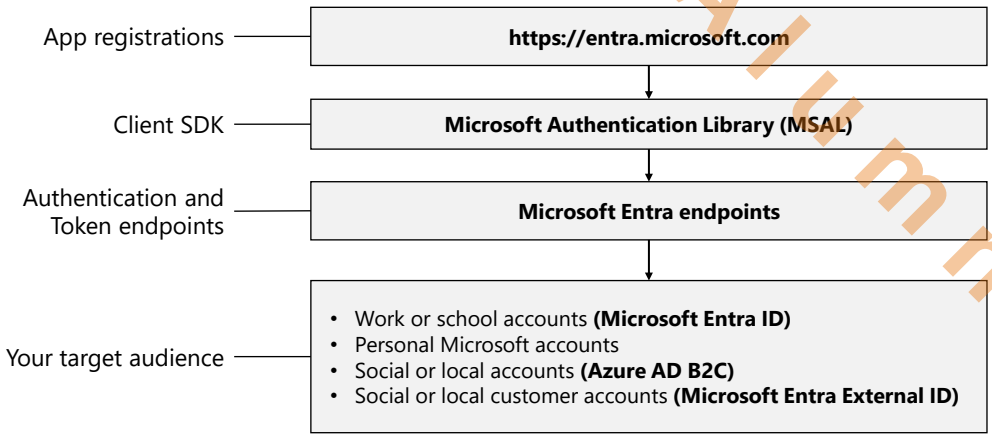
4

Introduction

- The Microsoft identity platform is a set of tools that includes authentication service, open-source libraries, and application management tools.
- Helps you build applications your users and customers can sign in to using their Microsoft identities or social accounts.

5

Explore the Microsoft identity platform



6

Explore service principals

When you register an app in the Azure portal, you choose whether it is:

- Single tenant: only accessible in your tenant
- Multi-tenant: accessible in other tenants

If you register an application in the portal, below objects are automatically created in your home tenant:

- Application object
- Service principal object - Application, Managed identity, Legacy

Relationship between application objects and service principals

An application object has:

- 1:1 relationship with the software application, and
- 1:many relationship with its corresponding service principal object

7

Discover permissions and consent (1 of 2)

Permission types

The Microsoft identity platform supports two types of permissions:

- *Delegated permissions* are used by apps that have a signed-in user present.
- *App-only access permissions* are used by apps that run without a signed-in user present

Consent types

There are three consent types:

- Static user consent
- Incremental and Dynamic user consent
- Admin consent

8

Discover permissions and consent (2 of 2)

In an OpenID Connect or OAuth 2.0 authorization request, an app can request the permissions it needs by using the scope query parameter.

```
GET https://login.microsoftonline.com/common/oauth2/v2.0/authorize?
client_id=6731de76-14a6-49ae-97bc-6eba6914391e
&response_type=code
&redirect_uri=http%3A%2F%2Flocalhost%2Fmyapp%2F
&response_mode=query
&scope=
   https%3A%2F%2Fgraph.microsoft.com%2Fcalendars.read%20
   https%3A%2F%2Fgraph.microsoft.com%2Fmail.send
&state=12345
```


9

Discover conditional access

Conditional Access enables developers to protect services in a multitude of ways including

- Multifactor authentication
- Allowing only Intune enrolled devices to access specific services
- Restricting user locations and IP ranges

Conditional Access impact on an app

Specifically, the following scenarios require code to handle Conditional Access challenges:

- Apps performing the on-behalf-of flow
- Apps accessing multiple services/resources
- Single-page apps using MSAL.js
- Web apps calling a resource

10

Summary and knowledge check

In this module, you learned how to:

- Identify the components of the Microsoft identity platform
- Describe the three types of service principals and how they relate to application objects
- Explain how permissions and user consent operate, and how conditional access impacts your application

1. Which of the types of permissions supported by the Microsoft identity platform is used by apps that have a signed-in user present?
2. What app scenarios require code to handle Conditional Access challenges?

11

Module 2: Implement authentication by using the Microsoft Authentication Library

12

Learning objectives

- Explain the benefits of using MSAL and the application types and scenarios it supports
- Instantiate both public and confidential client apps from code
- Register an app with the Microsoft identity platform
- Create an app that retrieves a token by using the MSAL.NET library

13

Introduction

- The Microsoft Authentication Library (MSAL) enables developers to acquire tokens from the Microsoft identity platform to authenticate users and access secured web APIs.

14

Explore the Microsoft Authentication Library (1 of 2)

- The Microsoft Authentication Library (MSAL) can be used to provide secure access to Microsoft Graph, other Microsoft APIs, third-party web APIs, or your own web API.
- MSAL supports many different application architectures and platforms including .NET, JavaScript, Java, Python, Android, and iOS.
- Application types and scenarios:
  - web applications
  - web APIs
  - single-page apps (JavaScript)
  - mobile and native applications
  - daemons and server-side applications

15

Explore the Microsoft Authentication Library (2 of 2)

Authentication flows

Flow	Description
Authorization code	Native and web apps securely obtain tokens in the name of the user.
Client credentials	Access to web APIs by using the identity of the application itself.
On-behalf-of	Access from an "upstream" web API to a "downstream" web API on behalf of the user.
Implicit grant	User sign-in and access to web APIs on behalf of the user. (No longer recommended.)
Device code	Enables sign-in to a device by using another device that has a browser.
Integrated Windows	Allows applications on domain or Microsoft Entra joined computers to acquire a token silently (without any UI interaction from the user).
Username/password	The application signs in a user by using their username and password. (Not recommended.)

16

Initialize client applications (1 of 2)

With MSAL.NET 3.x, the recommended way to instantiate an application is by using the `PublicClientApplicationBuilder` and `ConfidentialClientApplicationBuilder` application builders.

```
//Public client initialization

IPublicClientApplication app = PublicClientApplicationBuilder.Create(clientId).Build();


//Confidential client initialization

string redirectUri = "https://myapp.azurewebsites.net";
IConfidentialClientApplication app = ConfidentialClientApplicationBuilder.Create(clientId)
    .WithClientSecret(clientSecret)
    .WithRedirectUri(redirectUri )
    .Build();
```


17

Initialize client applications (2 of 2)

Modifier	Description
`.WithAuthority() 7 overrides`	Sets the application default authority to a Microsoft Entra authority, with the possibility of choosing the Azure Cloud, the audience, the tenant (tenant ID or domain name), or providing directly the authority URI.
`.WithTenantId(string tenantId)`	Overrides the tenant ID, or the tenant description.
`.WithClientId(string)`	Overrides the client ID.
`.WithRedirectUri(string redirectUri)`	Overrides the default redirect URI. In the case of public client applications, this will be useful for scenarios requiring a broker.
`.WithComponent(string)`	Sets the name of the library using MSAL.NET (for telemetry reasons).
`.WithDebugLoggingCallback()`	If called, the application will call `Debug.Write` simply enabling debugging traces.
`.WithLogging()`	If called, the application will call a callback with debugging traces.
`.WithTelemetry(TelemetryCallback telemetryCallback)`	Sets the delegate used to send telemetry.

18

Exercise: Implement interactive authentication by using MSAL.NET

In this exercise you learn how to use MSAL.NET to acquire a token interactively in a console application.

Objectives

- Register a new application
- Build the console app
- Review completed app
- Run the application to retrieve the token
- Clean up resources

19

Summary and knowledge check

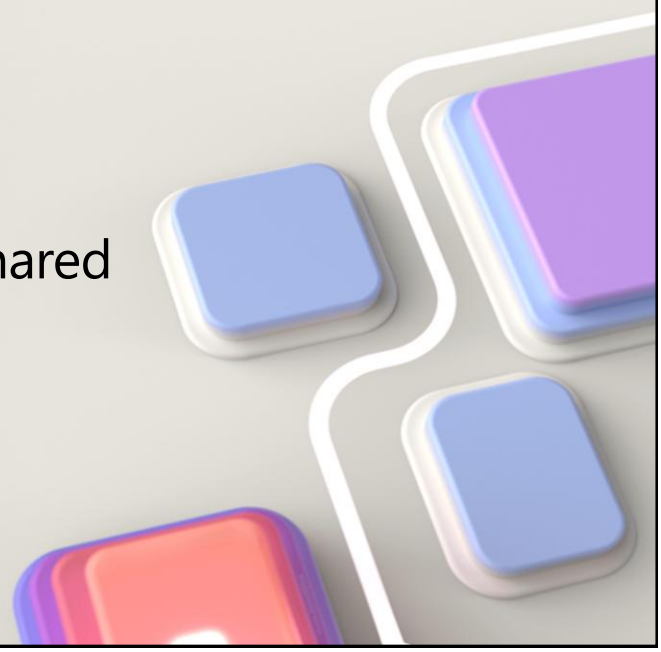
In this module, you learned how to:

- Explain the benefits of using MSAL and the application types and scenarios it supports
- Instantiate both public and confidential client apps from code
- Register an app with the Microsoft identity platform
- Create an app that retrieves a token by using the MSAL.NET

1. Which MSAL library supports single-page web apps?
2. What is the purpose of using `PublicClientApplicationBuilder` class in MSAL.NET?

20

Module 3: Implement shared access signatures



21

Learning objectives

- Identify the three types of shared access signatures
- Explain when to implement shared access signatures
- Create a stored access policy

22

Introduction

- A shared access signature (SAS) is a signed URI that points to one or more storage resources and includes a token that contains a special set of query parameters.
- The token indicates how the resources may be accessed by the client.

23

Discover shared access signatures (1 of 2)

Types of shared access signatures

- User delegation SAS
- Service SAS
- Account SAS

Best practices

- Always use HTTPS
- The most secure SAS is a user delegation SAS
- Set expiration time to smallest useful time
- Apply the rule of minimum-required privileges
- SAS isn't always the correct solution

24

Discover shared access signatures (2 of 2)

Use SAS to access data

- **URI:** `https://medicalrecords.blob.core.windows.net/patient-images/patient-116139-nq8z7f.jpg?`
- **SAS token:** `sp=r&st=2020-01-20T11:42:32Z&se=2020-01-20T19:42:32Z&spr=https&sv=2019-02-02&sr=b&sig=<...>`

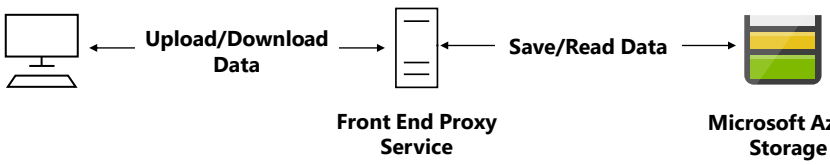
Component	Description
`sp=r`	Controls the access rights. The values can be **a** for add, **c** for create, **d** for delete, **l** for list, **r** for read, or **w** for write.
`st=2020-01-20T11:42:32Z`	The date and time when access starts.
`se=2020-01-20T19:42:32Z`	The date and time when access ends. This example grants eight hours of access.
`sv=2019-02-02`	The version of the storage API to use.
`sr=b`	The kind of storage being accessed. In this example, b is for blob.
`sig=<...>`	The cryptographic signature.

25

Choose when to use shared access signatures (1 of 2)

In a scenario where a storage account stores user data, there are two typical design patterns:

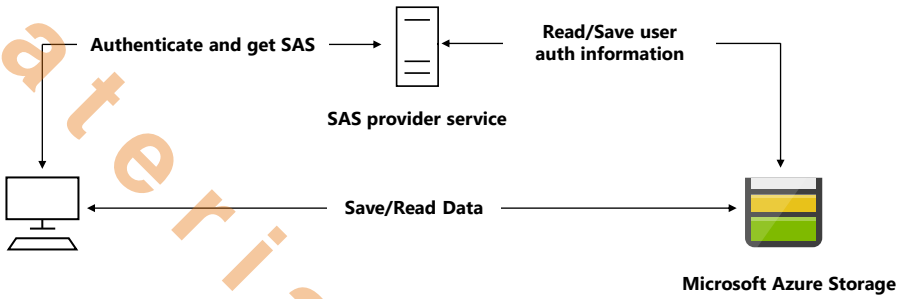
- Clients upload and download data via a front-end proxy service, which performs authentication. This front-end proxy service has the advantage of allowing validation of business rules, but for large amounts of data or high-volume transactions, creating a service that can scale to match demand may be expensive or difficult.



