# Serverless computing:

* Serverless computing is a microservice run on the cloud. It is FAAS (Function as a service).
* It’s a best way to host business logic on cloud. It provide scaling based on user request and pay as used, paying model.
  1. **Automatic Scaling**
  2. **No Server to manage**
  3. **Pay as you go model**

### Stateless logic:

* Function instances are create and destroyed on demand.
* If required, state can be stored in storage service.

### Disadvantages of Serverless funtions.

* Serverless function have timeout of 5 min and can configure up to 10 min.
* If function is using HTTPS request, https request have 2.5 min timeout.

### Trigger in functions app

* Timer, HTTP, Blob, queue, event-hub, azure cusmosDB

### Binding in function app

* Binding is a way of connecting with data. Normally a function have 2 type of binding.
  + Input binding -> it is used to provide input to function app
  + Output binding -> it is used to store the output from function.
* A function have multiple input/output binding.

### What is an HTTP trigger Authorization level?

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

There are three Authorization levels:

1. Function
2. Anonymous
3. 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.

## What is Azure Storage?

Azure Storage is Microsoft's cloud storage solution that supports all types of data, including blobs, queues, and NoSQL. The goal of Azure Storage is to provide data storage that's:

* Highly available
* Secure
* Scalable
* Managed

# Serverless computing:

* Serverless computing is a microservice run on the cloud. It is FAAS (Function as a service).
* It’s a best way to host business logic on cloud. It provide scaling based on user request and pay as used, paying model.
  + **Automatic Scaling**
  + **No Server to manage**
  + **Pay as you go model**

### Stateless logic:

* Function instances are create and destroyed on demand.
* If required, state can be stored in storage service.

### Disadvantages of Serverless funtions.

* Serverless function have timeout of 5 min and can configure up to 10 min.
* If function is using HTTPS request, https request have 2.5 min timeout.

### Trigger in functions app

* Timer, HTTP, Blob, queue, event-hub, azure cusmosDB

### Binding in function app

* Binding is a way of connecting with data. Normally a function have 2 type of binding.
  + Input binding -> it is used to provide input to function app
  + Output binding -> it is used to store the output from function.
* A function have multiple input/output binding.

### What is an HTTP trigger Authorization level?

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

There are three Authorization levels:

1. Function
2. Anonymous
3. 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.

## What is Azure Storage?

Azure Storage is Microsoft's cloud storage solution that supports all types of data, including blobs, queues, and NoSQL. The goal of Azure Storage is to provide data storage that's:

* Highly available
* Secure
* Scalable
* Managed

## 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. Finally, page blobs are made up of pages and are designed for frequent random read and write operations.

### What is a blob trigger?

A blob trigger is a trigger that executes a function when a file is uploaded or updated in Azure Blob storage. To create a blob trigger, you create an Azure Storage account and provide a location that the trigger monitors.

# Durable Functions

Durable functions is extension of azure functions which can run long lasting with stateful manner.

* Whereas azure function operates in stateless environment, durable function manage state between different function call.

**Benefits of using Durable functions:**

* Allow us to write event-driven code. (It waits for trigger based on one or more external events and then perform series of task in response to this events).
* You can chain function together. (passing output of one function as input of other function)
* You can orchestrate and coordinate function based on order you want.
* State is managed for you.

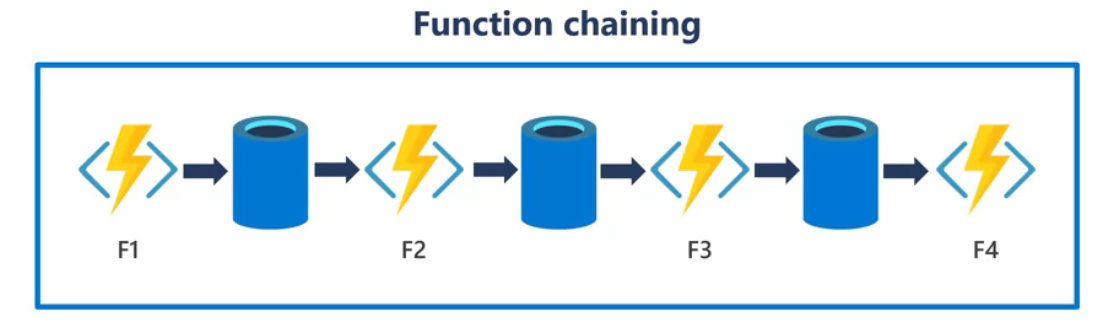
**Durable function types:**

1. Client Function: It can act as entry point for a function.
2. Orchestrator function: Describe how actions as executed and in what order.
3. Activity Function: Unit Work Item.

