**Create serverless applications**

**Choose the best Azure service to automate your business processes**

* Logic Apps
* Microsoft Power Automate
* WebJobs
* Azure Functions

**Design-first technologies**

They both include user interfaces in which you can draw out the workflow

* **Logic Apps** is a service within Azure that you can use to automate, orchestrate, and integrate disparate components of a distributed application. By using the design-first approach in Logic Apps, you can draw out complex workflows that model complex business processes.

You can create or edit a workflow in JSON

One reason why Logic Apps is so good at integration is that over 200 connectors are included. You can create your own connector if your system exposes a REST API.

* **Microsoft Power Automate** is a service that you can use to create workflows even when you have no development or IT Pro experience. You can create workflows that integrate and orchestrate many different components by using the website or the Microsoft Power Automate mobile app.

There are four different types of flow that you can create:

* + **Automated** a flow that is started by a trigger from some event.
  + **Button**
  + **Scheduled**
  + **Business process** a flow that models a business process such as the stock ordering process or the complaints procedure. The flow process can have: notification to required people; with their approval recorded; calendar dates for steps; and recorded time of flow steps.

*Under the hood, Microsoft Power Automate is built on Logic Apps. This fact means that Power Automate supports the same range of connectors and actions*



**Code-first technologies**

This is the case when you need more control over the performance of your workflow or need to write custom code as part of the business process.

* **WebJobs and the WebJobs SDK** WebJobs are a part of the Azure App Service that you can use to run a program or script automatically.  
  There are two kinds of WebJob:
  + **Continuous** run in a continuous loop. For example, you could use a continuous WebJob to check a shared folder for a new photo.
  + **Triggered** run when you manually start them or on a schedule.

The SDK includes a range of classes, such as **JobHostConfiguration** and **HostBuilder**, which   
reduce the amount of code required to interact with the Azure App Service. The WebJobs SDK only supports C# and the NuGet package manager.

* **Azure Functions** is a simple way for you to run small pieces of code in the cloud, without having to worry about the infrastructure required to host that code.

*In addition, with the consumption plan option, you only pay for the time when the code runs.* Azure automatically scales your function in response to the demand from users.

When you create an Azure Function, you can start by writing the code for it in the portal.

Alternatively, if you need source code management, you can use GitHub or Azure DevOps Services.

* + **HTTPTrigger**
  + **TimerTrigger**
  + **BlobTrigger**
  + **CosmosDBTrigger**



**How to choose a service**

**Create serverless logic with Azure Functions  
  
Execution time**

By default, functions have a timeout of 5 minutes. This timeout is configurable to a maximum of 10 minutes. If your function requires more than 10 minutes to execute, you can host it on a VM.   
If your service is initiated through an HTTP request and you expect that value as an HTTP response, the timeout is further restricted to 2.5 minutes. Finally, there's also an option called **Durable Functions** that allows you to orchestrate the executions of multiple functions without any timeout.

**Execution frequency**

If you expect your function to be executed continuously by multiple clients, it would be prudent to estimate the usage and calculate the cost of using functions accordingly. It might be cheaper to host your service on a VM.  
While scaling, only one function app instance can be created every 10 seconds, for up to 200 total instances. Keep in mind, each instance can service multiple concurrent executions, so there is no set limit on how much traffic a single instance can handle. Different types of triggers have different scaling requirements, so research your choice of trigger and investigate its limits.

**What is a function app?**

Functions are hosted in an execution context called a function app. You define function apps to logically group and structure your functions and a compute resource in Azure.

**Choose a service plan**

Function apps may use one of two types of service plans. The first service plan is the **Consumption service plan.**The plan that you choose when using the Azure serverless application platform. The Consumption service plan provides automatic scaling and bills you when your functions are running. The Consumption plan comes with a configurable timeout period for the execution of a function. By default, it is 5 minutes, but may be configured to have a timeout as long as 10 minutes.  
**Azure App Service plan** allows you to avoid timeout periods by having your function run continuously on a VM that you define. When using an App Service plan, you are responsible for managing the app resources the function runs on, so this is technically not a serverless plan. However, it may be a better choice if your functions are used continuously or if your functions require more processing power or execution time than the Consumption plan can provide.

**Storage account requirements**

When you create a function app, it must be linked to a storage account. You can select an existing account or create a new one. The function app uses this storage account for internal operations such as logging function executions and managing execution triggers.

**Triggers**

Functions are event driven, which means they run in response to an event.  
Azure supports triggers for the following services.

| **Service** | **Trigger description** |
| --- | --- |
| Blob storage | Starts a function when a new or updated blob is detected. |
| Azure Cosmos DB | Start a function when inserts and updates are detected. |
| Event Grid | Starts a function when an event is received from Event Grid. |
| HTTP | Starts a function with an HTTP request. |
| Microsoft Graph Events | Starts a function in response to an incoming webhook from the Microsoft Graph. Each instance of this trigger can react to one Microsoft Graph resource type. |
| Queue storage | Starts a function when a new item is received on a queue. The queue message is provided as input to the function. |
| Service Bus | Starts a function in response to messages from a Service Bus queue. |
| Timer | Starts a function on a schedule. |

### Bindings

Bindings are a declarative way to connect data and services to your function. Bindings know how to talk to different services, which means you don't have to write code in your function to connect to data sources and manage connections. The platform takes care of that complexity for you as part of the binding code. Each binding has a direction - your code reads data from input bindings, and writes data to output bindings. Each function can have zero or more bindings to manage the input and output data processed by the function.

A trigger is a special type of input binding that has the additional capability of initiating execution.

### Secure HTTP triggers

HTTP triggers let you use API keys to block unknown callers by requiring the key to be present on each request. When you create a function, you select the authorization level. By default, it's set to Function, which requires a function-specific API key, but it can also be set to Admin to use a global "master" key, or Anonymous to indicate that no key is required.

Because we specified Function when we created this function, we will need to supply the key when we send the HTTP request. You can send it as a query string parameter named code, or as an HTTP header (preferred) named x-functions-key.

**Execute an Azure Function with triggers**

**What is a timer trigger?**

A timer trigger is a trigger that executes a function at a consistent interval. To create a timer trigger, you need to supply two pieces of information.

* **A Timestamp** parameter name, which is simply an identifier to access the trigger in code.
* **A Schedule**, which is a CRON expression that sets the interval for the timer.

**What is a CRON expression?**

A CRON expression is a string that consists of six fields that represent a set of times. The order of the six fields in Azure is: {second} {minute} {hour} {day} {month} {day of the week}.

For example, a CRON expression to create a trigger that executes every five minutes looks like:

0 \*/5 \* \* \* \*

| **To build a CRON expression, you need to have a basic understanding of some of the special characters.** | | |
| --- | --- | --- |
| **Special character** | **Meaning** | **Example** |
| \* | Selects every value in a field | An asterisk "\*" in the day of the week field means *every* day. |
| , | Separates items in a list | A comma "1,3" in the day of the week field means just Mondays (day 1) and Wednesdays (day 3). |
| - | Specifies a range | A hyphen "10-12" in the hour field means a range that includes the hours 10, 11, and 12. |
| / | Specifies an increment | A slash "\*/10" in the minutes field means an increment of every 10 minutes. |

When you put all the fields together, the expression is read as "on the first second, of every fifth minute of every hour, of every day, of every month".

**Execute an Azure function with an HTTP request**

**What is an HTTP trigger Authorization level?**

An HTTP triggerAuthorization level is a flag that indicates if an incoming HTTP request needs an API key for authentication reasons.

**There are three Authorization levels:**

* Function
* Anonymous
* Admin

The Function and Admin levels are "key" based. To send an HTTP request, you must supply a key for authentication. There are two types of keys: function and host. The difference between the two keys is their scope. **Function keys are specific to a function. Host keys apply to all functions inside the function app.** If your Authorization level is set to Function, you can use either a function or a host key. If your Authorization level is set to Admin, you must supply a host key.  
The Anonymous level means that there's no authentication required. We use this level in our exercise.

**Execute an Azure function when a blob is created**

**What is Azure Blob storage?**

Azure Blob storage is an object storage solution that's designed to store large amounts of unstructured data. For example, Azure Blob storage is great at doing things like:

* Storing files
* Serving files
* Streaming video and audio
* Logging data

There are three types of blobs: block blobs, append blobs, and page blobs.

* **Block blobs** are the most common type. They allow you to store text or binary data efficiently.
* **Append blobs** are like block blobs, but they're designed more for append operations like creating a log file that's being constantly updated.
* **Page blobs** are made up of pages and are designed for frequent random read and write operations.

**How to create a blob trigger**

One setting that you'll want to look at is the Path. The Path tells the blob trigger where to monitor to see if a blob is uploaded or updated. By default, the Path value is:

samples-workitems/{name}

Let's break down this concept into two pieces: samples-workitems and {name}. The first part, samples-workitems, represents the blob container that the trigger monitors. The second part, {name} means that every type of file will cause the trigger to invoke the function. The function is invoked because there's no filter. For example, we could make the trigger invoke the function only when a PNG file is added by using syntax like:

samples-workitems/{name}.png

The last significant piece of information with this concept is the text name. The name represents a parameter in your Azure function that receives the name of the added file.

**True or false: an Azure Function can have multiple triggers associated with it?  
R:** *Every Azure Function must have exactly one trigger associated with it. If you want to use multiple triggers, you must create multiple functions.*

**Chain Azure Functions together using input and output bindings**



*You can see that we can't add more than one trigger. In fact, to change the trigger for our function, we would have to first delete the trigger, and create a new one. However, the Inputs and Outputs sections of this page display a plus sign (+) to add more bindings so we can accept more than one input value, and emit more than one output value.*

**What is a binding?**

Provide a declarative way to connect to data from within your code.   
This is powerful because you can connect to your data sources without having to code specific connection logic (like database connections or web API interfaces).

**Types of bindings**

* **Input binding** - An input binding is a connection to a data source. Our function can read data from these inputs.
* **Output binding** - An output binding is a connection to a data destination. Our function can write data to these destinations.

**Types of supported bindings**

* Blob Storage
* Azure Service Bus Queues
* Azure Cosmos DB
* Azure Event Hubs
* External Files
* External Tables
* HTTP endpoints

*A binding type can be used as an input, an output or both.*

**Binding properties**

Three properties are required in all bindings. You may have to supply additional properties based on the type of binding and storage you are using.

* **Name** - Defines the function parameter through which you access the data. For example, in a queue input binding, this is the name of the function parameter that receives the queue message content.
* **Type** - Identifies the type of binding, i.e., the type of data or service we want to interact with.
* **Direction** - Indicates the direction data is flowing, i.e., is it an input or output binding?

Additionally, most binding types also need a fourth property:

* **Connection** - Provides the name of an app setting key that contains the connection string. Bindings use connection strings stored in app settings to keep secrets out of the function code. This makes your code more configurable and secure.

{  
 "name": "headshotBlob",  
 "type": "blob",  
 "path": "thumbnail-images/{filename}",  
 "connection": "HeadshotStorageConnection",  
 "direction": "in"  
 }

The path property is required when using the Blob trigger, and should be provided in the style shown here, with curly braces around the filename portion of the path. This creates a **binding expression** that allows you to reference the blob's name in other bindings and in your function's code.

**Write data with output bindings**

As with input bindings, there are multiple types of output bindings. However not all types support both input and output. You'll use them anytime you want to send or store data. Here, we'll look at the types that support output bindings and when to use them.

**Output binding types**[**https://docs.microsoft.com/en-us/learn/modules/chain-azure-functions-data-using-bindings/6-write-data-with-output-bindings-portal-lesson**](https://docs.microsoft.com/en-us/learn/modules/chain-azure-functions-data-using-bindings/6-write-data-with-output-bindings-portal-lesson)

**Combining input and output bindings**

It's possible to apply multiple bindings to a single function. This allows you to define both input and output bindings, and the input and output can even be the same binding type.

**Input binding types**[**https://docs.microsoft.com/en-us/azure/azure-functions/functions-triggers-bindings?tabs=csharp#supported-bindings**](https://docs.microsoft.com/en-us/azure/azure-functions/functions-triggers-bindings?tabs=csharp#supported-bindings)

**What is a binding expression?**

A binding expression is specialized text in function.json, function parameters, or code that is evaluated when the function is invoked to yield a value. For example, if you have a Service Bus Queue binding, you could use a binding expression to get the name of the queue from App Settings.

**Types of binding expressions**

* App settings
* Trigger file name
* Trigger metadata
* JSON payloads
* New GUID
* Current date and time

Most expressions are identified by wrapping them in curly braces. However, app setting binding expressions are wrapped in percent signs rather than curly braces. For example if the blob output binding path is %Environment%/newblob.txt and the Environment app setting value is Development, a blob will be created in the Development container.

**Add an Azure Cosmos DB input binding**

In the Document ID field, enter {id}.

This syntax is known as a binding expression. The function is triggered by an HTTP request that uses a query string to specify the ID to look up.

An incoming HTTP request triggers the function, and an id query parameter is passed to the Azure Cosmos DB input binding. If the database finds a document that matches this ID, the bookmark parameter will be set to the located document. In that case, you construct a response that contains the URL value found in the bookmarked document. If no document is found matching this key, you would respond with a payload and status code that tells the user the bad news.

**Add an Azure Queue Storage output binding**

Azure Queue storage is a service for storing messages that can be accessed from anywhere in the world. The size of a single message can be as much as 64 KB, and a queue can contain millions of messages - up to the total capacity of the storage account in which it is defined. T



*The only task you performed was to create a queue binding. You never created the queue explicitly. You are witnessing the power of bindings! As the following notification declares, the queue is automatically created for you if it doesn't exist.*



In this exercise, we expanded your knowledge of bindings to output bindings, writing data to your Azure Cosmos DB. We went further and added another output binding to post messages to an Azure queue. This demonstrates the true power of bindings to help you shape and move data from incoming sources to a variety of destinations. We haven't written any database code or had to manage connection strings ourselves. Instead, we configured bindings declaratively and let the platform take care of securing connections, scaling our function, and scaling our connections.

**Create a long-running serverless workflow with Durable Functions**

**Durable Functions**

Is an extension of Azure Functions that enables you to perform long-lasting, stateful operations in Azure. Azure provides the infrastructure for maintaining state information. You can use Durable Functions to orchestrate a long-running workflow. Using this approach, you get all the benefits of a serverless hosting model, while letting the Durable Functions framework take care of activity monitoring, synchronization, and runtime concerns.

*Durable Functions is an extension of Azure Functions****. Whereas Azure Functions operate in a stateless environment, Durable Functions can retain state between function calls****. This approach enables you to simplify complex stateful executions in a serverless-environment.*

***Some benefits of using Durable Functions include:***

* They enable you to write event driven code. A durable function can wait asynchronously for one or more external events, and then perform a series of tasks in response to these events.
* You can chain functions together. You can implement common patterns such as fan-out/fan-in, which uses one function to invoke others in parallel, and then accumulate the results.
* You can orchestrate and coordinate functions, and specify the order in which functions should execute.
* The state is managed for you. You don't have to write your own code to save state information for a long-running function.

**An orchestration function provides these extra benefits:**

* You can define the workflows in code. You don't need to write a JSON description or use a workflow design tool.
* Functions can be called both synchronously and asynchronously. Output from the called functions is saved locally in variables and used in subsequent function calls.
* Azure checkpoints the progress of a function automatically when the function awaits. Azure may choose to dehydrate the function and save its state while the function waits, to preserve resources and reduce costs. When the function starts running again, Azure will rehydrate it and restore its state.

**Function types**

* **Client** functions are the entry point for creating an instance of a Durable Functions orchestration. They can run in response to an event from many sources, such as a new HTTP request arriving, a message being posted to a message queue, an event arriving in an event stream. You can write them in any of the supported languages.
* **Orchestrator** functions describe how actions are executed, and the order in which they are run. You write the orchestration logic in code (C# or JavaScript).
* **Activity** functions are the basic units of work in a durable function orchestration. An activity function contains the actual work performed by the tasks being orchestrated.

**Application patternsApplication patterns**

**Function chaining** - Executes a sequence of functions in a specified order. The output of one function is applied to the input of the next function in the sequence. The output of the final function is used to generate a result.

**Fan out/fan in** - This pattern runs multiple functions in parallel and then waits for all the functions to finish. The results of the parallel executions can be aggregated or used to compute a final result. 

**Async HTTP APIs** - This pattern addresses the problem of coordinating state of long-running operations with external clients. An HTTP call can trigger the long-running action. Then, it can redirect the client to a status endpoint. The client can learn when the operation is finished by polling this endpoint.



**Monitor -** This pattern implements a recurring process in a workflow, possibly looking for a change in state. For example, you could use this pattern to poll until specific conditions are met.



**Human interaction** - This pattern combines automated processes that also involve some human interaction. A manual process within an automated process is tricky because people aren't as highly available and as responsive as most computers. Human interaction can be incorporated using timeouts and compensation logic that runs if the human fails to interact correctly within a specified response time. An approval process is an example of a process that involves human interaction.



**Comparison with Logic Apps**



**Description of the design approval process**

1. A project design is submitted.
2. An approval task is allocated to a manager, so they can review the project design proposal.
3. The project design proposal is rejected or approved.
4. An escalation task is allocated if the approval task isn't completed within a pre-defined time limit.

**The following table shows how the workflow steps can be mapped to the function types we use in a Durable Functions workflow.**

****

**How to control long running tasks using durable timers & How to add an escalation path based on the timer**

**Timers in Durable Functions**

Durable Functions provides timers for use in the orchestrator functions, which you can use to implement delays or set up timeouts for asynchronous actions. You should use durable timers in orchestrator functions instead of the setTimeout() and setInterval() functions.

You create a durable timer by calling the createTimer() method of the DurableOrchestrationContext. This method returns a task that resumes on a specified date and time.

**Using timers for delay**

*You should always use currentUtcDateTime to obtain the current date and time, instead of Date.now or Date.UTC.*

*Durable Functions enables you to implement long-running workflows without requiring that you maintain state information manually. Azure provides the infrastructure in which Durable Functions run. You focus on the logic for the functions that perform the tasks in your workflow.*

*In this module we learned about Durable Functions and how to orchestrate our simple design proposal approvals process. Using durable timers, we can add an escalation path to our workflows, especially for those long-running, indeterminate tasks.*

**Develop, test, and publish Azure Functions by using Azure Functions Core Tools**

The Azure Functions Core Tools are command-line utilities that enable you to develop and run functions locally and publish them to Azure.

**Create and run Azure Functions locally by using the Core Tools**

The Azure Functions Core Tools let you develop and run functions on your local computer from the command line.

**What are the Azure Functions Core Tools?**

The Azure Functions Core Tools are a set of command-line tools that you can use to develop and test Azure Functions on your local computer.

1. Generate the files and folders you need to develop functions on your local computer
2. Run your functions locally so you can test and debug them
3. Publish your functions to Azure

**Function apps and functions projects**

Every function published to Azure belongs to a function app: a collection of functions that are published together into the same environment. All of the functions in an app share a common set of configuration values, and must all be built for the same language runtime. Each function app is an Azure resource that can be configured and managed independently.

When you develop functions locally, you work within a functions project: a folder that contains the code and configuration files that define your functions. A functions project on your computer is equivalent to a function app in Azure, and can contain multiple functions that use the same language runtime.

To get started developing, you need to create a functions project folder that's organized correctly. Every new function you add to the project requires additional code and configuration that must be complete and correctly structured, or your functions will not be able to run. If you wanted to, you could become familiar with the names and contents of the files needed in a functions project folder and create them yourself, but doing so would be time-consuming and error-prone.

With the Azure Functions Core Tools, you'll never need to do this! You can use the Core Tools to generate function projects and functions from scratch.

**When you create a new functions project two most critical project files are always present:**

* **host.json** stores runtime configuration values, such as logging options, for the function app. The settings stored in this file are used both when running functions locally and in Azure.
* **local.settings.json** stores configuration values that only apply to the function app when it is run locally with the Core Tools. This file contains two kinds of settings:
  + **local runtime settings** used to configure the local functions runtime itself
  + **custom application settings** which you can add and configure based on your app's needs and can be accessed and used by all the functions in the app.

**Run functions locally**

Functions aren't programs that can be run on their own: they must be hosted by the functions host. The host is what powers everything outside of your function code: it loads configuration, listens for triggers and HTTP requests, starts the worker process for the language your functions are written in, writes log output and more. In Azure, function apps run the function host automatically when they

You can use the Core Tools to run your own instance of the functions host and try out your functions locally before you publish them. By running your functions before publishing them, you can make sure your configuration and code loads correctly and test out your functions by making real HTTP calls to them without the need for Azure resources.

**Exercise - Create a function locally by using the Core Tools**

mkdir ~/loan-wizard

cd ~/loan-wizard

func init

func new

code .

***Replace the full contents of index.js***

func start

Ctrl+C

func start &> ~/output.txt &

curl "http://localhost:7071/api/simple-interest" -w "\n"

curl "http://localhost:7071/api/simple-interest?principal=5000&rate=.035&term=36" -w "\n"

pkill func

code ~/output.txt

**Create a function app**

Before you can use the Core Tools to publish a project, you need to create a function app in Azure. This is not a capability of the Core Tools: creating function apps is one of the responsibilities of the Azure management tools, which include the Azure portal, Azure CLI and Azure PowerShell.

*If you already have a local functions project you want to publish, make sure to create the function app with the same language runtime. If you try to deploy a project to an app with a different runtime, publishing will halt with an error.*

**Publish to Azure**

To publish a functions project to Azure, run **func azure functionapp publish <app\_name>** from the functions project folder. **<app\_name>** is the name of the target function app in Azure, not the name of your project folder, which can be different.

*The Core Tools don't ask you to sign in to Azure. Instead, they access your subscriptions and resources by loading your session information from the Azure CLI or Azure PowerShell. If you don't have an active session in one of those tools, publishing will fail. It's possible to publish from the Core Tools without the Azure CLI or Azure PowerShell, but it's much easier if you do have them, and we recommend you install one or the other and log in before trying to publish.*

**Things to know**

* The Core Tools do not validate or test your functions code during publishing.
* When you publish, any functions already present in the target app are stopped and deleted before the contents of your project are deployed. You can't combine functions from multiple projects into one app by publishing them in sequence - all of the functions you want in the app must be in one project.
* Publishing to Azure does not create any kind of relationship between the local project and the target function app. You can publish a single functions project to multiple function apps. You can also re-publish a project to the same app repeatedly as you work on your code.
* The invocation URLs displayed after you publish may include a code parameter in the query string, as in the screenshot above. By default, HTTP functions created by the Core Tools are configured with an authorization level of function, meaning they require callers to provide a secret key in the request headers or query string. The Core Tools includes the key in the query string of the displayed URL for your convenience.

**Publish a function to Azure by using the Core Tools**

RESOURCEGROUP=learn-1a731473-8658-4cd9-b51b-57b5612f8a15

STORAGEACCT=learnstorage$(openssl rand -hex 5)

FUNCTIONAPP=learnfunctions$(openssl rand -hex 5)

az storage account create \

--resource-group "$RESOURCEGROUP" \

--name "$STORAGEACCT" \

--kind StorageV2 \

--location centralus

az functionapp create \

--resource-group "$RESOURCEGROUP" \

--name "$FUNCTIONAPP" \

--storage-account "$STORAGEACCT" \

--runtime node \

--consumption-plan-location centralus \

--functions-version 2

1. The three lines at the top create shell variables with values that we use repeatedly in the following commands. For resource group, we specify the group created for you by the sandbox. The storage account and function app names include $(openssl rand -hex 5), which generates a random 5-character string, to ensure that the names meet the requirement of being globally unique.
2. az storage account create creates an Azure storage account that will be used by the function app. A storage account is a separate Azure resource that needs to be created before the function app can be created.
3. az functionapp create creates the function app. Our new app uses the node (JavaScript) runtime, and runs on the serverless, pay-as-you-go consumption billing plan.

cd ~/loan-wizard

func azure functionapp publish "$FUNCTIONAPP"

**Develop, test, and deploy an Azure Function with Visual Studio**

[**https://docs.microsoft.com/en-us/learn/modules/develop-test-deploy-azure-functions-with-visual-studio/2-develop-and-test-azure-functions-locally**](https://docs.microsoft.com/en-us/learn/modules/develop-test-deploy-azure-functions-with-visual-studio/2-develop-and-test-azure-functions-locally)

**Monitor GitHub events by using a webhook with Azure Functions**

**Webhooks** offer a lightweight mechanism for apps to be notified by another service when something of interest happens via an HTTP endpoint. You can use a webhook to trigger an Azure function, and then analyze the message, to determine what happened and how to respond.

In GitHub, webhooks can be set up on an organization or a specific repository. The webhook will be triggered each time one or more subscribed events occurs. For example, the **Gollum** event allows you to listen for wiki updates; specifically creation and updates for a wiki page.

**Setting up a webhook**

Setting up a webhook is a two-step process. You specify how you want your webhook to behave through GitHub and what events it should listen to. Then you set up your function in Azure Functions to receive and manage the payload received from the webhook.

**Webhooks require a couple of configuration options before you can use them. We'll go through each of these settings next.**

* **Payload URL** The payload URL is the URL of the server that will receive the webhook POST requests. Each event type has a specific payload format. That payload contains information about the event that triggered the webhook.
* **Content Type** Webhooks can be delivered using two different content types:
  + The application/json content type delivers the JSON payload directly as the body of the POST request.
  + The application/x-www-form-urlencoded content type sends the JSON payload as a form parameter, called payload.
* **Events** Are at the center of webhooks. Events occur whenever actions are taken in the repository. When the event occurs, the webhook fires off and calls the URL that you specify, sending along the payload and event information to your URL. For example, to respond whenever an issue is raised in a repository
  + Click Let me select individual events
  + Select Issues.
  + Make sure you select Active to receive issue events for triggered webhooks.

To listen for updates to the wiki for the repository, select the Wiki checkbox; this event is the Gollum event mentioned earlier.

<https://docs.microsoft.com/en-us/learn/modules/monitor-github-events-with-a-function-triggered-by-a-webhook/5-exercise-setup-webhook-for-github-repo>

**Secure Webhook payloads with a secret**

Once your function is configured to receive payloads, it will listen for any payload sent to the endpoint you configured. For security reasons, you might want to limit requests to those coming from GitHub. There are a few ways to go about this. For example, you could opt to approve requests from GitHub's IP address. An easier method is to set up a secret token and validate the request using this token.

**Enable automatic updates in a web application using Azure Functions and SignalR Service**

<https://docs.microsoft.com/en-us/learn/modules/automatic-update-of-a-webapp-using-azure-functions-and-signalr/>

**Expose multiple Azure Function apps as a consistent API by using Azure API Management**

**The Azure API Management (APIM)**

Is a fully managed cloud service that you can use to publish, secure, transform, maintain, and monitor APIs. API Management handles all the tasks involved in mediating API calls, including request authentication and authorization, rate limit and quota enforcement, request and response transformation, logging and tracing, and API version management. APIM enables you to create and manage modern API gateways for existing backend services.

**APIM Consumption Tier**

The consumption tier uses the same underlying service components as the previous tiers, but employs an entirely different architecture based on shared, dynamically allocated resources. The consumption tier aligns perfectly with serverless computing models; there is no infrastructure to manage, no idle capacity, high-availability, automatic scaling, and usage-based pricing, all of which make it an especially good choice for solutions that involve exposing serverless resources as APIs.

**How does API Management help?**

* Client apps are coupled to the API expressing business logic, not the underlying technical implementation with individual microservices. You can change the location and definition of the services without necessarily reconfiguring or updating the client apps.
* API Management acts as an intermediary. It forwards requests to the right microservice, wherever it is located, and returns responses to users. Users never see the different URIs where microservices are hosted.
* You can use API Management policies to enforce consistent rules on all microservices in the product. For example, you can transform all XML responses into JSON, if that is your preferred format.
* Policies also enable you to enforce consistent security requirements.

**Connect your services together**

**Choose a messaging model in Azure to loosely connect your services**

* Azure Storage queues

Azure Event Hubs

* Azure Event Grid
* Azure Service Bus.

**Choose whether to use messages or events**

**What is a message?**

* A message contains raw data, produced by one component, that will be consumed by another component.
* A message contains the data itself, not just a reference to that data.
* The sending component expects the message content to be processed in a certain way by the destination component. The integrity of the overall system may depend on both sender and receiver doing a specific job.

**What is an event?**

Events are lighter weight than messages, and are most often used for broadcast communications. The components sending the event are known as publishers, and receivers are known as subscribers.

With events, receiving components will generally decide in which communications they are interested, and will "subscribe" to those events. The subscription is managed by an intermediary, like **Azure Event Grid or Azure Event Hubs.** When publishers send an event, the intermediary will route that event to interested subscribers. This pattern is known as a **"publish-subscribe architecture."** It's not the only way to deal with events, but it is the most common.

* An event is a lightweight notification that indicates that something happened.
* The event may be sent to multiple receivers, or to none at all.
* Events are often intended to "fan out," or have a large number of subscribers for each publisher.
* The publisher of the event has no expectation about the action a receiving component takes.
* Some events are discrete units and unrelated to other events.
* Some events are part of a related and ordered series.

**How to choose messages or events**

* Events are more likely to be used for broadcasts and are often ephemeral, meaning a communication might not be handled by any receiver if none is currently subscribing.
* Messages are more likely to be used where the distributed application requires a guarantee that the communication will be processed.

**Does the sending component expect the communication to be processed in a particular way by the destination component?  
R:** If the answer is yes, choose to use a message. If the answer is no, you may be able to use events.

**Choose a message-based delivery with queues**

**What is Azure Queue Storage?**

**Queue storage** is a service that uses **Azure Storage** to store large numbers of messages that can be securely accessed from anywhere in the world using a simple **REST-based interface**. Queues can contain millions of messages, limited only by the capacity of the storage account that owns it.