26

Choose when to use shared access signatures (2 of 2)

- A lightweight service authenticates the client as needed and then generates a SAS. Once the client application receives the SAS, they can access storage account resources directly with the permissions defined by the SAS and for the interval allowed by the SAS. The SAS mitigates the need for routing all data through the front-end proxy service.



27

Explore stored access policies

- A stored access policy provides an additional level of control over service-level shared access signatures (SAS) on the server side.
- To create or modify a stored access policy, call the **Set ACL** operation for the resource with a request body that specifies the terms of the access policy.

```
BlobSignedIdentifier identifier = new
BlobSignedIdentifier
{
    Id = "stored access policy identifier",
    AccessPolicy = new BlobAccessPolicy
    {
        ExpiresOn = DateTimeOffset.UtcNow.AddHours(1),
        Permissions = "rw"
    }
};
```

```
az storage container policy create \
    --name <stored access policy identifier> \
    --container-name <container name> \
    --start <start time UTC datetime> \
    --expiry <expiry time UTC datetime> \
    --permissions <(a),(c),(d),(l),(r),(w) \
    --account-key <storage account key> \
    --account-name <storage account name>
```


28

Summary and knowledge check

In this module, you learned how to:

- Identify the three types of shared access signatures
- Explain when to implement shared access signatures
- Create a stored access policy

1. What type of shared access signatures (SAS) applies to Blob storage only?
2. What is the most flexible and secure way to use a service or account shared access signature (SAS)?

29

Module 4: Explore Microsoft Graph

30

Learning objectives

- Explain the benefits of using Microsoft Graph
- Perform operations on Microsoft Graph by using REST and SDKs
- Apply best practices to help your applications get the most out of Microsoft Graph

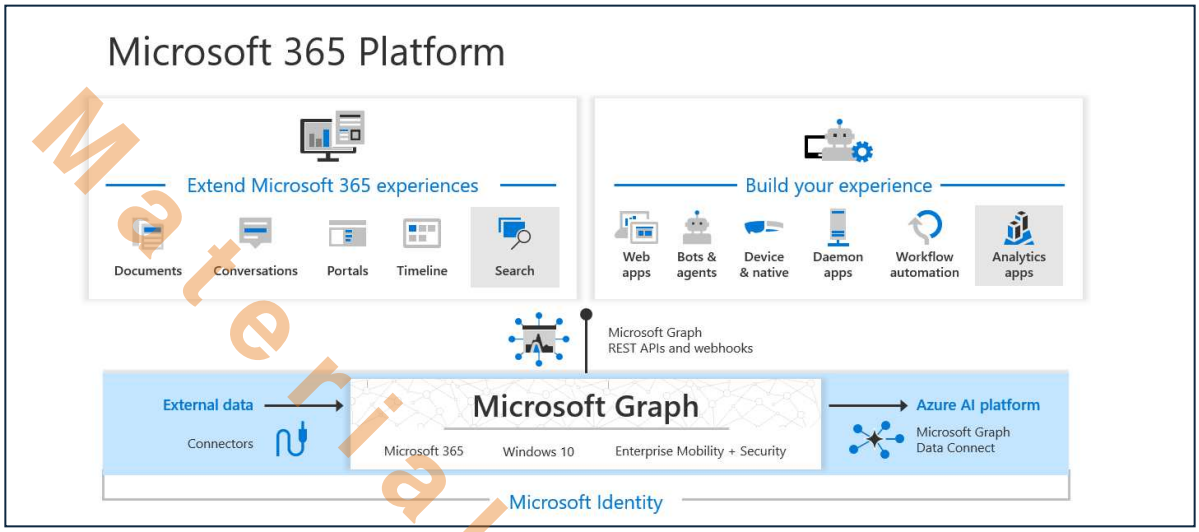
31

Introduction

- Microsoft Graph is the gateway to data and intelligence in Microsoft 365.
- It provides a unified programmability model that you can use to access the tremendous amount of data in Microsoft 365, Windows, and Enterprise Mobility + Security.

32

Discover Microsoft Graph



33

Query Microsoft Graph by using REST

Call a REST API method

- {HTTP method} `https://graph.microsoft.com/{version}/{resource}?{query-parameters}`
- HTTP methods (`GET`, `POST`, `PATCH`, `PUT`, `DELETE`)

{version}: Microsoft Graph currently supports two versions `v1.0` and `beta`

{resource}: A resource can be an entity or complex type, commonly defined with properties.

{query-parameters}: Query parameters can be OData system query options, or other strings that a method accepts to customize its response.

34

Query Microsoft Graph by using SDKs (1 of 4)

Microsoft.Graph

- Object-relational mapping tool for Microsoft Graph
- Contains classes mapped to the RESTful syntax of the Microsoft Graph API

Microsoft.Graph.Core

- Core library for making calls to Microsoft Graph

Authentication

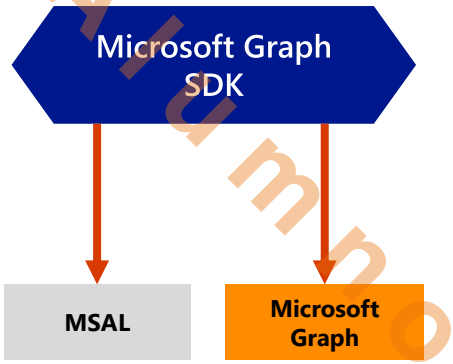
- Providers to integrate the Microsoft Graph SDK with the MSAL application builders
- Supports various authentication flows

35

Query Microsoft Graph by using SDKs (2 of 4)

Wrapper for the MSAL library:

- Supplies authentication provider helpers
- Uses MSAL "under the hood"
- Helpers automatically acquire tokens on your behalf
- Reduces the complexity of using Microsoft Graph in your application



36

Query Microsoft Graph by using SDKs (3 of 4)

Create a Microsoft Graph client (example)

```
// Build a client application.
IPublicClientApplication publicClientApplication = PublicClientApplicationBuilder
        .Create("INSERT-CLIENT-APP-ID")
        .Build();

// Create an authentication provider by passing in a client application and graph scopes.
DeviceCodeProvider authProvider = new DeviceCodeProvider(publicClientApplication, graphScopes);

// Create a new instance of GraphServiceClient with the authentication provider.
GraphServiceClient graphClient = new GraphServiceClient(authProvider);
```


37

Query Microsoft Graph by using SDKs (4 of 4)

Retrieve a list of entities (example)

```
// GET https://graph.microsoft.com/v1.0/me/messages?$select=subject,sender&$filter=<some
condition>&orderBy=receivedDateTime

var messages = await graphClient.Me.Messages
    .Request()
    .Select(m => new {
        m.Subject,
        m.Sender
    })
    .Filter("<filter condition>")
    .OrderBy("receivedDateTime")
    .GetAsync();
```


38

Apply best practices to Microsoft Graph

Authentication

- The HTTP Authorization request header, as a Bearer token
- The graph client constructor, when using a Microsoft Graph client library

Consent and authorization

- Use least privilege
- Use the correct permission type based on scenarios
- Consider the end user and admin experience
- Consider the end user and admin

Handle responses effectively

- Pagination
- Evolvable enumerations

Storing data locally

- Only cache or store data locally if necessary for a specific scenario
- Your application should also implement proper retention and deletion policies

39

Summary and knowledge check

In this module, you learned how to:

- Explain the benefits of using Microsoft Graph
- Perform operations on Microsoft Graph by using REST and SDKs
- Apply best practices to help your applications get the most out of Microsoft Graph

1. Which HTTP method is used to update a resource with new values?
2. Which of the components of the Microsoft 365 platform is used to deliver data external to Azure into Microsoft Graph services and applications?

40

41

Group discussion questions

- What are service principals and how can they be used?
- Describe a scenario where you would want to use conditional access? What impact on application code would that have?
- Describe some scenarios where it would be recommended to use a shared access signature.

42

Lab 06: Authenticate by using OpenID Connect, MSAL, and .NET SDKs

In this lab, you will register an application in Microsoft Entra ID, add a user, and then test the user's access to the application to validate that Entra ID can secure it. You will also use the Graph Explorer tool to build and test requests against the Graph API for an Entra ID user account.

http://aka.ms/az204labs

- Exercise 1: Configure a single-tenant Entra ID environment
- Exercise 2: Create a single-tenant ASP.NET web app

43

End of presentation

44

Microsoft

arelance
Building Digital Talent

AZ-204T00A

Learning Path 07: Implement secure cloud solutions

1

Agenda

- Implement Azure Key Vault
- Implement managed identities
- Implement Azure App Configuration

2

Module 1: Implement Azure Key Vault

3

Learning objectives

- Describe the benefits of using Azure Key Vault
- Explain how to authenticate to Azure Key Vault
- Set and retrieve a secret from Azure Key Vault by using the Azure CLI

4

Introduction

- Azure Key Vault is a cloud service for securely storing and accessing secrets.
- A secret is anything that you want to tightly control access to, such as API keys, passwords, certificates, or cryptographic keys.

5

Explore Azure Key Vault

Azure Key Vault provides

- **Secrets Management:** For safe storage and strict control
- **Key Management:** Used as a key management solution
- **Certificate Management:** Provision, manage, and deploy public and private SSL/TLS certificates

Key benefits of using Azure Key Vault

- **Centralized application secrets:** Control their distribution.
- **Securely store secrets and keys:** Requires proper authentication and authorization to gain access
- **Monitor access and use:** Enable logging to monitor activity.
- **Simplified administration of application secrets:** Follow the life cycle and have high availability.

6

Discover Azure Key Vault best practices (1 of 2)

Authenticating to Azure Key Vault

Three ways to authenticate

- **Managed identities for Azure resources**: When you deploy an app on a virtual machine in Azure, you can assign an identity to your virtual machine that has access to Key Vault.
- **Service principal and certificate**: You can use a service principal and an associated certificate that has access to Key Vault.
- **Service principal and secret**: Although you can use a service principal and a secret to authenticate to Key Vault, we don't recommend it.

7

Discover Azure Key Vault best practices (2 of 2)

Best practices

- **Use separate key vaults:** Recommended to use a vault per application per environment.
- **Control access to your vault:** Key Vault data is sensitive and business critical.
- **Backup:** Create regular back ups of your vault on update/delete/create of objects within a Vault.
- **Logging:** Be sure to turn on logging and alerts.
- **Recovery options**: Turn on soft-delete and purge protection if you want to guard against force deletion of the secret.

8

Authenticate to Azure Key Vault (1 of 2)

Two ways to obtain a service principal

- Enable a system-assigned managed identity for the application. deployed to a variety of services.
- If you cannot use managed identity, you instead register the application with your Azure AD tenant.

Note: It is recommended to use a system-assigned managed identity.

9

Authenticate to Azure Key Vault (2 of 2)

Authentication to Key Vault in application code

Key Vault SDK is using Azure Identity client library, which allows seamless authentication to Key Vault across environments.

.NET	Python	Java	JavaScript
Azure Identity SDK .NET	Azure Identity SDK Python	Azure Identity SDK Java	Azure Identity SDK JavaScript

Authentication to Key Vault with REST

Access tokens must be sent to the service using the HTTP Authorization header:

```
PUT /keys/MYKEY?api-version=<api_version> HTTP/1.1
Authorization: Bearer <access_token>
```


10

Exercise: Set and retrieve a secret from Azure Key Vault by using Azure CLI

In this exercise you learn how to use Azure CLI to create Azure Key Vault resources and create and retrieve a key.

