**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

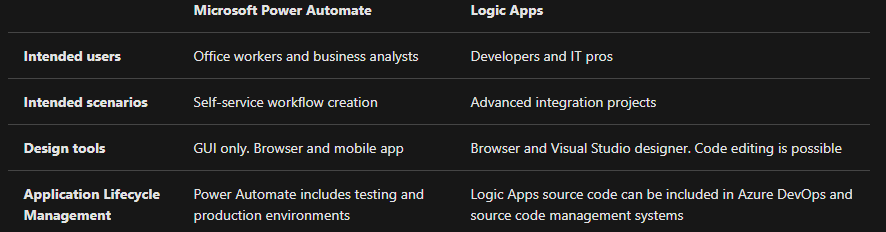
One reason why Logic Apps is so good at integration is that over 200 connectors are included. You can create your own connector if your system exposes a REST API.

* **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.

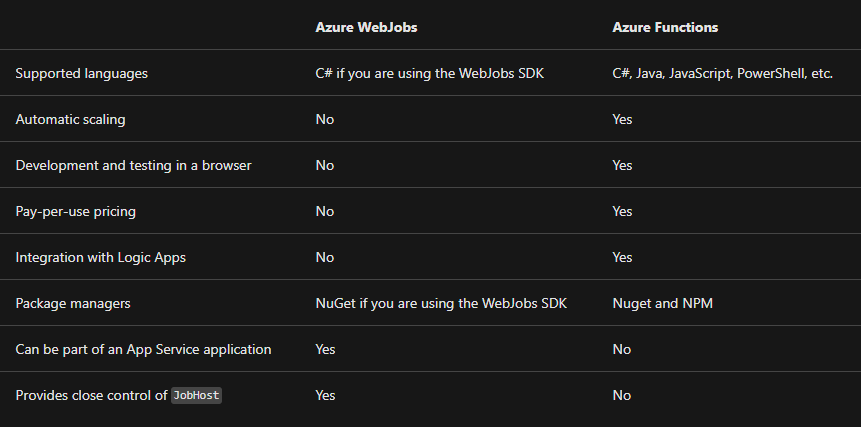
* **Azure Functions** is a simple way for you to run small pieces of code in the cloud, without having to worry about the infrastructure required to host that code.

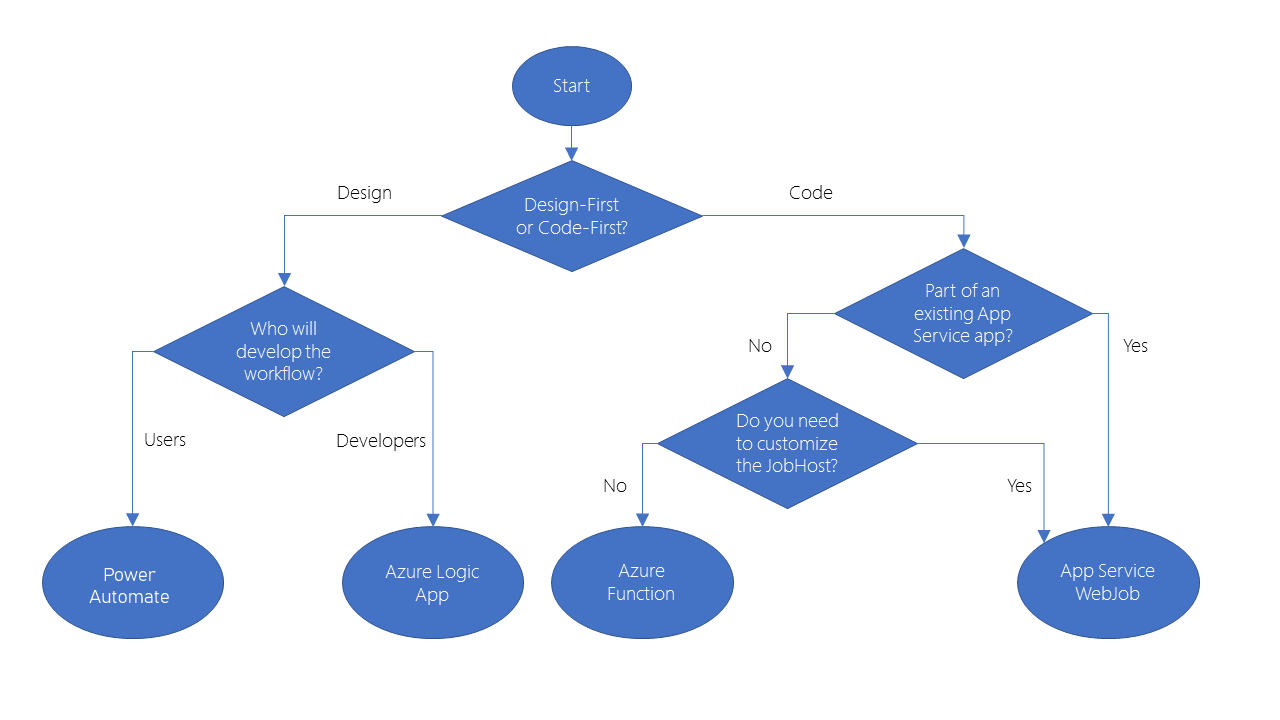
*In addition, with the consumption plan option, you only pay for the time when the code runs.* Azure automatically scales your function in response to the demand from users.

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}

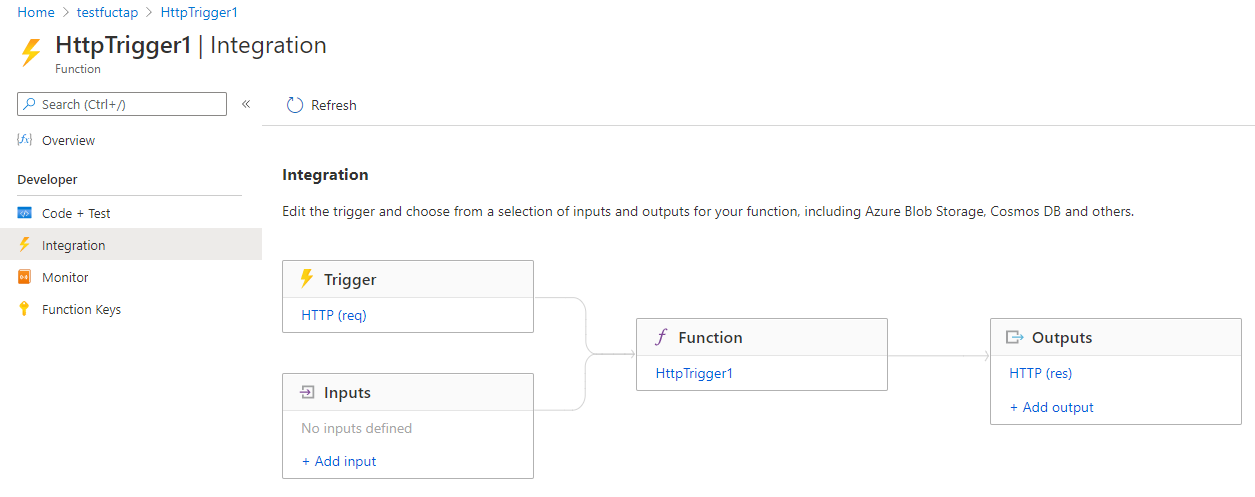
Let's break down this concept into two pieces: samples-workitems and {name}. The first part, samples-workitems, represents the blob container that the trigger monitors. The second part, {name} means that every type of file will cause the trigger to invoke the function. The function is invoked because 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**