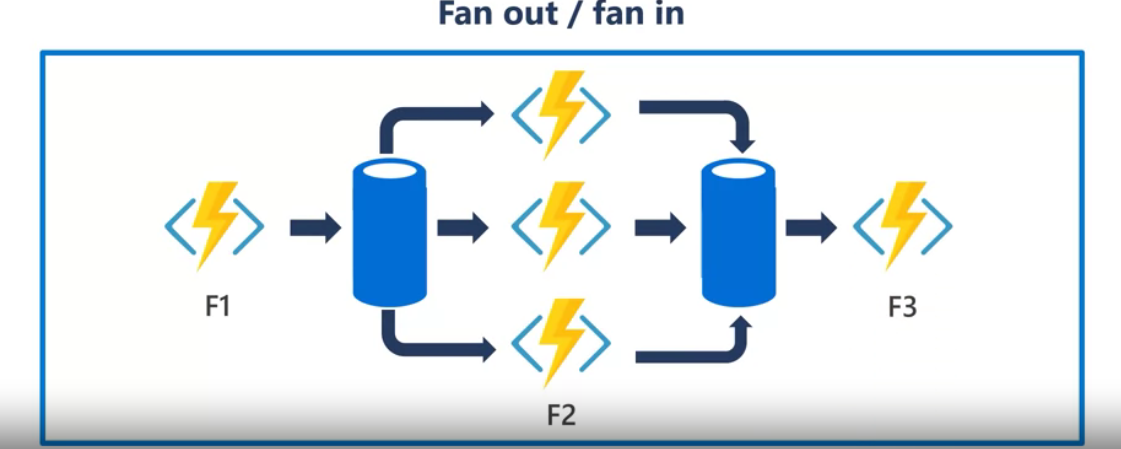
* **Client Function:**
  1. A client function is typically the entry point or trigger for a serverless workflow. It is a piece of code or function that initiates and manages the execution of a workflow. The client function interacts with the workflow orchestration system to initiate the workflow and pass any required input or data to it.
  2. The client function is responsible for triggering and monitoring the workflow, handling user interactions or external events, and providing input data to the orchestration.
* **Orchestration:**
  1. Orchestration, in the context of serverless computing, is a higher-level control mechanism that coordinates and manages the execution of multiple functions or activities in a specific order or workflow. It defines the workflow's logic, including which activities to execute and their sequence.
  2. Orchestration is responsible for maintaining the state of the workflow, handling retries, error handling, and ensuring that all activities are executed according to the defined logic. It often relies on a state machine or declarative definition to define the workflow's structure.
* **Activity Function:**
  1. Activity functions are the individual units of work within a serverless workflow. These functions perform specific tasks or operations, such as data processing, API calls, database interactions, or any other task required by the workflow.
  2. Activity functions do not have knowledge of the overall workflow's structure or state; they receive their input from the orchestration and perform their tasks independently. They report the results back to the orchestration, which then determines the next step based on the results.

**Application Pattern:**

1. Function Chaining (Series of function run in order and output of one function is act as input of another function.



1. Fan in/ Fan out

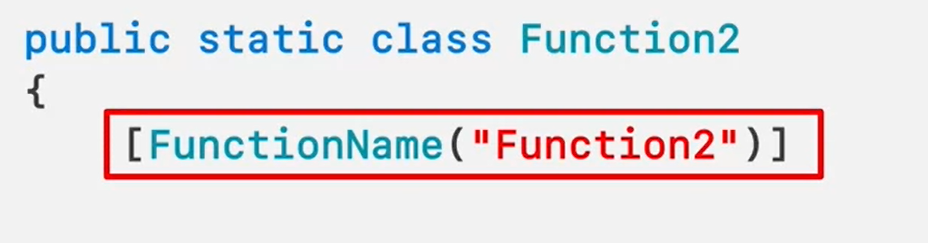


1. Async HTTP API
2. Monitor
3. Human Interaction.

# Azure Function App Core Tools

Azure offer a wide range of core tools such as CLI, Cloud Shell ect.