Objectives

- Create a Key Vault
- Add and retrieve a secret
- Clean up resources

11

Summary and knowledge check

In this module, you learned how to:

- Describe the benefits of using Azure Key Vault
- Explain how to authenticate to Azure Key Vault
- Set and retrieve a secret from Azure Key Vault by using the Azure CLI

1. What method of authenticating to Azure Key Vault is recommended for most scenarios?

2. Azure Key Vault protects data when it's traveling between Azure Key Vault and clients. What protocol does it use for encryption?

12

Module 2: Implement managed identities

13

Learning objectives

- Explain the differences between the two types of managed identities
- Describe the flows for user- and system-assigned managed identities
- Configure managed identities
- Acquire access tokens by using REST and code

14

Introduction

- A common challenge for developers is the management of secrets and credentials used to secure communication between different components making up a solution.
- Managed identities eliminate the need for developers to manage credentials.

15

Explore managed identities (1 of 2)

Types of managed identities

- A system-assigned managed identity is enabled directly on an Azure service instance.
- A user-assigned managed identity is created as a standalone Azure resource.

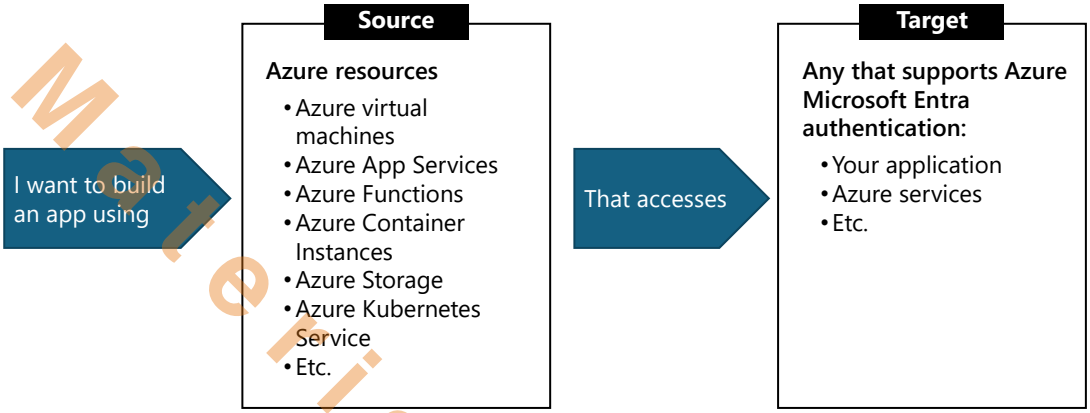
Characteristics of managed identities

Characteristic	System-assigned managed identity	User-assigned managed identity
Creation	Created as part of an Azure resource (for example, an Azure virtual machine or Azure App Service)	Created as a stand-alone Azure resource
Lifecycle	Shared lifecycle with the Azure resource that the managed identity is created with. When the parent resource is deleted, the managed identity is deleted as well.	Independent life-cycle. Must be explicitly deleted.
Sharing across Azure resources	Cannot be shared, it can only be associated with a single Azure resource.	Can be shared, the same user-assigned managed identity can be associated with more than one Azure resource.

16

Explore managed identities (2 of 2)

When to use managed identities



17

Discover the managed identities authentication flow

How a system-assigned managed identity works with an Azure virtual machine:

1. Azure Resource Manager (ARM) receives a request to enable the system-assigned managed identity on a virtual machine.
2. Azure Resource Manager creates a service principal in Microsoft Entra ID for the identity of the virtual machine.
3. ARM configures the identity on the virtual machine by updating the Azure Instance Metadata Service identity endpoint with the service principal client ID and certificate.
4. After the virtual machine has an identity, use the service principal information to grant the virtual machine access to Azure resources.
5. Your code that's running on the virtual machine can request a token from the Azure Instance Metadata service endpoint, accessible only from within the virtual machine.
6. A call is made to Microsoft Entra ID to request an access token by using the client ID and certificate configured. Azure Active Directory returns a JSON Web Token access token.
7. Your code sends the access token on a call to a service that supports Microsoft Entra authentication.

18

Configure managed identities (1 of 3)

System-assigned managed identity:

- To create, or enable, an Azure virtual machine with the system-assigned managed identity your account needs the Virtual Machine Contributor role assignment.
- No additional Microsoft Entra directory role assignments are required.

User-assigned managed identity:

- **Create the user-assigned identity**: Create a user-assigned managed identity using `az identity create`.
- **Assign the identity to a virtual machine:** Creates a virtual machine associated with the new user-assigned identity.

19

Configure managed identities (2 of 3)

Enable system-assigned identity during resource creation

```
az vm create --resource-group myResourceGroup \
    --name myVM --image win2016datacenter \
    --generate-ssh-keys \
    --assign-identity \
    --admin-username azureuser \
    --admin-password myPassword12
```

Enable user-assigned identity during resource creation

```
# Create the identity
az identity create -g myResourceGroup \
    -n myUserAssignedIdentity

# Assign identity during creation
az vm create \
    --resource-group <RESOURCE GROUP> \
    --name <VM NAME> \
    --image UbuntuLTS \
    --admin-username azureuser \
    --admin-password myPassword12 \
    --assign-identity <USER ASSIGNED IDENTITY NAME>
```


20

Configure managed identities (3 of 3)

Azure SDKs with managed identities for Azure resources support:

- Azure supports multiple programming platforms through a series of Azure SDKs.
- Several of them have been updated to support managed identities for Azure resources.

SDK	Sample
.NET	Manage resource from a virtual machine enabled with managed identities for Azure resources enabled
Java	Manage storage from a virtual machine enabled with managed identities for Azure resources
Node.js	Create a virtual machine with system-assigned managed identity enabled
Python	Create a virtual machine with system-assigned managed identity enabled
Ruby	Create Azure virtual machine with an system-assigned identity enabled

21

Acquire an access token

- A client application can request managed identities for Azure resources app-only access token for accessing a given resource.
- The token is based on the managed identities for Azure resources service principal.
- Recommended to use the `DefaultAzureCredential`

```
// When deployed to an azure host, the default azure credential will authenticate the
specified user assigned managed identity.

string userAssignedClientId = "<your managed identity client Id>";
var credential = new DefaultAzureCredential(new DefaultAzureCredentialOptions {
       ManagedIdentityClientId = userAssignedClientId });
var blobClient = new BlobClient(new Uri("URI"), credential);
```


22

Summary and knowledge check

In this module, you learned how to:

- Explain the differences between the two types of managed identities
- Describe the flows for user- and system-assigned managed identities
- Configure managed identities
- Acquire access tokens by using REST and code

1 A client app requests managed identities for an access token for a given resource. What is is the basis for the token?

23

Module 3: Implement Azure App Configuration

24

Learning objectives

- Explain the benefits of using Azure App Configuration
- Describe how Azure App Configuration stores information
- Implement feature management
- Securely access your app configuration information

25

Introduction

- Azure App Configuration provides a service to centrally manage application settings and feature flags.
- Programs running in a cloud, generally have many components that are distributed in nature.
- Spreading configuration settings across these components can lead to hard-to-troubleshoot errors during an application deployment.
- Use App Configuration to store all the settings for your application and secure their access in one place.

26

Explore the Azure App Configuration service (1 of 2)

Application configuration provides benefits:

- A fully managed service that can be set up in minutes
- Flexible key representations and mappings
- Tagging with labels
- Point-in-time replay of settings
- Dedicated UI for feature flag management
- Comparison of two sets of configurations on custom-defined dimensions
- Enhanced security through Azure-managed identities
- Complete data encryptions, at rest or in transit
- Native integration with popular frameworks

App Configuration implementation scenarios:

- Centralize management and distribution of hierarchical configuration data for different environments and geographies
- Dynamically change application settings without the need to redeploy or restart an application
- Control feature availability in real-time

27

Explore the Azure App Configuration service (2 of 2)

The easiest way to add an App Configuration store to your application is through a client library that Microsoft provides.

Programming language and framework	How to connect
.NET Core and ASP.NET Core	App Configuration provider for .NET Core
.NET Framework and ASP.NET	App Configuration builder for .NET
Java Spring	App Configuration client for Spring Cloud
JavaScript/Node.js	App Configuration client for JavaScript
Python	App Configuration client for Python
Others	App Configuration REST API

28

Create paired keys and values

Keys

- **Design key namespaces** – There are two general approaches to naming keys used for configuration data: flat or hierarchical.
- **Label keys** – Key values in App Configuration can optionally have a label attribute.
- **Version key values** – App Configuration doesn't version key values automatically as they're modified.
- **Query key values** – Each key value is uniquely identified by its key plus a label that can be null.

Values

- Values assigned to keys are also unicode strings.
- There's an optional user-defined content type associated with each value.
- Configuration data stored in an App Configuration store, which includes all keys and values, is encrypted at rest and in transit.

29

Manage application features (1 of 2)

Feature management is a modern software-development practice that decouples feature release from code deployment and enables quick changes to feature availability on demand.

Basic concepts	
Feature flag: A feature flag is a variable with a binary state of on or off. **Feature manager:** A feature manager is an application package that handles the lifecycle of all the feature flags in an application. **Filter:** A filter is a rule for evaluating the state of a feature flag.	**Components that implement effective feature management** • An application that makes use of feature flags. • A separate repository that stores the feature flags and their current states.

30

Manage application features (2 of 2)

Feature flag declaration

- Each feature flag has two parts: a name, and a list of one or more filters that are used to evaluate if a feature's state is on.
- When a feature flag has multiple filters, the filter list is traversed in order until one of the filters determines the feature should be enabled.
- The feature manager supports *appsettings.json* as a configuration source for feature flags.

Feature flag repository

- To use feature flags effectively, you need to externalize all the feature flags used in an application.
- Azure App Configuration is designed to be a centralized repository for feature flags.

31

Secure app configuration data (1 of 2)

Encrypt configuration data by using customer-managed keys

Enable customer-managed key capability:

- Standard tier Azure App Configuration instance
- Azure Key Vault with soft-delete and purge-protection features enabled
- An RSA or RSA-HSM key within the Key Vault

Allow Azure application configuration to use Key Vault keys:

- Assign a managed identity to the Azure App Configuration instance
- Grant the identity permissions in the target Key Vault's access policy.

32

Secure app configuration data (2 of 2)

Private endpoints and managed identities

Use private endpoints for Azure App Configuration

- Secure your application configuration details by configuring the firewall to block all connections to App Configuration on the public endpoint.
- Increase security for the virtual network (VNet) ensuring data doesn't escape from the VNet.
- Securely connect to the App Configuration store from on-premises networks that connect to the VNet using VPN or ExpressRoutes with private-peering.

Managed identities

- A system-assigned identity is tied to your configuration store. It's deleted if your configuration store is deleted. A configuration store can only have one system-assigned identity.
- A user-assigned identity is a standalone Azure resource that can be assigned to your configuration store. A configuration store can have multiple user-assigned identities.

33

Summary and knowledge check

In this module, you learned how to:

- Explain the benefits of using Azure App Configuration
- Describe how Azure App Configuration stores information
- Implement feature management
- Securely access your app configuration information

1. Which type of encryption does Azure App Configuration use to encrypt data at rest?
2. Which option evaluates the state of a feature flag?

34

Discussion and lab

35

Group discussion questions

- Contoso Inc has a web app where each developer has contributor access. What can be done to prevent developers from accessing application secrets?
- Managed identities are generally recommended to handle authentication between Azure resources in a solution. In what situations would you want to use a different authentication solution?
- Contoso Inc wants to perform A/B testing on a new feature for their app. What service(s) should they use and what kind of changes do they need to make to their code?