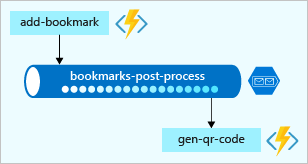
In the Document ID field, enter {id}.

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.

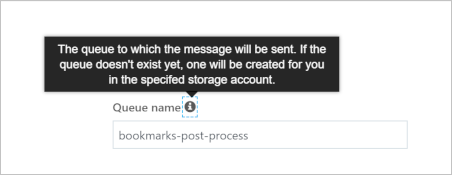
An incoming HTTP request triggers the function, and an id query parameter is passed to the Azure Cosmos DB input binding. If the database finds a document that matches this ID, the bookmark parameter will be set to the located document. In that case, you construct a response that contains the URL value 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**

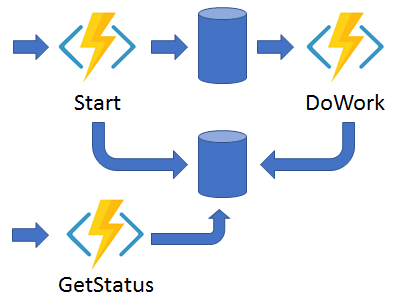
* **Client** functions are the entry point for creating an instance of a Durable Functions orchestration. They can run in response to an event from many sources, such as a new HTTP request arriving, a message being posted to a message queue, an event arriving in an event stream. You can write them in 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.



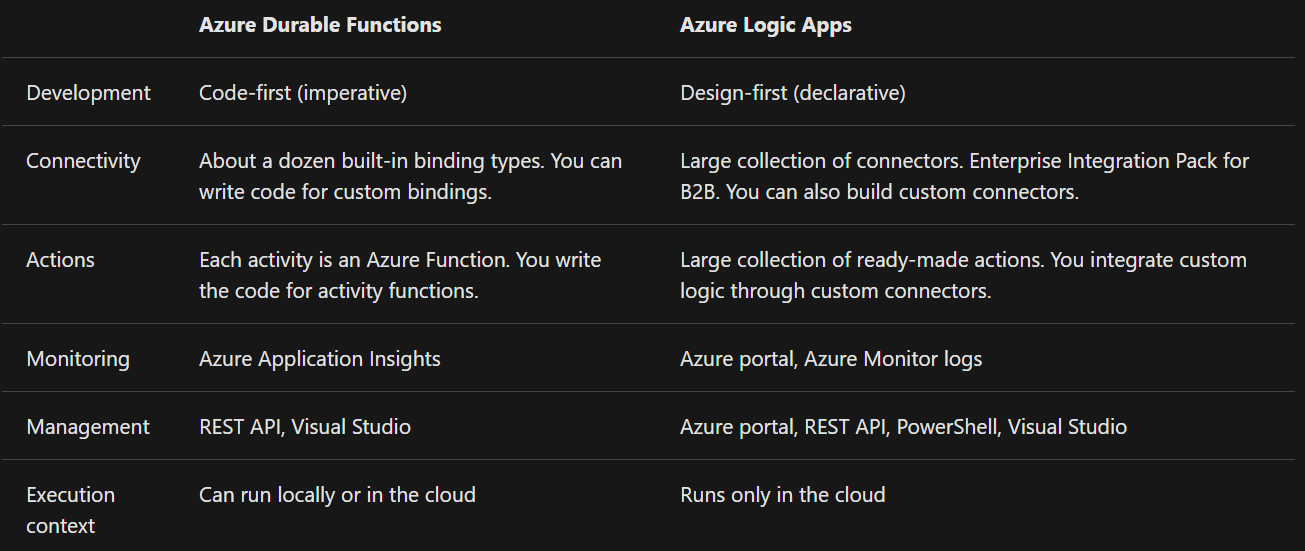
**Monitor -** This pattern implements a recurring process in a workflow, possibly looking for a change in state. For example, you could use this pattern to poll until specific conditions are met.



**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.



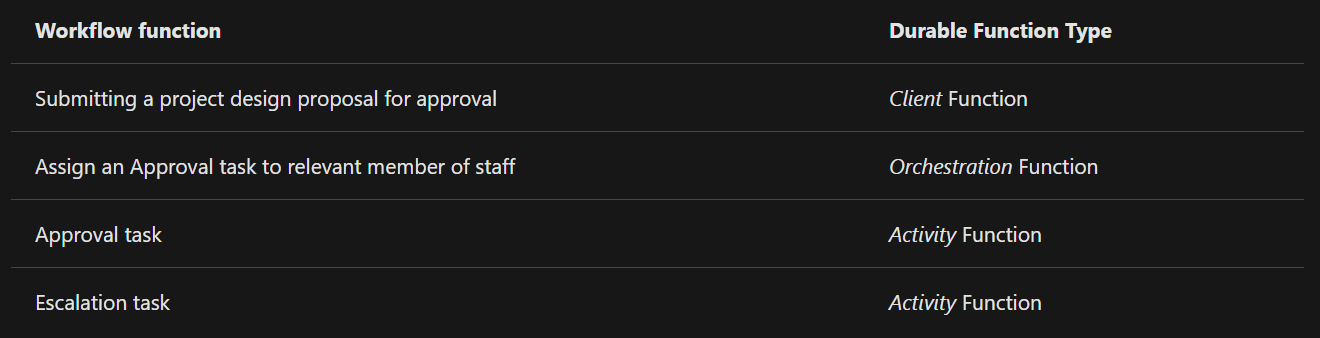
**Comparison with Logic Apps**



**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.

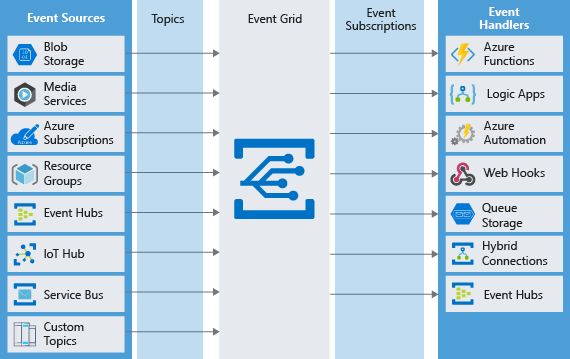
**Use Storage queues** when you want a simple and easy-to-code queue system. For more advanced needs, use **Service Bus queues**. If you have multiple destinations for a single message, but need queue-like behavior, use **Service Bus topics.**

**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?**

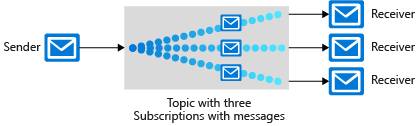
A queue is a simple temporary storage location for messages. A sending component adds a message to the queue. A destination component picks up the message at the front of the queue. Under ordinary circumstances, each message is received by only one receiver.