**What is Azure Service Bus Queues?**

**Service Bus** is a message broker system intended for enterprise applications. These apps often utilize multiple communication protocols, have different data contracts, higher security requirements, and can include **both cloud and on-premises services**. Service Bus is built on top of a dedicated messaging infrastructure designed for exactly these scenarios.

**What are Azure Service Bus Topics?**

**Azure Service Bus** topics are like queues, but can have multiple subscribers. When a message is sent to a topic instead of a queue, multiple components can be triggered to do their work.  
Internally, **topics use queues**. **When you post to a topic, the message is copied and dropped into the queue for each subscriptio**n. The queue means that the message copy will stay around to be processed by each subscription branch even if the component processing that subscription is too busy to keep up.

**Message delivery guarantees**

* At-Least-Once Delivery:
* At-Most-Once Delivery:
* First-In-First-Out (FIFO):

**Transactional support**

Message transactions succeed or fail as a single unit - just like in the database world. If the credit card details message delivery fails, so will the order details message.

**Use Service Bus topics if you:**

* Need multiple receivers to handle each message

**Use Service Bus queues if you:**

* Need an At-Most-Once delivery guarantee.
* Need a FIFO guarantee.
* Need to group messages into transactions.
* Want to receive messages without polling the queue.
* Need to provide a role-based access model to the queues.
* Need to handle messages larger than 64 KB but less than 256 KB.
* Queue size will not grow larger than 80 GB.
* Want to publish and consume batches of messages.

**Use Queue storage if you:**

* Need an audit trail of all messages that pass through the queue.
* Expect the queue to exceed 80 GB in size.
* Want to track progress for processing a message inside of the queue.

**Use Storage queues** when you want a simple and easy-to-code queue system. For more advanced needs, use **Service Bus queues**. If you have multiple destinations for a single message, but need queue-like behavior, use **Service Bus topics.**

**What is Azure Event Grid?**

Is a fully-managed **event routing service** running on top of Azure Service Fabric. Event Grid distributes events from different sources, such as Azure Blob storage accounts or Azure Media Services, to different handlers, such as Azure Functions or Webhooks. Event Grid was created to make it easier to build event-based and serverless applications on Azure.

There are several concepts in Azure Event Grid that connect a source to a subscriber:

* **Events:** What happened.
* **Event sources:** Where the event took place.
* **Topics:** The endpoint where publishers send events.
* **Event subscriptions:** The endpoint or built-in mechanism to route events, sometimes to multiple handlers. Subscriptions are also used by handlers to filter incoming events intelligently.
* **Event handlers:** The app or service reacting to the event.



**What is an event?**

Events are the data messages passing through Event Grid that describe what has taken place. Each event is self-contained, can be up to 64 KB, and contains several pieces of information based on a schema defined by Event Grid:



| **WHAT IS AN EVENT?** | |
| --- | --- |
| **Field** | **Description** |
| **topic** | The full resource path to the event source. Event Grid provides this value. |
| **subject** | Publisher-defined path to the event subject. |
| **id** | The unique identifier for event. |
| **eventType** | One of the registered event types for this event source. This is a value you can create filters against, e.g. CustomerCreated, BlobDeleted, HttpRequestReceived, etc. |
| **eventTime** | The time the event was generated based on the provider's UTC time. |
| **data** | Specific information that is relevant to the type of event. For example, an event about a new file being created in Azure Storage has details about the file, such as the lastTimeModified value. Or, an Event Hubs event has the URL of the Capture file. This field is optional. |
| **dataVersion** | The schema version of the data object. The publisher defines the schema version. |
| **metadataVersion** | The schema version of the event metadata. Event Grid defines the schema of the top-level properties. Event Grid provides this value. |

*Event Grid sends an event to indicate something has happened or changed. However, the actual object that was changed is not part of the event data. Instead, a URL or identifier is often passed to reference the changed object.*

**What is an event source?**

Event sources are responsible for sending events to Event Grid. Each event source is related to one or more event types. For example, Azure Storage is the event source for blob created events. IoT Hub is the event source for device created events.

Azure Event Hub has the concept of an event publisher which is often confused with the event source. A publisher to Event Hub is the user or organization that decides to send events to Event Grid. For example, Microsoft publishes events for several Azure services.

**What is an event topic?**

Event topics categorize events into groups. Topics are represented by a public endpoint and are where the event source sends events to.

For example, consider an application that sends events related to modifying user accounts and processing orders. It's unlikely any event handler wants both categories of events. Create two custom topics and let event handlers subscribe to the one that interests them. Event subscribers can filter for the event types they want from a specific topic.

* **System topics** are built-in topics provided by Azure services. You don't see system topics in your Azure subscription because the publisher owns the topics, but you can subscribe to them. To subscribe, you provide information about the resource you want to receive events from. As long as you have access to the resource, you can subscribe to its events.
* **Custom topics** are application and third-party topics. When you create or are assigned access to a custom topic, you see that custom topic in your subscription.

**What is an event subscription?**

Event Subscriptions define which events on a topic an event handler wants to receive. A subscription can also filter events by their type or subject, so you can ensure an event handler only receives relevant events.

**What is an event handler?**

An event handler (sometimes referred to as an event "subscriber") is any component (application or resource) that can receive events from Event Grid.

**Should you use Event Grid?**

* **Simplicity:** It is straightforward to connect sources to subscribers in Event Grid.
* **Advanced filtering:** Subscriptions have close control over the events they receive from a topic.
* **Fan-out:** You can subscribe to an unlimited number of endpoints to the same events and topics.
* **Reliability:** Event Grid retries event delivery for up to 24 hours for each subscription.
* **Pay-per-event:** Pay only for the number of events that you transmit.

**What is Azure Event Hubs?**

Event Hubs is an intermediary for the publish-subscribe communication pattern. Unlike Event Grid, however, it is optimized for extremely high throughput, a large number of publishers, security, and resiliency.

**Partitions**

As Event Hubs receives communications, it divides them into partitions. Partitions are buffers into which the communications are saved. Because of the event buffers, events are not completely ephemeral, and an event isn't missed just because a subscriber is busy or even offline. The subscriber can always use the buffer to "catch up." By default, events stay in the buffer for 24 hours before they automatically expire.

The buffers are called partitions because the data is divided amongst them. Every event hub has at least two partitions, and each partition has a separate set of subscribers.

**Choose Event Hubs if:**

* You need to support authenticating a large number of publishers.
* You need to save a stream of events to Data Lake or Blob storage.
* You need aggregation or analytics on your event stream.
* You need reliable messaging or resiliency.

Otherwise, if you need a simple event publish-subscribe infrastructure, with trusted publishers (for instance, your own web server), you should choose Event Grid.

Event Hubs lets you build a big data pipeline capable of processing millions of events per second with low latency. It can handle data from concurrent sources and route it to a variety of stream-processing infrastructures and analytics services. **It enables real-time processing and supports repeated replay of stored raw data.**

**What is a queue?**

A queue is a simple temporary storage location for messages. A sending component adds a message to the queue. A destination component picks up the message at the front of the queue. Under ordinary circumstances, each message is received by only one receiver.



A queue responds to high demand like this without needing to add resources to the system. However, for messages that need to be handled relatively quickly, adding additional instances of your destination component can allow them to share the load. Each message would be handled by only one instance. This is an effective way to scale your entire application while only adding resources to the components that actually need it.

**What is a topic?**

A topic is similar to a queue but can have multiple subscriptions. This means that multiple destination components can subscribe to a single topic, so each message is delivered to multiple receivers. Subscriptions can also filter the messages in the topic to receive only messages that are relevant. Subscriptions provide the same decoupled communications as queues and respond to high demand in the same way. Use a topic if you want each message to be delivered to more than one destination component.



*Topics are not supported in the Basic pricing tier.*

**What is a relay?**

A relay is an object that performs synchronous, two-way communication between applications. Unlike queues and topics, it is not a temporary storage location for messages. Instead, it provides bidirectional, unbuffered connections across network boundaries such as firewalls. Use a relay when you want direct communications between components as if they were located on the same network segment but separated by network security devices.

**Key advantages of Service Bus queues include:**

* Supports larger messages sizes of 256 KB (standard tier) or 1MB (premium tier) per message versus 64 KB
* Supports both at-most-once and at-least-once delivery - choose between a very small chance that a message is lost or a very small chance it is handled twice
* Guarantees first-in-first-out (FIFO) order - messages are handled in the same order they are added (although FIFO is the normal operation of a queue, it is not guaranteed for every message)
* Can group multiple messages into a transaction - if one message in the transaction fails to be delivered, all messages in the transaction will not be delivered
* Supports role-based security
* Does not require destination components to continuously poll the queue

**Advantages of storage queues:**

* Supports unlimited queue size (versus 80-GB limit for Service Bus queues)
* Maintains a log of all messages

**Create a Service Bus namespace**

In Azure Service Bus, a namespace is a container, with a unique fully qualified domain name, for queues, topics, and relays. You must start by creating the namespace.

Each namespace has primary and secondary shared access signature encryption keys. To gain access to the objects within the namespace, a sending or receiving component must provide these keys when it connects.

**Connection strings and keys**

Source components and destination components both need two pieces of information to connect to a queue in a Service Bus namespace:

* **The location of the Service Bus namespace, also known as an endpoint:** The location is specified as a fully qualified domain name within the servicebus.windows.net domain. For example: pizzaService.servicebus.windows.net.
* **An access key:** Service Bus restricts access to queues, topics, and relays by requiring an access key.

**Set filters on subscriptions**

If you want to control that specific messages sent to the topic are delivered to particular subscriptions, you can place filters on each subscription in the topic. In the pizza application, for instance, our storefronts are running Universal Windows Platform (UWP) applications. Each store can subscribe to the "OrderCancellation" topic but filter for its own StoreId. We save internet bandwidth because we are not sending unnecessary messages to distant store locations. Meanwhile, the payment processing component subscribes to all our cancellation messages.

**Filters can be one of three types:**

* **Boolean Filters:** The TrueFilter ensures that all messages sent to the topic are delivered to the current subscription. The FalseFilter ensures that none of the messages are delivered to the current subscription. (This effectively blocks or switches off the subscription.)
* **SQL Filters:** A SQL filter specifies a condition by using the same syntax as a WHERE clause in a SQL query. Only messages that return True when evaluated against this subscription will be delivered to the subscribers.
* **Correlation Filters:** A correlation filter holds a set of conditions that are matched against the properties of each message. If the property in the filter and the property on the message have the same value, it is considered a match.

*For our StoreId filter, we could use a SQL filter. SQL filters are the most flexible, but they're also the most computationally expensive and could slow down our Service Bus throughput. In this case, we choose a correlation filter instead.*

**True or false: you can add a message to an Azure Service Bus queue that is 2 MB in size.  
R:** *An Azure Storage queue message must be smaller than 64 KB. A service bus queue can be up to 256 KB for standard tier, and 1MB for the premium tier.*

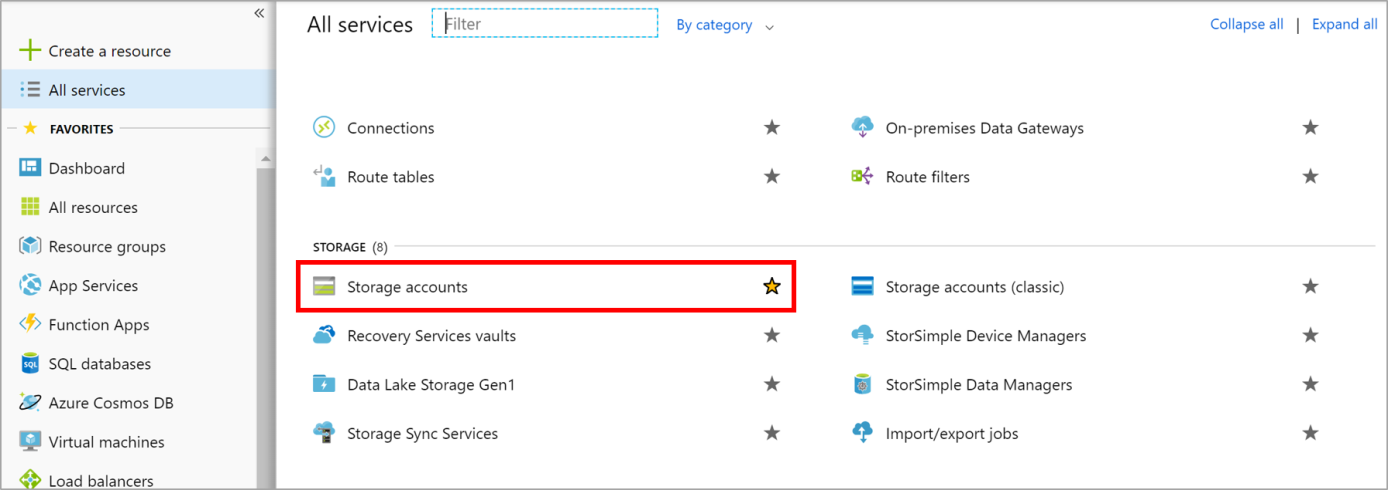
**Communicate between applications with Azure Queue storage**

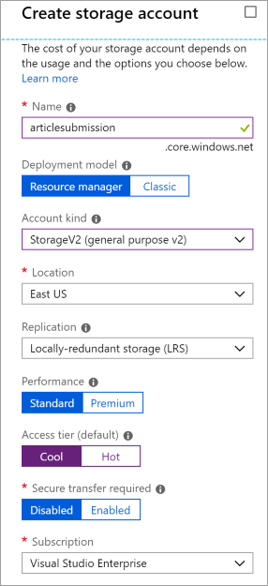
**What is Azure Queue storage?**

**Azure Queue storage is an Azure service that implements cloud-based queues. Each queue maintains a list of messages. Application components access a queue using a REST API or an Azure-supplied client library.**



**Creating a storage account**





**Queues are only available as part of Azure general-purpose storage accounts (v1 or v2). You cannot add them to Blob storage accounts.**

**Settings for queues**

* The Access tier setting which is shown for StorageV2 accounts applies only to Blob storage and does not affect queues.
* You should choose a location that is close to either the source components or destination components or (preferably) both.
* Data is always replicated to multiple servers to guard against disk failures and other hardware problems. You have a choice of replication strategies: Locally Redundant Storage (LRS) is low-cost but vulnerable to disasters that affect an entire data center while Geo-Redundant Storage (GRS) replicates data to other Azure data centers. Choose the replication strategy that meets your redundancy needs.
* The performance tier determines how your messages are stored: Standard uses magnetic drives while Premium uses solid-state drives. Choose Standard if you expect peaks in demand to be short. Consider Premium if queue length sometimes becomes long and you need to minimize the time to access messages.
* Require secure transfer if sensitive information may pass through the queue. This setting ensures that all connections to the queue are encrypted using Secure Sockets Layer (SSL).

**Identify a queue**

To access a queue, you need three pieces of information:

* **Storage account name**
* **Queue name**
* **Authorization token**

**Queue identity**

Every queue has a name that you assign during creation. The name must be unique within your storage account but doesn't need to be globally unique (unlike the storage account name).

The combination of your storage account name and your queue name uniquely identifies a queue.

**Authorization Type**

* **Azure Active Directory** You can use role-based authentication and identify specific clients based on AAD credentials.
* **Shared Key** Sometimes referred to as an account key, this is an encrypted key signature associated with the storage account. Every storage account has two of these keys that can be passed with each request to authenticate access. Using this approach is like using a root password - it provides full access to the storage account.
* **Shared access signature** A shared access signature (SAS) is a generated URI that grants limited access to objects in your storage account to clients. You can restrict access to specific resources, permissions, and scope to a data range to automatically turn off access after a period of time.

**Accessing queues**

You access a queue using a **REST API**. To do this, you'll use a URL that combines the name you gave the storage account with the domain **queue.core.windows.net** and the **path to the queue you want to work with**. For example: http**://<storage account>.**queue.core.windows.net/**<queue name>.** An Authorization header must be included with every request. The value can be any of the three authorization styles.

**Programmatically access a queue**

Queues hold messages - packets of data whose shape is known to the sender application and receiver application. The sender creates the queue and adds a message. The receiver retrieves a message, processes it, and then deletes the message from the queue. The following illustration shows a typical flow of this process.



Notice that get and delete are separate operations. **This arrangement handles potential failures in the receiver and implements a concept called at-least-once delivery.** After the receiver gets a message, **that message remains in the queue but is invisible for 30 seconds**. If the **receiver crashes or experiences a power failure** during processing, then **it will never delete the message from the queue**. **After 30 seconds**, the message **will reappear** in the queue and another instance of the receiver can process it to completion.

***While the total queue size can be up to 500 TB,*** *the individual messages in it* ***can only be up to 64 KB in size (48 KB when using Base64 encoding)****. If you need a larger payload you can combine queues and blobs – passing the URL to the actual data (stored as a Blob) in the message.* ***This approach would allow you to enqueue up to 200 GB for a single item****.*

There are several other commands available that you can try with the tools - check out both **az storage queue --help** and **az storage message --help** to explore them.

**Suppose you work for a government agency that plans the long-term expansion of the highway system. You receive traffic data from thousands of sensors and analyze it to make your recommendations. The amount of incoming data varies throughout the day; for example, it spikes during the morning and evening commuting hours. True or false: a server-side architecture consisting of an Azure Queue connected to a single virtual machine is a reasonable choice for this workload?  
R:** The queue will handle spikes in traffic and ensure no data is lost. If the VM cannot keep up with the flow of incoming messages, it will process the message backlog during low-traffic times.

**Enable reliable messaging for Big Data applications using Azure Event Hubs**

**What is an Azure Event Hub?**

Azure Event Hubs is a cloud-based, event-processing service that can receive and process millions of events per second. Event Hubs acts as a front door for an event pipeline, to receive incoming data and stores this data until processing resources are available.



Azure Event Hubs sits between these two entities to divide the production (from the publisher) and consumption (to a subscriber) of an event stream. This decoupling helps to manage scenarios where the rate of event production is much higher than the consumption.

**Consumer groups**

An Event Hub consumer group represents a specific view of an Event Hub data stream. By using separate consumer groups, multiple subscriber apps can process an event stream independently, and without affecting other apps. However, the use of many consumer groups isn't a requirement, and for many apps, the single default consumer group is sufficient.

**Define an Event Hubs namespace**

An Event Hubs namespace is a containing entity for managing one or more Event Hubs. Creating an Event Hubs namespace typically involves the following configuration:

Certain settings such as namespace capacity (configured using throughput units), pricing tier, and performance metrics are defined at the namespace level. These settings apply to all the Event Hubs within that namespace. If you don't define these settings, a default value is used: 1 for capacity and Standard for pricing tier.

Select a unique name for the namespace. The namespace is accessible through this URL: namespace.servicebus.windows.net

**Scale in Event Hubs is controlled by how many throughput units you purchase, with each throughput unit entitling you to 1 Megabyte per second, or 1000 events per second of ingress and twice that volume in egress. Event Hubs can automatically scale up throughput units when you reach the throughput limit if you use the Auto-Inflate feature;**

**Define the following optional properties:**

* Enable Kafka. This option enables Kafka apps to publish events to the Event Hub.
* Make this namespace zone redundant. Zone-redundancy replicates data across separate data centers with their independent power, networking, and cooling infrastructures.
* Enable Auto-Inflate and Auto-Inflate Maximum Throughput Units. Auto-Inflate provides an automatic scale-up option by increasing the number of throughput units up to a maximum value. This option is useful to avoid throttling in situations when incoming or outgoing data rates exceed the currently set number of throughput units.

**The following parameters are required to create an Event Hub:**

* Event Hub name - Event Hub name that is unique within your subscription and:
  + Is between 1 and 50 characters long.
  + Contains only letters, numbers, periods, hyphens, and underscores.
  + Starts and ends with a letter or number.
* Partition Count - The number of partitions required in an Event Hub (between 2 and 32). The partition count should be directly related to the expected number of concurrent consumers and can't be changed after the hub has been created. The partition separates the message stream so that consumer or receiver apps only need to read a specific subset of the data stream. If not defined, this value defaults to 4.
* Message Retention - The number of days (between 1 and 7) that messages will remain available if the data stream needs to be replayed for any reason. If not defined, this value defaults to 7.

*You can also optionally configure an Event Hub to stream data to an Azure Blob storage or Azure Data Lake Store account.*

**How can you test Event Hub resilience?**

Azure Event Hubs keeps received messages from your sender application, even when the hub is unavailable. Messages received after the hub becomes unavailable are successfully transmitted to our application as soon as the hub becomes available.

To test this functionality, you can use the Azure portal to disable your Event Hub.

When you re-enable your Event Hub, you can rerun your receiver application, and use Event Hubs metrics for your namespace to check whether all sender messages are successfully transmitted and received.

**Other useful metrics available in the Event Hubs include:**

* Throttled Requests: The number of throttled requests because the throughput exceeded unit usage.
* ActiveConnections: The number of active connections on a namespace or Event Hub.
* Incoming/Outgoing Bytes: The number of bytes sent to/received from the Event Hubs service over a specified period.

*Azure Event Hubs provides big data applications the capability to process large volume of data. It can also scale out during exceptionally high-demand periods, as required. Azure Event Hubs decouples the sending and receiving messages to manage the data processing. This helps eliminate the risk of overwhelming consumer application and data loss because of any unplanned interruption*

**Applications that publish messages to Azure Event Hub very frequently will get the best performance using Advanced Message Queuing Protocol (AMQP) because it establishes a persistent socket.  
R:** *Publishers can use either HTTPS or AMQP. AMQP opens a socket and can send multiple messages over that socket.*

**By default, how many partitions will a new Event Hub have?  
R:** *Event Hubs default to 4 partitions. Partitions are the buckets within an Event Hub. Each publication will go into only one partition. Each consumer group may read from one or more than one partition.*

**What is the maximum size for a single publication (individual or batch) that is allowed by Azure Event Hub?  
R:** *The maximum size for a single publication (individual or batch) that is allowed by Azure Event Hub is 1 MB.*

**Store data in Azure**

**Choose a data storage approach in Azure**

**Classify your data**

Application data can be classified in one of three ways:

* **Structured:** AKArelational data, is data that adheres to a strict schema, so all of the data has the same fields or properties. The shared schema allows this type of data to be easily searched with query languages such as SQL.   
  Structured data is often stored in database tables with rows and columns with key columns to indicate how one row in a table relates to data in another row of another table.
* **Semi-structured** Is less organized than structured data, and is not stored in a relational format, as the fields do not neatly fit into tables, rows, and columns. Semi-structured data contains tags that make the organization and hierarchy of the data apparent - for example, key/value pairs. Semi-structured data is also referred to as non-relational or NoSQL data. The expression and structure of the data in this style is defined by a serialization language.
* **Unstructured** The organization of unstructured data is ambiguous. Unstructured data is often delivered in files, such as photos or videos. The video file itself may have an overall structure and come with semi-structured metadata, but the data that comprises the video itself is unstructured. Therefore, photos, videos, and other similar files are classified as unstructured data.
  + Media files, such as photos, videos, and audio files
  + Office files, such as Word documents
  + Text files
  + Log files

*When deciding what storage solution to use, think about how your data will be used. How often will your data be accessed? Is your data read-only? Does query time matter? The answers to these questions will help you decide on the best storage solution for your data.*

**What is a transaction?**

A transaction is a logical group of database operations that execute together.

* **Atomicity** means a transaction must execute exactly once and must be atomic; either all of the work is done, or none of it is. Operations within a transaction usually share a common intent and are interdependent.
* **Consistency** ensures that the data is consistent both before and after the transaction.
* **Isolation** ensures that one transaction is not impacted by another transaction.
* **Durability** means that the changes made due to the transaction are permanently saved in the system. Committed data is saved by the system so that even in the event of a failure and system restart, the data is available in its correct state.

**OLTP vs OLAP**

* **OLTP (Online Transaction Processing)** systems commonly support lots of users, have quick response times, and handle large volumes of data. They are also highly available (meaning they have very minimal downtime), and typically handle small or relatively simple transactions.
* **OLAP (Online Analytical Processing**) systems commonly support fewer users, have longer response times, can be less available, and typically handle large and complex transactions.

**Product catalog data**

**Recommendation: Azure Cosmos DB**

* Supports semi-structured data, or NoSQL data, by design.
* Supports SQL for queries and every property is indexed by default.
* Is also ACID-compliant, so you can be assured that your transactions are completed according to those strict requirements.
* Enable you to replicate your data anywhere in the world with the click of a button.

**Photos and videos**

**Recommendation: Azure Blob storage**

* Supports storing files such as photos and videos
* Works with Azure Content Delivery Network (CDN)
* You can also move images from the hot storage tier to the cool or archive storage tier, to reduce costs and focus throughput on the most frequently viewed images and videos.

***Why not other Azure services?***

*You could upload your images to Azure App Service, so that the same server that is running your app is serving up your images. This solution would work if you didn't have many files. But if you have lots of files, and a global audience, you'll get more performance results by using Azure Blob storage with Azure CDN.*

**Business data**

**Recommendation: Azure SQL Database**

**Create an Azure Storage account**

**What is Azure Storage?**

Azure selected four of these data services and placed them together under the name Azure Storage. The four services are Azure Blobs, Azure Files, Azure Queues, and Azure Tables. The following illustration shows the elements of Azure Storage**.**

**What is a storage account?**

A storage account is a container that groups a set of Azure Storage services together. Only data services from Azure Storage can be included in a storage account (Azure Blobs, Azure Files, Azure Queues, and Azure Tables). The following illustration shows a storage account containing several data services.



**Azure data services like Azure SQL and Azure Cosmos DB are managed as independent Azure resources and cannot be included in a storage account**. The following illustration shows a typical arrangement: Blobs, Files, Queues, and Tables are inside storage accounts, while other services are not.



**Storage account settings**

* **Subscription:** The Azure subscription that will be billed for the services in the account.
* **Location:** The datacenter that will store the services in the account.
* **Performance:** Determines the data services you can have in your storage account and the type of hardware disks used to store the data. **Standard** allows you to have any data service (Blob, File, Queue, Table) and **uses magnetic disk drives. Premium** introduces additional services for storing data. For example, **storing unstructured object data as block blobs or append blobs, and specialized file storage used to store and create premium file shares. These storage accounts use solid-state drives (SSD) for storage.**
* **Replication:** Determines the strategy used to make copies of your data to protect against hardware failure or natural disaster. **At a minimum, Azure will automatically maintain three copies of your data** within the data center associated with the storage account. This is called **locally-redundant storage (LRS),** and guards against hardware failure but does not protect you from an event that incapacitates the entire datacenter. You can upgrade to one of the other options such **as geo-redundant storage (GRS)** to get replication at different datacenters across the world.
* **Access tier:** Controls how quickly you will be able to access the blobs in this storage account. Hot gives quicker access than Cool, but at increased cost. This applies only to blobs, and serves as the default value for new blobs.
* **Secure transfer required:** A security feature that determines the supported protocols for access. Enabled requires HTTPs, while disabled allows HTTP.
* **Virtual networks:** A security feature that allows inbound access requests only from the virtual network(s) you specify.

**How many storage accounts do you need?**

Do you have data that is specific to a country or region? If so, you might want to locate it in a data center in that country for performance or compliance reasons. You will need one storage account for each location.

Do you have some data that is proprietary and some for public consumption? If so, you could enable virtual networks for the proprietary data and not for the public data. This will also require separate storage accounts.

**Cost sensitivity**

A storage account by itself has no financial cost; however, the settings you choose for the account do influence the cost of services in the account. **Geo-redundant storage** costs more than locally-redundant storage. Premium performance and the Hot access tier increase the cost of blobs.

You can use multiple storage accounts to reduce costs. For example, you could partition your data into critical and non-critical categories. You could place your critical data into a storage account with **geo-redundant storage** and put your non-critical data in a different storage account with locally-redundant storage.

**Account kind**

Storage account kind is a set of policies that determine which data services you can include in the account and the pricing of those services. There are three kinds of storage accounts:

* **StorageV2 (general purpose v2):** the current offering that supports all storage types and all of the latest features
* **Storage (general purpose v1):** a legacy kind that supports all storage types but may not support all features
* **Blob storage:** a legacy kind that allows only block blobs and append blobs

**Network connectivity**

You can connect to your storage account either publicly, via public IP addresses or service endpoints, or privately, using a private endpoint.

* Public endpoint (all networks)
* Public endpoint (selected networks)
* Private endpoint

**Network routing**

Determine how to route your traffic as it travels from the source to its Azure endpoint. Microsoft network routing is recommended for most customers.

* Microsoft network routing (default)
* Internet routing

**Data protection**

* **Turn on soft delete for blobs**: Lets you recover your blob data in many cases where blobs or blob snapshots are deleted accidentally or overwritten.
* **Turn on soft delete for file shares:** File share soft delete lets you recover your blob data more easily at the folder level.

**Connect an app to Azure Storage**

**Explore Azure storage services**

*A single Azure subscription can host up to 200 storage accounts, each of which can hold 500 TB of data.*

**General-purpose v2 (GPv2)**

Support all of the latest features for blobs, files, queues, and tables. Pricing for GPv2 accounts has been designed to deliver the lowest per gigabyte prices.

**General-purpose v1 (GPv1)**

Provide access to all Azure Storage services but may not have the latest features or the lowest per gigabyte pricing. For example, cool storage and archive storage are not supported in GPv1. Pricing is lower for GPv1 transactions, so workloads with high churn or high read rates may benefit from this account type.

**Blob storage accounts**

A legacy account type, blob storage accounts support all the same block blob features as GPv2, but they are limited to supporting only block and append blobs. Pricing is broadly similar to pricing for general-purpose v2 accounts.

**Use the REST API**

The Storage REST APIs are accessible from anywhere on the Internet, by any app that can send an HTTP/HTTPS request and receive an HTTP/HTTPS response.