36

Lab 07: Access resource secrets more securely across services

In this lab, you will create a storage account and an Azure Function app that will access the storage account. To demonstrate the secure storage of connection string information, you will provision a Key Vault resource and manage the appropriate secrets to store the connection string information. You will also manage the service identity to gain secure access to the connection string information for the storage account.

http://aka.ms/az204labs

- Exercise 1: Create Azure resources
- Exercise 2: Configure secrets and identities
- Exercise 3: Build an Azure Functions app
- Exercise 4: Access Azure Blob Storage data

37

End of presentation

38

Microsoft

arelance
Building Digital Talent

AZ-204T00A

Learning Path 08: Implement API Management

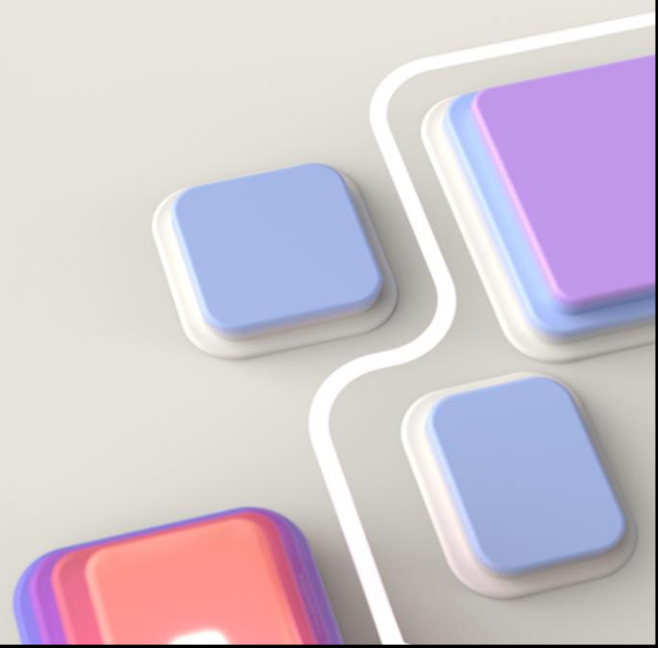
1

Agenda

- Explore API Management

2

Module 1: Explore API Management



3

Learning objectives

- Describe the components, and their function, of the API Management service.
- Explain how API gateways can help manage calls to your APIs.
- Secure access to APIs by using subscriptions and certificates.
- Create a backend API

4

Introduction

- API Management helps organizations publish APIs to external, partner, and internal developers to unlock the potential of their data and services.

5

Discover the API Management service (1 of 2)

The role of API management

- API management provides core functions to ensure a successful API program through developer participation, business insight, analysis, security, and protection.
- Each API consists of one or more operations, and each API can be added to one or more products.

The system is made up of the following components:

- API gateway
- Azure portal
- The Developer portal

6

Discover the API Management service (2 of 2)

Products
Products are how APIs are surfaced to developers.

Groups
Groups are used to manage the visibility of products to developers.

Developers
Developers represent the user accounts in an API Management service instance.

Policies
Allow the Azure portal to change the behavior of the API through configuration.

Developer portal
You can learn about your API, view and call operations, and subscribe to products.

7

Explore API gateways (1 of 2)

API gateway role

- An API gateway sits between clients and services.
- It acts as a reverse proxy, routing requests from clients to services.
- Applies policies and collects telemetry.
- Can also perform tasks such as authentication, SSL termination, and rate limiting.

8

Explore API gateways (2 of 2)

Potential issues when deploying an API without a gateway

- Lead to complex client code.
- Create coupling between the client and the backend.
- A single operation may need to call multiple services.
- Every public-facing service must be dealt with.
- The service must expose a client-friendly protocol.

Functional design patterns

- Gateway routing: Use a gateway as a reverse proxy to route requests to one or more back-end services
- Gateway aggregation: Use a gateway to aggregate multiple individual requests into one request.
- Gateway Offloading: Use the gateway to offload functionality from individual services to the gateway, particularly cross-cutting concerns.

9

Explore API Management policies (1 of 2)

The role of policies

- Allows the publisher to change the behavior of the API through configuration.
- A collection of statements that are executed sequentially in response to requests or responses to the API.
- The policy is applied in the gateway between the API consumer and the managed API.
- Policies can apply changes to inbound requests and outbound responses.

Policy configuration

- The policy definition is a simple XML document that describes a series of inbound and outbound statements.
- The configuration is divided into inbound, backend, outbound and error.
- If an error occurs during the processing of the request, any remaining steps will be skipped, and the statement that jumps to the error section will be executed.

10

Explore API Management policies (2 of 2)

Policy examples

```
<policies>
  <inbound>
    <!-- statements to be applied to the
       request go here -->
  </inbound>
  <backend>
    <!-- statements to be applied before the
       request is forwarded to
       the backend service go here -->
  </backend>
  <outbound>
    <!-- statements to be applied to the
       response go here -->
  </outbound>
  <on-error>
    <!-- statements to be applied if there
       is an error condition go here -->
  </on-error>
</policies>
```

```
<policies>
    <inbound>
        <cross-domain />
        <base />
        <find-and-replace from="xyz" to="abc" />
    </inbound>
</policies>
```


11

Create advanced policies (1 of 2)

API management policy

Control flow Conditionally applied based on the evaluation result of the Boolean expression.	**Forward request** Forward the request to the backend service.	**Limit concurrency** Prevent policies from executing more than the specified number of requests at a time.
Log to Event Hub Send the message in the specified format to the event center defined by the Logger entity.	**Mock response** The pipeline execution is aborted and the simulated response is returned directly to the caller.	**Retry** Retry the execution of the contained policy statement until the condition is met.

12

Create advanced policies (2 of 2)

Examples

```
<limit-concurrency key="expression" max-count="number">
    <!-- nested policy statements -->
</limit-concurrency>
```

```
<forward-request timeout="time in seconds" follow-redirects="true | false"/>
```

```
<log-to-eventhub logger-id="id of the logger entity" partition-id="index of the partition where
messages are sent" partition-key="value used for partition assignment">
    <!-- Expression returning a string to be logged -->
</log-to-eventhub>
```


13

Secure APIs by using subscriptions (1 of 3)

Subscription key scopes

Scope	Details
All APIs	Applies to every API accessible from the gateway
Single API	This scope applies to a single imported API and all of its endpoints
Product	A product is a collection of one or more APIs that you configure in API Management. You can assign APIs to more than one product. Products can have different access rules, usage quotas, and terms of use.

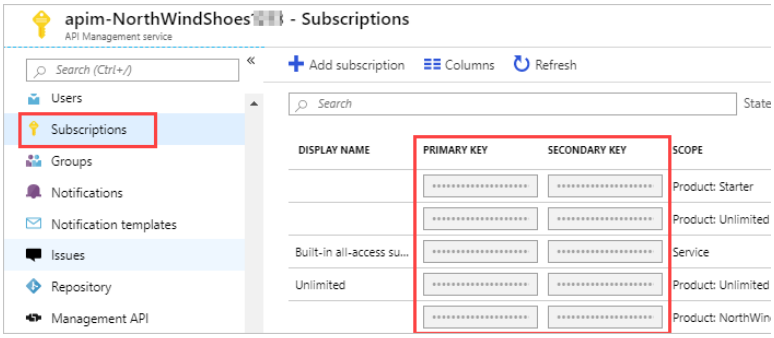
Note: API Management also supports other mechanisms for securing access to APIs, including: OAuth2.0, Client certificates, and IP allow listing.

14

Secure APIs by using subscriptions (2 of 3)

Applications that call protected APIs

- Must include the key in every request
- You can regenerate these subscription keys at any time.
- Every subscription has two keys, a primary and a secondary.

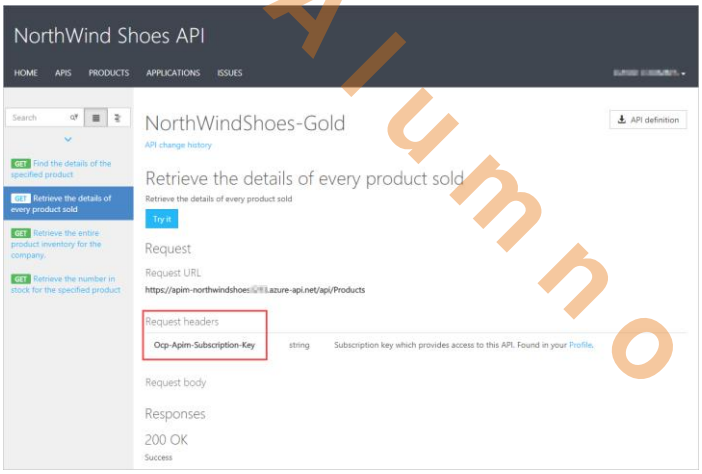


15

Secure APIs by using subscriptions (3 of 3)

Call an API with the subscription key

- Keys can be passed in the request header, or as a query string in the URL.
- The default header name is `Ocp-Apim-Subscription-Key`.
- Use the developer portal to test out API calls



16

Secure APIs by using certificates (1 of 4)

Transport Layer Security client authentication

Property	Description
Certificate Authority (CA)	Only allow certificates signed by a particular CA
Thumbprint	Allow certificates containing a specified thumbprint
Subject	Only allow certificates with a specified subject
Expiration Date	Only allow certificates that have not expired

Two methods to verify a certificate:

- Check who issued the certificate.
- If the certificate is issued by the partner, verify that it came from them.

17

Secure APIs by using certificates (2 of 4)

- Accepting client certificates in the Consumption tier
- Certificate Authorization Policies
- Check the thumbprint of a client certificate
- Check the thumbprint against certificates uploaded to API Management
- Check the issuer and subject of a client certificate

18

Secure APIs by using certificates (3 of 4)

Example

```
<!--Check the thumbprint of a client certificate-->
<choose>
    <when condition="@(context.Request.Certificate == null || context.Request.Certificate.Thumbprint
    != "desired-thumbprint")" >
        <return-response>
            <set-status code="403" reason="Invalid client certificate" />
        </return-response>
    </when>
</choose>
```


19

Secure APIs by using certificates (4 of 4)

Example

```
<!--Check the issuer and subject of a client certificate-->
<choose>
    <when condition="@(context.Request.Certificate == null || context.Request.Certificate.Issuer !=
    "trusted-issuer" || context.Request.Certificate.SubjectName.Name != "expected-subject-name")" >
        <return-response>
            <set-status code="403" reason="Invalid client certificate" />
        </return-response>
    </when>
</choose>
```


20

Exercise: Create a backend API

In this exercise you learn how to implement an API Management instance and import and configure an API.

Objectives

- Create an API Management instance
- Import a backend API
- Configure the backend settings
- Test the API
- Clean up resources

21

Summary and knowledge check

In this module, you learned how to:

- Describe the components, and their function, of the API Management service.
- Explain how API gateways can help manage calls to your APIs.
- Secure access to APIs by using subscriptions and certificates.
- Create a backend API.

1. What component of the API Management service would a developer use if they need to create an account and subscribe to get API keys?
2. What API Management policy would you use to apply a policy based on a condition?

22

23

Group discussion questions

- The security team of Contoso Inc noticed a particular IP Address is firing thousands of the requests against your API. What would do?
- You've setup API Management and a 3rd party partner is receiving 401 when calling your API. What can be the issue?
- Contoso Inc is hosting APIs on App Service for 3rd party integration. What can you do to increase the security?

24

Lab 08: Create a multi-tier solution by using Azure services

In this lab, you will create a containerized application to host a web app on Azure, as the source of information for the API. You will then build an API proxy using the Azure API Management capabilities to expose and test your APIs. Developers can query the APIs to test the service and validate its applicability.

http://aka.ms/az204labs

- Exercise 1: Create an Azure App Service resource by using a Docker container image
- Exercise 2: Build an API proxy tier by using Azure API Management

25

End of presentation

26

Microsoft

arelance
Building Digital Talent

AZ-204T00A

Learning Path 09: Develop event-based solutions

1

Agenda

- Explore Azure Event Grid
- Explore Azure Event Hubs

2

Module 1: Explore Azure Event Grid

3

Learning objectives

- Describe how Event Grid operates and how it connects to services and event handlers.
- Explain how Event Grid delivers events and how it handles errors.
- Implement authentication and authorization.
- Route custom events to web endpoint by using Azure CLI.