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.

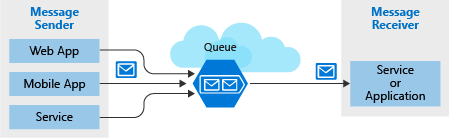
*For our StoreId filter, we could use a SQL filter. SQL filters are the most flexible, but they're also the most computationally expensive and could slow down our Service Bus throughput. In this case, we choose a correlation filter instead.*

**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.*

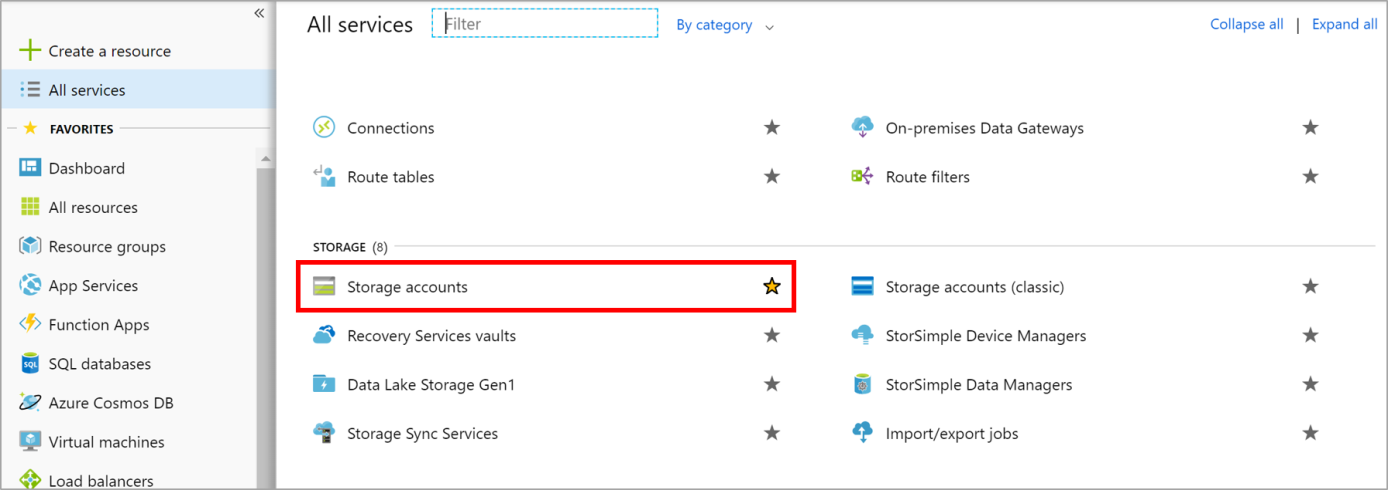
**Communicate between applications with Azure Queue storage**

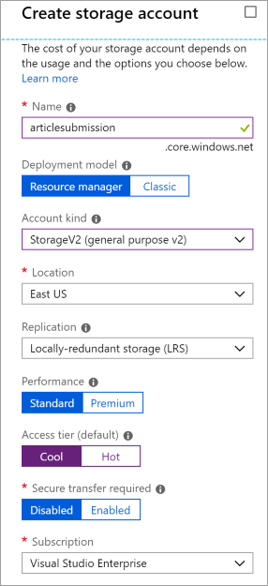
**What is Azure Queue storage?**

**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**

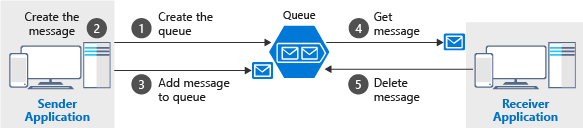
* **Azure Active Directory** You can use role-based authentication and identify specific clients based on AAD credentials.
* **Shared Key** Sometimes referred to as an account key, this is an encrypted key signature associated with the storage account. Every storage account has two of these keys that 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****.*

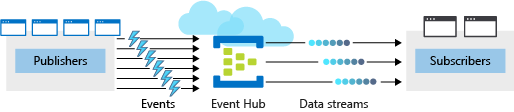
There are several other commands available that you can try with the tools - check out both **az storage queue --help** and **az storage message --help** to explore them.

**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**

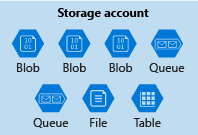
**Create an Azure Storage account**

**What is Azure Storage?**

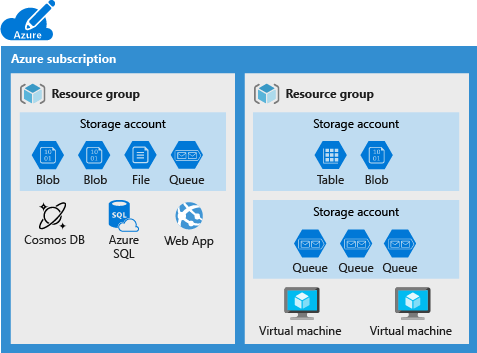
Azure selected four of these data services and placed them together under the name Azure Storage. The four services are Azure Blobs, Azure Files, Azure Queues, and Azure Tables. The following illustration shows the elements of Azure Storage**.**

**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)

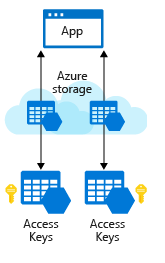
This would return an XML block with data specific to the account.