For example, if you wanted to list all the blobs in a container, you would send something like:

**GET** [**https://[url-for-service-account]/?comp=list&include=metadata**](https://[url-for-service-account]/?comp=list&include=metadata)

This would return an XML block with data specific to the account.

The REST endpoint is a combination of your storage account name, the data type, and a known domain. For example:

|  |  |
| --- | --- |
| **Data type** | **Example endpoint** |
| Blobs | https://[name].blob.core.windows.net/ |
| Queues | https://[name].queue.core.windows.net/ |
| Table | https://[name].table.core.windows.net/ |
| Files | https://[name].file.core.windows.net/ |

**Security access keys**

Each storage account has two unique access keys that are used to secure the storage account. If your app needs to connect to multiple storage accounts, your app will require an access key for each storage account.



**Connection strings**

The simplest way to handle access keys and endpoint URLs within applications is to use storage account connection strings. A connection string provides all needed connectivity information in a single text string

**Security**

Each storage account has two access keys. The reason for this is to allow keys to be rotated (regenerated) periodically as part of security best practice in keeping your storage account secure.

Rotating a key will invalidate the original key value immediately and will revoke access to anyone who obtained the key inappropriately.

**Shared access signatures (SAS)**

Access keys are the easiest approach to authenticating access to a storage account. However they provide full access to anything in the storage account, similar to a root password on a computer.

Storage accounts offer a separate authentication mechanism called shared access signatures that support expiration and limited permissions for scenarios where you need to grant limited access. **You should use this approach when you are allowing other users to read and write data to your storage account.** There are links to our documentation on this advanced topic at the end of the module.

**Secure your Azure Storage account**

* **Encryption at rest** All data written to Azure Storage is automatically encrypted by Storage Service Encryption (SSE) with a 256-bit Advanced Encryption Standard (AES) cipher, and is FIPS 140-2 compliant. SSE automatically encrypts data when writing it to Azure Storage. When you read data from Azure Storage, Azure Storage decrypts the data before returning it. This process incurs no additional charges and doesn't degrade performance. It can't be disabled.
* **Encryption in transit** Keep your data secure by enabling transport-level security between Azure and the client. Always use HTTPS to secure communication over the public internet. When you call the REST APIs to access objects in storage accounts, you can enforce the use of HTTPS by requiring secure transfer for the storage account. After you enable secure transfer, connections that use HTTP will be refused. This flag will also enforce secure transfer over SMB by requiring SMB 3.0 for all file share mounts.
* **CORS support** Azure Storage supports cross-domain access through cross-origin resource sharing (CORS). CORS uses HTTP headers so that a web application at one domain can access resources from a server at a different domain. By using CORS, web apps ensure that they load only authorized content from authorized sources. CORS support is an optional flag you can enable on Storage accounts. The flag adds the appropriate headers when you use HTTP GET requests to retrieve resources from the Storage account.
* **Role-based access control** Every request to a secure resource must be authorized. The service ensures that the client has the permissions required to access the data. You can choose from several access options. Arguably, the most flexible option is role-based access. To a security principal or a managed identity for Azure resources, you can assign RBAC roles that are scoped to a subscription, a resource group, a storage account, or an individual container or queue.
* **Auditing access** Auditing is another part of controlling access. You can audit Azure Storage access by using the built-in Storage Analytics service. Storage Analytics logs every operation in real time, and you can search the Storage Analytics logs for specific requests. Filter based on the authentication mechanism, the success of the operation, or the resource that was accessed.

**Understand storage account keys**

Azure Storage accounts can create authorized apps in Active Directory to control access to the data in blobs and queues. **This authentication approach is the best solution for apps that use Blob storage or Queue storage.**

For other storage models, clients can use a **shared key, or shared secret**. This authentication option is one of the easiest to use, and it supports blobs, files, queues, and tables. The client embeds the shared key in the **HTTP Authorization header** of every request, and the Storage account validates the key.

**Understand shared access signatures**

As a best practice, you shouldn't share storage account keys with external third-party applications. If these apps need access to your data, you'll need to secure their connections without using storage account keys.

For untrusted clients, use a shared access signature (SAS).

**Types of shared access signatures**

You can use a service-level SAS to allow access to specific resources in a storage account. You'd use this type of SAS, for example, to allow an app to retrieve a list of files in a file system, or to download a file. You can use an account-level SAS to allow the ability to create file systems.

**Understand Advanced Threat Protection for Azure Storage**



Azure Defender for Storage provides an extra layer of security intelligence that detects unusual and potentially harmful attempts to access or exploit storage accounts. This layer of protection allows you to address threats without being a security expert or managing security monitoring systems.

Security alerts are triggered when anomalies in activity occur.

**Explore security anomalies**

When storage activity anomalies occur, you receive an email notification with information about the suspicious security event. Details of the event include:



You can review and manage your current security alerts from Azure Security Center's Security alerts tile.

*Azure Storage provides a layered security model. Use this model to secure your storage accounts to a specific set of supported networks. When you set up network rules, only applications that request data over the specified networks can access your storage account.*

*Authorization is supported by a public preview of Azure Active Directory credentials (for blobs and queues), a valid account access key, or a shared access signature (SAS) token. Data encryption is enabled by default, and you can proactively monitor systems by using Advanced Threat Protection.*

**When configuring network access to your Azure Storage Account, what is the default network rule?  
R:** *Correct. A shared access signature is a string that contains a security token that can be attached to a URI. Use a shared access signature to delegate access to storage objects and specify constraints, such as the permissions and the time range of access.*

**Which Azure service detects anomalies in account activities and notifies you of potential harmful attempts to access your account?  
R***: Which Azure service detects anomalies in account activities and notifies you of potential harmful attempts to access your account?*

**Store application data with Azure Blob storage**

**Storage accounts, containers, and metadata**

In Blob storage, every blob lives inside a blob container. You can store an unlimited number of blobs in a container and an unlimited number of containers in a storage account. Containers are "flat" — they can only store blobs, not other containers.

Blobs and containers support metadata in the form of name-value string pairs. Your apps can use metadata for anything you like: a human-readable description of a blob's contents to be displayed by the app, a string that your app uses to determine how to process the blob's data, and so on.

*Blob storage does not provide any mechanism for searching or sorting blobs by metadata. For information about using Azure Cognitive Search, at the end of this module, see the Further Reading section.*

**Blob name prefixes (virtual directories)**

Technically, containers are "flat" and do not support any kind of nesting or hierarchy. But if you give your blobs hierarchical names that look like file paths (such as **finance/budgets/2017/q1.xls**), the API's listing operation can filter results to specific prefixes. This enables you to navigate the list as if it was a hierarchical system of files and folders.

**Which of the following describes a good strategy for creating storage accounts and blob containers for your application?  
R:** *Creating an Azure Storage account is an administrative activity and can be done prior to deploying an application. Container creation is lightweight and is often driven by run-time data which makes it a good activity to do in your application.*

**Deploy a website with Azure virtual machines**

**Compile a checklist for creating an Azure Virtual Machine**

**Start with the network**

Virtual networks (VNets) are used in Azure to provide private connectivity between Azure Virtual Machines and other Azure services. VMs and services that are part of the same virtual network can access one another. By default, services outside the virtual network cannot connect to services within the virtual network. You can, however, configure the network to allow access to the external service, including your on-premises servers.

**Segregate your network**

After deciding the virtual network address space(s), you can create one or more subnets for your virtual network. You do this to break up your network into more manageable sections. For example, you might assign 10.1.0.0 to VMs, 10.2.0.0 to back-end services, and 10.3.0.0 to SQL Server VMs.

***Azure reserves the first four addresses and the last address in each subnet for its use.***

**Secure the network**

By default, there is no security boundary between subnets, so services in each of these subnets can talk to one another. However, you can set up Network Security Groups (NSGs), which allow you to control the traffic flow to and from subnets and to and from VMs. NSGs act as software firewalls, applying custom rules to each inbound or outbound request at the network interface and subnet level. This allows you to fully control every network request coming in or out of the VM.

**Name the VM**

You can specify a name of up to 15 characters on a Windows VM and 64 characters on a Linux VM. A good convention is to include the following information in the name:

|  |  |  |
| --- | --- | --- |
| **Element** | **Example** | **Notes** |
| Environment | dev, prod, QA | Identifies the environment for the resource |
| Location | uw (US West), ue (US East) | Identifies the region into which the resource is deployed |
| Instance | 01, 02 | For resources that have more than one named instance (web servers, etc.) |
| Product or Service | service | Identifies the product, application, or service that the resource supports |
| Role | sql, web, messaging | Identifies the role of the associated resource |

**Determine the size of the VM**

The best way to determine the appropriate VM size is to consider the type of workload your VM needs to run. Based on the workload, you're able to choose from a subset of available VM sizes. Workload options are classified as follows on Azure:

|  |  |
| --- | --- |
| **Option** | **Description** |
| General purpose | General-purpose VMs are designed to have a balanced CPU-to-memory ratio. Ideal for testing and development, small to medium databases, and low to medium traffic web servers. |
| Compute optimized | Compute optimized VMs are designed to have a high CPU-to-memory ratio. Suitable for medium traffic web servers, network appliances, batch processes, and application servers. |
| Memory optimized | Memory optimized VMs are designed to have a high memory-to-CPU ratio. Great for relational database servers, medium to large caches, and in-memory analytics. |
| Storage optimized | Storage optimized VMs are designed to have high disk throughput and IO. Ideal for VMs running databases. |
| GPU | GPU VMs are specialized virtual machines targeted for heavy graphics rendering and video editing. These VMs are ideal options for model training and inferencing with deep learning. |
| High performance computes | High performance compute is the fastest and most powerful CPU virtual machines with optional high-throughput network interfaces. |

**What if my size needs change?**

The VM size can be changed while the VM is running, as long as the new size is available in the current hardware cluster the VM is running on.

*Be careful about resizing production VMs - they will be rebooted automatically which can cause a temporary outage and change some configuration settings such as the IP address.*

**Understanding the pricing model**

There are two separate costs the subscription will be charged for every VM

**Compute costs** - Compute expenses are priced on a per-hour basis but billed on a per-minute basis. For example, you are only charged for 55 minutes of usage if the VM is deployed for 55 minutes. You are not charged for compute capacity if you stop and deallocate the VM since this releases the hardware. The hourly price varies based on the VM size and OS you select. The cost for a VM includes the charge for the Windows operating system. Linux-based instances are cheaper because there is no operating system license charge.

***You might be able to save money by reusing existing licenses for Windows with the Azure Hybrid benefit.***

**Storage costs** - You are charged separately for the storage the VM uses. The status of the VM has no relation to the storage charges that will be incurred; even if the VM is stopped/deallocated and you aren’t billed for the running VM, you will be charged for the storage used by the disks.

**Storage for the VM**

Best practice is that all Azure virtual machines will have at least two virtual hard disks (VHDs). **The first disk stores the operating system, and the second is used as temporary storage**. You can add additional disks to store application data; the maximum number is determined by the VM size selection (typically two per CPU). It's common to create one or more data disks, particularly since the OS disk tends to be quite small. Also, separating out the data to different VHDs allows you to manage the security, reliability, and performance of the disk independently.

The data for each VHD is held in Azure Storage as page blobs, which allows Azure to allocate space only for the storage you use. It's also how your storage cost is measured; you pay for the storage you are consuming.

When you create disks, you will have two options for managing the relationship between the storage account and each VHD. You can choose either unmanaged disks or managed disks.

|  |  |
| --- | --- |
| **Option** | **Description** |
| **Unmanaged disks** | You are responsible for the storage accounts that are used to hold the VHDs that correspond to your VM disks. You pay the storage account rates for the amount of space you use. A single storage account has a fixed-rate limit of 20,000 I/O operations/sec. This means that a storage account is capable of supporting 40 standard virtual hard disks at full utilization. If you need to scale out with more disks, then you'll need more storage accounts, which can get complicated. |
| **Managed disks** | Are the newer and recommended disk storage model. They elegantly solve this complexity by putting the burden of managing the storage accounts onto Azure. You specify the size of the disk, up to 4 TB, and Azure creates and manages both the disk and the storage. You don't have to worry about storage account limits, which makes managed disks easier to scale out. |

**Describe the options available to create and manage an Azure Virtual Machine**

* **Azure Resource Manager -** Azure provides you with the option to create a template from which to create an exact copy of a VM.
  + 
* **Azure PowerShell** - Creating administration scripts is a powerful way to optimize your workflow. You can automate everyday, repetitive tasks, and after a script has been verified, it will run consistently, likely reducing errors. Azure PowerShell is ideal for one-off interactive tasks and/or the automation of repeated tasks.
* **Azure CLI** - The Azure CLI is Microsoft's cross-platform command-line tool for managing Azure resources such as virtual machines and disks from the command line. It's available for Windows, Linux and macOS, or in the browser using the Cloud Shell.
* **Programmatic (APIs)** - When it comes to more complex scenarios, where the creation and management of VMs form part of a larger application with complex logic, another approach is needed. You can interact with every type of resource in Azure **programmatically.**
* **Azure REST API** - The Azure Compute APIs give you programmatic access to virtual machines and their supporting resources
* **Azure Client SDK** - The Azure Client SDK encapsulates the Azure REST API, making it much easier for developers to interact with Azure.
* **Azure VM extensions** - Azure VM extensions are small applications that enable you to configure and automate tasks on Azure VMs after initial deployment. Azure VM extensions can be run with the Azure CLI, PowerShell, Azure Resource Manager templates, and the Azure portal. You bundle extensions with a new VM deployment, or run them against an existing system.
* **Azure Automation services** - Azure Automation enables you to integrate services that allow you to automate frequent, time-consuming, and error-prone management tasks with ease. These services include process automation, configuration management, and update management.

**What is an availability set?**

Allow you to create two or more virtual machines in different physical server racks in an Azure datacenter. **Microsoft guarantees a 99.95 percent SLA with an availability set for multiple-instance VMs deployed in an availability set.** That means that for the SLA to apply, there must be at least two instances of the VM deployed within an availability set.

**What is a fault domain?**

A logical representation of the physical rack in which a host computer is installed. By default, Azure assigns two fault domains to an availability set. If a problem occurs in one fault domain (one computer rack), the VMs in that fault domain will be affected, but VMs in the second fault domain will not. This protects you from unplanned maintenance events and unexpected downtime.

**What is an update domain?**

Designed to protect you from a situation where the host computer is being rebooted. When you create an availability set, Azure creates five update domains by default. These update domains are spread across the fault domains in the availability set.

If a reboot is required on computers in the availability set (whether host computers or VMs within the availability set), Azure will only reboot computers in one update domain at a time and it will wait 30 minutes for computers to recover from the reboot before it moves on to the next update domain. Update domains protect you from planned maintenance events.



**Failover across locations**

You can also replicate your infrastructure across sites to handle regional failover. Azure Site Recovery replicates workloads from a primary site to a secondary location. If an outage happens at your primary site, you can fail over to a secondary location. This failover enables users to continue to access your applications without interruption. You can then fail back to the primary location after it's up and running again. Azure Site Recovery is about replication of virtual or physical machines; it keeps your workloads available in an outage.

**Back up your virtual machines**

Azure Backup is a backup as a service offering that protects physical or virtual machines no matter where they reside: on-premises or in the cloud.

Azure Backup can be used for a wide range of data backup scenarios, such as:

* Files and folders on Windows OS machines (physical or virtual, local or cloud)
* Application-aware snapshots (Volume Shadow Copy Service)
* Popular Microsoft server workloads such as Microsoft SQL Server, Microsoft SharePoint, and Microsoft Exchange
* Native support for Azure Virtual Machines, both Windows, and Linux
* Linux and Windows 10 client machines

**Use Azure Backup**

Azure Backup uses several components that you download and deploy to each computer you want to back up. The component that you deploy depends on what you want to protect.

* Azure Backup agent
* System Center Data Protection Manager
* Azure Backup Server
* Azure Backup VM extension

Azure Backup uses a Recovery Services vault for storing the backup data. A vault is backed by Azure Storage blobs, making it a very efficient and economical long-term storage medium. With the vault in place, you can select the machines to back up, and define a backup policy (when snapshots are taken and for how long they’re stored).

**Suppose you want to run a network appliance on a virtual machine. Which workload option should you choose?  
R:** Compute optimized virtual machines are designed to have a high CPU-to-memory ratio. Suitable for medium traffic web servers, network appliances, batch processes, and application servers.

**Create a Linux virtual machine in Azure**

**Resources used in a Linux VM**

* A virtual machine that provides CPU and memory resources
* An Azure Storage account to hold the virtual hard disks
* Virtual disks to hold the OS, applications, and data
* A virtual network (VNet) to connect the VM to other Azure services or your on-premises hardware
* A network interface to communicate with the VNet
* An optional public IP address so you can access the VM

**Here are some guidelines based on the scenario you are targeting**

****

**Choosing storage options**

*There are two levels of SSD storage available: standard and premium. Choose Standard SSD disks if you have normal workloads but want better performance. Choose Premium SSD disks if you have I/O intensive workloads or mission-critical systems that need to process data very quickly.*

**By default, two virtual hard disks (VHDs) will be created for your Linux VM:**

* **The operating system disk:** This is your primary drive, and it has a maximum capacity of 2048 GB. It will be labeled as /dev/sda by default.
* **A temporary disk:** This provides temporary storage for the OS or any apps. On Linux virtual machines, the disk is /dev/sdb and is formatted and mounted to /mnt by the Azure Linux Agent. It is sized based on the VM size and is used to store the swap file.

***The temporary disk is not persistent. You should only write data to this disk that is not critical to the system.***

*Azure virtual disk sizes are measured in Gibibytes (GiB), which are not the same as Gigabytes (GB); one GiB is approximately 1.074 GB. Therefore, to obtain an approximate equivalent of your virtual disk size in GB, multiply the size in GiB by 1.074, and that will return a size in GB that is relatively close. For example, 32,767 GiB would be approximately 35,183 GB.*

**Unmanaged vs. managed disks**

* **With unmanaged disks**, you are responsible for the storage accounts that are used to hold the VHDs that correspond to your VM disks. You pay the storage account rates for the amount of space you use. A single storage account has a fixed rate limit of 20,000 I/O operations/sec. This means that a single storage account is capable of supporting 40 standard virtual hard disks at full throttle. If you need to scale out, then you need more than one storage account, which can get complicated.
* **Managed disks** are the newer and recommended disk storage model. They elegantly solve this complexity by putting the burden of managing the storage accounts onto Azure. You specify the disk type (Premium or Standard) and the size of the disk, and Azure creates and manages both the disk and the storage it uses. You don't have to worry about storage account limits, which makes them easier to scale out. They also offer several other benefits:
  + **Increased reliability:** Azure ensures that VHDs associated with high-reliability VMs will be placed in different parts of Azure Storage to provide similar levels of resilience.
  + **Better security:** Managed disks are real managed resources in the resource group. This means they can use role-based access control to restrict who can work with the VHD data.
  + **Snapshot support:** Snapshots can be used to create a read-only copy of a VHD. We recommend that you shut down the VM to clear out any processes that are in progress. Creating the snapshot only takes a few seconds. Once it's done, you can power on the VM and use the snapshot to create a duplicate VM to troubleshoot a production issue or roll back the VM to the point in time that the snapshot was taken.
  + **Backup support:** Managed disks can be automatically backed up to different regions for disaster recovery with Azure Backup without affecting the service of the VM.

**Decide an authentication method for SSH**

Before we can create a Linux virtual machine in Azure, we will need to think about remote access. We want to be able to sign in to our Linux web server to configure the software and perform maintenance. The default approach to administering Linux VMs hosted in Azure is SSH.

**What is SSH?**

Secure Shell (SSH) is an encrypted connection protocol that allows secure sign-ins over unsecured connections. SSH allows you to connect to a terminal shell from a remote location using a network connection.

There are two approaches we can use to authenticate an **SSH connection: username and password, or an SSH key pair.**

**Although SSH provides an encrypted connection, using passwords with SSH connections leaves the VM vulnerable to brute-force attacks of passwords.** A more secure and preferred method of connecting to a Linux VM with SSH is a public-private key pair, also known as SSH keys.

With an SSH key pair, you can sign in to Linux-based Azure virtual machines without a password. This is a more secure approach if you only plan to sign in to the VM from a few computers. **If you need to be able to access the Linux VM from a variety of locations, a username and password combination might be a better approach.** There are two parts to an SSH key pair: a public key and a private key.

* **The public key** is placed on your Linux VM or any other service that you wish to use with public-key cryptography. This can be shared with anyone.
* **The private key** is what you present to verify your identity to your Linux VM when you make an SSH connection. Consider this confidential information and protect this like you would a password or any other private data.

You can use the same single public-private key pair to access multiple Azure VMs and services.

* Create the SSH key pair

1. Run the following command in the Cloud Shell.
2. ssh-keygen -m PEM -t rsa -b 4096
3. Press Enter to accept the default location. The command creates two files: id\_rsa and id\_rsa.pub in the ~/.ssh directory. The files are overwritten if they exist.
4. Enter a passphrase that you'll remember. You'll need this passphrase when you use the SSH key to access the VM.

**Private key passphrase**

This passphrase is used to access the private SSH key file and is not the user account password.

**Use the SSH key when creating a Linux VM**

You can view the contents of the file in Azure Cloud Shell by running the following command:

**cat ~/.ssh/id\_rsa.pub**

To apply the SSH key while creating a new Linux VM, you will need to copy the contents of the public key and supply it to the Azure portal, or supply the public key file to the Azure CLI or Azure PowerShell command. We'll use this approach when we create our Linux VM.

**Azure VM IP addresses**

As we saw a moment ago, Azure VMs communicate on a virtual network. They can also have an optional public IP address assigned to them. With a public IP, we can interact with the VM over the Internet. Alternatively, we can set up a virtual private network (VPN) that connects our on-premises network to Azure - letting us securely connect to the VM without exposing a public IP. This approach is covered in another module and is fully documented if you are interested in exploring that option.

Public IP addresses in Azure are dynamically allocated by default. That means the IP address can change over time - for VMs the IP address assignment happens when the VM is restarted. You can pay more to assign static addresses, if you want to connect directly to an IP address and need to ensure that the IP address will not change.

**Connect to the VM with SSH**

To connect to the VM via SSH, you need the following items:

* Public IP address of the VM
* Username of the local account on the VM
* Public key configured in that account
* Access to the corresponding private key
* Port 22 open on the VM

**Open ports in Azure VMs**

By default, new VMs are locked down.

Apps can make outgoing requests, but the only inbound traffic allowed is from the virtual network (for example, other resources on the same local network) and from Azure Load Balancer (probe checks).

There are two steps for adjusting the configuration to support different protocols on the network. When you create a new VM, you have an opportunity to open a few common ports (RDP, HTTP, HTTPS, and SSH). However, if you require other changes to the firewall, you will need to adjust them manually.

The process for this involves two steps:

* Create a network security group.
* Create an inbound rule allowing traffic on the ports you need.

**What is a network security group?**

Virtual networks (VNets) are the foundation of the Azure networking model, and provide isolation and protection. Network security groups (NSGs) are the primary tool you use to enforce and control network traffic rules at the networking level. NSGs are an optional security layer that provides a software firewall by filtering inbound and outbound traffic on the VNet.

Security groups can be associated to a network interface (for per host rules), a subnet in the virtual network (to apply to multiple resources), or both levels.

**Security group rules**

NSGs use rules to allow or deny traffic moving through the network. Each rule identifies the source and destination address (or range), protocol, port (or range), direction (inbound or outbound), a numeric priority, and whether to allow or deny the traffic that matches the rule.



Each security group has a set of default security rules to apply the default network rules previously described. These default rules cannot be modified but can be overridden.

**How Azure uses network rules**

For inbound traffic, Azure processes the security group associated to the subnet, and then the security group applied to the network interface. Outbound traffic is handled in the opposite order (the network interface first, followed by the subnet).

***Keep in mind that security groups are optional at both levels. If no security group is applied, then all traffic is allowed by Azure. If the VM has a public IP, this could be a serious risk, particularly if the OS doesn't provide a built-in firewall.***

The rules are evaluated in **priority-order**, starting with the lowest priority rule. **Deny rules always stop the evaluation.** For example, if an outbound request is blocked by a network interface rule, any rules applied to the subnet will not be checked. In order for traffic to be allowed through the security group, it must pass through all applied groups.

**The last rule is always a Deny All rule.** This is a default rule added to every security group for both inbound and outbound traffic with a priority of 65500. That means to have traffic pass through the security group you must have an allow rule or it will be blocked by the default final rule.

***SMTP (port 25) is a special case. Depending on your subscription level and when your account was created, outbound SMTP traffic may be blocked. You can request to remove this restriction with business justification.***

**Update the NSG on the network interface**

Port 80 is open on the NSG applied to the subnet. But port 80 is blocked by the NSG applied to the network interface. Let's fix that so we can connect to the website.

**True or false: for security reasons, you must use an image from the official Azure Marketplace when creating a new virtual machine.  
R:** Azure lets you configure your virtual machines to meet your needs. This includes support for using your own VM images.

**What is the effect of the default network security settings for a new virtual machine?  
R:** Outbound requests are considered low risk, so they are allowed by default. Inbound traffic from within the virtual network is allowed. By placing a VM in a virtual network, the VM owner is implicitly opting-in to communication among the resources in the virtual network.

**Create a Windows virtual machine in Azure**

**Resources used in a Windows VM**

* A virtual machine that provides CPU and memory resources.
* An Azure Storage account to hold the virtual hard disks.
* Virtual disks to hold the OS, applications, and data.
* Virtual network (VNet) to connect the VM to other Azure services or your own on-premises hardware.
* A network interface to communicate with the VNet.
* A public IP address so you can access the VM. This is optional.

**Choose the VM image**

*You can also create and upload your own images. Check the documentation for more information.*

**Sizing your VM**

*There are quota limits on each subscription that can impact VM creation. By default, you cannot have more than 20 virtual cores across all VMs within a region. You can either split up VMs across regions or file an online request to increase your limits.*

****

**Mapping storage to disks**

**By default, two virtual hard disks (VHDs) will be created for your Windows VM:**

* The Operating System disk. This is your primary or C: drive and has a maximum capacity of 2048 GB.
* A Temporary disk. This provides temporary storage for the OS or any apps. It is configured as the D: drive by default and is sized based on the VM size, making it an ideal location for the Windows paging file.

*The temporary disk is not persistent. You should only write data to this disk that you are willing to lose at any time.*

**Opening ports in Azure VMs**

**The process for this involves two steps:**

* Create a Network Security Group.
* Create an inbound rule allowing traffic on port 20 and 21 for active FTP support, for example

**What is the final rule that is applied in every Network Security Group?  
R:** Deny All - This is a safe choice. It will block all traffic that you don't specifically allow.

**Manage resources in Azure**

**Control Azure services with the CLI**

The Azure CLI can also be used from a browser through the Azure Cloud Shell. In both cases, it can be used interactively or scripted

**How do you find the particular commands you need?**

Example - find the most popular commands related to the word blob  
**az find blob**

Example - Show me the most popular commands for an Azure CLI command group  
**az find "az vm"**

Example - Show me the most popular parameters and subcommands for an Azure CLI command  
**az find "az vm create"**

If you already know the name of the command you want, the --help argument for that command will get you more detailed information on the command, and for a command group, a list of the available subcommands.  
**az storage blob –help**

**How to create an Azure resource**

**Connect**

Since you're working with a local install of the Azure CLI, you'll need to authenticate before you can execute Azure commands, by using the Azure CLI login command.

**az login**

The Azure CLI will typically launch your default browser to open the Azure sign-in page. If this doesn't work, follow the command-line instructions and enter an authorization code at <https://aka.ms/devicelogin>

After a successful sign in, you'll be connected to your Azure subscription.

**Create**

**az group create --name <name> --location <location>**

**Verify**

**az group list**

To get a more concise view, you can format the output as a simple table:

**az account show --output table**

**Create an Azure website using the CLI**

Create several variables that you will use in later commands

**export RESOURCE\_GROUP=learn-5ba87f0c-8bd7-4fd4-8778-96f9e9645bf8  
export AZURE\_REGION=centralus  
export AZURE\_APP\_PLAN=popupappplan-$RANDOM  
export AZURE\_WEB\_APP=popupwebapp-$RANDOM**

If you have several items in the group list, you can filter the return values by adding a --query option.

**az group list --query "[?name == '$RESOURCE\_GROUP']"**

**Steps to create a service plan**

*When you run Web Apps using the Azure App Service, you pay for the Azure compute resources that are used by the app, and the resource costs depend on the App Service plan associated with your Web Apps. Service plans determine the region used for the app datacenter, number of VMs used, and pricing tier.*

Create an App Service plan to run your app. The following command specifies the free pricing tier, but you can run **az appservice plan create --help** to see the other pricing tiers

**az appservice plan create --name $AZURE\_APP\_PLAN --resource-group $RESOURCE\_GROUP --location $AZURE\_REGION --sku FREE**

Verify that the service plan was created successfully by listing all your plans in a table.

**az appservice plan list --output table**

To create the web app, you'll supply web app name and the name of the app plan you created above.

**az webapp create --name $AZURE\_WEB\_APP --resource-group $RESOURCE\_GROUP --plan $AZURE\_APP\_PLAN**

Verify that the app was created successfully by listing all your apps in a table.

**az webapp list --output table**

**Steps to deploy code from GitHub**

The final step is to deploy code from a GitHub repository to the web app

**az webapp deployment source config --name $AZURE\_WEB\_APP --resource-group $RESOURCE\_GROUP --repo-url "https://github.com/DiegoRomario/about-me" --branch master --manual-integration**

**Automate Azure tasks using scripts with PowerShell**

**How to create a resource group with Azure PowerShell**



1. **Import the Azure cmdlets.**
2. **Connect to your Azure subscription.**
3. **Create the resource group.**
4. **Verify that creation was successful.**

**Import the Azure cmdlets**

At startup, PowerShell loads only the core cmdlets by default. This means the cmdlets you need to work with Azure won't be loaded. The most reliable way to load the cmdlets you need is to import them manually at the start of your PowerShell session.