4

Introduction

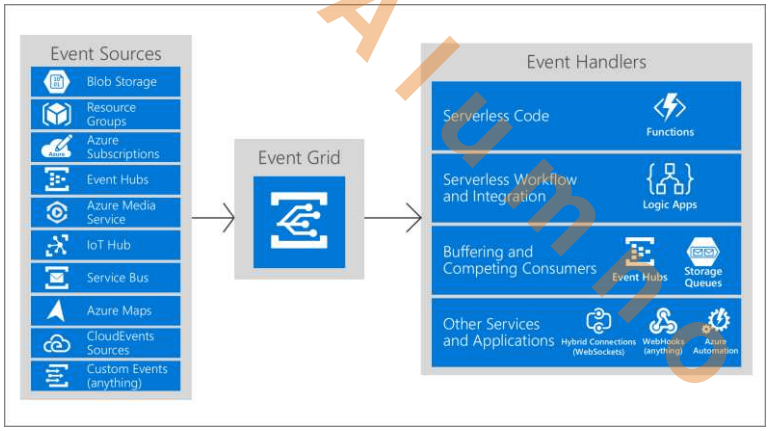
- Azure Event Grid is deeply integrated with Azure services and can be integrated with third-party services.
- It simplifies event consumption and lowers costs by eliminating the need for constant polling.
- Event Grid efficiently and reliably routes events from Azure, and non-Azure resources, and distributes the events to registered subscriber endpoints.

5

Explore Azure Event Grid (1 of 2)

What is the event grid

- Azure Event Grid is an eventing backplane
- Built-in support for events coming from Azure services
- Create events with custom topics
- Use filters to route specific events to different endpoints



6

Explore Azure Event Grid (2 of 2)

Six key concepts

1. **Publishers** - The application that sends events to Event Grid.
2. **Events and CloudEvents** - An event is the smallest amount of information that fully describes what is happening in the system. Event Grid conforms to Cloud Native Computing Foundation's open standard CloudEvents 1.0 specification.
3. **Event sources** - An event source is where the event happens. Each event source is related to one or more event types.
4. **Topics** - The event grid topic provides an endpoint where the source sends events. A topic is used for a collection of related events.
5. **Event subscriptions** - A subscription tells Event Grid which events on a topic you're interested in receiving. You can filter the events that are sent to the endpoint.
6. **Event handlers** - An event handler is the place where the event is sent. The handler takes some further action to process the event.

7

Discover event schemas (1 of 2)

Event properties

Property	Type	Required
topic	string	No
subject	string	Yes
eventType	string	Yes
eventTime	string	Yes
id	string	Yes
data	object	No
dataVersion	string	No
metadataVersion	string	No

```
[
  {
    "topic": string,
    "subject": string,
    "id": string,
    "eventType": string,
    "eventTime": string,
    "data":{
      object-unique-to-each-publisher
    },
    "dataVersion": string,
    "metadataVersion": string
  }
]
```


8

Discover event schemas (2 of 2)

Azure Blob storage event example

```
[{
    "topic": "...", "subject": "...",
    "eventType": "Microsoft.Storage.BlobCreated",
    "eventTime": "2017-06-26T18:41:00.9584103Z",
    "id": "831e1650-001e-001b-66ab-eeb76e069631",
    "data": {
        "api": "PutBlockList", "eTag": "0x8D4BCC2E4835CD0",
        "storageDiagnostics": { "batchId": "b68529f3-68cd-4744-baa4-3c0498ec19f0" },
        "clientRequestId": "6d79dbfb-0e37-4fc4-981f-442c9ca65760",
        "requestId": "831e1650-001e-001b-66ab-eeb76e000000",
        "contentType": "application/octet-stream", "contentLength": 524288,
        "blobType": "BlockBlob", "sequencer": "00000000000004420000000000028963",
        "url": "https://test.blob.core.windows.net/container/blob"
    },
    "dataVersion": "", "metadataVersion": "1"
}]
```


9

Explore event delivery durability

- **Retry schedule** - When Event Grid receives an error for an event delivery attempt, it decides whether it should retry the delivery, dead-letter the event, or drop the event based on the type of the error.
- **Retry policy** - You can customize the retry policy when creating an event subscription through the *maximum number of attempts* and *event time-to-live* settings.
- **Output batching** - You can configure Event Grid to batch events for delivery for improved HTTP performance in high-throughput scenarios.
- **Delayed delivery** - If an endpoint experiences delivery failures, Event Grid will delay the delivery and retry of events to that endpoint.
- **Dead-letter events** - If an event can not be delivered within a certain time period, or a certain number of attempts, it can send the undelivered event to a storage account.

10

Control access to events (1 of 2)

Built-in roles

Azure Event Grid allows you to control the level of access given to different users to do various management operations such as list event subscriptions, create new ones, and generate keys. Event Grid uses Azure role-based access control (Azure RBAC).

Role	Description
Event Grid Subscription Reader	Read Event Grid event subscriptions.
Event Grid Subscription Contributor	Manage Event Grid event subscription operations.
Event Grid Contributor	Create and manage Event Grid resources.
Event Grid Data Sender	Send events to Event Grid topics.

11

Control access to events (2 of 2)

Permissions for event subscriptions

If you're using an event handler that isn't a WebHook (such as an event hub or queue storage), you need write access to that resource. This permissions check prevents an unauthorized user from sending events to your resource.

Topic Type	Description
System topics	Need permission to write a new event subscription at the scope of the resource publishing the event.
Custom topics	Need permission to write a new event subscription at the scope of the event grid topic.

12

Receive events by using webhooks

Three Azure services

- Azure Logic Apps with Event Grid Connector
- Azure Automation via webhook
- Azure Functions with Event Grid Trigger

Two ways to verify subscription

- **Synchronous handshake** - At the time of event subscription creation, Event Grid sends a subscription validation event to your endpoint.
- **Asynchronous handshake** - In certain cases, you can't return the `ValidationCode` in response synchronously.

13

Filter events

- Event type filtering
  - By default, all event types for the event source are sent to the endpoint.
  - You can decide to send only certain event types to your endpoint.
- Subject filtering
  - For simple filtering by subject, specify a starting or ending value for the subject.
  - You can filter the subject with `/blobServices/default/containers/testcontainer` to get all events for that container but not other containers in the storage account.
- Advanced filtering
  - To filter by values in the data fields and specify the comparison operator, use the advanced filtering option.

14

Exercise: Route custom events to web endpoint by using Azure CLI

In this exercise you learn how to use the Azure CLI to Event Grid resources, subscribe to a custom topic, and send an event to a custom topic.

Objectives

- Create a resource group
- Enable an Event Grid resource provider
- Create a custom topic
- Create a message endpoint
- Subscribe to a custom topic
- Send an event to your custom topic
- Clean up resources

15

Summary and knowledge check

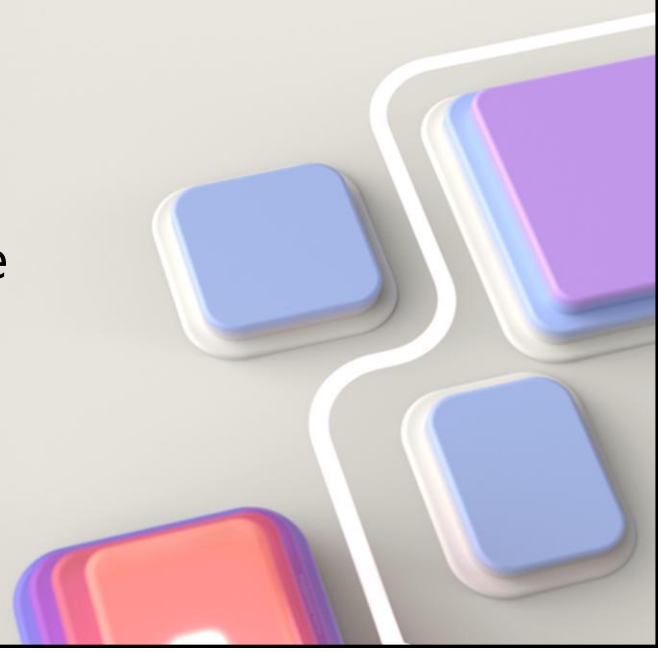
In this module, you learned how to:

- Describe how Event Grid operates and how it connects to services and event handlers.
- Explain how Event Grid delivers events and how it handles errors.
- Implement authentication and authorization.
- Route custom events to web endpoint by using Azure CLI.

1. Which event schema properties requires a value?
2. Which Event Grid built-in role is appropriate for managing Event Grid resources?

16

Module 2: Explore Azure Event Hubs



17

Learning objectives

- Describe the benefits of using Event Hubs and how it captures streaming data.
- Explain how to process events.
- Perform common operations with the Event Hubs client library.

18

Introduction

- Azure Event Hubs is a big data streaming platform and event ingestion service.
- It can receive and process millions of events per second.
- Data sent to an event hub can be transformed and stored by using any real-time analytics provider or batching/storage adapters.

19

Discover Azure Event Hubs (1 of 2)

- Azure Event Hubs provides a unified streaming platform with time retention buffer, decoupling event producers from event consumers.
- It is a scalable event-processing service that ingests and processes large volumes of events and data, with low latency and high reliability.

Feature	Description
Fully managed PaaS	Event Hubs is a fully managed service with little configuration or management overhead
Real-time and batch processing	Event Hubs uses a partitioned consumer model, enabling multiple applications to process the stream concurrently and letting you control the speed of processing.
Scalable	Scaling options, like Auto-inflate, scale the number of throughput units to meet your usage needs.
Rich ecosystem	Event Hubs for Apache Kafka ecosystems enables Apache Kafka (1.0 and later) clients and applications to talk to Event Hubs

20

Discover Azure Event Hubs (2 of 2)

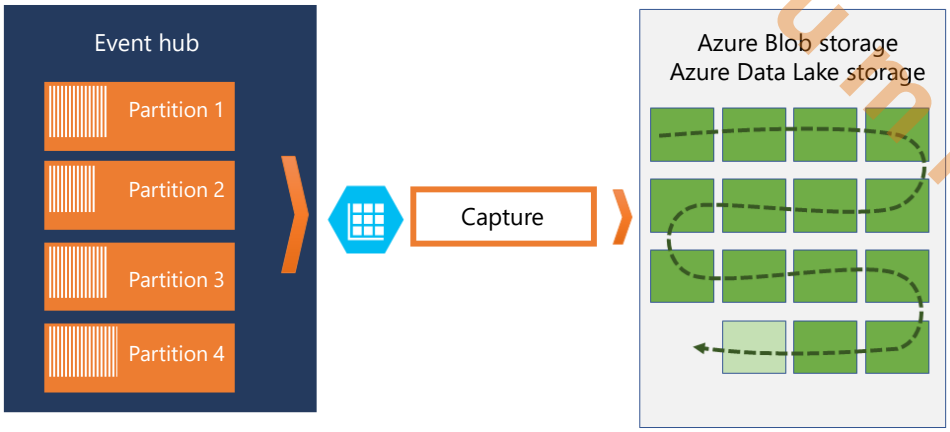
Key components

- An **Event Hub client** is the primary interface for developers interacting with the Event Hubs client library.
- An **Event Hub producer** is a type of client that serves as a source of telemetry data, diagnostics information, usage logs, or other log data, as part of an embedded device solution, a mobile device application, a game title running on a console or other device, some client or server based business solution, or a web site.
- An **Event Hub consumer** is a type of client which reads information from the Event Hub and allows processing of it. Processing may involve aggregation, complex computation and filtering.
- A **partition** is an ordered sequence of events that is held in an Event Hub. Partitions are a means of data organization associated with the parallelism required by event consumers.
- A **consumer group** is a view of an entire Event Hub. Consumer groups enable multiple consuming applications to each have a separate view of the event stream, and to read the stream independently at their own pace and from their own position.
- **Event receivers** - Any entity that reads event data from an event hub. All Event Hubs consumers connect via the AMQP 1.0 session.
- **Throughput units** or **processing units** - Pre-purchased units of capacity that control the throughput capacity of Event Hubs.