### Function App:

* Every Function app has a trigger associated with it. For example,
  1. Service Bus Queue trigger
  2. Http trigger
  3. Web Hook (when there was any changes to a thing hosted by web, it will place and http request to an defined URL.
  4. Timer trigger
  5. Queue trigger
  6. Blob trigger (When a file is uploaded or updated in Blob storage
  7. Event Hub Trigger (When event hub receive message)
* Azure function HTTP trigger
  + It might exposed publicly, to avoid this we need to restrict access.
    - Authorization Level
      1. **Anonymous**: No authentication is required
      2. **Function**: The http request must provide a key for function app to authorize. Key can be created separately and can be maintained using portal.
      3. **Admin**: It is same as Function Key but it is admin key which can access to any function in the FAPP. However, function key can only give you access to a specific function.
  + Anything returned from FAPP will be wrapped inside **IActionResult** Object.
  + In all cases, azure FAPP pass a parameter ILogger which is used to log information for debugging purposes.
  + 
    - This FunctionName attribute is identifier for a function.

## Deployment Options from local machine:

1. Deploy from Visual Studio
2. Continuous deployment from GitHub or azure devops
3. Zip Deployment.

# Create a function triggered by a webhook

It is Just normal Function App triggered by HTTP request.

### Set up a webhook for a GitHub repository:

-> In GitHub, webhook can be set-up on Organization Repository or Specific Repository. It get triggred when specific event occurs.

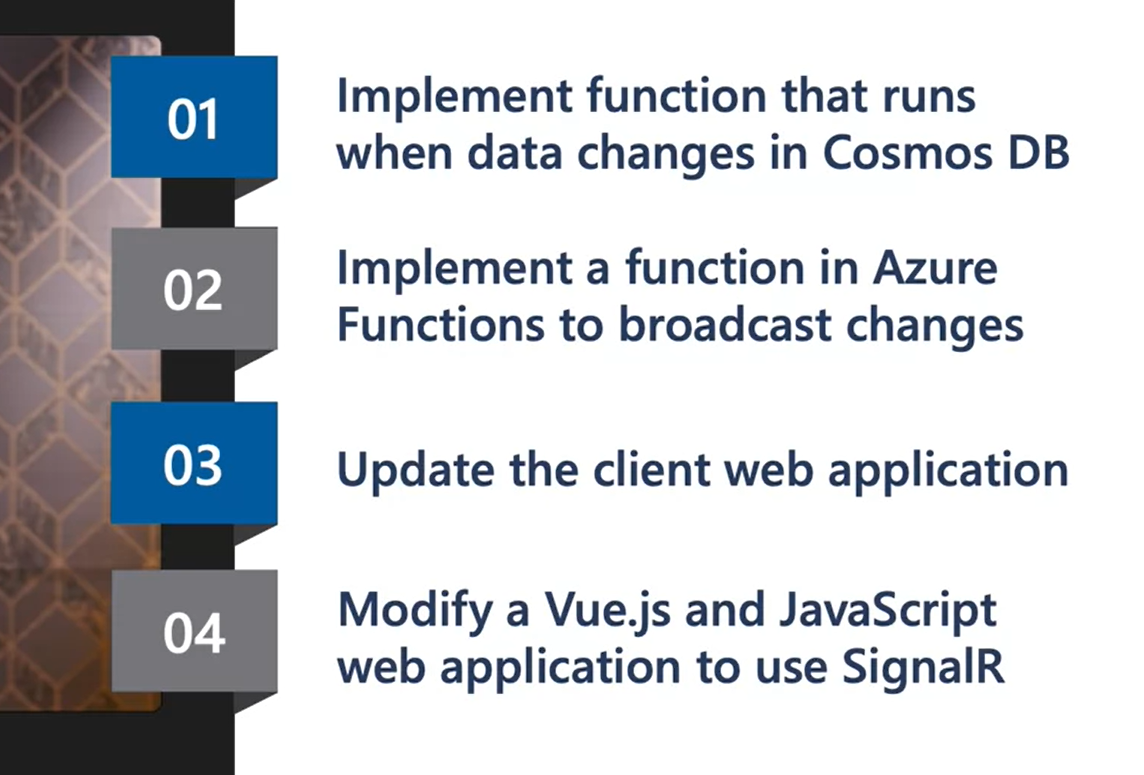
-> Events are at the center of webhooks. Events occur whenever actions are taken in the repository. When the event occurs, the webhook fires off and calls the URL that you specify, sending along the payload and event information to your URL.