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**

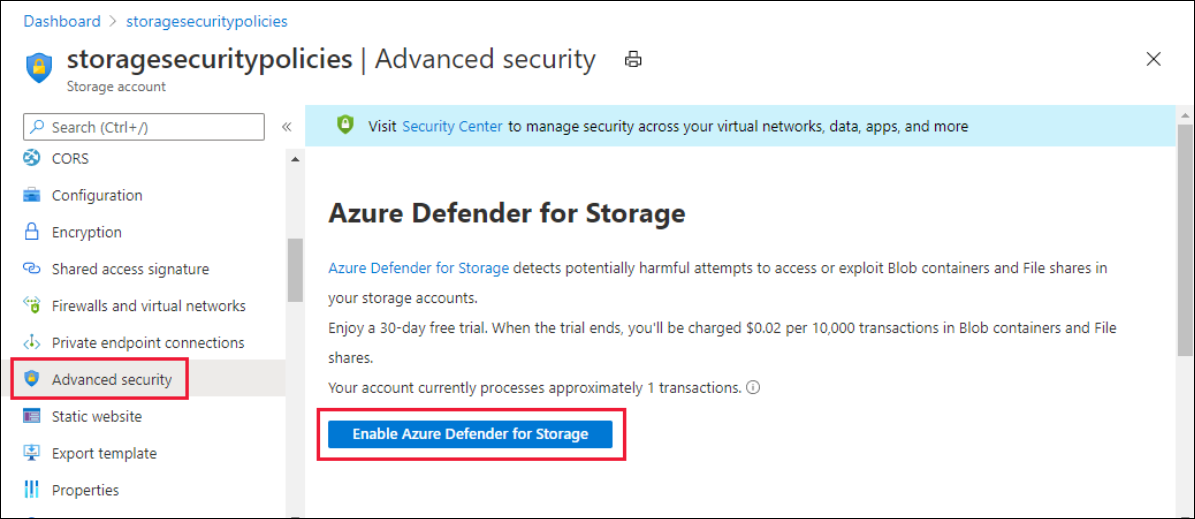
As a best practice, you shouldn't share storage account keys with external third-party applications. If these apps need access to your data, you'll need to secure their connections without using storage account keys.

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**

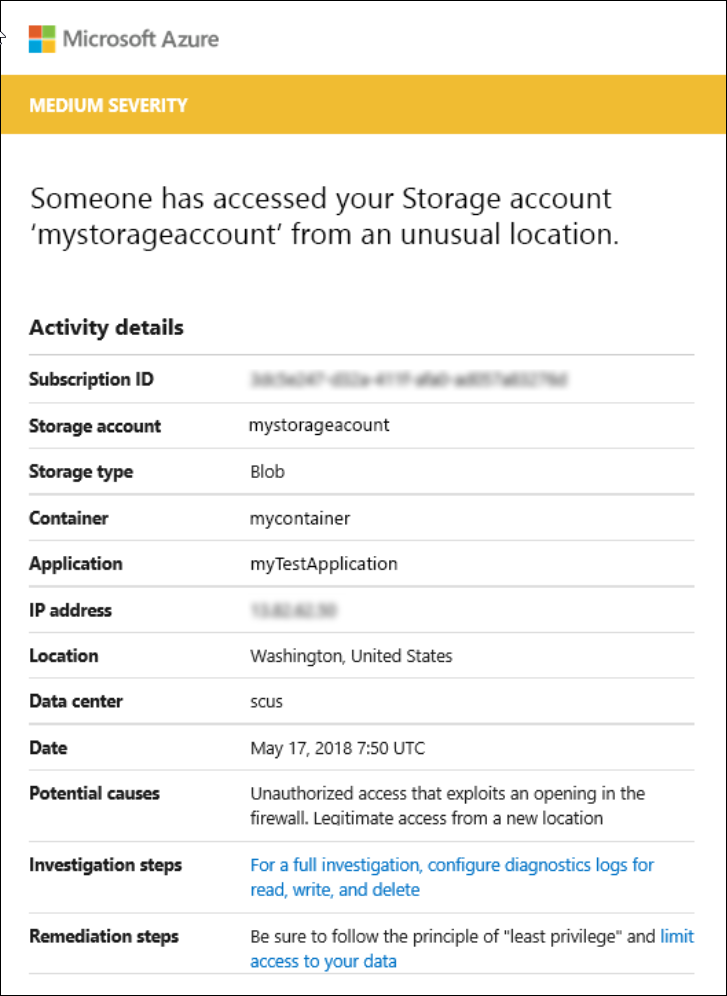


Azure Defender for Storage provides an extra layer of security intelligence that detects unusual and potentially harmful attempts to access or exploit storage accounts. This layer of protection allows you to address threats without being a security expert or managing security monitoring systems.

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**

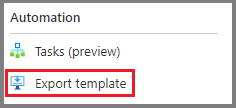
Best practice is that all Azure virtual machines will have at least two virtual hard disks (VHDs). **The first disk stores the operating system, and the second is used as temporary storage**. You can add additional disks to store application data; the maximum number is determined by the VM size selection (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.

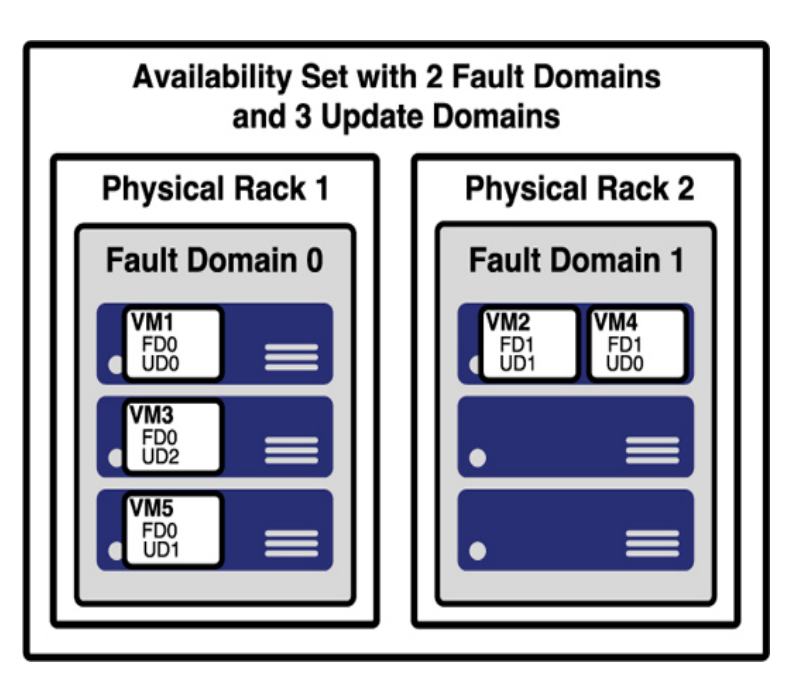
**What is a fault domain?**

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).