You use the Import-Module cmdlet to load modules. This cmdlet has many parameters to handle a variety of situations. For example, it can load multiple modules, a specific module version, part of a module, and so on.

For example, we can load all the cmdlets for Az with the following command in an elevated PowerShell session:

**Import-Module Az**

*If you find that you work with Azure PowerShell frequently, there are two ways you can automate the module-loading process. You can add an entry to your PowerShell profile to import the Azure module at startup or use the latest versions of PowerShell, which loads the containing module automatically when you use a cmdlet.*

**Connect**

When you are working with a local install of Azure PowerShell, you will need to authenticate before you can execute Azure commands. The Connect-AzAccount cmdlet prompts for your Azure credentials and then connects to your Azure subscription. It has many optional parameters, but if all you need is an interactive prompt, no parameters are needed:

**Connect-AzAccount**

You'll need to repeat these steps for every new PowerShell session you start since this module is not part of the core set.

**Work with subscriptions**

You can only be in one subscription at a time. Use **the Get-AzContext** cmdlet to determine which subscription is active. If it's not the correct one, you can change it.

Get a list of all subscription names in your account with the **Get-AzSubscription** command.

Change the subscription by passing the name of the one to select.

**Select-AzSubscription -SubscriptionId '53dde41e-916f-49f8-8108-558036f826ae'**

**Get a list of all resource groups**

**Get-AzResourceGroup**

To get a more concise view, you can send the output from the Get-AzResourceGroup to the Format-Table cmdlet using a pipe '|'.

**Get-AzResourceGroup | Format-Table or (| ft)**

**Create a resource group**

**New-AzResourceGroup -Name <name> -Location <location>**

**Verify the resources**

**Get-AzResource**

**You can also filter it to specific resource groups to only list resources associated with that group:**

**Get-AzResource -ResourceGroupName ExerciseResources**

**Create an Azure Virtual Machine**

**New-AzVm   
 -ResourceGroupName <resource group name>   
 -Name <machine name>   
 -Credential <credentials object>   
 -Location <location>   
 -Image <image name>**

You can supply these parameters directly to the cmdlet as shown above. Alternatively, other cmdlets can be used to configure the virtual machine, such as **Set-AzVMOperatingSystem, Set-AzVMSourceImage, Add-AzVMNetworkInterface**, and **Set-AzVMOSDisk.**

**Here's an example that strings the Get-Credential cmdlet together with the -Credential parameter:**

**New-AzVM -Name MyVm -ResourceGroupName ExerciseResources -Credential (Get-Credential) ...**

**Example: Getting information for a VM**

**$vm = Get-AzVM -Name MyVM -ResourceGroupName ExerciseResources**

The interesting thing is this is an object you can interact with. For example, you can take that object, make changes and then push changes back to Azure with the Update-AzVM command:

**$ResourceGroupName = "ExerciseResources"  
$vm = Get-AzVM -Name MyVM -ResourceGroupName $ResourceGroupName  
$vm.HardwareProfile.vmSize = "Standard\_DS3\_v2"  
Update-AzVM -ResourceGroupName $ResourceGroupName -VM $vm**

**Create a Linux VM with Azure PowerShell**

**New-AzVm -ResourceGroupName learn-c7434ead-1792-4e69-8578-a4ea8bc6c948 -Name "testvm-eus-01" -Credential (Get-Credential) -Location "East US" -Image UbuntuLTS -OpenPorts 22**

This will take a few minutes to complete. After completion, you can query it, and assign the VM object to a variable ($vm).

**$vm = (Get-AzVM -Name "testvm-eus-01" -ResourceGroupName learn-c7434ead-1792-4e69-8578-a4ea8bc6c948)**

Query the value to dump out the information about the VM.

**$vm**

You can reach into complex objects through a dot (".") syntax.

**$vm.StorageProfile.OsDisk**

To try out some more commands, let's delete the VM. We'll shut it down first.

**Stop-AzVM -Name $vm.Name -ResourceGroup $vm.ResourceGroupName**

Now, let's delete the VM by running the Remove-AzVM cmdlet.

**Remove-AzVM -Name $vm.Name -ResourceGroup $vm.ResourceGroupName**

Run this command to list all the resources in your resource group.

**Get-AzResource -ResourceGroupName $vm.ResourceGroupName | ft**

**Remove-AzVM** *command just deletes the VM. It doesn't clean up any of the other resources. At this point, we'd likely just delete the resource group itself, and be done with it.*

**What is a PowerShell script?**

A PowerShell script is a text file containing commands and control constructs. The commands are invocations of cmdlets. The control constructs are programming features like loops, variables, parameters, comments, etc. supplied by PowerShell.

PowerShell script files have a .ps1 file extension. You can create and save these files with any text editor.

*If you’re writing PowerShell scripts under Windows, you can use the Windows PowerShell Integrated Scripting Environment (ISE). This editor provides features such as syntax coloring and a list of available cmdlets.*



Once you have written the script, execute it from the PowerShell command line by passing the name of the file preceded by a dot and a backslash:

**.\myScript.ps1**

**Parameters**

When you execute a script, you can pass arguments on the command line. You can provide names for each parameter to help the script extract the values. For example:

**param([string]$location, [int]$size)**

Inside the script, you capture the values into variables. In this example, the parameters are matched by name:

**param([string]$location, [int]$size)**

You can omit the names from the command line. For example:

**.\setupEnvironment.ps1 5 "East US"**

Inside the script, you rely on position for matching when the parameters are unnamed:

**param([int]$size, [string]$location)**

**Plan and manage your Azure costs**

**What's the TCO Calculator?**

Helps you estimate the cost savings of operating your solution on Azure over time, instead of in your on-premises datacenter.

With the TCO Calculator, you enter the details of your on-premises workloads. Then you review the suggested industry average cost (which you can adjust) for related operational costs. These costs include electricity, network maintenance, and IT labor. You're then presented with a side-by-side report. Using the report, you can compare those costs with the same workloads running on Azure.



How does the TCO Calculator work?



**Define your workloads:** First, you enter the specifications of your on-premises infrastructure into the TCO Calculator, based on these four categories:

* Servers
  + This category includes operating systems, virtualization methods, CPU cores, and memory (RAM).
* Databases
  + This category includes database types, server hardware, and the Azure service you want to use, which includes the expected maximum concurrent user sign-ins.
* Storage
  + This category includes storage type and capacity, which includes any backup or archive storage.
* Networking
  + This category includes the amount of network bandwidth you currently consume in your on-premises environment.

Adjust assumptions

* specify whether your current on-premises licenses are enrolled for Software Assurance, which can save you money by reusing those licenses on Azure.
* specify whether you need to replicate your storage to another Azure region for greater redundancy.
* See the key operating cost assumptions across several different areas, which vary among teams and organizations.
  + Electricity price per kilowatt hour (KWh).
  + Hourly pay rate for IT administration.
  + Network maintenance cost as a percentage of network hardware and software costs.

**View the report**

Choose a time frame between one and five years. the TCO Calculator generates a report that's based on the information you've entered. Here's an example:



For each category (compute, datacenter, networking, storage, and IT labor), you can also view a side-by-side comparison of the cost breakdown of operating those workloads on-premises versus operating them on Azure. Here's an example:



**What types of Azure subscriptions can I use?**

* **Free trial** provides you with 12 months of popular free services, a credit to explore any Azure service for 30 days, and more than 25 services that are always free. Your Azure services are disabled when the trial ends or when your credit expires for paid products, unless you upgrade to a paid subscription.
* **Pay-as-you-go** enables you to pay for what you use by attaching a credit or debit card to your account. Organizations can apply for volume discounts and prepaid invoicing.
* **Member offers** your existing membership to certain Microsoft products and services might provide you with credits for your Azure account and reduced rates on Azure services. For example, member offers are available to Visual Studio subscribers, Microsoft Partner Network members, Microsoft for Startups members, and Microsoft Imagine members.

**How do I purchase Azure services?**

1. **Through an Enterprise Agreement** Larger customers, known as enterprise customers, can sign an Enterprise Agreement with Microsoft. This agreement commits them to spending a predetermined amount on Azure services over a period of three years. The service fee is typically paid annually. As an Enterprise Agreement customer, you'll receive the best customized pricing based on the kinds and amounts of services you plan on using.
2. **Directly from the web** here, you purchase Azure services directly from the Azure portal website and pay standard prices. You're billed monthly, as a credit card payment or through an invoice. This purchasing method is known as Web Direct.
3. **Through a Cloud Solution Provider** (CSP) is a Microsoft Partner who helps you build solutions on top of Azure. Your CSP bills you for your Azure usage at a price they determine. They also answer your support questions and escalate them to Microsoft, as needed.

**How can I estimate the total cost?**

The Pricing calculator displays Azure products in categories. You add these categories to your estimate and configure according to your specific requirements. You then receive a consolidated estimated price, with a detailed breakdown of the costs associated with each resource you added to your solution. You can export or share that estimate or save it for later. You can load a saved estimate and modify it to match updated requirements.



**What's the best way to ensure that the development team doesn't provision too many virtual machines at the same time?  
R:** Deallocate virtual machines when they're not in use. (*Although you can delete your virtual machines when they're not in use, you also lose any associated hard disks. It can take some time to re-create the environment at the start of each week.)*

**Control and organize Azure resources with Azure Resource Manager**

**Secure resources with role-based access control**

**RBAC** provides fine-grained access management for Azure resources, enabling you to grant users the specific rights they need to perform their jobs. RBAC is considered a core service and is included with all subscription levels at no cost.

**Using RBAC, you can:**

* Allow one user to manage VMs in a subscription, and another user to manage virtual networks.
* Allow a database administrator (DBA) group to manage SQL databases in a subscription.
* Allow a user to manage all resources in a resource group, such as VMs, websites, and virtual subnets.
* Allow an application to access all resources in a resource group.

**Best Practices for RBAC**

Here are some best practices you should use when setting up resources.

* Segregate duties within your team and grant only the amount of access to users that they need to perform their jobs. Instead of giving everybody unrestricted permissions in your Azure subscription or resources, allow only specific actions at a particular scope.
* When planning your access control strategy, grant users the lowest privilege level that they need to do their work.
* Use Resource Locks to ensure critical resources aren't modified or deleted (as you'll see in the next unit).

**Tags can be applied to any type of resource on Azure?  
R:** Not all resources support tags, so you will want to confirm that your resource type supports them.   
You can apply a maximum of 50 tags to a single resource or resource group in Azure.

**Move Azure resources to another resource group**

**Check the limitations on resource types**

After identifying the resource types of your resources, you must investigate whether they can be moved, and the restrictions that might be in place. Check your resource types against the move support for resources list. The list shows whether each resource type can be moved between resource groups or between subscriptions. For example, these resources can be moved:

* Azure Storage accounts
* Azure virtual machines
* Azure virtual networks

**These resources can't be moved:**

* Azure Active Directory domain services
* Azure Backup vaults
* Azure App Service gateways

**Virtual machines have their own limitations you must keep in mind. Here's a summary of limitations for virtual machines:**

* If you want to move a virtual machine, all of its dependants must go with it.
* You can't move virtual machines with certificates in Azure Key Vault between subscriptions.
* You can't move virtual machine scale sets with standard load balancers or a standard public IP.
* You can't move any managed disks that are in availability zones to different subscriptions.

**Validate resources in Azure**

You'll also see how to use **the Azure REST API** validate move operation to test and validate your moves.

**REST API** is a programmatic interface that you can call by sending HTTP requests. Programmers often call REST APIs in their custom code from clients such as mobile apps. To call a specific Azure REST method, such as the validate move operation, you can use the Azure CLI:

* Your Azure subscription ID.
* The name of the resource group currently holding your resources.
* The resource ID for each of the resources in your original resource group.
* The resource ID for the destination resource group where you want to move your resources.
* Your account access token.

*When you use the Azure CLI to call an Azure REST API operation, you don't have to provide a subscription ID or an access token. The CLI includes these values automatically.*

**You can send a POST request with the following details:**

POST https://management.azure.com/subscriptions/<your-subscription-id>/resourceGroups/<your-source-group>/validateMoveResources?api-version=2019-05-10

Authorization: Bearer <your-access-token>

Content-type: application/json

**The body of your POST request must contain the following information:**

{

"resources": ["<your-resource-id-1>", "<your-resource-id-2>", "<your-resource-id-3>"],  
"targetResourceGroup": "/subscriptions/<your-subscription-id>/resourceGroups/<your-target-group>"

}

**To submit this POST request with the correct body by using the Azure CLI, run this command:**

**az rest --method post /**

**--uri https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/<your-source-group>/validateMoveResources?api-version=2019-05-10 /**

**--body "{\"resources\": [\"<your-resource-id-1>\", \"<your-resource-id-2>\", \"<your-resource-id-3>\"], \"targetResourceGroup\": \"/subscriptions/<your-subscription-id>/resourceGroups/<your-target-group>\"}" /**

**--verbose**

**If your request is accepted, the API returns a status code of 202:**

**Response Code: 202  
cache-control: no-cache  
pragma: no-cache  
expires: -1  
location:** [**https://management.azure.com/subscriptions/<your-subscription-id>/operationresults/<your-operation-id>?api-version=2018-02-01**](https://management.azure.com/subscriptions/%3cyour-subscription-id%3e/operationresults/%3cyour-operation-id%3e?api-version=2018-02-01) **retry-after: 15**

*At this stage, the API has only validated your request. It hasn't yet validated whether your move will be successful. This response gives you a location URL. You use this location URL to test your move. Wait for the amount of time shown in the retry-after value in the request validation, before attempting to test your validation. In this example, the value is 15 seconds.*

**you then send a GET request to the location URL:**

**GET <location-url>**

**Authorization: Bearer <your-access-token>**

**OR  
  
az rest --method get --uri <location-url>**

**Move resources between subscriptions**

Depending on the resource type, you can move your resources between subscriptions, or between resource groups within the same subscription.

Move resources by using the Azure CLI / Move resources by using Azure PowerShell

<https://docs.microsoft.com/en-us/learn/modules/move-azure-resources-another-resource-group/6-move-verify-resources>

**Host a web application with Azure App Service**

**What is Azure App Service**

Azure App Service is a fully managed web application hosting platform. This platform as a service (PaaS) offered by Azure allows you to focus on designing and building your app while Azure takes care of the infrastructure to run and scale your applications.

**Deployment slots**

Using the Azure portal, you can easily add deployment slots to an App Service web app**. For instance, you can create a staging deployment slot where you can push your code to test on Azure.** Once you are happy with your code, you can easily swap the staging deployment slot with the production slot. You do all this with a few simple mouse clicks in the Azure portal.



**Continuous integration/deployment support**

The Azure portal provides out-of-the-box continuous integration and deployment with Azure DevOps, GitHub, Bitbucket, FTP, or a local Git repository on your development machine. Connect your web app with any of the above sources and App Service will do the rest for you by automatically syncing your code and any future changes on the code into the web app. Furthermore, with Azure DevOps, you can define your own build and release process that compiles your source code, runs the tests, builds a release, and finally deploys the release into your web app every time you commit the code. All that happens implicitly without any need to intervene.



**Built-in auto scale support (automatic scale-out based on real-world load)**

Baked into the web app is the ability to scale up/down or scale out. Depending on the usage of the web app, you can scale your app up/down by increasing/decreasing the resources of the underlying machine that is hosting your web app. Resources can be number of cores or the amount of RAM available.

***Scaling out, on the other hand, is the ability to increase the number of machine instances that are running your web app.***

**Operating systems**

**Selecting Windows activates the Monitoring tab, where you have the option to enable Application Insights.** Enabling this feature will configure your app to automatically send detailed performance telemetry to the Application Insights monitoring service without requiring any changes to your code. Application Insights can be used from Linux-hosted apps as well, **but this turnkey, no-code option is only available on Windows.**

**App Service plans**

Is a set of virtual server resources that run App Service apps. A plan's size (sometimes referred to as its sku or pricing tier) determines the performance characteristics of the virtual servers that run the apps assigned to the plan and the App Service features that those apps have access to. Every App Service web app you create must be assigned to a single App Service plan that runs it.

A single App Service plan can host multiple App Service web apps. In most cases, the number of apps you can run on a single plan will be limited by the performance characteristics of the apps and the resource limitations of the plan.

**App Service plans are the unit of billing for App Service. The size of each App Service plan in your subscription, in addition to the bandwidth resources used by the apps deployed to those plans, determines the price that you pay. The number of web apps deployed to your App Service plans has no effect on your bill.**

**Automated deployment**

* **Azure DevOps:** You can push your code to Azure DevOps (previously known as Visual Studio Team Services), build your code in the cloud, run the tests, generate a release from the code, and finally, push your code to an Azure Web App.
* **GitHub:** Azure supports automated deployment directly from GitHub. When you connect your GitHub repository to Azure for automated deployment, any changes you push to your production branch on GitHub will be automatically deployed for you.
* **Bitbucket:** With its similarities to GitHub, you can configure an automated deployment with Bitbucket.
* **OneDrive:** Microsoft's cloud-based storage. You must have a Microsoft Account linked to a OneDrive account to deploy to Azure.
* **Dropbox:** Azure supports deployment from Dropbox, which is a popular cloud-based storage system that is similar to OneDrive.

**Manual deployment**

* **Git:** App Service web apps feature a Git URL that you can add as a remote repository. Pushing to the remote repository will deploy your app.
* **az webapp up:** webapp up is a feature of the az command-line interface that packages your app and deploys it. Unlike other deployment methods*, az webapp up can create a new App Service web app for you if you haven't already created one.*
* **ZIP deploy:** Use az webapp deployment source config-zip to send a ZIP of your application files to App Service. ZIP deploy can also be accessed via basic HTTP utilities such as curl.
* **WAR deploy:** It's an App Service deployment mechanism specifically designed for deploying Java web applications using WAR packages. WAR deploy can be accessed using the Kudu HTTP API located at http://<your-app-name>.scm.azurewebsites.net/api/wardeploy. If this fails try: https://<your-app-name>.scm.azurewebsites.net/api/wardeploy.
* **Visual Studio:** Visual Studio features an App Service deployment wizard that can walk you through the deployment process.
* **FTP/S:** FTP or FTPS is a traditional way of pushing your code to many hosting environments, including App Service.

**Deploy with ZIP deploy**

**First, use dotnet publish to build the final app files and zip to package them into a zip file:**

**cd ~/BestBikeApp**

**dotnet publish -o pub**

**cd pub**

**zip -r site.zip \***

**Finally, perform the deployment with az webapp deployment source config-zip. Replace <your-app-name> in the following command with the name of your Azure web app and run it.**

**az webapp deployment source config-zip \**

**--src site.zip \**

**--resource-group learn-eb2e7eb6-1bd6-4023-bfc4-afb2cbe6d1d0 \**

**--name <your-app-name>**

**Publish a web app to Azure with Visual Studio**

**Pricing and reliability levels**

* **Shared compute:** Free and Shared, the two base tiers, run an app on the same Azure VM as other App Service apps, including apps of other customers. These tiers allocate CPU quotas to each app that runs on the shared resources, and the resources cannot scale-out.

Free and Shared plans are best for small-scale personal projects with limited traffic demands, with a set limit of 165 MB of outbound data every 24 hours.

* **Dedicated compute:** The Basic, Standard, Premium, and Premium V2 tiers run apps on dedicated Azure VMs. Only apps in the same App Service plan share the same compute resources. The higher the tier, the more VM instances are available to you for scale out.
  + **The Standard** service plan is best suited for live production workloads, where you are publishing commercial applications to customers.
  + **The Premium** service plans support high-capacity web apps where you do not want the additional costs of a dedicated (isolated) plan.
* **Isolated:** This tier runs dedicated Azure VMs on dedicated Azure virtual networks, which provide network isolation on top of compute isolation to your apps. It provides the maximum scale-out capabilities. You would only select an Isolated service plan when you have a specific requirement for the highest levels of security and performance. Isolate your app into a new App Service plan when:
  + The app is resource-intensive
  + You want to scale the app independently from the other apps in the existing plan
  + The app needs resources in a different geographical region

*Your App Service plan can be scaled up and down at any time. You can choose a lower pricing tier at first and scale up later when you need more App Service features.*

**Publish your ASP.NET Core web app to Azure**

*ASP.NET Core apps are cross-platform. This means they support running on the Linux version of App Service with no code changes. However, the Linux version doesn't support a shared hosting model, so you'll be using a Windows App Service for this exercise.*

**Stage a web app deployment for testing and rollback by using App Service deployment slots**

**Use a deployment slot**

Within a single **Azure App Service web app**, you can create multiple deployment slots. Each slot is a separate instance of that web app, and it has a separate hostname. **You can deploy a different version of your web app into each slot.**

**Use additional slots** to host new versions of your web app. Against these instances, **you can run tests such as integration tests, acceptance tests, and capacity tests.**

After you're satisfied with the test results for a new app version, deploy it by swapping its slot with the production slot. Unlike a code deployment, a slot swap is instantaneous. When you swap slots, the slot hostnames are exchanged, immediately sending production traffic to the new version of the app.

When you use slot swaps to deploy, your app is never exposed to the public web in a partially deployed state.

**Understand slots as separate Azure resources**

When you use more than one deployment slot for a web app, those slots are treated as separate instances of that web app. For example, they're listed separately on the **All resources page in the Azure portal. They each have their own URL. However, each slot shares the resources of the App Service plan, including virtual machine memory and CPU as well as disk space.**

**Maximum number of slots you can create**

|  |  |
| --- | --- |
| **Tier** | **Maximum staging slots** |
| Free | 0 |
| Shared | 0 |
| Basic | 0 |
| Standard | 5 |
| Premium | 20 |
| Isolated | 20 |

**Avoid a cold start during swaps**

When you swap a slot into production, you "warm-up" the app because your action sends a request to the root of the site. The warm-up request ensures that all compilation and caching tasks finish. After the swap, the site responds as fast as if it had been deployed for days.

**Create a deployment slot**

*Although you can clone settings to a new slot, you can't clone content. New slots always begin with no content. You must deploy content by using git or another deployment strategy. The clone operation copies the configuration to the new slot. After you clone the settings, the configuration of the two slots can be changed independently.*

**Access a slot**

The new slot's hostname is derived from the web app name and the name of the slot. You see this hostname when you select the slot on the Deployment Slots page:



You can control access to a slot by using IP address restrictions. Create a list of IP address ranges that you'll allow accessing the slot or a list of ranges that you'll deny access to the slot. These lists are like the allow ranges and deny ranges that you can set up on a firewall. Use this list to permit access only to computers that belong to your company or development team.

**Manage the configuration for a swap**

When you swap two slots, the app's configuration travels to the new slot along with the app. You can override this behavior for individual application settings and configuration strings by configuring them as slot settings.

Suppose, for example, you have two databases. You use one for production, and the other for acceptance testing. You always want the app version in the staging slot to use the testing database. The app version in the production slot should always use the production database. To achieve this, you can configure the database connection string as a slot setting.

**Configure slot settings**

1. On the Azure portal menu or from the Home page, select All resources, and select the deployment slot you want to configure.
2. Navigate to the Configuration page.
3. On the Application settings tab, observe whether the settings you're interested in contain a checkmark in the deployment slot setting field. To set or unset a checkmark on a given setting, select the setting's pencil button to edit it, then toggle the deployment slot setting checkbox to the desired value, and select OK.

**Notice how the swap will affect settings. The value of the APP\_VERSION setting will be exchanged between the slots, but the value of the ENVIRONMENT slot setting won't be swapped. (ENVIRONMENT was checked as ‘deployment slot setting’)**

**Understand the slot-swapping preview**

When you swap slots, the settings in the target slot (which is typically the production slot) are applied to the app version in the source slot before the hostnames are swapped. You might discover problems at this point.

* Phase 1: Slot settings from the target slot are applied to the web app in the source slot. Then Azure warms up the staging slot. At this point, the swap operation pauses so you can test the app in the source slot to make sure it works with the target slot configuration. If you find no problems, begin the next phase.
* Phase 2: The hostnames for the two sites are swapped. The version of the app now in the source slot receives its slot settings.

**Auto swap**

Auto swap brings the zero-downtime and easy rollback benefits of swap-based deployment to automated deployment pipelines. When you configure a slot for auto swap, Azure automatically swaps it whenever you push code or content into that slot.

When you use auto swap, you can't test the new app version in the staging slot before the swap. Auto swap mainly benefits users who want zero-downtime deployments and simple automated deployment pipelines.

If you want to be able to test before you swap, you'll need a more complex deployment pipeline that requests the slot swap itself. Alternatively, you can deploy to a separate slot that's dedicated for testing.

***Auto swap is not available in App Service on Linux.***

**When you consider using deployment slots, remember that Azure warms up an app before a swap, and traffic redirection is instantaneous. The result is that your app is deployed without service interruptions or performance drops.**

**Scale an App Service web app to efficiently meet demand with App Service scale up and scale out**

**Monitor and scale a web app**

When you create a web app, you can either create a new App Service plan or use an existing one. If you select an existing plan, any other web apps that use the same plan will share resources with your web app. They'll all scale together, so they need to have the same scaling requirements. If your apps have different requirements, use a separate App Service plan for each one.

**Scale up a web app**

Scaling out enables you to run more instances of a web app, but the resources available to each instance are determined by the pricing tier used by the App Service plan that hosts the web service. Each pricing tier specifies the computing power provided, together with the memory and maximum number of instances that can be created.

Scaling up can cause an interruption in service to client apps running at the time. **They might need to disconnect from the service and reconnect if the scale-up occurs during an active call to the web app**. And **new connections might be rejected until scaling finishes.** Also, scaling up can cause the outgoing **IP addresses for the web app to change**. If your web app depends on other services that have firewalls restricting incoming traffic, you'll need to reconfigure these services.

As with scale-out, you should monitor the performance of your system to ensure that scaling up (or down) has the desired effect. It's also important to understand that scale up and scale out can work cooperatively together. If you have scaled out to the maximum number of instances available for your pricing tier, you must scale up before you can scale out further.

**Deploy and run a containerized web app with Azure App Service**

**What is Container Registry?**

Azure Container Registry enables you to store Docker images in the cloud, in an Azure storage account.

Container Registry is an Azure service that you can use to create your own private Docker registries. Like Docker Hub, Container Registry is organized around repositories that contain one or more images. **Container Registry also lets you automate tasks such as redeploying an app when an image is rebuilt.**

**Security is an important reason to choose Container Registry instead of Docker Hub:**

* You have much more control over who can see and use your images.
* You can sign images to increase trust and reduce the chances of an image becoming accidentally (or intentionally) corrupted or otherwise infected.
* All images stored in a container registry are encrypted at rest.

**Working with images in Container Registry is like working with Docker Hub, but offers a few unique benefits:**

* Container Registry runs in Azure. The registry can be replicated to store images near where they're likely to be deployed.
* Container Registry is highly scalable, providing enhanced throughput for Docker pulls that can span many nodes concurrently. The Premium SKU of Container Registry includes 500 GiB of storage.

**Use Container Registry**

In addition to storing and hosting images, **you can also use Container Registry to build images.** Instead of building an image yourself and pushing it to Container Registry, use the CLI to upload the Docker file and other files that make up your image. Container Registry will then build the image for you. Use the acr build command to run a build.

**az acr build --file Dockerfile --registry myregistry --image myimage .**

**Deploy a web app by using an image from an Azure Container Registry repository**

You can deploy a web app to Azure App Service directly from Azure Container Registry**.**

**Deploy a web app from a repository in Azure Container Registry**

**When you create a web app from a Docker image, you configure the following properties:**

* **Registry that contains the image:** The registry can be Docker Hub, Azure Container Registry, or some other private registry.
* **Image:** This item is the name of the repository.
* **Tag:** This item indicates which version of the image to use from the repository. By convention, the most recent version is given the tag latest when it's built.
* **Startup file:** This item is the name of an executable file or a command to be run when the image is loaded. It's equivalent to the command that you can supply to Docker when running an image from the command line by using docker run. If you're deploying a ready-to-run, containerized app that already has the ENTRYPOINT and/or COMMAND values configured, you don't need to fill this in.

***After you've configured the web app, the Docker image is pulled, and runs as a cold start operation the first time a user attempts to visit the site. The app might take a few seconds to start initially, but thereafter, it will be available immediately.***

**What is a webhook?**

Azure App Service supports continuous deployment using webhooks. A webhook is a service offered by Azure Container Registry. Services and applications can subscribe to the webhook to receive notifications about updates to images in the registry. A web app that uses App Service can subscribe to an Azure Container Registry webhook to receive notifications about updates to the image that contains the web app. **When the image is updated, and App Service receives a notification, your app automatically restarts the site and pulls the latest version of the image.**

**What is the Container Registry tasks feature?**

You use the tasks feature of Container Registry to rebuild your image whenever its source code changes automatically. **You configure a Container Registry task to monitor the GitHub repository that contains your code and trigger a build each time it changes**. If the build finishes successfully, Container Registry can store the image in the repository. If your web app is set up for continuous integration in App Service, it receives a notification via the webhook and updates the app.

**Enable continuous integration from App Service**

The Container settings page of an App Service resource in the Azure portal automates the setup of continuous integration**. If you turn on Continuous Deployment, App Service configures a webhook in your container registry to notify an App Service endpoint.** Notifications from the registry that reach this endpoint cause your app to restart and pull the latest version of the container image.

**Extend continuous integration to source control by using a Container Registry task**

Container Registry tasks must be created from the command line. Unlike the az acr build command that we ran earlier to build our image, the **az acr task create** command creates and registers a long-lived task.