21

Explore Event Hubs Capture (1 of 2)

- Azure Event Hubs enables you to automatically capture the streaming data in Event Hubs in an Azure Blob storage or Azure Data Lake Storage account of your choice.
- Event Hubs Capture enables you to process real-time and batch-based pipelines on the same stream.



22

Explore Event Hubs Capture (2 of 2)

Capture windowing

- Event Hubs Capture enables you to set up a window to control capturing.
- Each partition captures independently and writes a completed block blob at the time of capture, named for the time at which the capture interval was encountered.

Scaling to throughput units

- Event Hubs traffic is controlled by throughput units.
- Event Hubs Capture copies data directly from the internal Event Hubs storage, bypassing throughput unit egress quotas and saving your egress for other processing readers.
- Event Hubs Capture runs automatically when you send your first event.

23

Scale your processing application (1 of 2)

The key to scale for Event Hubs is the idea of *partitioned consumers*. In contrast to the competing consumers pattern, the partitioned consumer pattern enables high scale by removing the contention bottleneck and facilitating end to end parallelism.

Example scenario

When designing the consumer in a distributed environment, the solution must handle the following requirements: scale, load balance, seamless resume on failures, and event consumption.

Event processor or consumer client

- You don't need to build your own solution to meet these requirements.
- The Azure Event Hubs SDKs provide this functionality.
- For most production scenarios use the event processor client for reading and processing events.

24

Scale your processing application (2 of 2)

Partition ownership tracking

- An event processor instance typically owns and processes events from one or more partitions.
- Each event processor is given a unique identifier and claims ownership of partitions by adding or updating an entry in a checkpoint store.

Receive messages

- When you create an event processor, you specify the functions that will process events and errors.
- We recommend that you do things relatively fast. That is, do as little processing as possible.

Checkpointing

- Checkpointing is a process by which an event processor marks or commits the position of the last successfully processed event within a partition.
- By specifying a lower offset from this checkpoint process, you can return to older data.

Thread safety and processor instances

By default, the function that processes the events is called sequentially for a given partition. As the event pump continues to run in the background of other threads, subsequent events from the same partition and calls to this function are queued in the background.

25

Control access to events

Azure built-in roles:

- Azure Event Hubs Data Owner: Use this role to give *complete access* to Event Hubs resources.
- Azure Event Hubs Data Sender: Use this role to give *send access* to Event Hubs resources.
- Azure Event Hubs Data Receiver: Use this role to give *receiving access* to Event Hubs resources.

You can:

- Authorize access with managed identities
- Authorize access with Microsoft Identity Platform
- Authorize access to Event Hubs publishers with shared access signatures
- Authorize access to Event Hubs consumers with shared access signatures

26

Perform common operations with the Event Hubs client library (1 of 3)

Inspect an Event Hub

```
await using (var producer = new EventHubProducerClient(connectionString, eventHubName))
{
    string[] partitionIds = await producer.GetPartitionIdsAsync();
}
```


27

Perform common operations with the Event Hubs client library (2 of 3)

Publish events to an Event Hub

```
await using (var producer = new EventHubProducerClient(connectionString, eventHubName))
{
    using EventDataBatch eventBatch = await producer.CreateBatchAsync();
    eventBatch.TryAdd(new EventData(new BinaryData("First")));
    eventBatch.TryAdd(new EventData(new BinaryData("Second")));

    await producer.SendAsync(eventBatch);
}
```


28

Perform common operations with the Event Hubs client library (3 of 3)

Read events from an Event Hub

```
string consumerGroup = EventHubConsumerClient.DefaultConsumerGroupName;

await using (var consumer = new EventHubConsumerClient(consumerGroup, connectionString, eventHubName))
{
    using var cancellationSource = new CancellationTokenSource();
    cancellationSource.CancelAfter(TimeSpan.FromSeconds(45));

    await foreach (PartitionEvent receivedEvent in consumer.ReadEventsAsync(cancellationSource.Token))
    {
        // At this point, the loop will wait for events to be available in the Event Hub.  When an event
        // is available, the loop will iterate with the event that was received.  Because we did not
        // specify a maximum wait time, the loop will wait forever unless cancellation is requested using
        // the cancellation token.
    }
}
```


29

Summary and knowledge check

In this module, you learned how to:

- Describe the benefits of using Event Hubs and how it captures streaming data.
- Explain how to process events.
- Perform common operations with the Event Hubs client library.

1. Which Event Hubs concept represents an ordered sequence of events that is held in an Event Hubs?
2. Which process represents when an event processor marks or commits the position of the last successfully processed event within a partition?

30

Discussion and lab

31

Group discussion questions

- Can you describe the differences between the capabilities of Azure Event Grid and Azure Event Hubs? When would you use one over the other?
- Contoso Inc is using Event Grid in their solution, but many events are being lost during delivery. What can they do to improve delivery and retain the events being lost?
- Can you list the three Azure built-in roles for Event Hubs and what permissions they grant?

32

Lab 09: Publish and subscribe to Event Grid events

In this lab, you will start with a proof-of-concept web app, hosted in a container, that will be used to subscribe to your Event Grid. This app will allow you to submit events and receive confirmation messages that the events were successful.

http://aka.ms/az204labs

- Exercise 1: Create Azure resources
- Exercise 2: Create an Event Grid subscription
- Exercise 3: Publish Event Grid events from .NET

33

End of presentation

34

Microsoft

arelance
Building Digital Talent

AZ-204T00A

Learning Path 10: Develop message-based solutions

1

Agenda

- Discover Azure message queues

2

Module 1: Discover Azure message queues

3

Learning objectives

- Choose the appropriate queue mechanism for your solution.
- Explain how the messaging entities that form the core capabilities of Service Bus operate.
- Send and receive message from a Service Bus queue by using .NET.
- Identify the key components of Azure Queue Storage
- Create queues and manage messages in Azure Queue Storage by using .NET.

4

Introduction

- Azure supports two types of queue mechanisms: *Service Bus queues* and *Storage queues*.
- Service Bus queues are part of a broader Azure messaging infrastructure that supports queuing, publish/subscribe, and more advanced integration patterns.
- Storage queues are part of the Azure Storage infrastructure, and they allow you to store large numbers of messages.

5

Choose a message queue solution

Consider using Service Bus queues when:

- Your solution needs to receive messages without having to poll the queue
- Your solution requires the queue to provide a guaranteed first-in-first-out (FIFO) ordered delivery.
- Your solution needs to support automatic duplicate detection.
- You want your application to process messages as parallel long-running streams
- Your solution requires transactional behavior and atomicity when sending or receiving multiple messages from a queue.
- Your application handles messages that can exceed 64 KB but won't likely approach the 256-KB, or 1-MB limit.

Consider using Storage queues when:

- Your application must store over 80 gigabytes of messages in a queue.
- Your application wants to track progress for processing a message in the queue. It's useful if the worker processing a message crashes. Another worker can then use that information to continue from where the prior worker left off.
- You require server-side logs of all the transactions executed against your queues.

6

Explore Azure Service Bus

Some common scenarios are:

- Messaging - Transfer business data, such as sales or purchase orders, journals, or inventory movements.
- Decouple applications - Improve reliability and scalability of applications
- Topics and subscriptions - Enable 1:n relationships between publishers and subscribers
- Message sessions - Implement workflows that require message ordering or message deferral.

Premium	Standard
High throughput	Variable throughput
Predictable performance	Variable latency
Fixed pricing	Pay as you go variable pricing
Ability to scale workload up and down	N/A
Message size up to 100 MB.	Message size up to 256 KB

7

Discover Service Bus queues, topics, and subscriptions

Queues

- Queues offer First In, First Out (FIFO) message delivery to one or more competing consumers.

Receive modes

- You can specify two different modes in which Service Bus receives messages: Receive and delete or Peek lock.

Topics and subscriptions

- Provides a one-to-many form of communication in a publish and subscribe pattern.

Rules and actions

- You can configure subscriptions to find messages that have desired properties and then perform certain modifications to those properties.
- You can use filter actions to copy a subset of those messages to the virtual subscription queue.

8

Explore Service Bus message payloads and serialization

Message routing and correlation patterns

- *Simple request/reply* - A publisher sends a message into a queue and expects a reply from the message consumer.
- *Multicast request/reply* - As a variation of the prior pattern, a publisher sends the message into a topic and multiple subscribers become eligible to consume the message
- *Multiplexing* - This session feature enables multiplexing of streams of related messages through a single queue or subscription
- *Multiplexed request/reply* - This session feature enables multiplexing of streams of related messages through a single queue or subscription

Payload serialization

- The `ContentType` property enables applications to describe the payload
- The .NET Framework version of the Service Bus API supports creating `BrokeredMessage` instances by passing arbitrary .NET objects into the constructor.
- When using the legacy SBMP protocol, those objects are then serialized with the default binary serializer, or with a serializer that is externally supplied.
- When using the AMQP protocol, the object is serialized into an AMQP Bytes

9

Exercise: Send and receive message from a Service Bus queue by using .NET.

In this exercise you learn how to use the Azure CLI to create a Service Bus namespace and queue. You also create a .NET console app to send and receive messages from the queue.

Objectives

- Create Azure resources
- Create console app to send messages to the queue
- Review results
- Update project to receive messages to the queue
- Clean up resources

10

Explore Azure Queue Storage

- Azure Queue Storage is a service for storing large numbers of messages.
- A queue message can be up to 64 KB in size
- The Queue service contains the following components:
  - URL format
  - Storage
  - Queue
  - Message

The Queue service contains the following components:

- *URL format* - Queues are addressable using the URL format `https://<account>.queue.core.windows.net/<queue>`.
- *Storage account* - All access to Azure Storage is done through a storage account.
- *Queue* - A queue contains a set of messages. All messages must be in a queue.
- *Message* - A message, in any format, of up to 64 KB. For version 2017-07-29 or later, the maximum time-to-live can be any positive number, or -1 indicating that the message doesn't expire. If this parameter is omitted, the default time-to-live is seven days.

11

Create and manage Azure Queue Storage queues and messages by using .NET (1 of 3)

The following code examples rely on the following NuGet packages:

- Azure.Core library for .NET: This package provides shared primitives, abstractions, and helpers for modern .NET Azure SDK client libraries.
- Azure.Storage.Common client library for .NET: This package provides infrastructure shared by the other Azure Storage client libraries.
- Azure.Storage.Queues client library for .NET: This package enables working with Azure Queue Storage for storing messages that may be accessed by a client.
- System.Configuration.ConfigurationManager library for .NET: This package provides access to configuration files for client applications.

12

Create and manage Azure Queue Storage queues and messages by using .NET (2 of 3)

```
// The QueueClient class enables you to retrieve queues stored in Queue storage.

QueueClient queueClient = new QueueClient(connectionString, queueName);
```

```
// This example shows how to create a queue if it does not already exist

// Get the connection string from app settings
string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];

// Instantiate a QueueClient which will be used to create and manipulate the queue
QueueClient queueClient = new QueueClient(connectionString, queueName);

// Create the queue
queueClient.CreateIfNotExists();
```


13

Create and manage Azure Queue Storage queues and messages by using .NET (3 of 3)

```
// To insert a message into an existing queue, call the SendMessage method.

// Get the connection string from app settings
string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];

// Instantiate a QueueClient which will be used to create and manipulate the queue
QueueClient queueClient = new QueueClient(connectionString, queueName);

// Create the queue if it doesn't already exist
queueClient.CreateIfNotExists();

if (queueClient.Exists())
{
    // Send a message to the queue
    queueClient.SendMessage(message);
}
```


14

Summary and knowledge check

In this module, you learned how to:

- Choose the appropriate queue mechanism for your solution.
- Explain how the messaging entities that form the core capabilities of Service Bus operate.
- Send and receive message from a Service Bus queue by using .NET.
- Identify the key components of Azure Queue Storage
- Create queues and manage messages in Azure Queue Storage by using .NET.