-> [Exercise - Set up a webhook for a GitHub repository - Training | Microsoft Learn](https://learn.microsoft.com/en-us/training/modules/monitor-github-events-with-a-function-triggered-by-a-webhook/5-exercise-setup-webhook-for-github-repo)

-> When an event occurs, an Azure Functions function is triggered. The function can parse the payload to retrieve and process the data sent to extract the fields from the payload, and take the appropriate actions.

-> Secure this passing of payload with a secure Key. ([Exercise - Secure webhook payloads with a secret - Training | Microsoft Learn)](https://learn.microsoft.com/en-us/training/modules/monitor-github-events-with-a-function-triggered-by-a-webhook/9-exercise-secure-webhook-payloads-with-secret)

### I.e. Built a stock price showing website and it update price when ever there is a change in DB.\



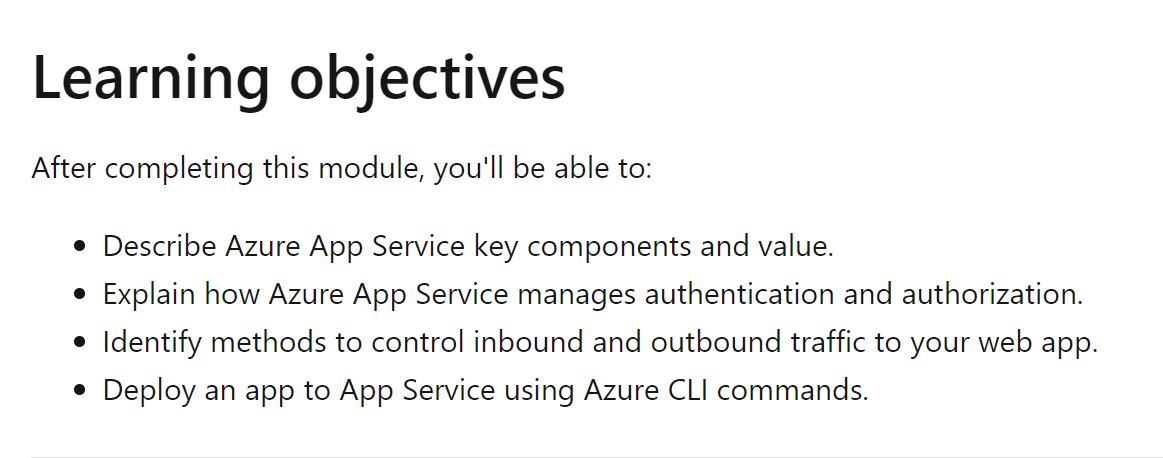
**Analyze the limitations of a polling-based web app**

**->** Polling-Based web App: First consider an example that we are building an stock app which will show stocks price. Now in polling-based web app, we develop a function which will update the data on website every 5 seconds(Even If there is not change in data). This fetching data every 5 seconds (when application grows and demands huge data), it will start making trouble for server and overhead the process

New Design:

1. Only update data on front-end if data changes in DB (Using SignalR)
2. SignalR:
   1. It allows for two-way communication between the client and server.
   2. It handles connection management automatically.
   3. It lets you broadcast messages to all connected clients simultaneously.
   4. It does support sending messages to specific clients

# Azure App Service



Azure app service is an http based web hosting platform that can host REST API, Web backend, and mobile back-end. It can easily work with window and Linux based environments.