**az acr task create --registry <container\_registry\_name> --name buildwebapp --image webimage --context https://github.com/MicrosoftDocs/mslearn-deploy-run-container-app-service.git --file Dockerfile --git-access-token <access\_token>**

**Configure continuous deployment and create a webhook**

1. Set Continuous Deployment to On, and then select Save. This setting configures a webhook that Container Registry uses to alert the web app that the Docker image has changed.
2. In Container Registry, in the left menu pane, under Services, if you select Webhooks for your container registry, you'd see the newly configured webhook on the Webhooks/Container registry pane that appears.
3. Update the web app and test the webhook

**Secure your cloud data**

**What's Azure Security Center?**

Is a monitoring service that provides visibility of your security posture across all of your services, both on Azure and on-premises. The term security posture refers to cybersecurity policies and controls, as well as how well you can predict, prevent, and respond to security threats.

* Monitor security settings across on-premises and cloud workloads.
* Automatically apply required security settings to new resources as they come online.
* Provide security recommendations that are based on your current configurations, resources, and networks.
* Continuously monitor your resources and perform automatic security assessments to identify potential vulnerabilities before those vulnerabilities can be exploited.
* Use machine learning to detect and block malware from being installed on your virtual machines (VMs) and other resources. You can also use adaptive application controls to define rules that list allowed applications to ensure that only applications you allow can run.
* Detect and analyze potential inbound attacks and investigate threats and any post-breach activity that might have occurred.
* Provide just-in-time access control for network ports. Doing so reduces your attack surface by ensuring that the network only allows traffic that you require at the time that you need it to.

**Understand your security posture**

**Let's say that Tailwind Traders must comply with the Payment Card Industry's Data Security Standard (PCI DSS). This report shows that the company has resources that it needs to remediate.**



**Resource security hygiene section,** Tailwind Traders can see the health of its resources from a security perspective. To help prioritize remediation actions, recommendations are categorized as low, medium, and high. Here's an example:



**What's secure score?**



Secure score helps you:

* Report on the current state of your organization's security posture.
* Improve your security posture by providing discoverability, visibility, guidance, and control.
* Compare with benchmarks and establish key performance indicators (KPIs).

**Protect against threats**

* **Just-in-time VM access -** Helps to protect your VMs from attacks on management ports. When JIT access is enabled, users must request access to a VM in order to remote into it. Until someone is given JIT access, management ports on the VM are closed so they can’t be accessed. Once JIT access is given to a user, the ports are open for a specific period of time as requested by the user. Once that time period has elapsed, the management ports are closed again.
* **Adaptive application controls -** Can control which applications are allowed to run on its virtual machines. In the background, Security Center uses machine learning to look at the processes running on a virtual machine. It creates exception rules for each resource group that holds the virtual machines and provides recommendations. This process provides alerts that inform the company about unauthorized applications that are running on its VMs.
* **Adaptive network hardening -** Can monitor the internet traffic patterns of the VMs and compare those patterns with the company's current network security group (NSG) settings. From there, Security Center can make recommendations on whether the NSGs should be locked down further and provide remediation steps.
* **File integrity monitoring -** Can also configure the monitoring of changes to important files on both Windows and Linux, registry settings, applications, and other aspects that might indicate a security attack.

**Respond to security alerts**

Can use Security Center to get a centralized view of all of its security alerts. From there, the company can dismiss false alerts, investigate them further, remediate alerts manually, or use an automated response with a workflow automation.

Workflow automation **uses Azure Logic Apps and Security Center connectors**. The logic app can be triggered by a threat detection alert or by a Security Center recommendation, filtered by name or by severity. You can then configure the logic app to run an action such as sending an email or posting a message to a Microsoft Teams channel.

**Azure Sentinel**

Is Microsoft's cloud-based **SIEM** system. It uses intelligent security analytics and threat analysis.

A **SIEM** system aggregates security data from many different sources (as long as those sources support an open-standard logging format). It also provides capabilities for threat detection and response.

**Azure Sentinel capabilities**

* Collect cloud data at scale
  + Collect data across all users, devices, applications, and infrastructure, both on-premises and from multiple clouds.
* Detect previously undetected threats
  + Minimize false positives by using Microsoft's comprehensive analytics and threat intelligence.
* Investigate threats with artificial intelligence
  + Examine suspicious activities at scale, tapping into years of cybersecurity experience from Microsoft.
* Respond to incidents rapidly
  + Utilize built-in orchestration and automation of common tasks.

**Connect your data sources**

Azure Sentinel supports a number of data sources, which it can analyze for security events. These connections are handled by built-in connectors or industry-standard log formats and APIs.

* Connect Microsoft solutions
* Connect other services and solutions (AWS CloudTrail for example)
* Connect industry-standard data sources

**Detect threats**

* **Built in analytics** use templates designed by Microsoft's team of security experts and analysts based on known threats, common attack vectors, and escalation chains for suspicious activity.
* **Custom analytics** are rules that you create to search for specific criteria within your environment. You can preview the number of results that the query would generate (based on past log events) and set a schedule for the query to run. You can also set an alert threshold.

**Investigate and respond**

When Azure Sentinel detects suspicious events, Tailwind Traders can investigate specific alerts or incidents (a group of related alerts). With the investigation graph, the company can review information from entities directly connected to the alert and see common exploration queries to help guide the investigation.



The company will also use Azure Monitor Workbooks to automate responses to threats. For example, it can set an alert that looks for malicious IP addresses that access the network and create a workbook that does the following steps:

1. When the alert is triggered, open a ticket in the IT ticketing system.
2. Send a message to the security operations channel in Microsoft Teams or Slack to make sure the security analysts are aware of the incident.
3. Send all of the information in the alert to the senior network admin and to the security admin. The email message includes two user option buttons: Block or Ignore.

**Store and manage secrets by using Azure Key Vault**

**Azure Key Vault**

Is a centralized cloud service for storing an application's secrets in a single, central location. It provides secure access to sensitive information by providing access control and logging capabilities.

**What can Azure Key Vault do?**

* **Manage secrets** you can use Key Vault to securely store and tightly control access to tokens, passwords, certificates, API keys, and other secrets.
* **Manage encryption keys** you can use Key Vault as a key management solution. Key Vault makes it easier to create and control the encryption keys that are used to encrypt your data.
* **Manage SSL/TLS certificates** Key Vault enables you to provision, manage, and deploy your public and private Secure Sockets Layer / Transport Layer Security (SSL/TLS) certificates for both your Azure resources and your internal resources.
* **Store secrets backed by hardware security modules (HSMs)** these secrets and keys can be protected either by software or by FIPS 140-2 Level 2 validated HSMs.

**What are the benefits of Azure Key Vault?**

* **Centralized application secrets -** Centralizing the storage for your application secrets enables you to control their distribution and reduces the chances that secrets are accidentally leaked.
* **Securely stored secrets and keys** - Azure uses industry-standard algorithms, key lengths, and HSMs. Access to Key Vault requires proper authentication and authorization.
* **Access monitoring and access control -** By using Key Vault, you can monitor and control access to your application secrets.
* **Simplified administration of application secrets -** Key Vault makes it easier to enroll and renew certificates from public certificate authorities (CAs). You can also scale up and replicate content within regions and use standard certificate management tools.
* **Integration with other Azure services -** You can integrate Key Vault with storage accounts, container registries, event hubs, and many more Azure services. These services can then securely reference the secrets stored in Key Vault.

**Host your Azure virtual machines on dedicated physical servers by using Azure Dedicated Host**

Some organizations must follow regulatory compliance that requires them to be the only customer using the physical machine that hosts their virtual machines. Azure Dedicated Host provides dedicated physical servers to host your Azure VMs for Windows and Linux.



**What are the benefits of Azure Dedicated Host?**

* Gives you visibility into, and control over, the server infrastructure that's running your Azure VMs.
* Helps address compliance requirements by deploying your workloads on an isolated server.
* Lets you choose the number of processors, server capabilities, VM series, and VM sizes within the same host.

**Availability considerations for Dedicated Host**

For high availability, you can provision multiple hosts in a host group and deploy your virtual machines across this group. VMs on dedicated hosts can also take advantage of maintenance control. This feature enables you to control when regular maintenance updates occur, within a 35-day rolling window.

**What's the easiest way for Tailwind Traders to combine security data from all of its monitoring tools into a single report that it can take action on?  
R:** Collect security data in Azure Sentinel. Azure Sentinel is Microsoft's cloud-based SIEM. A SIEM aggregates security data from many different sources to provide additional capabilities for threat detection and responding to threats.

**Summary**

* Azure Security Center provides visibility of your security posture across all of your services, both on Azure and on-premises.
* Azure Sentinel aggregates security data from many different sources and provides additional capabilities for threat detection and response.
* Azure Key Vault stores your applications' secrets, such as passwords, encryption keys, and certificates, in a single, central location.
* Azure Dedicated Host provides dedicated physical servers to host your Azure VMs for Windows and Linux.

**Top 5 security items to consider before pushing to production**

**Activating Azure Security Center**

Azure Security Center provides unified security management and advanced threat protection for hybrid cloud workloads and is offered in two tiers: Free and Standard. The free tier provides security policies, assessments, and recommendations while the Standard tier provides a robust set of features, including threat intelligence.

**To upgrade a subscription to the Standard tier, you must be assigned the role of Subscription Owner, Subscription Contributor, or Security Admin.**

**Configure security policies to manage data**

**Protect data at rest**

**Apply disk encryption to help safeguard your data.**

Use **Microsoft Azure Disk Encryption**, which enables IT administrators to encrypt both Windows infrastructure as a service (IaaS) and Linux IaaS virtual machine (VM) disks. Disk encryption combines the industry-standard BitLocker feature and the Linux DM-Crypt feature to provide volume encryption for the operating system (OS) and the data disks. ‎Azure Storage and Azure SQL Database encrypt data at rest by default, and many services offer encryption as an option.

**Protect data in transit**

For data moving between your on-premises infrastructure and Azure, consider appropriate safeguards such as **HTTPS or VPN**. When sending encrypted **traffic between an Azure virtual network and an on-premises** location over the public internet, **use Azure VPN Gateway.**

PROTECT DATA IN TRANSIT

|  |  |
| --- | --- |
| **Best practice** | **Solution** |
| Secure access from multiple workstations located on-premises to an Azure virtual network | **Use site-to-site VPN.** |
| Secure access from an individual workstation located on-premises to an Azure virtual network | **Use point-to-site VPN.** |
| Move large data sets over a dedicated high-speed wide-area network (WAN) link | **Use Azure ExpressRoute. If you choose to use ExpressRoute, you can also encrypt the data at the application level by using SSL/TLS or other protocols for added protection.** |
| Interact with Azure Storage through the Azure portal | **All transactions occur via HTTPS. You can also use Storage REST API over HTTPS to interact with Azure Storage and Azure SQL Database.** |

**Data discovery**

Provides advanced capabilities built into Azure SQL Database for discovering, classifying, labeling and protecting sensitive data (such as business, personal data (PII), and financial information) in your databases.

**Data discovery and classification** is part of the **Advanced Data Security** offering, which is a unified package for advanced Microsoft SQL Server security capabilities. You access and manage data discovery and classification via the central SQL Advanced Data Security portal.

Data discovery and classification introduces a set of advanced services and SQL capabilities, forming a SQL Information Protection paradigm aimed at protecting the data, not just the database:

* **Discovery and recommendations** - The classification engine scans your database and identifies columns containing potentially sensitive data. It then provides you with a more natural way to review and apply the appropriate classification recommendations via the Azure portal.
* **Labeling** - Sensitivity classification labels can be persistently tagged on columns using new classification metadata attributes introduced into the SQL Server Engine. This metadata can then be utilized for advanced sensitivity-based auditing and protection scenarios.
* **Query result set sensitivity** - The sensitivity of the query result set is calculated in real-time for auditing purposes.
* **Visibility** - You can view the database classification state in a detailed dashboard in the Azure portal. Additionally, you can download a report (in Microsoft Excel format) that you can use for compliance and auditing purposes, in addition to other needs.

**Steps for discovery, classification, and labeling**

Classifications have two metadata attributes:

* **Labels** - These are the main classification attributes used to define the sensitivity level of the data stored in the column.
* **Information Types** - These provide additional granularity into the type of data stored in the column.

**SQL Information Protection (SQL IP)**

SQL IP brings a set of advanced services and SQL capabilities, forming a new information protection paradigm in SQL aimed at protecting the data, not just the database:

* **Azure SQL Auditing –** Azure SQL Auditing tracks database events and writes them to an audit log in your Azure storage account, Log Analytics workspace or Event Hub.
* **Data Discovery & Classifications –** Is built into Azure SQL Database, Azure SQL Managed Instance, and Azure Synapse Analytics. It provides advanced capabilities for discovering, classifying, labeling, and reporting the sensitive data in your databases.
* **Dynamic data masking –** Azure SQL Database, Azure SQL Managed Instance, and Azure Synapse Analytics support dynamic data masking. Dynamic data masking limits sensitive data exposure by masking it to non-privileged users.
* **Security Center –** Scans your database and makes recommendations to improve security. Also allows you to set up and monitor Security Alerts.
* **Transparent data encryption –** Transparent data encryption encrypts your databases, backups, and logs at rest without any changes to your application. To enable encryption, go to each database.

**Classify your SQL DB**

[**https://docs.microsoft.com/en-us/learn/modules/configure-security-policies-to-manage-data/3-exercise-classify-sql-database**](https://docs.microsoft.com/en-us/learn/modules/configure-security-policies-to-manage-data/3-exercise-classify-sql-database)

**Monitor access to sensitive data**

An important aspect of the IP paradigm is the ability to monitor access to sensitive data. Azure SQL Database Auditing has been enhanced to include a new field in the audit log. The data\_sensitivity\_information field logs the sensitivity classifications (labels) of the actual data that was returned by the query. 

Consider configuring Azure SQL Database Auditing for monitoring and auditing access to your classified sensitive data.

**Explore data recovery, retention, and disposal**

A data retention policy defines the principles for data recovery and disposal and enforced in the same manner as data reclassification.

Failure to maintain a data retention policy could mean data loss or failure to comply with regulatory and legal discovery requirements.

**Immutable storage and data retention**

Immutable storage for Azure Blob Storage enables users to store business-critical data in a write once, read many (WORM) state. This state makes the data unerasable and unmodifiable for a user-specified interval. Blobs can be created and read, but not modified or deleted, for the duration of the retention interval.

* **Time-based retention policy support** - Users set policies to store data for a specified interval. Legal hold policy support - When the retention interval is not known, users can set legal holds to store data immutably until the legal hold is cleared. When a legal hold is set, blobs can be created and read, but not modified or deleted. Each legal hold is associated with a user-defined alphanumeric tag that is used as an identifier string (such as a case ID).
* **Support for all blob tiers** - WORM policies are independent of the Azure Blob Storage tier and apply to all tiers: hot, cool, and archive. Users can transition data to the most cost-optimized tier for their workloads while maintaining data immutability.
* **Container-level configuration** - Users can configure time-based retention policies and legal hold tags at the container level. By using simple container-level settings, users can create and lock time-based retention policies, extend retention intervals, set and clear legal holds, and more. These policies apply to all the blobs in the container, both existing and new.
* **Audit logging support** - Each container includes an audit log, which displays up to five time-based retention commands for locked time-based retention policies. However, the log has a maximum of three logs for retention interval extensions or time-based retention. The log contains the user ID, command type, time stamps, and retention interval. For legal holds, the log contains the user ID, command type, time stamps, and legal hold tags.

**Understand data sovereignty**

Digital information is always subject to the laws of the country or region where it's stored. This concept is known as data sovereignty. Many of the concerns that surround data sovereignty relate to enforcing privacy regulations and preventing data that are stored in a foreign country from being subpoenaed by the host country or region's government.

In Azure, customer data can be replicated within a geographic area for enhanced data durability if there's a significant data center disaster.

**Paired regions**

Each Azure region is paired with another region within the same geography, forming a region pair. The exception is Brazil South, which is paired with a region outside its geography.

Across the region pairs, Azure serializes platform updates (or planned maintenance) so that only one region is updated at a time. If an outage affecting multiple regions occurs, one region in each pair will be prioritized for recovery.



**An example of paired regions**

The following illustration shows a hypothetical app that uses a regional pair for DR. The arrows highlight the cross-region activities of three Azure services (Azure Compute, Azure Storage, and Azure Database), and how they are configured to replicate across regions.



**Configure and manage secrets in Azure Key Vault**

**Vaults**

You use Azure Key Vault to create multiple secure containers, called vaults. Vaults help reduce the chances of accidental loss of security information by centralizing application secrets storage.

Key vaults also control and log the access to anything stored in them.

**Keys**

Microsoft and your apps don't have access to the stored keys directly once a key is created or added to a key vault. Applications must use your keys by calling cryptography methods on the Key Vault service. The Key Vault service performs the requested operation within its hardened boundary. The application never has direct access to the keys.

There are two variations on keys in Key Vault: hardware-protected, and software-protected.

**Hardware protected keys**

The Key Vault service supports using HSMs that provide a hardened, tamper-resistant environment for cryptographic processing and key generation. Azure has dedicated HSMs validated to FIPS 140-2 Level 2 that Key Vault uses to generate or store keys. These HSM-backed keys are always locked to the boundary of the HSM. When you ask the Key Vault service to decrypt or sign with a key, the operation is performed inside an HSM.

*You can import keys from your own hardware security modules (HSMs) and transfer them to Key Vault without leaving the HSM boundary. This scenario is often referred to as bring your own key, or* ***BYOK***

**Software protected keys**

Key Vault can also generate and protect keys using software-based RSA and ECC algorithms. In general, software-protected keys offer most of the features as HSM-protected keys except the FIPS 140-2 Level 2 assurance:

* **Your key is still isolated from the application (and Microsoft) in a container that you manage**
* **It's stored at rest encrypted with HSMs**
* **You can monitor usage using Key Vault logs**

The primary difference (besides price) with a software-protected key is when cryptographic operations are performed, they are done in software using Azure compute services while for HSM-protected keys the cryptographic operations are performed within the HSM.

*For production use, it's recommended to use HSM-protected keys and use software-protected keys in only test/pilot scenarios. There is an additional charge for HSM-backed keys per-month if the key is used in that month. The summary page has a link to the pricing details for Azure Key Vault.*

**Secrets**

Secrets are small (less than 10K) data blobs protected by a HSM-generated key created with the Key Vault. Secrets exist to simplify the process of persisting sensitive settings that almost every application has: storage account keys, .PFX files, SQL connection strings, data encryption keys, etc.

**Key vault uses**

* **Secrets management**. Azure Key Vault can securely store (with HSMs) and tightly control access to tokens, passwords, certificates, API keys, and other secrets.
* **Key management.** Azure Key Vault is a cloud-based key management solution, making it easier to create and control the encryption keys used to encrypt your data. Azure services such as App Service integrate directly with Azure Key Vault and can decrypt secrets without knowledge of the encryption keys.
* **Certificate management.** Azure Key Vault is also a service that lets you easily provision, manage, and deploy public and private SSL/TLS certificates for use with Azure and your internal connected resources. It can also request and renew TLS certificates through partnerships with certificate authorities, providing a robust solution for certificate lifecycle management.

**Authentication**

Azure Key Vault uses Azure Active Directory (Azure AD) to authenticate users and apps that try to access a vault. Authentication is always performed by associating the Azure AD tenant of the subscription that the Key Vault is part of, and every user or app making a request having to be known to Azure AD. There is no support for anonymous access to a Key Vault.

**Authorization**

Management operations (creating a new Azure Key Vault) use role-based access control (RBAC). There is a built-in role Key Vault Contributor that provides access to management features of key vaults, but doesn't allow access to the key vault data. This is the recommended role to use. There's also a Contributor role that includes full administration rights - including the ability to grant access to the data plane.

**Restricting network access**

You should determine the minimum network access required - for example you can restrict Key Vault endpoints to specific Azure Virtual Network subnets, specific IP addresses, or trusted Microsoft services including Azure SQL, Azure App Service, and various data and storage services that use encryption keys.

**Adding certificates to a Key Vault**



1. In the previous diagram, your application is creating a certificate which internally begins by creating a key in your Azure Key Vault.
2. Key Vault returns a Certificate Signing Request (CSR) to your application.
3. Your application passes the CSR to your chosen CA.
4. Your chosen CA responds with an X.509 Certificate.
5. Your application completes the new certificate creation with a merger of the X.509 Certificate from your CA.

**Third, you can connect your Key Vault with a trusted certificate issuer (referred to as an integrated CA) and create the certificate directly in Azure Key Vault. This approach requires a one-time setup to connect the certificate authority.**



1. In the previous diagram, your application is creating a certificate which internally begins by creating a key in your key vault.
2. Key Vault sends a SSL Certificate Request to the CA.
3. Your application polls, in a loop and wait process, for your Key Vault for certificate completion. The certificate creation is complete when Key Vault receives the CA’s response with x509 certificate.
4. The CA responds to Key Vault's SSL Certificate Request with an X509 SSL Certificate.
5. Your new certificate creation completes with the merger of the X509 Certificate for the CA.

**Finally, you can import existing certificates - this allows you to add certificates to Key Vault that you are already using. The imported certificate can be in either PFX or PEM format and must contain the private key. For example, here's a PowerShell script to upload a certificate:**

$pfxFilePath = "C:\WebsitePrivateCertificate.pfx"

$pwd = "password-goes-here"

$flag = [System.Security.Cryptography.X509Certificates.X509KeyStorageFlags]::Exportable

$pkcs12ContentType = [System.Security.Cryptography.X509Certificates.X509ContentType]::Pkcs12

$collection = New-Object System.Security.Cryptography.X509Certificates.X509Certificate2Collection

$collection.Import($pfxFilePath, $pwd, $flag)

$clearBytes = $collection.Export($pkcs12ContentType)

$fileContentEncoded = [System.Convert]::ToBase64String($clearBytes)

$secret = ConvertTo-SecureString -String $fileContentEncoded -AsPlainText –Force

$secretContentType = 'application/x-pkcs12'

# Replace the following <vault-name> and <key-name>.

Set-AzKeyVaultSecret -VaultName <vault-name> -Name <key-name> -SecretValue $secret -ContentType $secretContentType

**Azure App Service integration**

Once you have a public/private key pair certificate in your Azure Key Vault, you can easily associate it to your web app through the Azure portal.

1. **Select TLS/SSL settings under Settings.**
2. **Select the Private Key Certificate (.pfx) tab.**
3. **Select + Import Key Vault Certificate as shown in the following screenshot.**



1. **You can then select the vault, which must be in the same subscription, and the secret containing the certificate.**
   1. **The certificate must be an X.509 cert with a content type of application/x-pkcs12 and cannot have a password.**

**Secure your Azure resources with Azure role-based access control (Azure RBAC)**

**Azure subscriptions**

**E**ach Azure subscription is associated with a single Azure AD directory. Users, groups, and applications in that directory can manage resources in the Azure subscription. The subscriptions use Azure AD for single sign-on (SSO) and access management. You can extend your on-premises Active Directory to the cloud by using Azure AD Connect. This feature allows your employees to manage their Azure subscriptions by using their existing work identities. When you disable an on-premises Active Directory account, it automatically loses access to all Azure subscriptions connected with Azure AD.

**What is Azure RBAC?**

Azure role-based access control (Azure RBAC) is an authorization system built on Azure Resource Manager that provides fine-grained access management of resources in Azure. With Azure RBAC, you can grant the exact access that users need to do their jobs. For example, you can use Azure RBAC to let one employee manage virtual machines in a subscription while another manages SQL databases within the same subscription.

The scope of a role assignment can be a subscription, a resource group, or a single resource. A role assigned at a parent scope also grants access to the child scopes contained within it. For example, a user with access to a resource group can manage all the resources it contains, like websites, virtual machines, and subnets. The Azure role that you assign dictates what resources the user, group, or application can manage within that scope.

**How does Azure RBAC work?**

**1. Security principal (who)**

A security principal is just a fancy name for a user, group, or application that you want to grant access to.



**2. Role definition (what you can do)**

A role definition is a collection of permissions. It's sometimes just called a role. A role definition lists the permissions that can be performed, such as read, write, and delete. Roles can be high-level, like Owner, or specific, like Virtual Machine Contributor.



Azure includes several built-in roles that you can use. The following lists four fundamental built-in roles:

* **Owner** - Has full access to all resources, including the right to delegate access to others.
* **Contributor** - Can create and manage all types of Azure resources, but can’t grant access to others.
* **Reader** - Can view existing Azure resources.
* **User Access Administrator** - Lets you manage user access to Azure resources.

**3. Scope (where)**

Scope is where the access applies to. This is helpful if you want to make someone a Website Contributor, but only for one resource group.

In Azure, you can specify a scope at multiple levels: management group, subscription, resource group, or resource. Scopes are structured in a parent-child relationship. When you grant access at a parent scope, those permissions are inherited by the child scopes. For example, if you assign the Contributor role to a group at the subscription scope, that role is inherited by all resource groups and resources in the subscription.



**Role assignment**

Once you have determined the who, what, and where, you can combine those elements to grant access. A role assignment is the process of binding a role to a security principal at a particular scope, for the purpose of granting access. To grant access, you create a role assignment. To revoke access, you remove a role assignment.



**NotActions**

Use NotActions to create a set of not allowed permissions. The access granted by a role, the effective permissions, is computed by subtracting the NotActions operations from the Actions operations. For example, the Contributor role has both Actions and NotActions. The wildcard (\*) in Actions indicates that it can perform all operations on the control plane. Then you subtract the following operations in NotActions to compute the effective permissions:

* Delete roles and role assignments
* Create roles and role assignments
* Grants the caller User Access Administrator access at the tenant scope
* Create or update any blueprint artifacts
* Delete any blueprint artifacts

**Suppose an administrator needs to generate a report of the role assignments for the last week. Where in the Azure portal would they generate that report?  
R:** Search for Activity log and filter on the Create role assignment (roleAssignments) operation.In the Activity log, filter on the Operation name field to find role assignments.

**Secure your Azure SQL Database**

**Create and configure a Linux virtual machine**

Two things to note. First**, we don't need a password because we generated an SSH key pair as part of the VM creation.** Second, on the initial shell connection into the VM it will give you a prompt about the authenticity of the host. This occurs because we are connecting to an IP address instead of a host name. **Answering "yes" will save the IP as a valid host for connection and allow the connection to proceed.**

**Firewall rules**

Azure SQL Database has a built-in firewall that is used to allow and deny network access to both the database server itself, as well as individual databases. Initially, all public access to your Azure SQL Database is blocked by the SQL Database firewall. To access a database server, you must specify one or more server-level IP firewall rules that enable access to your Azure SQL Database. You use the IP firewall rules to specify which IP address ranges from the Internet are allowed, and whether Azure applications can attempt to connect to your Azure SQL Database.

Firewall rules are configured at the server and/or database level, and will specifically state which network resources are allowed to establish a connection to the database. Depending on the level, the rules you can apply will be as follows:

**Server-level firewall rules**

These rules enable clients to access your entire Azure SQL server, that is, all the databases within the same logical server. There are three types of rules that can be applied at the server level.

* **Allow access to Azure services** - Rule allows services within Azure to connect to your Azure SQL Database. When enabled, this setting allows communications from all Azure public IP addresses. This includes all Azure Platform as a Service (PaaS) services, such as Azure App Service and Azure Container Service, as well as Azure VMs that have outbound Internet access. This rule can be configured through the ON/OFF option in the firewall pane in the portal, or by an IP rule that has 0.0.0.0 as the start and end IP addresses.



***This rule is used when you have applications running on PaaS services in Azure, such as Azure Logic Apps or Azure Functions, that need to access your Azure SQL Database. Many of these services don't have a static IP address, so this rule is needed to ensure they are able to connect to the database.***

* **IP address rules** - Are rules that are based on specific public IP address ranges. IP addresses connecting from an allowed public IP range will be permitted to connect to the database.



These rules can be used when you have a static public IP address that needs to access your database.

* **Virtual network rules** - Allow you to explicitly allow connection from specified subnets inside one or more Azure virtual networks (VNets). Virtual network rules can provide greater access control to your databases and can be a preferred option depending on your scenario. Since Azure VNet address spaces are private, you can effectively eliminate exposure to public IP addresses and secure connectivity to those addresses you control.



**Database-level firewall rules**

These rules allow access to an individual database on a logical server and are stored in the database itself. For database-level rules, only IP address rules can be configured. They function the same as when applied at the server-level, but are scoped to the database only.



The benefits of database-level rules are their portability. When replicating a database to another server, the database-level rules will be replicated, since they are stored in the database itself.

**Restricting network access in practice**

Whenever possible, as a best practice, use database-level IP firewall rules to enhance security and to make your database more portable.

**TLS network encryption**

Azure SQL Database enforces Transport Layer Security (TLS) encryption at all times for all connections, **which ensures all data is encrypted "in transit" between the database and the client**. By using TLS encryption, you can ensure that anyone who may have intercepted the traffic between the app server and database would not be able to read the data. TLS encryption is a standard of securing traffic over the internet, and in this case ensures your network traffic to and from your Azure SQL database is secure by default.

**Transparent data encryption**

Azure SQL Database protects your data at rest using transparent data encryption (TDE). **TDE performs real-time encryption and decryption of the database**, associated backups, and transaction log files at rest without requiring changes to the application. Using a database encryption key, transparent data encryption performs real-time I/O encryption and decryption of the data at the page level. Each page is decrypted when it's read into memory and then encrypted before being written to disk.

By default, TDE is enabled for all newly deployed Azure SQL databases. It's important to check that data encryption hasn’t been turned off, and older Azure SQL Server databases may not have TDE enabled.

**Dynamic data masking**

Data masking rules consist of the column to apply the mask to, and how the data should be masked. You can create your own masking format, or use one of the standard masks such as:

* Default value, which displays the default value for that data type instead.
* Credit card value, which only shows the last four digits of the number, converting all other numbers to lower case x’s.
* Email, which hides the domain name and all but the first character of the email account name.
* Number, which specifies a random number between a range of values. For example, on the credit card expiry month and year, you could select random months from 1 to 12 and set the year range from 2018 to 3000.
* Custom string, which allows you to set the number of characters exposed from the start of the data, the number of characters exposed from the end of the data, and the characters to repeat for the remainder of the data.