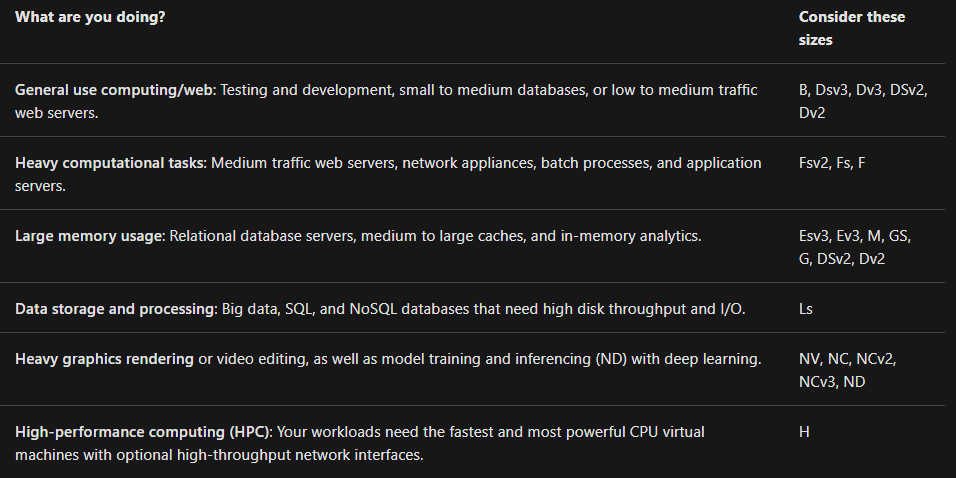
**Suppose you want to run a network appliance on a virtual machine. Which workload option should you choose?  
R:** Compute optimized virtual machines are designed to have a high CPU-to-memory ratio. Suitable for medium traffic web servers, network appliances, batch processes, and application servers.

**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.

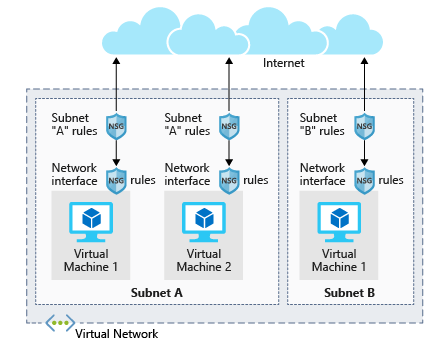
**What is a network security group?**

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**

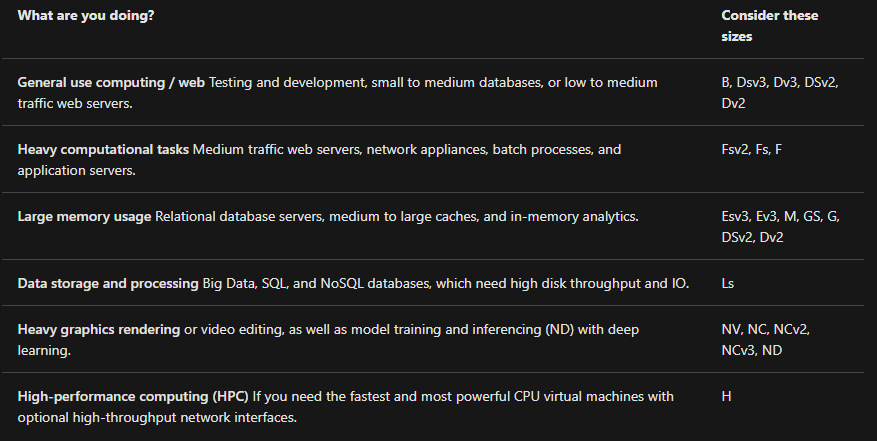
* A virtual machine that provides CPU and memory resources.
* An Azure Storage account to hold the virtual hard disks.
* Virtual disks to hold the OS, applications, and data.
* Virtual network (VNet) to connect the VM to other Azure services or your own on-premises hardware.
* A network interface to 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

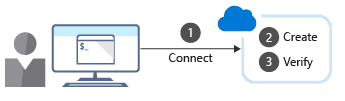
**How do you find the particular commands you need?**

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**

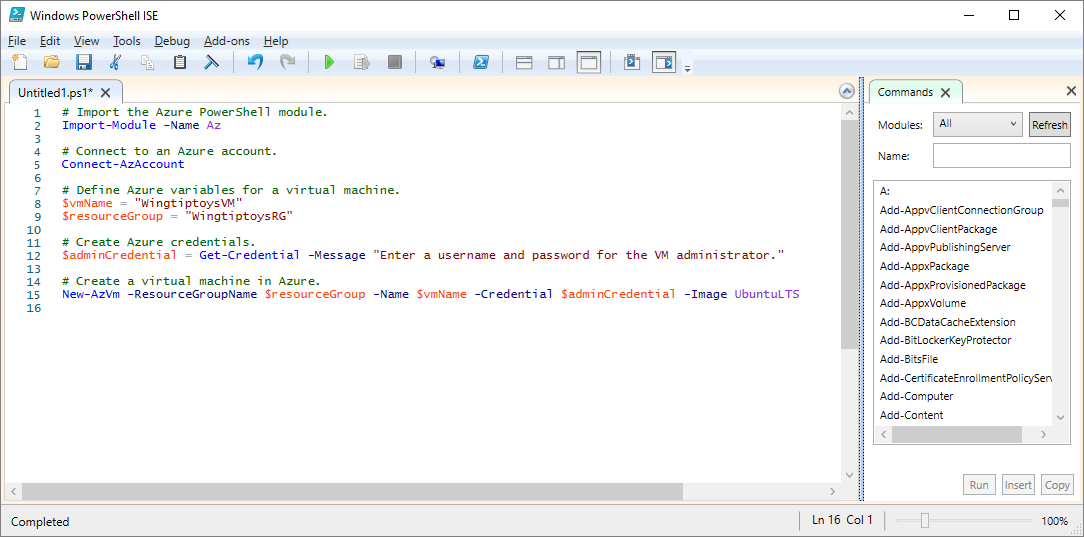
**Remove-AzVM** *command just deletes the VM. It doesn't clean up any of the other resources. At this point, we'd likely just delete the resource group itself, and be done with it.*