* It has built in auto scaling support
  + Depending upon the usage you can scale-up or scale-down the resource that is hosting your services.
  + Resources include the number of cores or the amount of RAM available. Scaling out/in is the ability to increase, or decrease, the number of machine instances that are running your web app.
* CI/CD support
  + Azure portal provides out of box support to CI/CD with azure DevOps services, GitHub, Bitbucket, FTP or a local Git Repo.
* Deployment Slots
  + P02, UAT, QA, PRD (working environment)
  + It is like a different environment of development.
  + Using Deployment slots you can create testing, developing version of resource in same the same resource and when ok with testing, you can swap production slot to testing slot.
* Limitations
  + App Service on Linux isn’t supported on Shared Pricing tier.

## **Azure App Service Plans**

* In App Service, an app always runs in an App Service Plan.
* An App Service plan defines a set of compute resources for a web app to run. One or more apps can run on same app service plan.
* App service plan defines
  + Operating system (window, Linux)
  + Region
  + Number of VM instances
  + Size of VM instance
  + Pricing tier (free, shared, basic, standard, premium, ..)
* The *pricing tier* of an App Service plan determines what App Service features you get and how much you pay for the plan. There are a few categories of pricing tiers:
  + **Shared compute**: **Free** and **Shared**, the two base tiers, runs 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 can't scale out.
  + **Dedicated compute**: The **Basic**, **Standard**, **Premium**, **PremiumV2**, and **PremiumV3** 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.
  + **Isolated**: The **Isolated** and **IsolatedV2** tiers run dedicated Azure VMs on dedicated Azure Virtual Networks. It provides network isolation on top of compute isolation to your apps. It provides the maximum scale-out capabilities.
* Your App Service plan can be scaled up and down at any time. It's as simple as changing the pricing tier of the plan. If your app is in the same App Service plan with other apps, you may want to improve the app's performance by isolating the compute resources. You can do it by moving the app into a separate App Service plan.

## **Continuous Deployment**

* CD is a process used to push out new features and bug fixes ina fast and repetitive pattern with minimal effect on end user.
* Azure support CD directly from
  + Azure DevOps
  + GitHub
  + Bitbucket

## **Authentication and Authorization in App Service**

* Azure provides built-in authentication and authorization support, so you can sign in users and access data by writing minimal code.

## **App Service Networking features**

* By defaut, apps hosted are accessible directly through the internet and can reach only **internet-hosted endpoints**.

# Azure Function

-> Azure functions is a serverless solution that allows you to write less code, maintain less infrastructure, and save on costs.

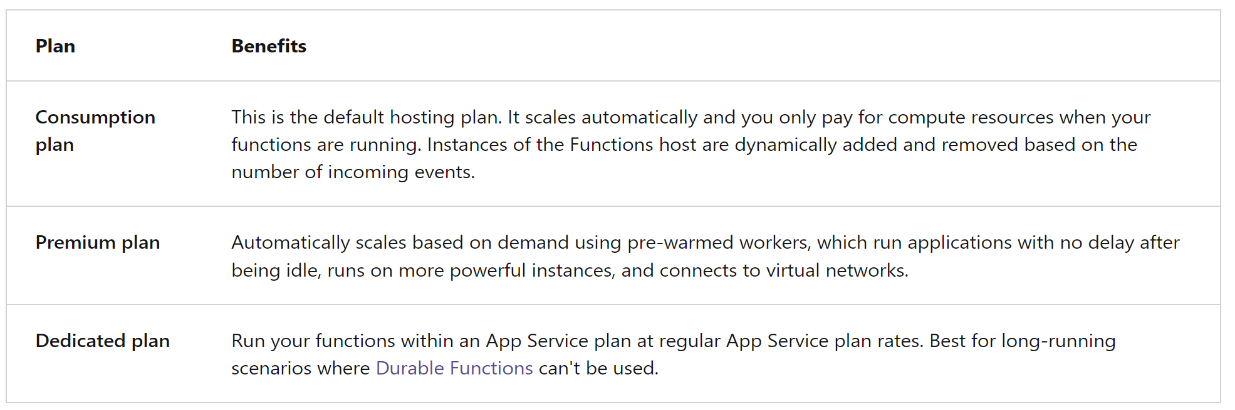
-> Azure Functions supports triggers, which are ways to start execution of your code, and bindings, which are ways to simplify coding for input and output data.