1. What advanced feature of Azure Service Bus creates a first-in, first-out (FIFO) guarantee?

2. In Azure Service Bus messages are durably stored which enables a load-leveling benefit. What is the benefit of load-leveling relative to a consuming application's performance?

15

Discussion and lab

16

Group discussion questions

- Under what conditions would it best better to use Azure Service Bus for messages? Azure Queue Storage queues?
- Describe the four components of Azure Queue Storage.
- Describe Service Bus queues, topics, and subscriptions. What are the receive modes?

17

Lab 10: Asynchronously process messages by using Azure Service Bus Queues

In this lab, you will create a .NET Core project that will publish messages to the system, and a second .NET Core application that will read messages from the queue. The first app will simulate data coming from a sensor, while the second app will simulate the system that will read the messages from the queue for processing.

http://aka.ms/az204labs

- Exercise 1: Create Azure resources
- Exercise 2: Create a .NET Core project to publish messages to a Service Bus queue
- Exercise 3: Create a .NET Core project to read messages from a Service Bus queue

18

19

Microsoft

arelance
Building Digital Talent

AZ-204T00A

Learning Path 11: Troubleshoot solutions by using Application Insights

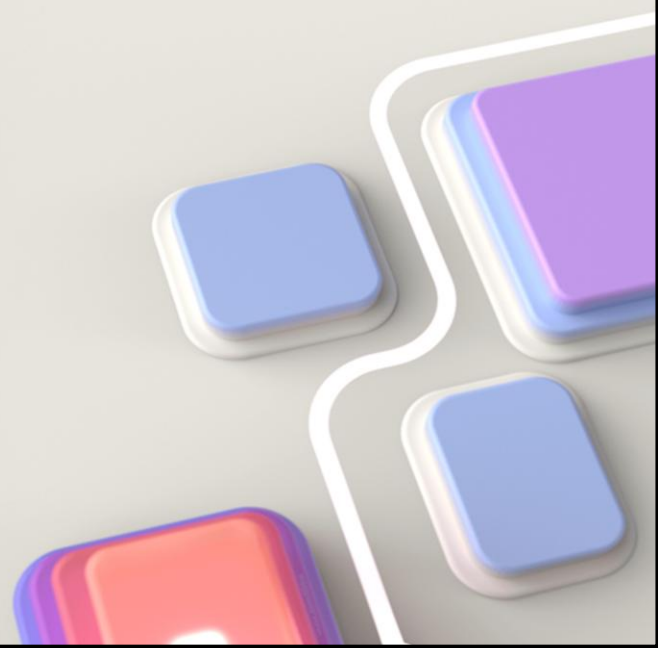
1

Agenda

- Monitor app performance

2

Module 1: Monitor app performance



3

Learning objectives

- Describe how Application Insights works and how it collects events and metrics.
- Instrument an app for monitoring, perform availability tests, and use Application Map to help you monitor performance and troubleshoot issues.

4

Introduction

- Instrumenting and monitoring, your apps helps you maximize their availability and performance.
- Application Insights is an extension of Azure Monitor and provides Application Performance Monitoring (also known as "APM") features.
- In addition to collecting metrics and application telemetry data, which describe application activities and health, Application Insights can also be used to collect and store application trace logging data.

5

Explore Application Insights (1 of 2)

Features include, but not limited to:

Feature	Description
Live Metrics	Observe activity from your deployed application in real time with no effect on the host environment.
Availability	Also known as "Synthetic Transaction Monitoring", probe your applications external endpoint(s) to test the overall availability and responsiveness over time.
GitHub/DevOps integration	Create GitHub or Azure DevOps work items in context of Application Insights data.
Usage	Understand which features are popular with users and how users interact and use your application
Smart Detection	Automatic failure and anomaly detection through proactive telemetry analysis.
Application Map	A high-level top-down view of the application architecture and at-a-glance visual references to component health and responsiveness.
Distributed Tracing	Search and visualize an end-to-end flow of a given execution or transaction.

6

Explore Application Insights (2 of 2)

Application Insights monitors:

- Request rates, response times, and failure rates
- Dependency rates, response times, and failure rates
- Exceptions
- Page views and load performance
- AJAX calls from web pages
- User and session counts.
- Performance counters
- Host diagnostics
- Diagnostic trace logs
- Custom events and metrics

Several ways to get started monitoring and analyzing performance:

- At run time
- At development time
- Instrument your web pages
- Analyze mobile app usage
- Availability tests

7

Discover log-based metrics

Log-based metrics

- The Application Insights backend stores all collected events as logs.
- The Application Insights blades in the Azure portal act as an analytical and diagnostic tool for visualizing event-based data from logs.
- Using logs to retain a complete set of events can bring great analytical and diagnostic value.
- Collecting a complete set of events may be impractical (or even impossible) for applications that generate a large volume of telemetry.

Standard metrics

- Stored as pre-aggregated time series, and only with key dimensions
- The newer SDKs (Application Insights 2.7 SDK or later for .NET) pre-aggregate metrics during collection
- For the SDKs that don't implement pre-aggregation, the Application Insights backend still populates the new metrics by aggregating the events received by the Application Insights event collection endpoint

8

Instrument an app for monitoring (1 of 2)

Autoinstrumentation

- Autoinstrumentation allows you to enable application monitoring with Application Insights without changing your code.
- Just enable and, in some cases, configure the agent, which will collect the telemetry automatically.

Manual instrumention

- Coding against Appliction Insights or OpenTelemetry API.

9

Instrument an app for monitoring (2 of 2)

Terminology changes

Terms in Application Insights

- Some legacy terms in Application Insights are confusing because of the industry convergence on OpenTelemetry.

App Insights	OpenTelemetry
Autocollectors	Instrumentation Libraries
Channel	Exporter
Codeless / Agent-based	Autoinstrumentation
Traces	Logs
Requests	Server Spans
Dependencies	Other Span Types (Client, Internal, etc.)
Operation ID	Trace ID
ID or Operation Parent ID	Span ID

10

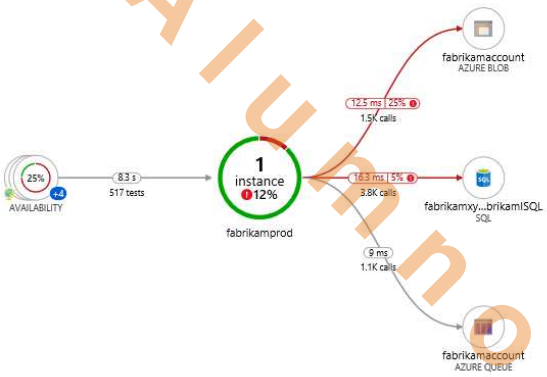
Select an availability test

- You can set up availability tests for any HTTP or HTTPS endpoint
- You don't have to make any changes to the website you're testing
- It doesn't even have to be a site that you own, you can test the availability of a REST API that your service depends on.
- You can create up to 100 availability tests per Application Insights resource, and there are three types of availability tests
  - Standard test
  - Custom TrackAvailability

11

Troubleshoot app performance by using Application Map

- Application Map helps you spot performance bottlenecks or failure hotspots across all components of your distributed application.
- Each node on the map represents an application component or its dependencies; and has health KPI and alerts status.
- You can click through from any component to more detailed diagnostics
- Components are independently deployable parts of your distributed/microservices application



12

Summary and knowledge check

In this module, you learned how to:

- Describe how Application Insights works and how it collects events and metrics.
- Instrument an app for monitoring, perform availability tests, and use Application Map to help you monitor performance and troubleshoot issues.

1. What availability test is recommended for authentication tests?

2. What metric collection type provides near real-time querying and alerting on dimensions of metrics, and more responsive dashboards?

13

Discussion and lab

14

Group discussion questions

- Can you describe how Application Insights and Log Analytics are related to each other?
- What are the performance differences between log-based and pre-aggregated metrics? When would you choose one over the other?
- What process would you follow, and what tools would you choose, to monitor and optimize the performance of an app?

15

Lab 11: Monitor services that are deployed to Azure

In this lab, you will create an Application Insights resource in Azure that will be used to monitor and log application insight data for later review. The API will be set to automatically scale if demand increases to a certain threshold and logging the data will help determine how the service is being utilized.

http://aka.ms/az204labs

- Exercise 1: Create and configure Azure resources
- Exercise 2: Monitor a local web API by using Application Insights
- Exercise 3: Monitor a web API using Application Insights

16

17

Microsoft

arelance
Building Digital Talent

AZ-204T00A

Learning Path 12: Implement caching for solutions

1

Agenda

- Develop for Azure Cache for Redis
- Develop for storage on CDNs

2

Module 1: Develop for Azure Cache for Redis

3

Learning objectives

- Explain the key scenarios Azure Cache for Redis covers and its service tiers.
- Identify the key parameters for creating an Azure Cache for Redis instance and interact with the cache.
- Connect an app to Azure Cache for Redis by using .NET Core.

4

Introduction

- Caching is a common technique that aims to improve the performance and scalability of a system.
- It does this by temporarily copying frequently accessed data to a location close to the application.

5

Explore Azure Cache for Redis (1 of 2)

- Azure Cache for Redis provides an in-memory data store based on the Redis software.
- Redis improves the performance and scalability of applications that use backend data stores.
- Processes large volumes of application requests by keeping frequently accessed data in the server memory.
- Brings a critical low-latency and high-throughput data storage solution to modern applications.

6

Explore Azure Cache for Redis (2 of 2)

Key scenarios

Azure Cache for Redis improves application performance by supporting common application architecture patterns.

- Data cache
- Content cache
- Session store
- Job and message queuing
- Distributed transactions

Service tiers

Azure Cache for Redis is available in these tiers:

- Basic
- Standard
- Premium
- Enterprise
- Enterprise Flash

7

Configure Azure Cache for Redis (1 of 4)

Create and configure an Azure Cache for Redis instance

There are several parameters you will need to decide in order to configure the cache properly for your purposes.

8

Configure Azure Cache for Redis (2 of 4)

Accessing the Redis instance

- Redis has a command-line tool for interacting with an Azure Cache for Redis as a client.
- Redis supports a set of known commands.

Command	Description	Command	Description
`ping`	Ping the server. Returns "PONG".	`get [key]`	Gets a value from the cache.
`set [key] [value]`	Sets a key/value in the cache. Returns "OK" on success.	`exists [key]`	Returns '1' if the **key** exists in the cache, '0' if it doesn't.
`incr [key]`	Increment the given value associated with **key** by '1'. The value must be an integer or double value. This returns the new value.	`incrby [key] [amount]`	Increment the given value associated with **key** by the specified amount. The value must be an integer or double value. This returns the new value.
`type [key]`	Returns the type associated to the value for the given **key**.	`del [key]`	Deletes the value associated with the **key**.
`flushdb`	Delete all keys and values in the database.		

9

Configure Azure Cache for Redis (3 of 4)

Adding an expiration time to values

- In Redis we expire values when they are stale by applying a time to live (TTL) to a key.
- When the TTL elapses, the key is automatically deleted, exactly as if the DEL command were issued.
  - Expirations can be set using seconds or milliseconds precision.
  - The expire time resolution is always 1 millisecond.

```
> set counter 100
OK
> expire counter 5
(integer) 1
> get counter
100
... wait ...
> get counter
(nil)
```


10

Configure Azure Cache for Redis (4 of 4)

Accessing a Redis cache from a client

To connect to an Azure Cache for Redis instance, you'll need the host name, port, and an access key for the cache. You can retrieve this information in the Azure portal.

- The host name is the public Internet address of your cache, which was created using the name of the cache. For example, sportsresults.redis.cache.windows.net.
- The access key acts as a password for your cache. There are two keys created: primary and secondary.