**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:

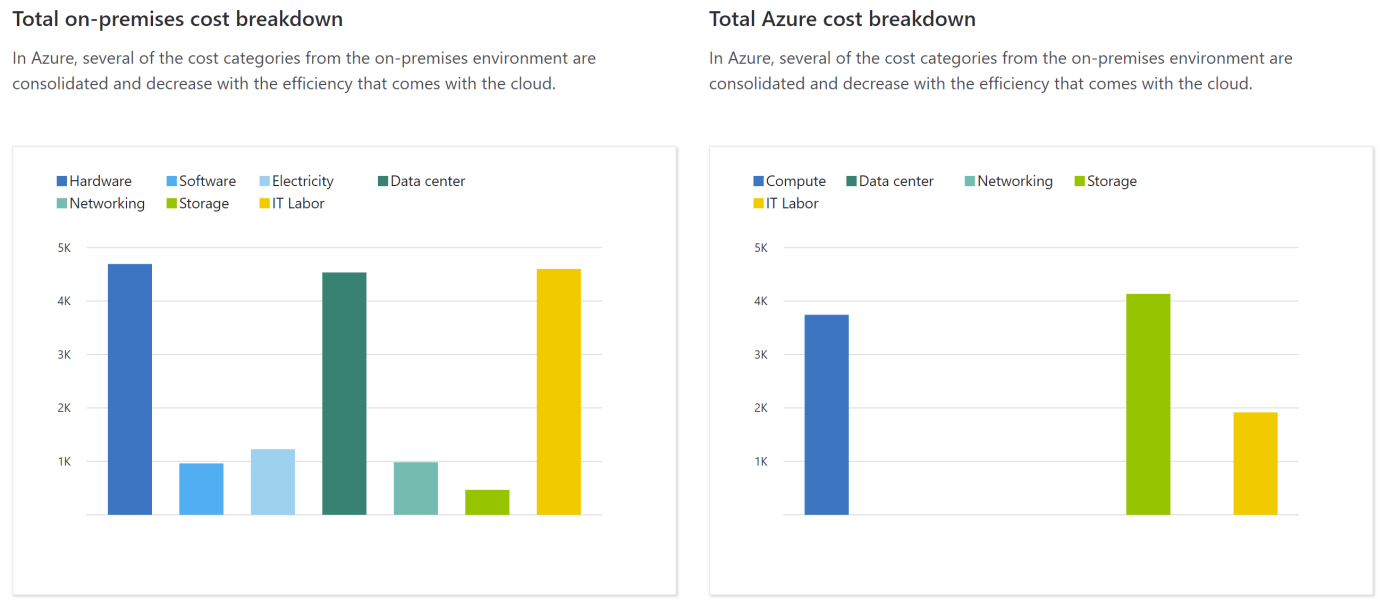
**param([int]$size, [string]$location)**

**Plan and manage your Azure costs**

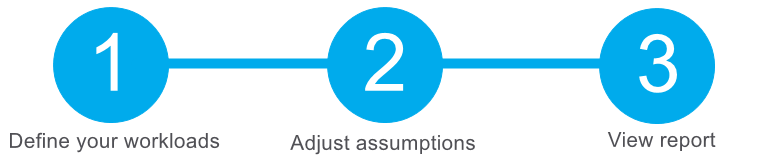
**What's the TCO Calculator?**

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:

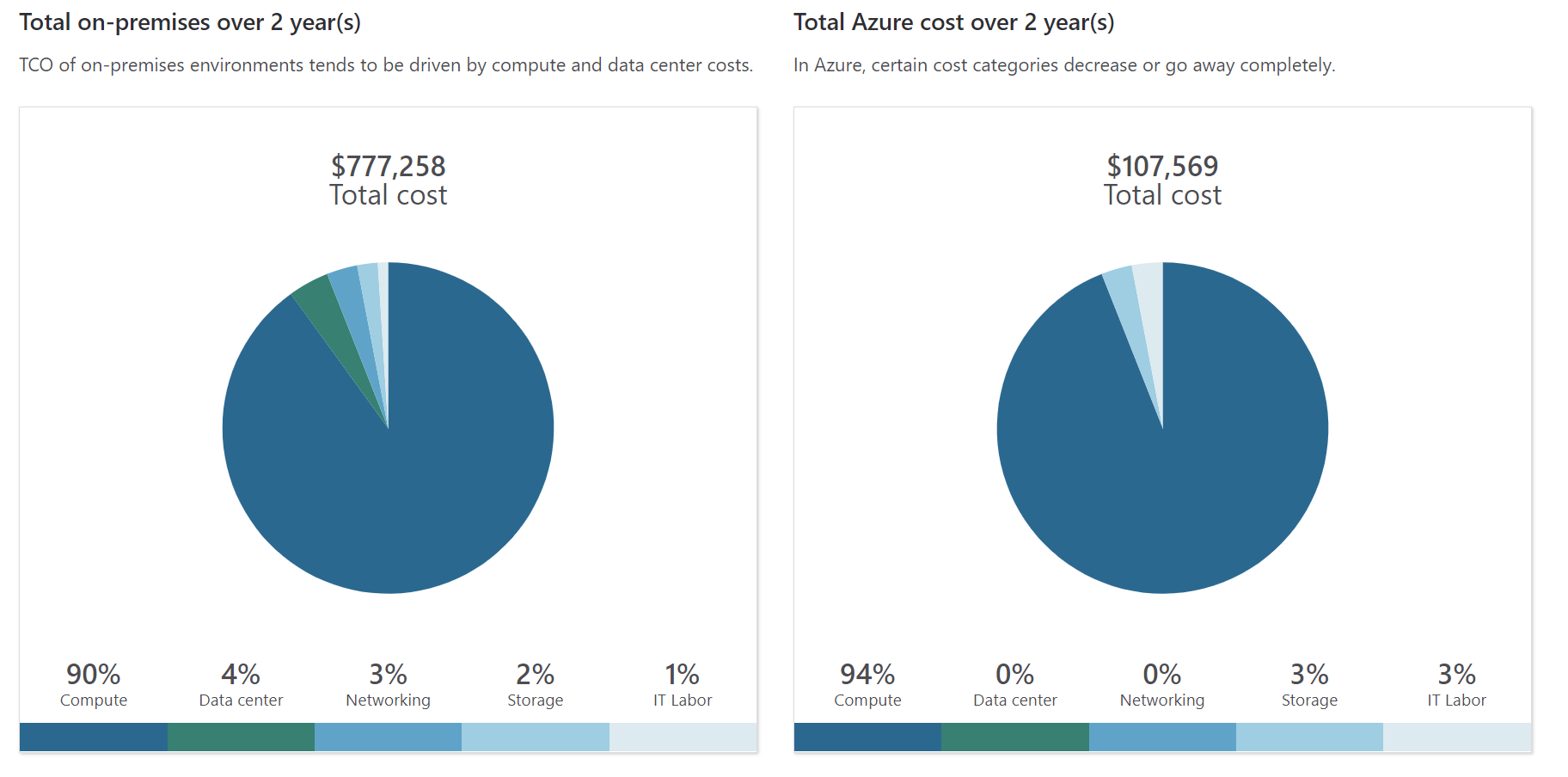
* Servers
  + This category includes operating systems, virtualization methods, CPU cores, and memory (RAM).
* Databases
  + This category includes database types, server hardware, and the Azure service you want to use, which includes the expected maximum concurrent user sign-ins.
* Storage
  + This 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