**Azure SQL Database auditing**

By enabling auditing, operations that occur on the database are stored for later inspection or to have automated tools analyze them. Auditing is also used for compliance management or understanding how your database is used. Auditing is also required if you wish to use Azure threat detection on your Azure SQL database.

**You can use SQL database auditing to:**

* Retain an audit trail of selected events. You can define categories of database actions to be audited.
* Report on database activity. You can use pre-configured reports and a dashboard to get started quickly with activity and event reporting.
* Analyze reports. You can find suspicious events, unusual activity, and trends.

Audit logs are written to Append Blobs in an Azure Blob storage account that you designate. Audit policies can be applied at the server-level or database-level. Once enabled, you can use the Azure portal to view the logs, or send them to Log Analytics or Event Hub for further processing and analysis.

**Advanced Data Security for Azure SQL Database**

(ADS) provides a set of advanced SQL security capabilities, including data discovery & classification, vulnerability assessment, and Advanced Threat Protection.

* **Data discovery & classification (currently in preview)** provides capabilities built into Azure SQL Database for discovering, classifying, labeling & protecting the sensitive data in your databases. It can be used to provide visibility into your database classification state, and to track the access to sensitive data within the database and beyond its borders.
* **Vulnerability assessment** is an easy to configure service that can discover, track, and help you remediate potential database vulnerabilities. It provides visibility into your security state, and includes actionable steps to resolve security issues, and enhance your database fortifications.
* **Advanced Threat Protection** detects anomalous activities indicating unusual and potentially harmful attempts to access or exploit your database. It continuously monitors your database for suspicious activities, and provides immediate security alerts on potential vulnerabilities, SQL injection attacks, and anomalous database access patterns. Advanced Threat Protection alerts provide details of the suspicious activity and recommend action on how to investigate and mitigate the threat.

**The Data Discovery & Classification panel** shows columns within your tables that need to be protected. Some of the columns may have sensitive information or may be considered classified in different countries or regions.



**Vulnerability Assessment**

The Vulnerability Assessment lists configuration issues on your database and the associated risk. For example, in the image above, you can see the server-level firewall needs to be set up.



**Threat Detection**

This displays a list of detected threats. For example, here you can see one potential SQL injection attack listed.



**Transparent Data Encryption will encrypt which database files?  
R:** Database files, log files, and backup files

**Optimize your web applications by caching read-only data with Redis**

**What are Redis caching architectures?**

Redis caching architecture is how we distribute our data in the cache. Redis distributes data in three major ways:

* Single node
* Multiple node
* Clustered

**Redis caching architectures are split across Azure by tiers:**

* **Basic**: Basic cache ideal for development/testing. Is limited to a single server, 53 GB of memory, and 20,000 connections. There is no SLA for this service tier.
* **Standard**: Production cache which supports replication and includes an 99.99% SLA. It supports two servers (master/slave), and has the same memory/connection limits as the Basic tier.
* **Premium**: Enterprise tier which builds on the Standard tier and includes persistence, clustering, and scale-out cache support. This is the highest performing tier with up to 530 GB of memory and 40,000 simultaneous connections.

You can control the amount of cache memory available on each tier - this is selected by choosing a cache level from C0-C6 for Basic/Standard and P0-P4 for Premium. Check the [pricing page](https://azure.microsoft.com/en-us/pricing/details/cache/) for full details.

**The Premium tier allows you to persist data in two ways to provide disaster recovery:**

* **RDB persistence (Redis Database)** takes a periodic snapshot and can rebuild the cache using the snapshot in case of failure.
* **AOF persistence (Append Only File)** saves every write operation to a log that is saved at least once per second. **This creates bigger files than RDB but has less data loss.**

**Clustering support**

With a premium tier Redis cache, you can implement clustering to automatically split your dataset among multiple nodes. To implement clustering, you specify the number of shards to a maximum of 10. The cost incurred is the cost of the original node, multiplied by the number of shards.

**Redis supports a set of** [**known commands**](https://redis.io/commands)**. A command is typically issued as COMMAND parameter1 parameter2 parameter3.**

|  |  |
| --- | --- |
| **Command** | **Description** |
| **ping** | Ping the server. Returns "PONG". |
| **set [key] [value]** | Sets a key/value in the cache. Returns "OK" on success. |
| **get [key]** | Gets a value from the cache. |
| **exists [key]** | Returns '1' if the key exists in the cache, '0' if it doesn't. |
| **type [key]** | Returns the type associated to the value for the given key. |
| **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. |
| **del [key]** | Deletes the value associated with the key. |
| **flushdb** | Delete all keys and values in the database. |

**Adding an expiration time to values**

Caching is important because it allows us to store commonly used values in memory. **However, we also need a way to expire values when they are stale**. In Redis this is done 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. Here are some notes on TTL expirations.

* Expirations can be set using seconds or milliseconds precision.
* The expire time resolution is always 1 millisecond.
* Information about expires are replicated and persisted on disk, the time virtually passes when your Redis server remains stopped (this means that Redis saves the date at which a key will expire).

**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***.**

* The access key acts as a password for your cache. There are two keys created: primary and secondary. You can use either key, two are provided in case you need to change the primary key. You can switch all of your clients to the secondary key, and regenerate the primary key. This would block any applications using the original primary key. Microsoft recommends periodically regenerating the keys - much like you would your personal passwords.

**Suppose you're storing 400 MB of highly-accessed data in a Redis cache. How should you organize it in the cache to ensure high performance?  
R:** Redis works best with data that is 100 K or less. Splitting and storing larger data with multiple keys will reduce network latency and out-of-memory issues.

**Work with mutable and partial data in Azure Cache for Redis**

**Creating and running transactions**

Transactions in Redis work by queueing multiple commands to be executed as a group. When a transaction is executed, the commands queued inside of it are guaranteed to execute without any other commands from other clients interleaved between them.

To begin a transaction block, enter the **MULTI** command. Further commands will be queued and not executed immediately. Running the **EXEC** command will execute all of the queued commands as a transactional unit. If you decide you want to abort an open transaction while queuing commands, running the **DISCARD** command will close the transaction block without running any of the queued commands.

**What is data expiration?**

Data expiration is a feature that can automatically delete a key and value in the cache after a set amount of time.

**How to use data expiration in Azure Cache for Redis**

* **EXPIRE**: Sets the timeout of a key in seconds
* **PEXPIRE**: Sets the timeout of a key in milliseconds
* **EXPIREAT**: Sets the timeout of a key using an absolute Unix timestamp in seconds
* **PEXPIREAT**: Sets the timeout of a key using an absolute Unix timestamp in milliseconds
* **TTL**: Returns the remaining time a key has to live in seconds
* **PTTL**: Returns the remaining time a key has to live in milliseconds
* **PERSIST**: Makes a key never expire

**What is an eviction policy?**

**Azure Cache for Redis supports eviction policies, which indicate how data should be handled when you run out of memory.**

An eviction policy is a plan that determines how your data should be managed when you exceed the maximum amount of memory available. For example, using an eviction policy, you could tell Azure Cache for Redis to delete a random key to make room for the new data being inserted.

**Types of eviction policies**

|  |  |
| --- | --- |
| **Policy** | **Description** |
| **noeviction** | Returns an error if the memory limit has been reached when trying to insert more data |
| **allkeys-lru** | Evicts the least recently used keys out of all keys |
| **allkeys-lfu** | Evicts the least frequently used keys out of all keys |
| **allkeys-random** | Randomly evicts keys out of all keys |
| **volatile-lru** | Evicts the least recently used keys out of all keys with an “expire” field set |
| **volatile-lfu** | Evicts the least frequently used keys out of all keys with an “expire” field set |
| **volatile-random** | Randomly evicts keys with an “expire” field set |
| **volatile-ttl** | Evicts the shortest time-to-live keys out of all keys with an “expire” field set. |

**What is the cache-aside pattern?**

The cache-aside pattern describes how you can implement a cache in conjunction with a database, to return the most commonly accessed data as quickly as possible.

The cache-aside pattern dictates that when you need to retrieve data from a data source, like a relational database, you should first check for the data in your cache. If the data is in your cache, use it. If the data is not in your cache, then query the database, and when you're returning the data back to the user, add it to your cache. This will then allow you to access the data from your cache the next time it's needed.



**How to manage updating data**

The solution to this problem in the cache-aside pattern is to invalidate the data in the cache. When you update data in your application, you should first delete the data in the cache and then make the changes to the data source directly. By doing this, next time the data is requested, it won't be present in the cache, and the process will repeat.

**Considerations for using the cache-aside pattern**

* **Lifetime:** For cache-aside to be effective, make sure that the expiration policy matches the access frequency of the data. Making the expiration period too short can cause applications to continually retrieve data from the data store and add it to the cache.
* **Evicting:** Caches have a limited size compared to typical data stores, and they'll evict data if necessary. Make sure you choose an appropriate eviction policy for your data.
* **Priming:** To make the cache-aside pattern effective, many solutions will prepopulate the cache with data that they think will be accessed often.
* **Consistency:** Implementing the cache-aside pattern doesn't guarantee consistency between the data store and the cache. Data in a data store can be changed without notifying the cache. This can lead to serious synchronization issues.

**What happens if a command in a Redis transaction fails?  
R:** If a command is queued with incorrect syntax, the transaction will be automatically discarded if you try to execute it. If a command fails during execution, the transaction will complete as normal.

In Redis, commands that fail during execution do not stop a transaction. Commands queued with incorrect syntax will prevent the transaction from being executed.

**Capture and view page load times in your Azure web app with Application Insights**

**What is Application Insights?**

Application Insights is an Azure service that helps you to monitor the performance and behavior of web applications.

It mostly captures two kinds of data: **events and metrics.**

* **Events** are individual data points that can represent any kind of event that occurs in an app. These events can be technical events that occur within the application runtime or those that are related to the business domain of the application or actions taken by users.
* **Metrics** are measurements of values, typically taken at regular intervals, that aren't tied to specific events. Like events, metrics can be related to the application's runtime or infrastructure (like the length of a queue) or related to the application's business domain or users (like how many videos are viewed in an hour).

**Application Insights resources**

Application Insights is represented in **Azure as a resource deployed to one of your subscriptions**. Each Application Insights resource you create is a repository for application telemetry data. It includes the various displays and configuration tools that you can view from the Azure portal. To send telemetry data to an Application Insights resource from an app, you need to configure the app with the instrumentation key of the Application Insights resource. After the resource starts to receive telemetry data, you can use the Azure portal to explore and analyze it.

**Visualizations**

* **Live metrics streams:** Charts that display performance values as they vary in near-real time.
* **Metrics explorer:** Tool that shows how metrics vary over time.
* **Alerts:** Messages automatically sent to app admins when target metrics exceed specified thresholds. You can use alerts to ensure your team is aware of critical issues immediately.
* **Profiler:** Shows how a set of requests, like those for a single web page, were delivered. You can use these profiles, for example, to see which page elements load slowly.
* **Application Map:** Displays the components of an application and how they link to each other. You can use the data shown with each component to diagnose performance bottlenecks and failure hotspots.
* **Usage analysis:** Information about your app's users. For example, you can see numbers of unique users and sessions and information about user retention.

**There are two ways to configure your app to send data to Application Insights:**

* **Runtime instrumentation:** Captures telemetry without requiring you to change the web app's source code
* **Build-time instrumentation:** With this method, developers add a server-side SDK to the web app's code. For example, in an ASP.NET Core app, a developer could reference a NuGet package to access the SDK. When you instrument your app with the Application Insights SDK, you can enable full functionality and the richest set of visualizations in Application Insights. This type of instrumentation also enables you to add custom events and telemetry to your code to monitor unusual or unique behavior.

**Web app requirements**

**Runtime instrumentation and automatic client-side instrumentation** is supported only on **Windows** web apps. These features rely on capabilities of IIS, the web server technology that powers Windows apps on App Service. The use of Application Insights **in Linux apps is fully supported**, but you need to **modify application code to reference the Application Insights SDK.**

**Enabling automatic client-side telemetry**

To automatically inject the JavaScript SDK and necessary configuration into pages served by your web app, add a new application setting named **APPINSIGHTS\_JAVASCRIPT\_ENABLED** and set the value to true.

**You have a Linux-based web app that runs in the Azure App Service. You want to display basic performance data in an Application Insights dashboard. Which method should you use to instrument your app?  
R:** Build-time instrumentation. Because your app is based on Linux, you need to use build-time instrumentation. Runtime instrumentation and automatic client-side telemetry are available only for Windows apps.

**You create a custom chart and pin it to an Application Insights dashboard. Other users of the dashboard can't see the new chart. What should you do to resolve the problem?  
R:** Re-publish the Application Insights dashboard. When you make changes to a dashboard in Azure, other dashboard users won't see the changes until you publish them.

**Capture Web Application Logs with App Service Diagnostics Logging**

**Enable diagnostics logging for apps in Azure App Service**

* **App Service**
  + **Monitoring**
    - **App Service Logs**

<https://docs.microsoft.com/en-us/azure/app-service/troubleshoot-diagnostic-logs>

**What is live log streaming?**

Live log streaming is an easy and efficient way to view live logs for troubleshooting purposes. Live log streaming is designed to provide a quick view of all messages that are being sent to the app logs in the file system, without having to go through the process of locating and opening these logs. To use live logging, you connect to the live log service from the command line, and can then see text being written to the app's logs in real time.

**What logs can be streamed?**

The log streaming service adds a redirect from the file system logs, so you'll see the same information as is saved to those log files. So, if you enable verbose logging for ASP.NET Windows apps, for example, the live log stream will show all your logged messages.



**How to use live log streaming**

**az webapp log tail --name <app name> --resource-group <resource group name>**

**Log file storage locations**

The Azure infrastructure used to run Windows Web apps is not the same as that for Linux apps, and log files are not stored in the same locations.

Windows app log files

For Windows apps, file system log files are stored in a virtual drive that is associated with your Web app. This drive is addressable as **D:\Home**, and includes a LogFiles folder; within this folder are one or more subfolders:

* **Application** - Contains application-generated messages, if File System application logging has been enabled.
* **DetailedErrors** - Contains detailed Web server error logs, if Detailed error messages have been enabled.
* **http** - Contains IIS-level logs, if Web server logging has been enabled.
* **W3SVC<number>** - Contains details of all failed http requests, if Failed request tracing has been enabled.

***Where storage to a Blob container has been enabled, logs are stored in year, month, date, and hour folders, for example:***

2019  
 01  
 10  
 08 - log entries for the period 08:00:00 to 08:59:59 on January 10th 2019  
 09 - log entries for the period 09:00:00 to 09:59:59 on January 10th 2019

**Linux app log files**

For Linux Web apps, the Azure tools currently support fewer logging options than for Windows apps. Redirections to **STDERR** and **STDOUT** are managed through the underlying Docker container that runs the app, and these messages are stored in Docker log files. To see messages logged by underlying processes, such as Apache, you will need to open an SSH connection to the Docker container.

**Methods for retrieving log files**

* **Azure CLI**
  + **az webapp log download --log-file \<\_filename\_\>.zip --resource-group \<\_resource group name\_\> --name \<\_app name\_\>**
* **KUDU**
  + All Azure Web apps have an associated Source Control Management (SCM) service site. This site runs the Kudu service, and other Site Extensions; it is Kudu that manages deployment and troubleshooting for Azure Web Apps, including options for viewing and downloading log files. The specific functionality available in KUDU, and how you download logs, depends on the type of Web app. For Windows apps you can browse to the log file location, and then download the logs; for Linux apps, there may be a download link. One way to access the KUDU console is navigate to **https://<app name>.scm.azurewebsites.net**, and then sign in using deployment credentials.
* **Azure Storage Explorer**
  + To access Windows logs saved to an Azure Blob Storage container, you can use the Azure portal; to view and download the contents of the log file container, select Storage Explorer. Open the relevant year, month, date, and hour folder, then double-click a CSV file to download it to your computer

**Architect API integration in Azure**

**Publish and manage your APIs with Azure API Management**

**Azure API management**

The Azure API management service is hosted in the Azure cloud and is positioned between your APIs and the Internet. An Azure API gateway is an instance of the Azure API management service.

When you publish your APIs, you use the Azure portal to control how particular APIs are exposed to consumers. You might want some APIs to be freely available to developers, for demo purposes, and access to other APIs to be tightly controlled.

**Why use API Management?**

* **API documentation -** Documentation of APIs enables calling clients to quickly integrate their solutions. API Management allows you to quickly expose the structure of your API to calling clients through modern standards like Open API. You can have more than one version of an API. With multiple versions, you can stage app updates as your consuming apps don't have to use the new version straight away.
* **Rate limiting access -** If your API could potentially access a large amount of data, it's a good idea to limit the rate at which clients can request data. Rate limiting helps maintain optimal response times for every client. API Management let you set rate limits as a whole or for specific individual clients.
* **Health monitoring -** APIs are consumed by remote clients. So it can be difficult to identify potential problems or errors. API Management lets you view error responses and log files, and filter by types of responses.
* **Modern formats like JSON -** APIs have used many different data exchange formats over the years from XML to CSV and many more. API Management enables you to expose these formats using modern data models like JSON.
* **Connections to any API -** In many businesses, APIs are located across different countries and use different formats. API Management lets you add all of these disparate APIs into single modern interface.
* **Analytics -** As you develop your APIs, it's useful to see how often your APIs are being called and by which types of systems. API Management allows you to visualize this data within the Azure portal.
* **Security -** Security is paramount when dealing with system data. Unauthorized breaches can cost companies money, time lost in reworking code, and reputational loss. Security tools that you can use with Azure API management include OAuth 2.0 user authorization, and integration with Azure Active Directory.

**Pricing tiers**

When you create an Azure API management gateway, you must select from one of several pricing tiers:

* **Developer**. You use the Developer tier for evaluating the API management service. You shouldn't use this tier for production deployments.
* **Basic**. Entry level production use. 99.9% SLA. 1000 requests/sec. Two scale units.
* **Standard**. Medium level production use. 99.9% SLA. 2500 requests/sec. Four scale units.
* **Premium**. Multi region deployment. High volume use. 99.95% SLA. 4000 requests/sec. 10 scale units per region.
* **Consumption**. The serverless consumption tier plan lets you pay for what you use, rather than having dedicated resources. You can quickly set up ad-hoc testing, and you can scale up API access when demand increases. The consumption tier has built-in high availability and autoscaling. Because it is serverless, you can provision a consumption tier gateway in a lot less time than the other server-based tiers.

**API frameworks**

There are various API frameworks and standards. API Management provides you with several options for importing APIs.

|  |  |
| --- | --- |
| **Type** | **Details** |
| **Blank API** | You can import an API with a blank API definition. You then manually specify all the required parameters. |
| **Open API** | Open API is a specification that documents all the endpoints and operations for RESTful APIs, and all input and output parameters. OpenAPI was originally called Swagger. |
| **WADL** | Web Application Description Language is an XML description of HTTP-based web services. It is a simpler format and more lightweight than WSDL. |
| **WSDL** | Web Service Description Language is an XML description of any network service, not just HTTP. |
| **Logic App** | Logic apps are used to orchestrate and automate workflows and integrations with various data sources. |
| **API App** | An API hosted within an API app service in Azure. |
| **Function App** | Serverless code that can be called through triggers. |

**Policies**

Policies provide powerful capabilities to change the behavior of an API through configuration. They exist as a collection of statements that are executed sequentially on the request or response of an API.

**Popular configurations include:**

* Conversion from XML to JSON
* Call rate limiting to restrict the number of incoming calls.
* Setting inbound and outbound headers

**Product**

A product is a collection of APIs. You can assign APIs to more than one product.

For all pricing tiers except consumption, there two default products: **Starter and Unlimited.**

* **Unlimited** product is designed for production API management, as it has no restrictions on the number of attached APIs. You can create as many new products as you need.
* **Starter** product has a limit of five API calls/minute, and a maximum of 100 API calls/week.

**Subscriptions and keys**

To protect your APIs, you can use a subscription key.

A subscription key is a unique auto-generated string which needs to be passed through in the headers of the client request. The key is directly related to a subscription which can be scoped to different areas, this gives granular control over permissions and policies. The three main subscription scopes are:

* **All APIs**
* **A Single API**
* **A product**



Every subscription has two keys, a primary and a secondary. Having two keys makes it easier when you do need to regenerate a key. For example, if you want to change the primary key and avoid downtime, use the secondary key in your apps.

For products where subscriptions are enabled, you must supply a key when making calls to APIs in that product. As a developer, **you can obtain a key by submitting a subscription request. This is a core part of the APIM workflow.**

**Call an API with the subscription key**

The default header name is **Ocp-Apim-Subscription-Key**, and the default query string is **subscription-key.**

**Improve the performance of an API by adding a caching policy in Azure API Management**

**When do policies execute?**

In Azure API Management, policies execute at four different times**:**

* **Inbound. These policies execute when a request is received from a client.**
* **Backend. These policies execute before a request is forwarded to a managed API.**
* **Outbound. These policies execute before a response is sent to a client.**
* **On-Error. These policies execute when an exception is raised.**

In the policy XML, there is a separate tag for each of these execution times:

**Policy Scopes**

A policy's scope determines how broadly it will be applied. There are four possible scopes that you can choose from.

* **The global policy scope** - Policies applied at the global scope affect all APIs within the instance of API Management.
* **The product policy scope** - Policies applied to the product scope affect all the APIs in that product. APIs in other products are unaffected.
* **The API policy scope** - Policies applied at the API scope affect only a single API.
* **Operation policy scope** - Policies applied at the operation scope affect only one operation within the API. In the example below, the administrator has selected the GetSpeaker operation within the Demo Conference API and can set inbound, outbound, or backend policies that apply only to that operation:  
  <https://docs.microsoft.com/en-us/learn/modules/improve-api-performance-with-apim-caching-policy/media/2-operation-scope.png>

**Which order are policies applied in?**

You can use the <base /> tag to determine when policies from a higher scope are applied. For example, consider this policy, applied at the API scope:

**<policies>  
 <inbound>  
 <base />  
 <find-and-replace from="game" to="board game" />  
 </inbound>  
</policies>**

Because the <base> tag appears above the <find-and-replace> tag, Azure applies policies from the **global and product scopes first**, and then executes the find-and-replace policy.

Advanced policies

These policies can be of use in scenarios when you want non-standard behavior:

For example, if you want to apply a policy only when the response passes a specific test, use the Control flow policy.

Use the Forward request policy to forward a request to a backend server.

To control what happens when an action fails, use the Retry policy. Policy statements enclosed in Retry will execute repeated until a condition is met. Execution will repeat at the specified time intervals up until the retry count value is reached.

The Send one-way request policy can send a request to a URL without waiting for a response.

If you want to store a value for use in a later calculation or test, use the Set variable policy to persist a value in a named variable.

**How to control the API Management cache**

To set up a cache, you use an outbound policy named **cache-store** to store responses. You also use an inbound policy named **cache-lookup** to check if there is a cached response for the current request.

It's also possible to store individual values in the cache, instead of a complete response. Use the **cache-store-value** policy to add the value, with an identifying key. Retrieve the value from the cache by using the **cache-lookup-value** policy. If you want to remove a value before it expires, use the **cache-remove-value** policy

**Using vary-by tags**

With this policy, the cache will store and separate responses for each product because they have different part numbers. The cache will not store separate responses for each customer because that query parameter is not listed.

Azure does not, by default, examine HTTP headers to determine whether a cached response is suitable for a given request. If a header can make a significant difference to a response use the **<vary-by-header>** tag.

Within the **<cache-lookup>** tag**, there is also the vary-by-developer attribute**, which is required to be present and set **to false by default**. When this attribute is set to true, API Management examines the subscription key supplied with each request. **It serves a response from the cache only if it was originally requested with the same subscription key.** Set this attribute to true when each user should see a different response for the same URL. If each user group should see a different response for the same URL, set the **vary-by-developer-group** attribute to true.

**Using an external cache**

You might choose to use an external cache because:

* You want to avoid the cache being cleared when the API Management service is updated.
* You want to have greater control over the cache configuration than the internal cache allows.
* You want to cache more data than can be store in the internal cache.

Another reason to configure an external cache is that you want to use caching with the consumption pricing tier. **This tier follows serverless design principal and you should use it with serverless web APIs. For this reason, it has no internal cache.** If you want to use caching with an API Management instance in the consumption tier, you must use an external cache.

**Protect your APIs on Azure API Management**

**API management setup**

To set up API management, you will perform the following tasks:

* Create an API Management gateway. In this step, you create the API Management resource in the Azure portal. You also assign properties to the gateway, such as an FQDN and a pricing tier.
* Register an existing Web API with the gateway. In this step, you add the web API to the gateway. The API already has its own Azure app service host but you must add it to API Management in order to use policies and other API Management tools.
* Remove headers from the response. In this step, you will apply a policy that removes the insecure headers from all responses.

**Remove headers**

Now we'll add a policy to remove the X-Powered-By header from responses sent by the API:

* Select Census Data and then, at the top of the screen, select the Design tab.
* Select All operations, and then in the **Outbound** processing section, click the </> icon.
* Replace the default <outbound> tag with this code:
  + <outbound>
  + <set-header name="X-Powered-By" exists-action="delete" />
  + <base />
  + </outbound>

**Transformation policy**

A transformation policy modifies the content of an API call. Some transformation policies apply to the header and others apply to the body. The following transforms are available:

|  |  |
| --- | --- |
| **Transform** | **Detail** |
| **Convert JSON to XML** | Converts a request or response body from JSON to XML. |
| **Convert XML to JSON** | Converts a request or response body from XML to JSON. |
| **Find and replace string in body** | Finds a request or response substring and replaces it with a different substring. |
| **Mask URLs in content** | Rewrites links in the response body so that they point to the equivalent link through the gateway. |
| **Set backend service** | Changes the backend service for an incoming request. |
| **Set body** | Sets the message body for incoming and outgoing requests. |
| **Set HTTP header** | Assigns a value to an existing response or request header, or adds a new response or request header. |
| **Set query string parameter** | Adds, replaces the value of, or deletes a request query string parameter. |
| **Rewrite URL** | Converts a request URL from its public form to the form expected by the web service. |
| **Transform XML using an XSLT** | Applies an XSL transformation to the XML in the request or response body. |

**Mask URLs in content**

1. On the top of the screen, select the Design tab, and then select All operations.
2. In the **Outbound** processing section, click the </> icon.
3. Inside the <outbound> element, below the <set-header-name> element you added previously, add the following element:
   1. **<redirect-content-urls />**

**Throttle API requests**

It is common to find that a few users over-use an API to the extent that you incur extra costs or that responsiveness to other uses is reduced. You can use throttling to limit access to API endpoints by putting limits on the number of times an API can be called within a specified period of time.

**Limit by subscription throttling**

Subscription throttling allows you to set the rate limits by a specific API operation. It does not discriminate by the client. Instead, every request to the API or the specified operation is throttled in the same way. Using our census example, we could use subscription throttling to limit the number of times any of the APIs are called within a certain period. This configuration would result in clients receiving a 429 error when that limit was reached. The problem with this type of throttling is that it allows one client to use up all the requests before another client can use it.

For example, the following code demonstrates an example configuration that applies to all API operations:

**<rate-limit calls="3" renewal-period="15" />**

Alternatively, this configuration can be used to target a particular API operation:

**<rate-limit calls="number" renewal-period="seconds">  
 <api name="API name" id="API id" calls="number" renewal-period="seconds" />  
 <operation name="operation name" id="operation id" calls="number" renewal-period="seconds" />  
 </api>  
</rate-limit>**

**Limit by key throttling**

Key throttling allows you to configure different rate limits by any client request value. This type of throttling offers a better way of managing the rate limits as it applies the limit to a specified request key, often the client IP address. It gives every client equal bandwidth for calling the API When you choose to throttle by key, you will need to decide on specific requirements for rate limiting. For example, the table below lists three common ways of specifying the counter-key:

|  |  |
| --- | --- |
| **Value** | **Detail** |
| **context.Request.IpAddress** | Rates limited by client IP address |
| **context.Subscription.Id** | Rates limited by subscription ID |
| **context.Request.Headers.GetValue("My-Custom-Header-Value")** | Rates limited by a specified client request header value |

You may decide that you want each individual client IP to have its own bandwidth set, in which case you would use the context.Request.IpAddress. Alternatively, it could be that you want all requests from a particular domain name to be throttled as certain domains have many calls to the API. In that **case you would specify context.Request.Headers.GetValue("host") which would rate limit by the domains from which the call was made.**

**Control authentication for your APIs with Azure API Management**

**Use client certificates to secure access to an API**

Certificates can be used to provide TLS mutual authentication between the client and the API gateway. You can configure the API Management gateway to allow only requests with certificates containing a specific thumbprint. The authorization at the gateway level is handled through inbound policies.

**TLS client authentication**

With TLS client authentication, the API Management gateway can inspect the certificate contained within the client request and check for properties like:

|  |  |
| --- | --- |
| **Property** | **Reason** |
| **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 |

These properties are not mutually exclusive and **they can be mixed together** to form your own policy requirements. For instance, you can specify that the certificate passed in the request is signed by a certain certificate authority and hasn't expired.

Client certificates are signed to ensure that they are not tampered with. When a partner sends you a certificate, verify that it comes from them and not an imposter. There are two common ways to verify a certificate:

* Check who issued the certificate. If the issuer was a certificate authority that you trust, you can use the certificate. You can configure the trusted certificate authorities in the Azure portal to automate this process.
* If the certificate is issued by the partner, verify that it came from them. For example, if they deliver the certificate in person, you can be sure of its authenticity. These are known as self-signed certificates.