11

Interact with Azure Cache for Redis by using .NET (1 of 3)

Executing commands on the Redis cache

A popular high-performance Redis client for the .NET language is StackExchange.Redis. The package is available through NuGet and can be added to your .NET code using the command line or IDE.

```
//Creating a connection

using StackExchange.Redis;
...
var connectionString = "[cache-name].redis.cache.windows.net:6380,password=[password-here],ssl=True,abortConnect=False";
var redisConnection = ConnectionMultiplexer.Connect(connectionString);
```


12

Interact with Azure Cache for Redis by using .NET (2 of 3)

Once you have a `ConnectionMultiplexer`, there are 3 primary things you might want to do:

- Access a Redis Database (example below)
- Make use of the publisher/subscript features of Redis. (Outside the scope of this module.)
- Access an individual server for maintenance or monitoring purposes.

```
// Accessing a database
IDatabase db = redisConnection.GetDatabase();

// Example of storing a key/value in the cache
bool wasSet = db.StringSet("favorite:flavor", "i-love-rocky-road");

// Retrieving the value
string value = db.StringGet("favorite:flavor");
Console.WriteLine(value); // displays: ""i-love-rocky-road""
```


13

Interact with Azure Cache for Redis by using .NET (3 of 3)

Other common operations

Method	Description
`CreateBatch`	Creates a group of operations that will be sent to the server as a single unit, but not necessarily processed as a unit.
`CreateTransaction`	Creates a group of operations that will be sent to the server as a single unit and processed on the server as a single unit.
`KeyDelete`	Delete the key/value.
`KeyExists`	Returns whether the given key exists in cache.
`KeyExpire`	Sets a time-to-live (TTL) expiration on a key.
`KeyRename`	Renames a key.
`KeyTimeToLive`	Returns the TTL for a key.
`KeyType`	Returns the string representation of the type of the value stored at key. The different types that can be returned are: string, list, set, zset and hash.

14

Exercise: Connect an app to Azure Cache for Redis by using .NET Core

In this exercise you learn how to create a new Redis Cache instance by using Azure CLI commands and create a .NET Core console app to add and retrieve values from the cache.

Objectives

- Create Azure resources
- Create the console application
- Clean up resources

15

Summary and knowledge check

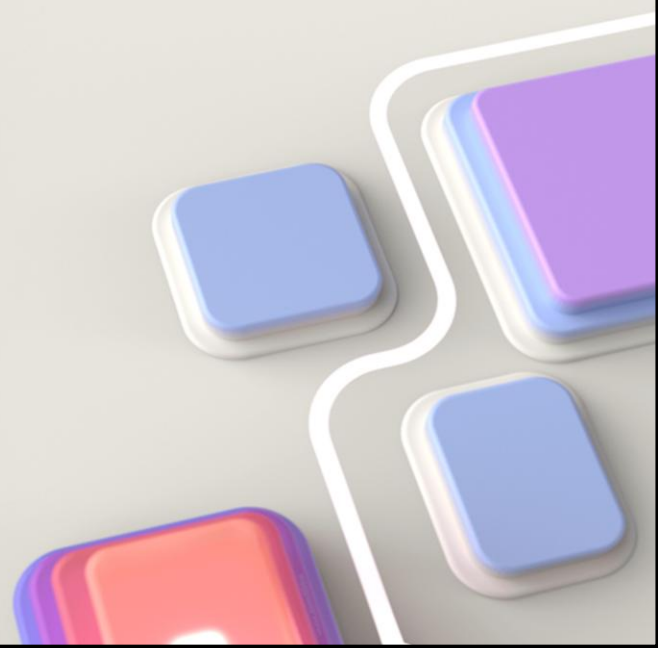
In this module, you learned how to:

- Explain the key scenarios Azure Cache for Redis covers and its service tiers
- Identify the key parameters for creating an Azure Cache for Redis instance and interact with the cache
- Connect an app to Azure Cache for Redis by using .NET Core

1. What is the lowest service tier of Azure Cache for Redis recommended for use in production scenarios?
2. What is the expire time resolution when applying a time to live (TTL) to a key in Redis?

16

Module 2: Develop for storage on CDNs



17

Learning objectives

- Explain how the Azure Content Delivery Network works and how it can improve the user experience.
- Control caching behavior and purge content.
- Perform actions on Azure CDN by using the Azure CDN Library for .NET.

18

Introduction

- A content delivery network (CDN) is a distributed network of servers that can efficiently deliver web content to users.
- CDNs store cached content on edge servers in point-of-presence (POP) locations that are close to end users, to minimize latency.

19

Explore Azure Content Delivery Networks (1 of 4)

Overview

- Azure Content Delivery Network (CDN) delivers high-bandwidth content to users by caching content at strategically placed physical nodes across the world.
- Azure CDN can also accelerate dynamic content, which cannot be cached, by leveraging various network optimizations using CDN POPs.

Benefits

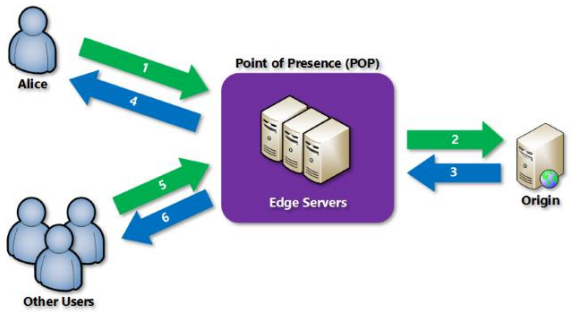
- Better performance and improved user experience for end users.
- Large scaling to better handle instantaneous high loads.
- Distribution of user requests and serving of content directly from edge servers so that less traffic is sent to the origin server.

20

Explore Azure Content Delivery Networks (2 of 4)

How Azure Content Delivery Network works

1. Alice requests a file by using a URL with a special domain name, such as `<endpoint name>.azureedge.net`.
2. If no edge servers in the POP have the file in their cache, the POP requests the file from the origin server.
3. The origin server returns the file to an edge server in the POP.
4. An edge server in the POP caches the file and returns the file to the original requestor (Alice).
5. Additional users can then request the same file by using the same URL that Alice used.
6. If the TTL for the file hasn't expired, the POP edge server returns the file directly from the cache.



21

Explore Azure Content Delivery Networks (3 of 4)

Requirements

- You need to create at least one CDN profile, which is a collection of CDN endpoints.
- Every CDN endpoint represents a specific configuration of content deliver behavior and access.
- To organize your CDN endpoints by internet domain, web application, or some other criteria, you can use multiple profiles.

Limitations

Each Azure subscription has default limits for the following resources:

- The number of CDN profiles that can be created.
- The number of endpoints that can be created in a CDN profile.
- The number of custom domains that can be mapped to an endpoint.

22

Explore Azure Content Delivery Networks (4 of 4)

Azure Content Delivery Network (CDN) includes three products:

- Azure CDN Standard from Microsoft
- Azure CDN Standard from Edgio (formerly Verizon)
- Azure CDN Premium from Edgio (formerly Verizon)

23

Control cache behavior on Azure Content Delivery Networks (1 of 4)

Overview

- Because a cached resource can potentially be out-of-date, it is important for any caching mechanism to control when content is refreshed.
- To save time and bandwidth consumption, a cached resource is not compared to the version on the origin server every time it is accessed.
- If a cached resource is considered to be fresh, it is assumed to be the most current version and is sent directly to the client.
- A cached resource is considered to be fresh when its age is less than the age or period defined by a cache setting.

24

Control cache behavior on Azure Content Delivery Networks (2 of 4)

Controlling caching behavior

- Azure CDNs provide two mechanisms for caching files: caching rules, and query string caching. However, these configuration settings depend on the tier you've selected.
- **Caching rules:** Azure CDN provides global and custom types of caching rules.
  - Global caching rules - You can set one global caching rule for each endpoint in your profile, which affects all requests to the endpoint.
  - Custom caching rules - Set one or more custom caching rules for each endpoint in your profile. Overrides the global caching rule, if set.
- **Query string caching:** Adjust how the Azure CDN treats caching for requests with query strings. If the file isn't cacheable, the query string caching setting has no effect, based on caching rules and content delivery network default behaviors.

25

Control cache behavior on Azure Content Delivery Networks (3 of 4)

Caching and time to live

- If you publish a website through Azure CDN, the files on that site are cached until their TTL expires. The `Cache-Control` header contained in the HTTP response from origin server determines the TTL duration.
- If you don't set a TTL on a file, Azure CDN sets a default value. However, this default may be overridden if you have set up caching rules in Azure. Default TTL values are as follows:
  - Generalized web delivery optimizations: seven days
  - Large file optimizations: one day
  - Media streaming optimizations: one year

26

Control cache behavior on Azure Content Delivery Networks (4 of 4)

Content updating

- In normal operation:
  - An Azure CDN edge node will serve an asset until its TTL expires.
  - The edge node reconnects to the origin server when the TTL expires and a client makes a request to the same asset.
  - The node will fetch another copy of the asset, resetting the TTL in the process.
- To ensure that users always receive the latest version of an asset, consider including a version string in the asset URL. This approach causes the CDN to retrieve the new asset immediately.
- You can purge cached content from the edge nodes, which refreshes the content on the next client request.

27

Interact with Azure Content Delivery Networks by using .NET (1 of 3)

You can use the Azure CDN Library for .NET to automate creation and management of CDN profiles and endpoints. Install the `Microsoft.Azure.Management.Cdn` directly from the Visual Studio Package Manager console or with the .NET Core CLI.

Common actions

- Create a CDN client
- List CDN profiles and endpoints
- Create CDN profiles and endpoints
- Purge an endpoint

28

Interact with Azure Content Delivery Networks by using .NET (2 of 3)

Create a CDN client

```
static void Main(string[] args)
{
    // Create CDN client
    CdnManagementClient cdn = new CdnManagementClient(new TokenCredentials(authResult.AccessToken))
        { SubscriptionId = subscriptionId };
}
```


29

Interact with Azure Content Delivery Networks by using .NET (3 of 3)

List CDN profiles and endpoints

```
private static void ListProfilesAndEndpoints(CdnManagementClient cdn)
{
    // List all the CDN profiles in this resource group
    var profileList = cdn.Profiles.ListByResourceGroup(resourceGroupName);
    foreach (Profile p in profileList)
    {
        Console.WriteLine("CDN profile {0}", p.Name);

        //List all the CDN endpoints on this CDN profile
        Console.WriteLine("Endpoints:");
        var endpointList = cdn.Endpoints.ListByProfile(p.Name, resourceGroupName);
        foreach (Endpoint e in endpointList)
        {
            Console.WriteLine("-{0} ({1})", e.Name, e.HostName);
        }
        Console.WriteLine();
    }
}
```


30

Summary and knowledge check

In this module, you learned how to:

- Explain how the Azure Content Delivery Network works and how it can improve the user experience.
- Control caching behavior and purge content.
- Perform actions on Azure CDN by using the Azure CDN Library for .NET.

1. When publishing a website through Azure CDN, the files on that site are cached until their time-to-live (TTL) expires. What is the default TTL for large file optimizations?

31

Discussion and lab

32

Group discussion questions

- You are building a cloud-native application using Azure that users in three regions will use: North America, Asia, and West Europe. You decide to implement a caching strategy with Redis. Describe what would be a good way to determine in which region the Redis cache should be physically located and why.
- Describe how an Azure CDN works.
- Can you describe a scenario where you would recommend Azure CDN over Azure Cache for Redis?

33

Lab 12: Enhance a web application by using the Azure Content Delivery Network

In this lab, you will implement the Azure Content Delivery Network capabilities to provide a caching solution based on customer locations. The lab configures a storage account for image and video files, which are impacted the most by the latency issues. You will use the Azure Content Delivery Network to implement the caching solution to aid in reducing latency for these image and video files.

http://aka.ms/az204labs

- Exercise 1: Create Azure resources
- Exercise 2: Configure Content Delivery Network and endpoints
- Exercise 3: Upload and configure static web content
- Exercise 4: Use Content Delivery Network endpoints

34

35