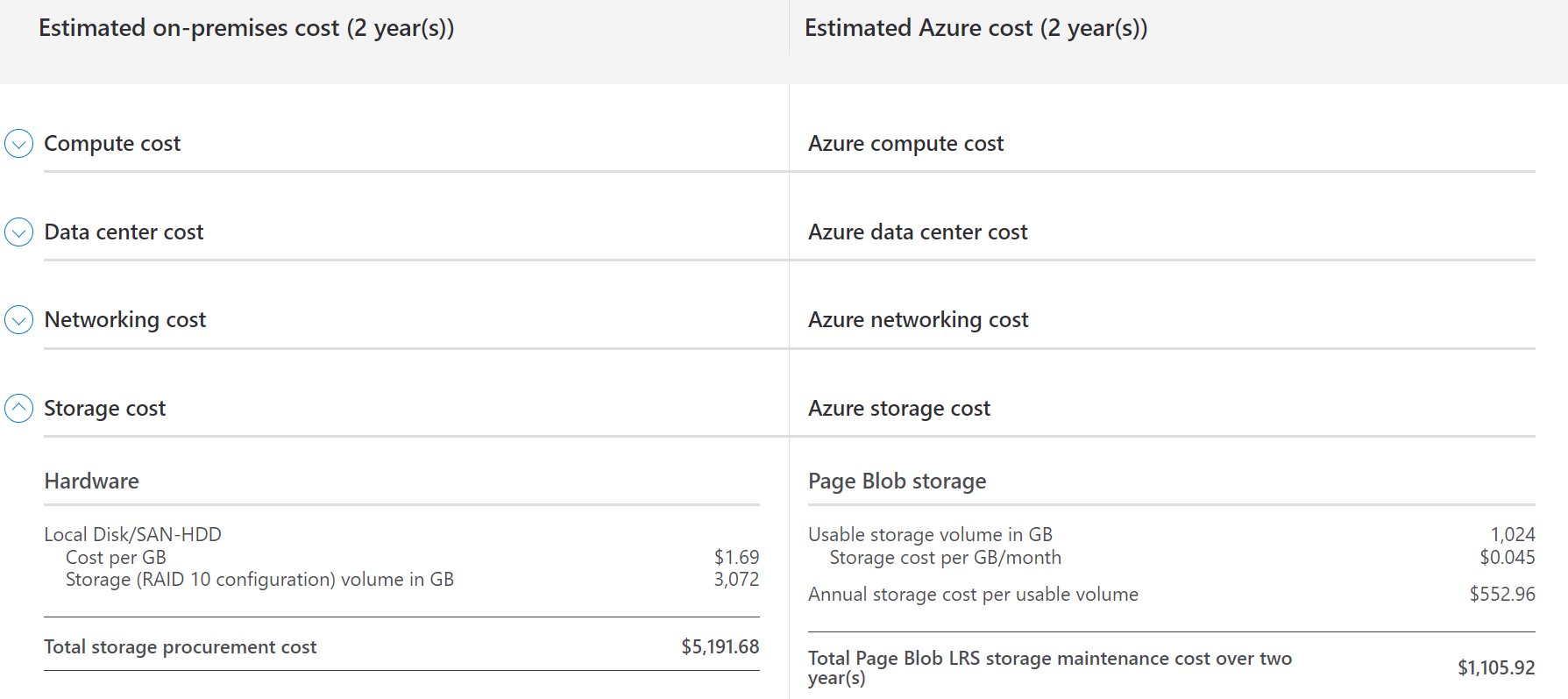
* specify whether your current on-premises licenses are enrolled for Software Assurance, which can save you money by reusing those licenses on Azure.
* specify whether you need to replicate your storage to another Azure region for greater redundancy.
* See the key operating cost assumptions across several 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?**

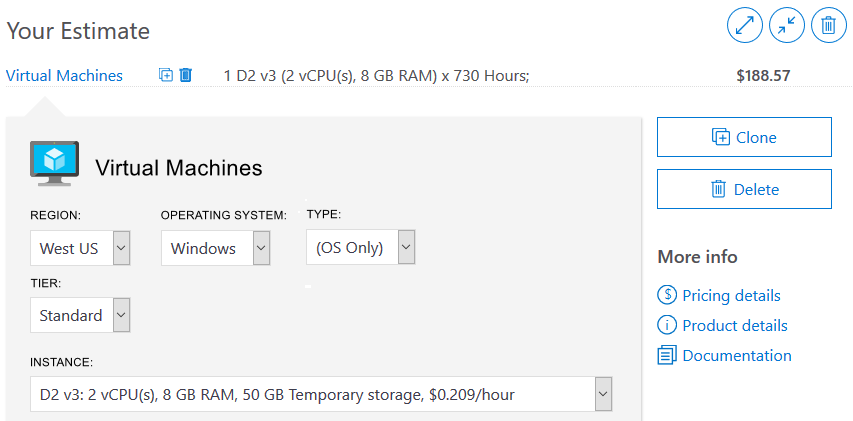
* **Free trial** provides you with 12 months of popular free services, a credit to explore any Azure service for 30 days, and more than 25 services that are always free. Your Azure services are disabled when the trial ends or when your credit expires for paid products, unless you upgrade to a paid subscription.
* **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>

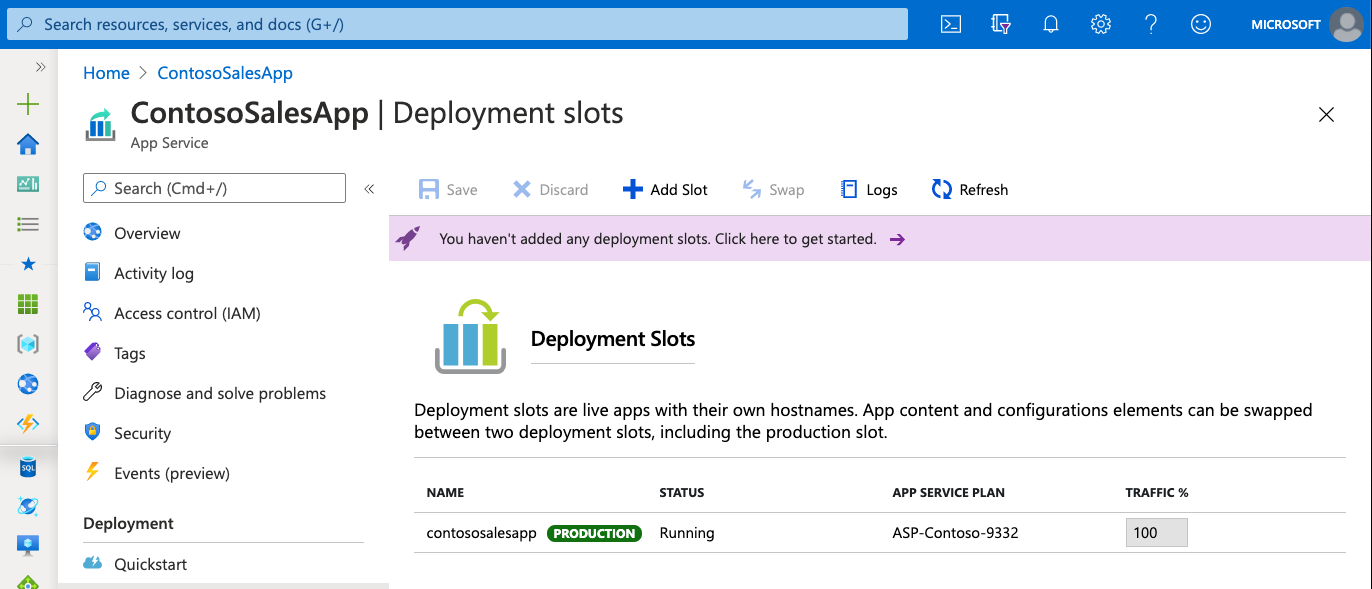
**Host a web application with Azure App Service**