**Accept client certificates in the Consumption tier**

The Consumption tier in API Management is designed to conform with serverless design principals. If you build your APIs from serverless technologies, such as Azure Functions, this tier is a good fit. In the Consumption tier, you must explicitly enable the use of client certificates, which you can do on the Custom domains page. This step is not necessary in other tiers.



**Check the thumbprint of a client certificate**

The thumbprint ensures that the values in the certificate have not been altered since the certificate was issued by the certificate authority. You can check the thumbprint in your policy. The following example checks the thumbprint of the certificate passed in the request.

<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>

**Check the thumbprint against certificates uploaded to API Management**

<when condition="@(context.Request.Certificate == null || !context.Request.Certificate.Verify() || !context.Deployment.Certificates.Any(c => c.Value.Thumbprint == context.Request.Certificate.Thumbprint))" >

**Check the issuer and subject of a client certificate**

<when condition="@(context.Request.Certificate == null || context.Request.Certificate.Issuer != "trusted-issuer" || context.Request.Certificate.SubjectName.Name != "expected-subject-name")" >

**Build a containerized web application with Docker**

**Customize a Docker image to run your own web app**

The following example shows a Dockerfile that builds a .NET Core 2.2 application and packages it into a new image.



* The FROM statement downloads the specified image and creates a new container based on this image.
* The WORKDIR command sets the current working directory in the container, used by the following commands.
* The COPY command copies files from the host computer to the container. The first argument (myapp\_code) is a file or folder on the host computer. The second argument (.) specifies the name of the file or folder to act as the destination in the container. In this case, the destination is the current working directory (/app).
* The RUN command executes a command in the container. Arguments to the RUN command are command-line commands.
* The EXPOSE command creates configuration in the new image that specifies which ports are intended to be opened when the container is run. If the container is running a web app, it's common to EXPOSE port 80.
* The ENTRYPOINT command specifies the operation the container should run when it starts. In this example, it runs the newly built app. You specify the command to be run and each of its arguments as a string array.

**docker build -t myapp:v1 .**

The docker build command creates a new image by running a Dockerfile. The -f flag indicates the name of the Dockerfile to use. The -t flag specifies the name of the image to be created, in this example, myapp:v1. The final parameter, ., provides the build context for the source files for the COPY command: the set of files on the host computer needed during the build process.

**Deploy a Docker image to an Azure Container Instance**

Azure Container Instance loads and runs Docker images on demand. The Azure Container Instance service can retrieve the image from a registry such as Docker Hub or Azure Container Registry.

Your organization wants to use Azure to run its web apps. For this reason, it makes sense to store the images in Azure Container Registry, and run them using the Azure Container Instance service.

**Use Azure Container Registry to store a container**

Azure Container Registry is a registry hosting service provided by Azure. Each Azure Container Registry resource you create is a separate registry with a unique URL. These registries are private: they require authentication to push or pull images. Azure Container Registry runs in the cloud, and provides similar levels of scalability and availability to many other Azure services.

**You create a registry either using the Azure portal, or using the acr create command in the Azure Command Line Interface, as shown in the following example:**

**az acr create --name myregistry --resource-group mygroup --sku standard --admin-enabled true**

**Azure Container Registry repositories are private -** they do not support unauthenticated access. To pull images from an Azure Container Registry repository, use the docker login command and specify the URL of the login server for the registry. The login server URL for a registry in Azure Container Registry has the form <registry\_name>.azurecr.io.

**docker login myregistry.azurecr.io**

You'll be prompted for a username and password. To find this information, you can either go to the Azure portal and look up the access keys for the registry, or you can run the following command.

**az acr credential show --name myregistry**

The following example creates an alias for the myapp:v1 image

**docker tag myapp:v1 myregistry.azurecr.io/myapp:v1**

Upload the image to the registry in Azure Container Registry.

**docker push myregistry.azurecr.io/myapp:v1**

Verify that the image has been uploaded correctly by querying the repositories in the registry with the acr repository list command.

**az acr repository list --name myregistry**

**Use Azure Container Instance to run an image**

**az container create --resource-group mygroup --name myinstance --image myregistry.azurecr.io/myapp:latest --dns-name-label mydnsname --registry-username <username> --registry-password <password>**

**Upload image to Azure Container Registry**

1. In your local command line, run the following command to tag the reservationsystem image with the name of your registry. Replace <registry-name> with the name of your registry in Azure Container Registry.

**docker tag reservationsystem:latest <registry-name>.azurecr.io/reservationsystem:latest**

1. Sign in to your registry in Azure Container Registry. Use the docker login command and specify the login server for the registry that you noted earlier. Enter the username and password for the registry when prompted.

**docker login <login-server>**

1. Upload the image to the registry in Azure Container Registry with the docker push command.

**docker push <registry-name>.azurecr.io/reservationsystem:latest**

**Build and store container images with Azure Container Registry**

**Introduction to Azure Container Registry**

Azure Container Registry is a managed Docker registry service based on the open-source Docker Registry 2.0. Container Registry is private, hosted in Azure, and allows you to build, store, and manage images for all types of container deployments.

Container images can be pushed and pulled with Container Registry using the Docker CLI or the Azure CLI. Azure portal integration allows you to visually inspect the container images in your container registry. In distributed environments, the Container Registry geo-replication feature can be used to distribute container images to multiple Azure datacenters for localized distribution.

In addition to storing container images, Azure Container Registry Tasks can build container images in Azure. Tasks use a standard Dockerfile to create and store a container image in Azure Container Registry without the need for local Docker tooling. With Azure Container Registry Tasks, you can build on demand or fully automate container image builds using DevOps processes and tooling.

*(premium registry SKU is deployed. The premium SKU is required for geo-replication.)*

**Create a container image with Azure Container Registry Tasks**

A standard Dockerfile provides build instructions. Azure Container Registry Tasks enables you to reuse any Dockerfile currently in your environment, including multi-staged builds.

Run the following Azure CLI command to build the container image from the Dockerfile. $ACR\_NAME is the variable you defined in the preceding unit to hold your container registry name.

**az acr build --registry $ACR\_NAME --image helloacrtasks:v1 .**

*. = Since we didn't specify the name of a file with the --file parameter, the command looks for a file called Dockerfile in our current directory.*

Run the following command in the Cloud Shell to verify that the image has been created and stored in the registry.

**az acr repository list --name $ACR\_NAME --output table**

**About registry authentication**

Azure Container Registry doesn't support unauthenticated access and require authentication for all operations. Registries support two types of identities:

* **Azure Active Directory identities**, including both user and service principals. Access to a registry with an Azure Active Directory identity is role-based, and identities can be assigned one of three roles: reader (pull access only), contributor (push and pull access), or owner (pull, push, and assign roles to other users).
* **The admin account** included with each registry. The admin account is disabled by default.

**Geo-replication**  
Suppose your company has compute workloads deployed to several regions to make sure you have a local presence to serve your distributed customer base.

Your aim is to place a container registry in each region where images are run. This strategy will allow for network-close operations, enabling fast, reliable image layer transfers.

Geo-replication enables an Azure container registry to function as a single registry, serving several regions with multi-master regional registries.

**A geo-replicated registry provides the following benefits:**

* Single registry/image/tag names can be used across multiple regions
* Network-close registry access from regional deployments
* No additional egress fees, as images are pulled from a local, replicated registry in the same region as your container host
* Single management of a registry across multiple regions

**Create a replicated region for an Azure Container Registry**

1. Run the following command to replicate your registry to another region. In this example, we're replicating to the japaneast region. $ACR\_NAME is the variable you defined earlier in the module to hold your container registry name.  
   **az acr replication create --registry $ACR\_NAME --location japaneast**
2. As a final step, retrieve all container image replicas created by running the following command.  
   **az acr replication list --registry $ACR\_NAME --output table**



**Suppose you use container images to run compute workloads in multiple regions throughout the world. You plan to enable the geo-replication feature of Azure Container Registry to decrease the time required to provision an instance. In which regions should you configure the Azure Container Registry geo-replication feature?  
R:** Place a container registry in each region where images are run. Placing a registry in each region that runs the images will ensure network-close registry access everywhere it is needed.

**Run Docker containers with Azure Container Instances**

**Create a container**

**az container create \  
 --resource-group learn-deploy-aci-rg \  
 --name mycontainer \  
 --image microsoft/aci-helloworld \  
 --ports 80 \  
 --dns-name-label $DNS\_NAME\_LABEL \  
 --location eastus**

**Check its status.**

**az container show \  
 --resource-group learn-deploy-aci-rg \  
 --name mycontainer \  
 --query "{FQDN:ipAddress.fqdn,ProvisioningState:provisioningState}" \  
 --out table**

**What are container restart policies?**

**Restart policy Description**

|  |  |
| --- | --- |
| Always | Containers in the container group are always restarted. This policy makes sense for long-running tasks such as a web server. This is the default setting applied when no restart policy is specified at container creation. |
| Never | Containers in the container group are never restarted. The containers run one time only. |
| OnFailure | Containers in the container group are restarted only when the process executed in the container fails (when it terminates with a nonzero exit code). The containers are run at least once. This policy works well for containers that run short-lived tasks. |

**--restart-policy OnFailure**

*Azure Container Instances starts the container and then stops it when its process (a script, in this case) exits. When Azure Container Instances stops a container whose restart policy is* ***Never or OnFailure****, the container's status is set to* ***Terminated****.*

**Set environment variables**

Environment variables enable you to dynamically configure the application or script the container runs. You can use the Azure CLI, PowerShell, or the Azure portal to set variables when you create the container. Secured environment variables enable you to prevent sensitive information from displaying in the container's output.

**COSMOS\_DB\_NAME=aci-cosmos-db-$RANDOM**

**COSMOS\_DB\_ENDPOINT=$(az cosmosdb create \  
 --resource-group learn-deploy-aci-rg \  
 --name $COSMOS\_DB\_NAME \  
 --query documentEndpoint \  
 --output tsv) – (**[**https://aci-cosmos-db-666.documents.azure.com:443/**](https://aci-cosmos-db-666.documents.azure.com:443/)**)**

**COSMOS\_DB\_MASTERKEY=$(az cosmosdb keys list \  
 --resource-group learn-deploy-aci-rg \  
 --name $COSMOS\_DB\_NAME \  
 --query primaryMasterKey \  
 --output tsv) – (bvd0HU0k870WUfA76ZONKpkccWvccVp127tcbjh8oawUzNMcpew9JkQxYXzvmwQeL79QexP7LyN8Zqzu7OtyOA==)**

**az container create \  
 --resource-group learn-deploy-aci-rg \  
 --name aci-demo \  
 --image microsoft/azure-vote-front:cosmosdb \  
 --ip-address Public \  
 --location eastus \  
 --environment-variables \  
 COSMOS\_DB\_ENDPOINT=$COSMOS\_DB\_ENDPOINT \  
 COSMOS\_DB\_MASTERKEY=$COSMOS\_DB\_MASTERKEY**

*Secure environment variables prevent clear text output. To use secure environment variables, you use the* ***--secure-environment-variables*** *argument instead of the --environment-variables argument.*

[ {

"name": "COSMOS\_DB\_ENDPOINT",  
 "secureValue": null,  
 "value": null  
 },  
 {  
 "name": "COSMOS\_DB\_MASTERKEY",  
 "secureValue": null,  
 "value": null  
 }]

**Use data volumes**

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.

**Get storage credentials**

To mount an Azure file share as a volume in Azure Container Instances, you need these three values:

1. **The storage account name**
2. **The share name**
3. **The storage account access key**

**Run this az container create command to create a container that mounts /aci/logs/ to your file share.**

**az container create \  
 --resource-group learn-deploy-aci-rg \  
 --name aci-demo-files \  
 --image microsoft/aci-hellofiles \  
 --location eastus \  
 --ports 80 \  
 --ip-address Public \  
 --azure-file-volume-account-name $STORAGE\_ACCOUNT\_NAME \  
 --azure-file-volume-account-key $STORAGE\_KEY \  
 --azure-file-volume-share-name aci-share-demo \  
 --azure-file-volume-mount-path /aci/logs/**

**Troubleshoot Azure Container Instances**

To help you understand basic ways to troubleshoot container instances, here you'll perform some basic operations such as:

* Pulling container logs
* Viewing container events
* Attaching to a container instance

**Get logs from your container instance**Run the following az container logs command to see the output from the container's running application.

**az container logs \  
 --resource-group learn-deploy-aci-rg \  
 --name mycontainer**

**Get container events**

***The az container attach command provides diagnostic information during container startup. Once the container has started, it also writes standard output and standard error streams to your local terminal.***

Run az container attach to attach to your container.

**az container attach \  
 --resource-group learn-deploy-aci-rg \  
 --name mycontainer**

**Execute a command in your container**

az container exec \  
 --resource-group learn-deploy-aci-rg \  
 --name mycontainer \  
 --exec-command /bin/sh

**Monitor CPU and memory usage on your container**

* Run the az monitor metrics list command to retrieve CPU usage information.

**az monitor metrics list \  
 --resource $CONTAINER\_ID \  
 --metric CPUUsage \  
 --output table**

**Note the --metric argument. Here, CPUUsage specifies to retrieve CPU usage.**

* Run this az monitor metrics list command to retrieve memory usage information.

**az monitor metrics list \  
--resource $CONTAINER\_ID \**

**CPU Usage measured in millicores: One millicore is 1/1000th of a CPU core, so 500 millicores represents usage of 0.5 CPU core.**

**--metric MemoryUsage \  
--output table**

**Here, you specify MemoryUsage for the --metric argument to retrieve memory usage information.**

**Which troubleshooting command can be used to view container startup events?  
R: The az container attach command** shows container events and logs. By contrast, the az container logs only shows the logs and not the startup events.

**Create an Azure Cosmos DB database built to scale**

**What is an Azure Cosmos DB account?**

An Azure Cosmos DB account is an Azure resource that acts as an organizational entity for your databases. It connects your usage to your Azure subscription for billing purposes.

Each Azure Cosmos DB account is associated with one of the several data models Azure Cosmos DB supports, and you can create as many accounts as you need.

**Provisioning throughput for containers and databases**

In Azure Cosmos DB, you provision throughput for your containers to run writes, reads, updates, and deletes. **You can provision throughput for an entire database** and have it shared among containers within the database. **You can also provision throughput dedicated to specific containers (it's evenly distributed among its physical partitions.).**

*If your requests consume all of the provisioned throughput, Azure Cosmos DB will rate-limit your requests. Operations will have to wait and retry, likely causing higher latency.*

**What is a request unit?**

Azure Cosmos DB measures throughput using something called a request unit (RU). Request unit usage is measured per second, so the unit of measure is request units per second (RU/s). You must reserve the number of RU/s you want Azure Cosmos DB to provision in advance, so it can handle the load you've estimated, and you can scale your RU/s up or down at any time to meet current demand.

**Request unit basics**

A single request unit, one RU, is equal to the approximate cost of performing a single GET request on a 1-KB document using a document's ID. Performing a GET by using a document's ID is an efficient means for retrieving a document, and thus the cost is small. **Creating, replacing, or deleting the same item requires additional processing by the service, and therefore requires more request units.**

*The number of request units consumed for an operation changes depending on the document size, the number of properties in the document, the operation being performed, and some additional* ***concepts such as consistency and indexing policy.***

**Azure Cosmos DB guarantees that the number of RUs for a given database operation for the same dataset is deterministic.**

**Request Unit considerations**

* **Item size:** As the size of an item increases, the number of RUs consumed to read or write the item also increases.
* **Item indexing:** By default, each item is automatically indexed. Fewer RUs are consumed if you choose not to index some of your items in a container.
* **Item property count:** Assuming the default indexing is on all properties, the number of RUs consumed to write an item increases as the item property count increases.
* **Indexed properties:** An index policy on each container determines which properties are indexed by default. To reduce the RU consumption for write operations, limit the number of indexed properties.
* **Data consistency:** The strong and bounded staleness consistency levels consume approximately two times more RUs on read operations when compared to that of other relaxed consistency levels.
* **Query patterns:** The complexity of a query affects how many RUs are consumed for an operation. Factors that affect the cost of query operations include:
  + The number of query results
  + The number of predicates
  + The nature of the predicates
  + The number of user-defined functions
  + The size of the source data
  + The size of the result set
  + Projections
* **Script usage:** As with queries, stored procedures and triggers consume RUs based on the complexity of their operations. As you develop your application, inspect the request charge header to better understand how much RU capacity each operation consumes.

**Exceeding throughput limits**

If you attempt to use throughput higher than the one provisioned, your request will be rate-limited. When a request is rate-limited, the request has to be retried again after a specified interval. The .NET SDK will automatically retry your request after waiting the amount of time specified in the retry-after header.

**When you create an account, you can provision a minimum of 400 RU/s, or a maximum of 250,000 RU/s in the portal. If you need even more throughput, fill out a ticket in the Azure portal.**

**What is a partition strategy?**

***A partition strategy ensures that your database can grow to efficiently run queries and transactions.***

If you continue to add new data to a single server or a single partition, it will eventually run out of space. A partitioning strategy enables you to add more partitions to your database when need them. This scaling strategy is called scale out or horizontal scaling.

*A partition key defines the partition strategy,* ***it's set when you create a container and can't be changed.*** *Selecting the right partition key is an important decision to make early in your development process.*

**What is a partition key?**

***A partition key is the value by which Azure organizes your data into logical divisions.*** It should aim to evenly distribute operations across the database to avoid hot partitions. **A hot partition is a single partition** that receives many more requests than the others, which can create a throughput **bottleneck**.

*In our online retail scenario, using the userID or productId value as the partition key is a good choice because it will be unique and likely used to look up records. userID is a good choice, as your application frequently needs to retrieve the personalization settings, shopping cart, order history, and profile information for the user, just to name a few. productId is also a good choice, as your application needs to query inventory levels, shipping costs, color options, warehouse locations, and more.*

The amount of required RU's and storage determines the number of **required physical partitions** for the container, which are completely managed by Azure Cosmos DB. When additional physical partitions are needed, Cosmos DB automatically creates them by splitting existing ones. There is no downtime or performance impact for the application.

The storage space for the data associated with each partition key can't exceed 20 GB, which is the size of one physical partition in Azure Cosmos DB. Think about using a composite key instead so that each record is smaller. An example of a composite key would be userID-date, which would look like CustomerName-08072018. This composite key approach would enable you to create a new partition for each day a user visited the site.

**Best practices**

* Don't be afraid of choosing a partition key that has a large number of values. The more values your partition key has, the more scalability you have.
* To determine the best partition key for a read-heavy workload, review the top three to five queries you plan on using. The value most frequently included in the WHERE clause is a good candidate for the partition key.
* For write-heavy workloads, you'll need to understand the transactional needs of your workload, because the partition key is the scope of multi-document transactions.

**True or false: You can add a partition key to an Azure Cosmos DB container after it has been created?  
R:** False

**Choose the appropriate API for Azure Cosmos DB**

Azure Cosmos DB was chosen because it is flexible. At the lowest level, Azure Cosmos DB stores data in atom-record-sequence **(ARS) format**. The data is then abstracted and projected as an API, which you specify when you are creating your database.

**Core (SQL) API -** Core (SQL) is the default API for Azure Cosmos DB, which provides you with a view of your data that resembles a traditional NoSQL document store. You can query the hierarchical JSON documents with a SQL-like language. Core (SQL) uses JavaScript's type system, expression evaluation, and function invocation.

**{  
 "id": "cc410485-e177-4cbf-95e1-708f7d5e9297",  
 "productName": "Industrial Saw",  
 "description": "Cuts through anything",  
 "supplier": "Hammer & Nail Inc",  
 "quantity": 261,  
 "unitCost": "$10.47",  
 "retailPrice": "$29.99",  
 "categories" : [  
 {"name": "hammers"},  
 {"name": "hand tools"}  
 ]  
}**

**SELECT c.productName FROM Items c**

**Core (SQL) provides several familiar SQL statements and clauses; for example:**

* SELECT
* FROM
* WHERE
* BETWEEN
* COUNT
* SUM
* MIN
* MAX
* ORDER BY

**MongoDB API**

Azure Cosmos DB's API for MongoDB supports the MongoDB wire protocol. This API allows existing MongoDB client SDKs, drivers, and tools to interact with the data transparently, as if they are running against an actual MongoDB database. The data is stored in document format, which is the same as using Core (SQL). Azure Cosmos DB's API for MongoDB is currently compatible with 3.2 version of the MongoDB wire protocol.

**db.Items.find({},{productName:1,\_id:0})**

**Cassandra API**

Makes it possible to query data by using the Cassandra Query Language (CQL), and your data will appear to be a partitioned row store. Just like the MongoDB API, any clients or tools should be able to connect transparently to Azure Cosmos DB; only your connection settings should need to be updated. Cosmos DB's Cassandra API currently supports version 4 of the CQL wire protocol.

To create a table that would store the JSON information listed earlier, you might use the following syntax:

**CREATE TABLE Catalog.Items(id text, productName text, description text, supplier text, quantity int, unitCost float, retailPrice float, categories map<text,text>, primary key (id));**

To retrieve a product's name, you could use this CQL command:

**SELECT id, productName FROM catalog.items**

**Azure Table API**

Provides support for applications that are written for Azure Table Storage that need premium capabilities like global distribution, high availability, scalable throughput. The original Table API only allows for indexing on the Partition and Row keys; there are no secondary indexes. Storing table data in Cosmos DB automatically indexes all the properties, and requires no index management.

Querying is accomplished by using OData and LINQ queries in code, and the original REST API for GET operations.

To retrieve a product's name, you could use this SQL command within a stored procedure or user-defined function:

**SELECT i.productName FROM Items i**

**Gremlin (graph) API**

Choosing Gremlin as the API provides a graph-based view over the data. Remember that at the lowest level, all data in any Azure Cosmos DB is stored in an ARS format. A graph-based view on the database means data is either a vertex (which is an individual item in the database), or an edge (which is a relationship between items in the database).

You typically use a traversal language to query a graph database, and Azure Cosmos DB supports Apache Tinkerpop's Gremlin language.

*This kind of graph might be useful when you are creating a product recommendation application for an e-commerce website, or you could return detailed information for each of the related purchases by using the following query:*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Core (SQL) | MongoDB | Cassandra | Azure Table | Gremlin |
| New projects being created from scratch | ✔ |  |  |  |  |
| Existing MongoDB, Cassandra, Azure Table, or Gremlin data |  | ✔ | ✔ | ✔ | ✔ |
| Analysis of the relationships between data |  |  |  |  | ✔ |
| All other scenarios | ✔ |  |  |  |  |

**Use Core (SQL) to store a product catalog**

<https://docs.microsoft.com/en-us/learn/modules/choose-api-for-cosmos-db/4-use-the-core-sql-api-to-store-a-product-catalog>

**Use the Gremlin (graph) API as a recommendation engine**

<https://docs.microsoft.com/en-us/learn/modules/choose-api-for-cosmos-db/5-use-the-gremlin-graph-api-as-a-recommendation-engine>

**Use MongoDB to import historical order data**

<https://docs.microsoft.com/en-us/learn/modules/choose-api-for-cosmos-db/6-use-the-mongodb-api-to-import-historical-order-data>

**Use Cassandra for web analytics**

<https://docs.microsoft.com/en-us/learn/modules/choose-api-for-cosmos-db/7-use-the-cassandra-api-for-web-analytics>

**Use the Azure Table API to store IoT data**

<https://docs.microsoft.com/en-us/learn/modules/choose-api-for-cosmos-db/8-use-the-azure-table-api-to-store-iot-data>

**Summary**

Your default choice for new Azure Cosmos DB accounts should be Core (SQL). However, you should also consider the following situations:

If your data is better represented in a graph, then the Gremlin (graph) API might be a good choice.

If you already have an existing application or database that is using one of the other APIs, then the current API might be a better choice for your specific scenario. Using the current API might make it easier to:

* Migrate your application or database to Azure Cosmos DB
* Reuse your existing code with minimal changes
* Leverage the existing knowledge and experience of your development team.

You should only use the Azure Table API if you are migrating from Azure Table Storage, as Core (SQL) offers far more features and flexibility.

**Insert and query data in your Azure Cosmos DB database**

**What is the Data Explorer?**

The Azure Cosmos DB Data Explorer is a tool included in the Azure portal that is used to manage data stored in an Azure Cosmos DB. It provides a UI for navigating and viewing data, querying and modifying data, and creating and running stored procedures. The Data Explorer is a great tool to get you acquainted with the inner workings and functionality provided by Azure Cosmos DB.

**Geospatial queries**

Geospatial queries enable you to perform spatial queries using GeoJSON Points. Using the coordinates in the database, you can calculate the distance between two points and determine whether a Point, Polygon, or LineString is within another Point, Polygon, or LineString.

**Define stored procedures and user-defined functions**

The way to perform these transactions in Azure Cosmos DB is by using stored procedures and user-defined functions (**UDFs**). Stored procedures are the only way to ensure **ACID** (Atomicity, Consistency, Isolation, Durability) transactions because they are run on the server, and are thus referred to as server-side programming. UDFs are also stored on the server and are used during queries to perform computational logic on values or documents within the query.

**Stored procedure basics**

Stored procedures perform complex transactions on documents and properties. Stored procedures are written in JavaScript and are stored in a container on Azure Cosmos DB. By performing the stored procedures on the database engine and close to the data, you can improve performance over client-side programming.

Stored procedures are the only way to achieve atomic transactions within Azure Cosmos DB; the client-side SDKs do not support transactions.

Performing batch operations in stored procedures is also recommended because of the reduced need to create separate transactions.

**Stored procedure example**

function helloWorld() {  
 var context = getContext();  
 var response = context.getResponse();  
 response.setBody("Hello, World");  
}

**User-defined function**

UDFs are used to extend the Azure Cosmos DB SQL query language grammar and implement custom business logic, such as calculations on properties and documents. UDFs can be called only from inside queries and, **unlike stored procedures**, **they do not have access to the context object, so they cannot read or write documents.**

function producttax(price) {  
 if (price == undefined)   
 throw 'no input';

var amount = parseFloat(price);

if (amount < 1000)   
 return amount \* 0.1;  
 else if (amount < 10000)   
 return amount \* 0.2;  
 else  
 return amount \* 0.4;  
}

**Run complex operations on your data**

<https://docs.microsoft.com/en-us/learn/modules/access-data-with-cosmos-db-and-sql-api/6-javascript-programming>

**Store and access graph data in Azure Cosmos DB with the Graph API**

**What is a graph database?**

In order to understand graph databases, you first need to know what we mean by a graph. A graph is a structure that's composed of vertices and edges. Both vertices and edges can have an arbitrary number of properties.

|  |  |
| --- | --- |
| **Component** | **Description** |
| **Vertices or Nodes** | Vertices represent objects. For example: a person, a place, or a product. |
| **Edges or Relationships** | Edges denote relationships between vertices. For example: a person might know another person, or have visited a place. |
| **Properties** | Properties express information about the vertices and edges. For example:   * Vertices' properties might include the name and age of a person. * Edge properties might include a time stamp of a purchase or a hierarchical affiliation between coworkers. |



**Performance**

With traditional relational databases, the performance of relationship queries decreases as the number of relationships increase. With graph databases, the performance stays constant even as the data and complexity continues to grow.

**Flexibility**

In graph databases, it's easy to add to an existing structure without affecting functionality. This flexibility allows the graph database model to dictate change, rather than being forced to adapt to a tabular way of seeing your data in a standard relational database.

**Deficiencies**

Graph databases are not as efficient at processing high volumes of transactions, nor are they as effective at handling queries that traverse an entire database.

Graph databases do not create better relationships; instead, they provide rapid data retrieval for connected data. This increases the need for efficient data design, because any performance gains from graph searches can be reduced by failing to model the relationships between your nodes efficiently.

**Query graph data from an application using Gremlin API**

<https://docs.microsoft.com/en-us/learn/modules/store-access-data-cosmos-graph-api/6-exercise-query-graph-data-from-an-application-using-gremlin-api?pivots=csharp>

**Store and Access NoSQL Data with Azure Cosmos DB and the Table API**

Differences between Azure Storage tables and Azure Cosmos DB tables

* You are charged for the capacity of an Azure Cosmos DB table as soon as it is created, even if that capacity isn't used. This charging structure is because Azure Cosmos DB uses a reserved-capacity model to ensure that clients can read data within 10 ms. In Azure Storage tables, you are only charged for used capacity, but read access is only guaranteed within 10 seconds.
* Query results from Azure Cosmos DB are not sorted in order of partition key and row key as they are from Storage tables.
* Row keys in Azure Cosmos DB are limited to 255 bytes.
* Batch operations are limited to 2 MBs.
* Cross-Origin Resource Sharing (CORS) is supported by Azure Cosmos DB.
* Table names are case-sensitive in Azure Cosmos DB. They are not case-sensitive in Storage tables.

**How to choose a storage location**

**HOW TO CHOOSE A STORAGE LOCATION**

|  |  |  |
| --- | --- | --- |
| **Priority** | **Azure Storage tables** | **Azure Cosmos DB tables** |
| **Latency** | **Responses are fast, but there is no guaranteed response time.** | **< 10 ms for reads, < 15 ms for writes.** |
| **Throughput** | **Maximum 20,000 operations/sec** | **No upper limit on throughput. Over 10 million operations/sec/table.** |
| **Global distribution** | **Single region for writes. A secondary read-only region is possible with read-access geo-redundant replication.** | **Replication of data for read and write to more than 30 regions.** |
| **Indexes** | **A single primary key on the partition key and the row key. No other indexes.** | **Indexes are created automatically on all properties.** |
| **Data consistency** | **Strong in the primary region. If you are using read-access geo-redundant replication, it may take time for changes to reach the secondary region.** | **You can choose from five different consistency levels depending on your needs for availability, latency, throughput, and consistency.** |
| **Pricing** | **Optimized for storage.** | **Optimized for throughput.** |
| **SLAs** | **99.99% availability.** | **99.99% availability for single region and relaxed consistency databases. 99.999% availability for multi-region databases.** |

**How to migrate an app to Azure Cosmos DB**

* **Azure Cosmos DB Data Migration Tool -** This open source tool is built specifically to import data into Azure Cosmos DB from many different sources, including tables in Azure Storage, SQL databases, MongoDB, text files in JSON and CSV formats, HBase, and other databases. The tool has both a command-line version and a GUI version. You supply the connection strings for the data source and the Azure Cosmos DB target, and you can filter the data before migration.
* **AzCopy -** This command-line only tool is designed to enable developers to copy data to and from Azure Storage accounts. The process has two stages:
  + Export the data from the source to a local file.
  + Import from that local file to a database in Cosmos, specifying the destination database by using its URL and access key.