## **Azure Functions and logic apps**

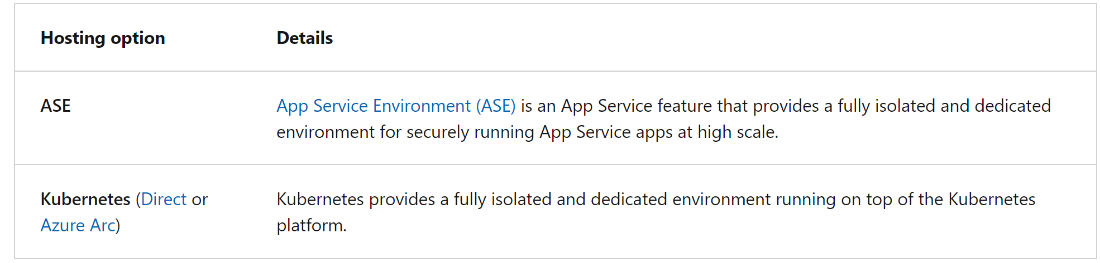
* Azure Function -> serverless compute service
* Azure Logic Apps -> serverless workflow integration platform.
  + For Azure Functions, you develop orchestrations by writing code and using the **Durable Functions extension**. For Logic Apps, you create orchestrations by using a GUI or editing configuration files.

## **Azure Function hosting options**

* There are three basic hosting options for Azure functions
  1. Consumption Plan
  2. Premium Plan
  3. App service Plan (dedicated)
* The hosting plan you choose dictates the following behaviors:
  1. How your function app is scaled.
  2. The resources available to each function app instance.
  3. Support for advanced functionality, such as Azure Virtual Network connectivity.

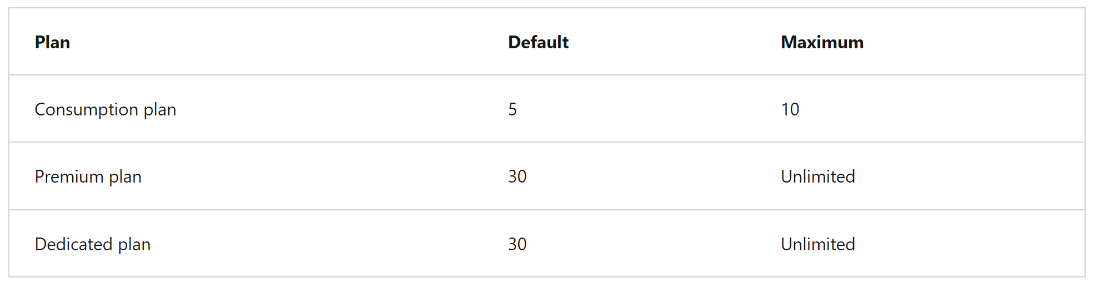


Other Hosting options:

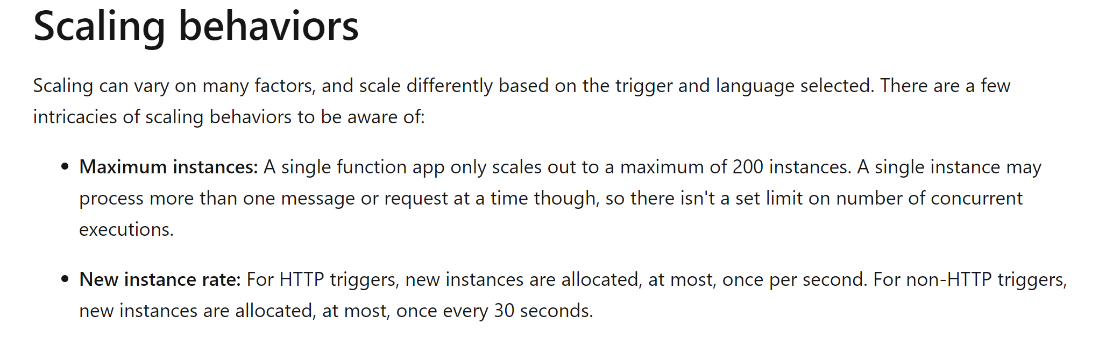


## **Function App Timeout Duration**

The following table shows the default and maximum values (in minutes) for specific plans



-> Azure Functions uses a component called the scale controller to monitor the rate of events and determine whether to scale out or scale in.