**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.

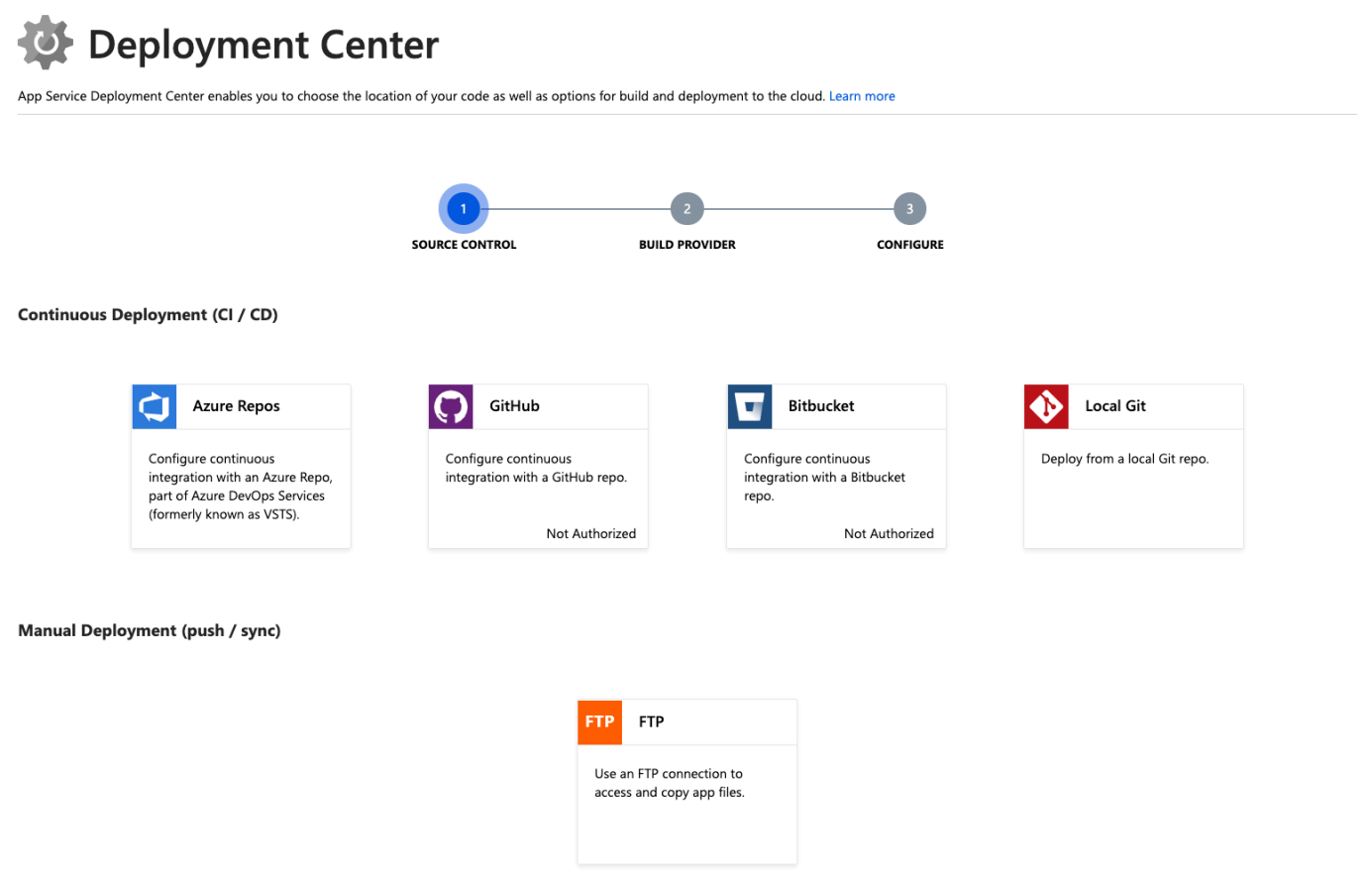
**Deployment slots**

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**

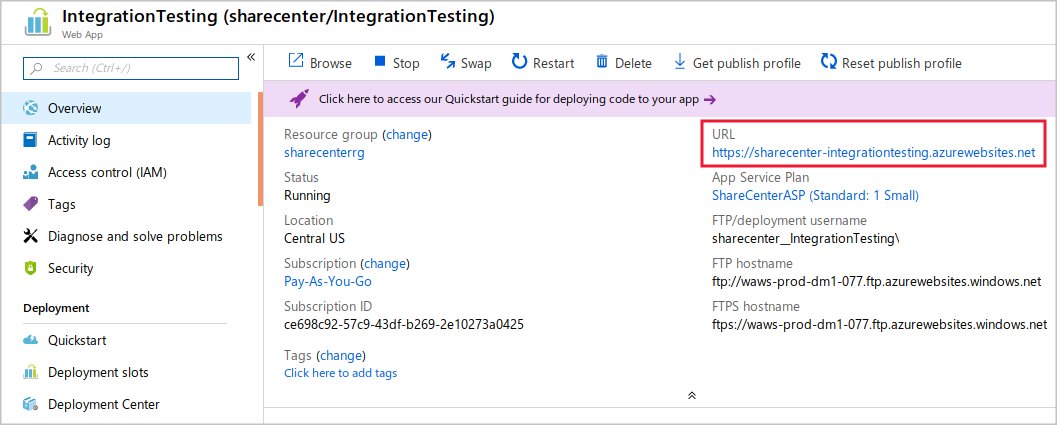
When you swap a slot into production, you "warm-up" the app because your action sends a request to the root of the site. The warm-up request ensures that all compilation and caching tasks finish. After the swap, the site responds as fast as if it had been deployed for days.

**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.***

Summary

In this module, you learned how to use deployment slots in Azure App Service. You used slots to test and stage new versions of your app, and you swapped those deployment slots. You can swap slots both to deploy a tested app to production, and to roll back a deployed app when unexpected problems arise.

**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