**Optimize the performance of Azure Cosmos DB by using partitioning and indexing strategies**

**Measure the rate of data processing in Request Units**

In Azure Cosmos DB, throughput is the rate at which data is processed. It's measured in Request Units (RUs). An RU is the amount of CPU, disk I/O, and memory required to read 1 **KB of data in 1 second. Other operations like reading more data, writing data, or querying data take more RUs.** The greater the demand on the underlying physical resources, the higher the number of RUs used.

The number of RUs that a specific operation uses depends on the following factors:

* How the data is distributed across the physical resources in Azure
* Volume of data that's read and written
* Whether the operation is a read or a write
* Number of fields in your database that are indexed, and the indexing mode
* Complexity of the operation for queries
* Data consistency for geographically replicated collections

**To operate an Azure Cosmos DB database efficiently, you need to:**

* Configure enough throughput to meet performance demands.
* Minimize unused excess throughput to keep your costs down.
* Regularly review metrics to maximize the efficiency of your provisioned throughput.

**Billing based on capacity you've configured**

RUs are billed on an hourly basis, whether you consume them. The amount that you're charged for your Azure Cosmos DB collection is fixed. It's based only on the configured capacity in RUs.

**Azure Cosmos DB concepts**

* **Resources** - An Azure Cosmos DB account is a container for one or more databases. An Azure Cosmos DB database is a container for one or more collections. A collection contains documents. A document is an unstructured set of key/value pairs, read and written in JSON format.
* **Partitioning** - Partitioning is the distribution and grouping of your data across the underlying resources. Documents are grouped in a partition based on the value of the partition key. You specify the partition key when you create the collection. An effective partitioning strategy distributes data and access evenly across partitions, and across time. Querying documents from within the same partition is less expensive than querying across partitions. You choose how to partition your data at design time. The partitioning configuration can't be changed after a collection is provisioned.
* **Index** - Is a catalog of document properties and their values. It includes links to documents that contain properties equal to each property value. Indexing makes searching a collection more efficient. But the search efficiency is balanced with the resources required to insert or change a document. When a document is inserted or changed, Azure Cosmos DB has to update the index. The optimal indexing strategy for your collection depends on your workload.

The partition key is a document property. Documents with the same partition key value are always located on the same logical partition. A partition supports a fixed maximum amount of storage and Request Units (RUs). When the capacity of a logical partition gets close to the maximum storage, Azure Cosmos DB allocates another physical partition. Azure Cosmos DB seamlessly splits the logical partitions, the groups of documents with the same partition key value, among the physical partitions.

**Identify a partition strategy for your Azure Cosmos DB data**

**Avoid hot partitions**

The Azure Cosmos DB throughput you've configured is divided evenly among partitions. A partition key design that doesn't evenly distribute throughput requests can create hot partitions. A hot partition is accessed more than the other partitions. The result is an inefficient use of the total configured throughput. If the demand on the hot partition is high enough, the partition becomes overloaded, and traffic to the database is rate-limited.

**Estimate the scale of your data needs**

* What's the approximate size of your documents, or range of sizes?
* What's the required throughput in number of reads per second and writes per second?
* What's the volume of documents being queried?

**Understand the workload**

* Do you have a read-heavy or write-heavy workload, or both?
* If it's read-heavy, what are the top five queries?
* If it's write-heavy, do you need transactions?

**Propose some partition key options**

* Does the key choice have a large number of possible values or large cardinality?
* Do the values have a consistent spread across the data?
* Are some values accessed more than others?
* For read-heavy workloads, can the query be within a single partition?
* For write-heavy transactional workloads, can the transaction be within a single partition?

**Identify indexing strategies for your Azure Cosmos DB data**

An index is extra information that sits alongside a collection to make querying more efficient. Queries use the index to locate documents.

The index is updated every time a document is updated or added to a collection. That update adds to the RUs used for each write operation.

The right indexing strategy depends on the patterns of access to your collections. Read-intensive workloads call for a different indexing strategy from write-intensive ones. Unlike the partitioning configuration, you can change the Azure Cosmos DB indexing configuration after a collection is provisioned. When your database is up and running, measure the performance of your index configuration and tune it.

**By default, all document properties in Azure Cosmos DB are indexed.**

**Indexing modes**

* **Consistent**: The index is updated synchronously every time a new document is written to the collection. New queries on the collection use the updated index immediately. Query results are consistent with the updated documents in the collection.
* **Lazy**: The index is updated at a lower priority. The reads and writes from the collection take a higher priority. In lazy mode, writes are cheaper because the index isn't updated immediately. When the index is fully updated depends on the demands on the collection. Query results don't include the updated documents until the index is consistent with the collection.
* **None**: No index is created. Queries are expensive on collections that aren't indexed. If you're using your Azure Cosmos DB collection to read records directly rather than querying the collection, it's possible to avoid the overhead of indexing.

One scenario for the Orders collection workload is write-intensive:

* Orders are placed or updated at any time of the day.
* Reports are generated less frequently for the quantity and type of items sold.

In this case, you might choose to keep the indexing on all properties of the documents, but set the indexing mode to lazy.

**Compare RUs across indexing strategies**

|  |  |  |
| --- | --- | --- |
| Operation | Indexing strategy | Approximate consumption (RUs) |
| Read document directly | N/A | 1 |
| Query document | All properties indexed | 3 |
| Query document by indexed property | Partial | 3+ |
| Query document by non-indexed property | Partial | 10+ |
| Query document | No properties indexed | 20+ |
| Insert document | All properties indexed | 13 |
| Insert document | All properties indexed lazily | 7 |
| Insert document | Some properties indexed | 6 |
| Insert document | No properties indexed | 5 |

You see that as indexing complexity goes up, the write consumption goes up, and the read consumption goes down. The indexing strategy that you choose depends on your data and the workloads that it supports.

**Distribute your data globally with Azure Cosmos DB**

**Global distribution basics**

Global distribution enables you to replicate data from one region into multiple Azure regions. You can add or remove regions in which your database is replicated at any time, and Azure Cosmos DB ensures that when you add an additional region, your data is available for operations within 30 minutes, assuming your data is 100 TBs or less.

**There are two common scenarios for replicating data in two or more regions:**

* Delivering low-latency data access to end users no matter where they are located around the globe
* Adding regional resiliency for business continuity and disaster recovery (BCDR)

When a database is replicated, the throughput and storage are replicated equally as well. So if your original database had 10GB of storage, and throughput of 1,000 RU/s, and if you replicated that to three additional regions, each region would have 10GB of data and 1,000 RU/s of throughput

**What is multi-master support?**

Multi-master support is an option that can be enabled on new Azure Cosmos DB accounts. Once the account is replicated in multiple regions, each region is a master region that equally participates in a write-anywhere model, also known as an active-active pattern.

**What are the benefits of multi-master support?**

* Single-digit write latency – Multi-master accounts have an improved write latency of <10 ms for 99% of writes, up from <15 ms for non-multi-master accounts.
* 99.999% read-write availability - The write availability multi-master accounts increases to 99.999%, up from the 99.99% for non-multi-master accounts.
* Unlimited write scalability and throughput – With multi-master accounts, you can write to every region, providing unlimited write scalability and throughput to support billions of devices.
* Built-in conflict resolution – Multi-master accounts have three methods for resolving conflicts to ensure global data integrity and consistency.

**Conflict resolution**

With the addition of multi-master support comes the possibility of encountering conflicts for writes to different regions. Conflicts are rare in Azure Cosmos DB and can only occur when an item is simultaneously changed in multiple regions, before propagation between the regions has happened.

There are three conflict resolution modes offered by Azure Cosmos DB.

* **Last-Writer-Wins (LWW),** in which conflicts are resolved based on the value of a user-defined integer property in the document. By default \_ts is used to determine the last written document. Last-Writer-Wins is the default conflict handling mechanism.
* **Custom - User-defined function,** in which you can fully control conflict resolution by registering a User-defined function to the collection. A User-defined function is a special type of stored procedure with a specific signature. If the User-defined function fails or does not exist, Azure Cosmos DB will add all conflicts into the read-only conflicts feed they can be processed asynchronously.
* **Custom - Async,** in which Azure Cosmos DB excludes all conflicts from being committed and registers them in the read-only conflicts feed for deferred resolution by the user’s application. The application can perform conflict resolution asynchronously and use any logic or refer to any external source, application, or service to resolve the conflict.

[**https://docs.microsoft.com/en-us/learn/modules/distribute-data-globally-with-cosmos-db/5-data-consistency-levels**](https://docs.microsoft.com/en-us/learn/modules/distribute-data-globally-with-cosmos-db/5-data-consistency-levels)

**Deploy Azure infrastructure by using ARM templates**

**What is infrastructure as code?**

Infrastructure as code enables you to describe, through code, the infrastructure that you need for your application.

With infrastructure as code, you can maintain both your application code and everything you need to deploy your application in a central code repository. The advantages to infrastructure as code are:

* Consistent configurations
* Improved scalability
* Faster deployments
* Better traceability

**What is an ARM template?**

(ARM templates) allow you to specify your project's infrastructure in a declarative and reusable way. The templates can be versioned and saved in the same source control as your development project.

ARM templates are JavaScript Object Notation (JSON) files that define the infrastructure and configuration for your deployment. The template uses a declarative syntax.

ARM templates allow you to declare what you intend to deploy **without having to write the sequence of programming commands to create it.** In an ARM template, you specify the resources and the properties for those resources. **Then Azure Resource Manager uses that information to deploy the resources in an organized and consistent manner.**

ARM templates are **idempotent**, which means you can deploy the same template many times and get the same resource types in the same state.

Resource Manager orchestrates the deployment of the resources so they're created in the correct order. **When possible, resources will also be created in parallel, so ARM template deployments finish faster than scripted deployments.**

**ARM template file structure**

* **schema**: A required section that defines the location of the JSON schema file that describes the structure of JSON data. The version number you use depends on the scope of the deployment and your JSON editor.
* **contentVersion**: A required section that defines the version of your template (such as 1.0.0.0). You can use this value to document significant changes in your template to ensure you're deploying the right template.
* **apiProfile**: An optional section that defines a collection of API versions for resource types. You can use this value to avoid having to specify API versions for each resource in the template.
* **parameters**: An optional section where you define values that are provided during deployment. These values can be provided by a parameter file, by command-line parameters, or in the Azure portal.
* **variables**: An optional section where you define values that are used to simplify template language expressions.
* **functions**: An optional section where you can define user-defined functions that are available within the template. User-defined functions can simplify your template when complicated expressions are used repeatedly in your template.
* **resources**: A required section that defines the actual items you want to deploy or update in a resource group or a subscription.
* **output**: An optional section where you specify the values that will be returned at the end of the deployment.

**Deploy an ARM template to Azure**

You can deploy an ARM template to Azure in one of the following ways:

* Deploy a local template.
* Deploy a linked template.
* Deploy in a continuous deployment pipeline.

**Add resources to the template**

To add a resource to your template, you'll need to know the resource provider and its types of resources. The syntax for this combination is in the form of {resource-provider}/{resource-type}. (Microsoft.Storage/storageAccounts.)

**Create and deploy an Azure Resource Manager template**

[**https://docs.microsoft.com/en-us/learn/modules/create-azure-resource-manager-template-vs-code/3-exercise-create-and-deploy-template?pivots=powershell**](https://docs.microsoft.com/en-us/learn/modules/create-azure-resource-manager-template-vs-code/3-exercise-create-and-deploy-template?pivots=powershell)

**Add flexibility to your Azure Resource Manager template by using parameters and outputs**

**ARM template parameters**

In the parameters section of the template, you specify which values you can input when you deploy the resources. You're limited to 256 parameters in a template. Parameter definitions can use most template functions.

**The allowed types of parameters are:**

* string
* secureString
* integers
* boolean
* object
* secureObject
* array

For security reasons, never hard code or provide default values for usernames and/or passwords in templates. Always use parameters for usernames and passwords (or secrets). Use secureString for all passwords and secrets. If you pass sensitive data in a JSON object, use the secureObject type. Template parameters with secureString or secureObject types can't be read or harvested after the deployment of the resource.

**ARM template outputs**

* **output-name:** Must be a valid JavaScript identifier.
* **condition:** (Optional) A Boolean value that indicates whether this output value is returned. When true, the value is included in the output for the deployment. When false, the output value is skipped for this deployment. When not specified, the default value is true.
* **type:** The type of the output value.
* **value:** (Optional) A template language expression that's evaluated and returned as an output value.
* **copy:** (Optional) Copy is used to return more than one value for an output.

**Add parameters and outputs to your Azure Resource Manager template**

[**https://docs.microsoft.com/en-us/learn/modules/create-azure-resource-manager-template-vs-code/5-exercise-parameters-output?pivots=powershell**](https://docs.microsoft.com/en-us/learn/modules/create-azure-resource-manager-template-vs-code/5-exercise-parameters-output?pivots=powershell)

**Deploy to multiple Azure environments by using ARM template features**

**What are ARM template functions?**

ARM template functions add flexibility to your ARM template by dynamically getting values during deployment.

* Array functions for working with arrays. For example, first and last.
* Comparison functions for making comparisons in your templates. For example, equals and greater.
* Date functions for working with dates. For example, utcNow and dateTimeAdd.
* Deployment value functions for getting values from sections of the template and values related to the deployment. For example, environment and parameters.
* Logical functions for working with logical conditions. For example, if and not.
* Numeric functions for working with integers. For example, max and mod.
* Object functions for working with objects. For example, contains and length.
* Resource functions for getting resource values. For example, resourceGroup and subscription.
* String functions for working with strings. For example, length and startsWith.

**Recommendations for ARM template variables**

Template variables are specified in camel case. They're best used for values that you need to specify more than once, especially if that value is a complex expression.

Don't use the reference function in the variables section of a template. The reference function is resolved at runtime, and variables are resolved when the template is parsed. Also, don't use variables for apiVersion on a resource. The API version determines the schema of the resource, and often you can't change the version without changing the properties for the resource.

**What is an ARM template parameter file**

An ARM template parameter file holds values that will be passed in to the ARM template when the template is executed and that file is specified. If you use a parameter file for each environment that an ARM template will be deployed to, you ensure that the correct parameters are set for that specific environment. You also ensure that you can track the history and maintenance of those parameter values in source control.

**Summary**

* Created an expression for a unique value by using Azure Resource Manager template functions
* Reused your expression by storing it in an Azure Resource Manager template variable
* Organized and tracked your deployed Azure resources by using resource tags
* Managed multiple deployment parameters by using Azure Resource Manager template parameter files

**Preview changes and validate Azure resources by using what-if and the ARM template test toolkit**

**Predict what a deployment will do by using what-if**

Using the what-if operation allows you to estimate what would happen if you were to deploy, by comparing the current state model to the desired state model. The what-if operation confirms if the changes made by your template match your expectations without applying those changes to real resources or to the state of those resources.

**Change types**

1. **Create**. The resource doesn't currently exist but is defined in the template. The resource will be created.
2. **Delete**. This change type applies only when you're using complete mode for deployment. The resource exists but isn't defined in the template. With complete mode, the resource will be deleted. This change type includes only resources that support deletion through complete mode.
3. **Ignore**. The resource exists but isn't defined in the template. The resource won't be deployed or modified.
4. **NoChange**. The resource exists and is defined in the template. The resource will be redeployed, but the properties of the resource won't change. This change type is returned when ResultFormat is set to FullResourcePayloads, which is the default value.
5. **Modify**. The resource exists and is defined in the template. The resource will be redeployed, and the properties of the resource will change. This change type is returned when ResultFormat is set to FullResourcePayloads, which is the default value.
6. **Deploy**. The resource exists and is defined in the template. The resource will be redeployed. The properties of the resource might or might not change. The operation returns this change type when it doesn't have enough information to determine if any properties will change. You see this condition only when ResultFormat is set to ResourceIdOnly.

**Result format**

* **FullResourcePayloads**. By including this parameter, you get a verbose output that consists of a list of resources that will change. The output also shows details about all the properties that will change in accordance with the template.
* **ResourceIdOnly**. This mode returns a list of resources that will change, but not all the details.

**Removal or deletion of resources and deployment modes**

There are times when you'll want to confirm the removal or deletion of resources as you deploy the template. To that end, the what-if operation supports using deployment mode. There are two deployment modes:

* **Incremental** **mode**. The default deployment mode is incremental. In this mode, Resource Manager leaves unchanged resources that exist in the resource group but aren't specified in the template. Resources in the template are added to the resource group.
* **Complete mode.** When you use this mode, resources not specified in the template are deleted. If you know for sure that what's in the template file constitutes the full state of your deployment, then go ahead and use this mode. If you use tools like the Azure CLI or PowerShell to update your state gradually, then incremental mode is the way to go.

**Confirmation**

To preview changes before deploying a template, use the confirm switch parameter with the deployment command. If the changes are as you expected, acknowledge that you want the deployment to finish.

**Deploy the modified template in the same environment**

* Purple and ~ for any modifications
* Green and + for new resources to be created
* Orange and - for deletions

**The test toolkit**

**There is such a tool: the ARM template test toolkit. It addresses the problems mentioned earlier by running a series of tests. The tests can be grouped into the following categories:**

* **Validating the user's intent.** This category looks at whether the declared variables and parameters are all used and warns if they're not.
* **Following security practices**. Another important aspect is ensuring that nothing is returned from the template that might be sensitive, like API secrets.
* **Using appropriate language constructs.** You should use language constructs or helper functions so that you're not relying on hard-coded values.

**Apply good practices to your template by using the ARM template test toolkit**

<https://docs.microsoft.com/en-us/learn/modules/arm-template-test/5-exercise-test-toolkit?pivots=windows>

**Automate the deployment of ARM templates by using GitHub Actions**

**Deploy your ARM templates**

**You'll sometimes need to deploy from different locations. For example:**

* From a local file
* From an external or remote location by using a Uniform Resource Identifier (URI)

**Linked template**

Linked template refers to the act of connecting separate template files, referenced by a link from a main template. Linked templates allow you to create reusable, composable, and modular deployments that comprise many individual ARM templates.

**Nested template**

Nested template refers to the act of embedding template syntax within a main template. Nested templates allow for advanced deployment scenarios like deploying to multiple Azure Resource Manager scopes or multiple resource groups from a single template file. Unlike linked templates, where each template is stored in its own template files, nested templates allow you to store many individual templates in one file. There are several reasons why you might want to do this, such as when you're deploying resources to multiple resource groups or deployment scopes.

When you're using a nested template, you can specify whether template expressions are evaluated within the scope of the parent template or the nested template. The scope determines how parameters, variables, and functions like resourceGroup and subscription are resolved.

**Deploy ARM templates as part of your CI/CD efforts with GitHub Actions**

<https://docs.microsoft.com/en-us/learn/modules/deploy-templates-command-line-github-actions/6-github-actions>

**Extend ARM templates by using deployment scripts**

**What are deployment scripts?**

Deployment scripts in Azure Resource Manager (ARM) templates enable custom automation for your environment management.

deploymentScripts resources are either PowerShell or Bash scripts that run in a Docker container as part of your template deployment. The default container images have either the Azure CLI or Azure PowerShell available. These scripts run during the processing of the ARM template, so you can add custom behavior to the deployment process.

The deploymentScripts resource produces output that other resources in the deployment can use. You can then look up information from an external system or provide data based on the current state of your environment to affect the rest of the deployment.

**Manage complex cloud deployments by using advanced ARM template features**

**How can I define dependencies among Azure resources?**

There's such a construct in Resource Manager templates, called dependsOn. Using this construct makes resources wait until the pointed out resource has finished deploying.

It's a key-value pair that enables you to define the deployment order between resources. Sometimes you need to ensure something exists before something else.

**What are child resources?**

A child resource is a resource that only exists within the context of another resource.

**Manage secrets in your ARM template by reading from Azure Key Vault**

**Configure permissions**

1. **Enable Key Vault for deployment.** Key Vault needs to be allowed to be read from during deployment. You can pass this setting at creation or amend it later.
2. **The user needs deployment permission on the resource group.** This is no different than other deployments. However, if you haven't created the resource group, you can assign a role to ensure you can deploy to it.
3. **The user needs proper access to the key vault.** You must ensure that you have the proper access to the key vault when you read from it during deployment. When you create a key vault, it creates an access policy giving the user rights to manage all the secrets, keys, and certificates.

**Configure Key Vault for deployment**

To enable Key Vault for deployment, there's a setting you can pass, either during creation of the Key Vault instance or something you can amend at a later point. In PowerShell, it's a switch called -**EnabledForTemplateDeployment**. In the Azure CLI, it's an argument **called --enabled-for-template-deployment** that needs the value true to be passed as well.

**User needs deployment permission on Key Vault and resource group**

* **Owner**: If you created the key vault, you automatically have this role.
* **Contributor**: This role grants you access to manage all the secrets. If you didn't create the key vault, the easiest approach is to give yourself the Contributor **role.**

*Another choice is to create and assign a custom role, and ensure that role contains the Microsoft.KeyVault/vaults/deploy/action permission.*

**Add conditional logic to your ARM templates**

**Pre-existing resource.** When you specify a resource in a template and deploy it, one of two things happens. Either the resource is deployed, or it's not deployed if it already exists. Checking whether a resource exists is something Azure Resource Manager does for you; it's implicit. The question is if you can use this mechanism to your advantage when you reason about how you can check the preexistence of something.

**Branching logic.** Depending on what parameters you pass to a template, at deployment time, you might want to deploy a different set of resources. What you are expressing is something known as branching logic. If the parameter has a certain type of value, then select the first branch. Otherwise, select the second or third branch to deploy. The branching logic continues in this manner.

**Conditional deployment**

The condition construct enables you to express whether you want something deployed or not. It's a property, with a value of either true or false, that you attach to a resource element. You'd typically find a condition construct looking like the following JSON in your template:

**Evaluation**

There are two ways in which the condition construct can be evaluated. Knowing about these two ways might affect how you choose to express your conditional logic. The two different ways are:

1. The value is either true/false. For example, consider the following construct:
   1. "condition": "[parameters('deployAccount')]"
   2. The deployAccount value is a parameter who's value can be passed at deploy time, or it can fall back to the default value. Regardless of the approach used, the value is strictly false or true. Attempting to assign another value that's not a Boolean results in an error.
2. There's an expression evaluating to true/false. Here, instead of assigning a strict true/false value to the condition construct, you use the built-in template function equals(arg1, arg2). arg1 needs to be equal to arg2 for the function to evaluate to true. Your condition construct can now be expressed like this:
   1. "condition": "[equals(parameters('newOrExisting'),'new')]"
   2. Using the equals() function, the value you pass into a parameter no longer needs to be true or false. It needs to match the second argument in the equals() function. In the preceding JSON example, the value of the newOrExisting parameter needs to match the string new for the function to evaluate to true.

**Create multiple resource instances by using the copy element**

**Create multiple instances**

The copy element is piece of JSON that you can use on many types of constructs, like resources, properties, variables, and output. The syntax for the copy element consists of the key copy, and an array as the value. For example: "copy": []

**Troubleshoot an App Service app with Azure Monitor**

**Azure Monitor**

Azure Monitor maximizes the availability and performance of your applications and services by delivering a comprehensive solution for collecting, analyzing, and acting on telemetry from your cloud and on-premises environments.

**Log Analytics Workspace**

With Azure Monitor Log Analytics you can easily store, retain and query data collected from your monitored resources in Azure and other environments for valuable insights.

A Log Analytics workspace is the logical storage unit where your log data is collected and stored.  
A Log Analytics workspace is the basic management unit of Log Analytics.

**Create a Log Analytics Workspace**

Azure Monitor stores log data in a Log Analytics workspace. A workspace is a container that includes data and configuration information.

**Create a diagnostic setting**

Diagnostic settings can be used to collect metrics for certain Azure services into Azure Monitor Logs for analysis with other monitoring data using log queries. For this tutorial, you enable the web server and standard output/error logs. See supported log types for a complete [list of log types](https://docs.microsoft.com/en-us/azure/app-service/troubleshoot-diagnostic-logs#supported-log-types) and descriptions.

You run the following commands to create diagnostic settings for AppServiceConsoleLogs (standard output/error) and AppServiceHTTPLogs (web server logs). Replace <app-name> and <workspace-name> with your values.

az monitor diagnostic-settings create --resource $resourceID \  
 --workspace $workspaceID \  
 -n myMonitorLogs \  
 --logs '[{"category": "AppServiceConsoleLogs", "enabled": true},  
 {"category": "AppServiceHTTPLogs", "enabled": true}]'

**Log queries**

Log queries help you to fully leverage the value of the data collected in Azure Monitor Logs. You use log queries to identify the logs in both AppServiceHTTPLogs (**Logs generated through your application and pushed to Azure Monitoring.)** and AppServiceConsoleLogs (**Console logs generated from application or container.).**

**Join AppServiceHTTPLogs and AppServiceConsoleLogs**

**A query has been prepared for you that does the following:**

* Filters HTTPLogs for 500 errors
* Queries console logs
* Joins the tables on TimeGenerated

let myHttp = AppServiceHTTPLogs | where ScStatus == 500 | project TimeGen=substring(TimeGenerated, 0, 19), CsUriStem, ScStatus;

let myConsole = AppServiceConsoleLogs | project TimeGen=substring(TimeGenerated, 0, 19), ResultDescription;

myHttp | join myConsole on TimeGen | project TimeGen, CsUriStem, ScStatus, ResultDescription;

**Analyze your Azure infrastructure by using Azure Monitor logs**

**Data collection in Azure Monitor**

Azure Monitor collects two fundamental types of data: metrics and logs. Metrics tell you how the resource is performing, and the other resources that it's consuming. Logs contain records that show when resources are created or modified.

Because Azure Monitor is an automatic system, it begins to collect data from these sources as soon as you create Azure resources such as virtual machines and web apps. You can extend the data that Azure Monitor collects by:

**Enabling diagnostics:** For some resources, such as Azure SQL Database, you receive full information about a resource only after you have enabled diagnostic logging for it. You can use the Azure portal, the Azure CLI, or PowerShell to enable diagnostics.

**Adding an agent:** For virtual machines, you can install the Log Analytics agent and configure it to send data to a Log Analytics workspace. This agent increases the amount of information that's sent to Azure Monitor.

Your developers might also want to send data to Azure Monitor from custom code, such as a web app, an Azure function, or a mobile app. They send data by calling the Data Collector API. You communicate with this REST interface through HTTP. This interface is compatible with a variety of development frameworks, such as .NET Framework, Node.js, and Python. Developers can choose their favorite language and framework to log data in Azure Monitor.

**Logs**

Logs contain time-stamped information about changes made to resources.

You log data from Azure Monitor in a Log Analytics workspace. Azure provides an analysis engine and a rich query language. The logs show the context of any problems and are useful for identifying root causes.

**Metrics**

Metrics are numerical values that describe some aspect of a system at a point in time. Azure Monitor can capture metrics in near real time. The metrics are collected at regular intervals and are useful for alerting because of their frequent sampling. You can use a variety of algorithms to compare a metric to other metrics and observe trends over time.

**Analyzing logs by using Kusto**

To retrieve, consolidate, and analyze data, you specify a query to run in Azure Monitor logs. You write a log query with the Kusto query language, which is also used by Azure Data Explorer.

**What is Microsoft Graph?**

**Microsoft Graph**

Provides access to data stored across Microsoft 365 services. Custom applications can use the Microsoft Graph API to connect to data and use it in custom applications to enhance organizational productivity.

**Data and intelligence like the following types can be accessed through the Microsoft Graph REST APIs and client libraries**:

* Users and groups
* Teams data
* Tasks
* Files
* Mail
* Meetings and calendars
* Organizational charts

In addition to making direct calls to Microsoft Graph REST APIs, you can use the Microsoft Graph SDK (software development kit) and client libraries to simplify the process of calling an API. The next modules in this learning path give examples of using the SDK.

**Identity and access management**

The Microsoft Graph API for Azure Active Directory (Azure AD) helps organizations build a secure identity and access foundation. Developers can use Microsoft Graph to connect to Azure AD identity management services and automate administrative workflows. It can be a time saver for admins if processes like profile maintenance, employment onboarding/termination, or tracking assignments are automated through Microsoft Graph.

**Productivity and collaboration**

Developers can enhance the app experience by adding a chatbot that can schedule meetings between colleagues and customers, check calendar availability, and remind salespeople about the to-do list for the day.

You can build a chatbot that consumes the Microsoft Graph Outlook calendar API and to-do API as a productivity solution.

The same chatbot idea can be used for collaboration purposes. A sales team can store its files on a SharePoint site and add its tasks to Microsoft Planner in the group. If the team needs to access any file or task, a chatbot can get the required data by using the Microsoft Graph API for SharePoint and Planner.

**People and workspace intelligence**

Microsoft Graph services for people and workplace intelligence can help you access many insights about users and groups in Microsoft 365. For instance, a salesperson participates in meetings, reads emails, and collaborates with colleagues and customers through different channels in an ordinary workday. Hundreds of documents can be shared during collaboration, and a salesperson needs to quickly locate the files to make effective decisions.