## Azure web App

* While creating web apps, we need to decide what will be the run time stack. There are many available run time stack but if the one you wants are not available, then you can select Docker Container in Publish section and you can insert your code in it and you also need to add compiler as well.

App Service Plan:

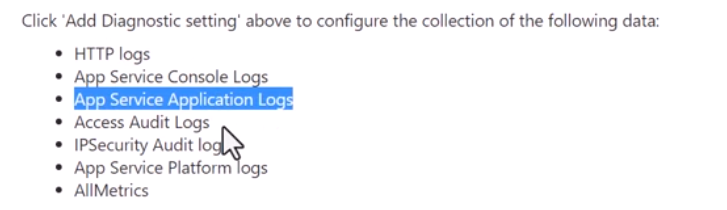
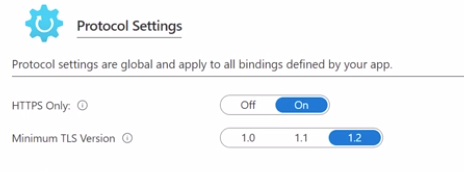
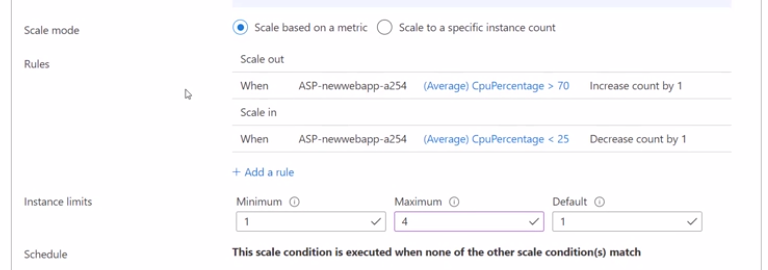
* Its like selecting size and CPU for a VM. Different app service plan have different pricing model. One app service plan can host many resources.

# Azure Compute Solution

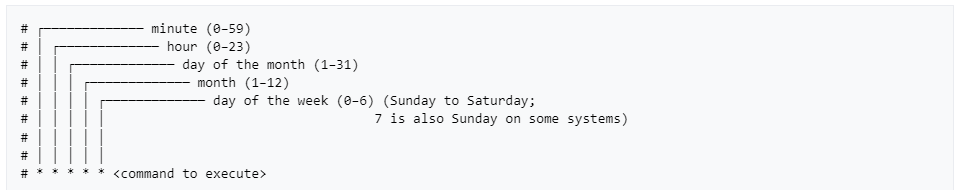
## Implement containerized solutions

* What is Container in development
  + Containers are lightweight packages of your application code together with dependencies such as specific versions of programming language runtimes and libraries required to run your software services.
  + Azure Container Instances (ACI) is a service that enables a developer to deploy containers on the Microsoft Azure public cloud without having to provision or manage any underlying infrastructure. The service supports both Linux containers and Windows containers.
* Create and manage container images for solutions
  + Container image can be managed in Container Registry.
* Publish an image to Azure Container Registry
  + Azure Container Registry is like a directory of Container image and you can add custom Images into it.
  + You can deploy your images which are developed, tested and ready to deploy to container here.
  + Just go to repository under services and there you can deploy images.
* Run containers by using Azure Container Instance
  + Container Instance is the easiest way to deploy container in azure.
  + You can select available image or deploy image to container registry while deploying it.
* Create solutions by using Azure Container Apps
  + To deploy Images in Container Registry, first we need to develop image.
  + We can use visual studio to develop code for image and we need Docker Desktop installed and running. Docker is industries accepted image format. We need to convert that code into docket image for uploading in container repo.
  + For converting code to docker image, we need to add docker support code into our solution and for adding that we need visual studio container tool installed in visual studio.
  + Docket desktop, we can run this app for window and linux (it is like different version but you can switch from app).
  + In your vs solution, you can go to add and then click on docker support and it will open a pop up window asking window or linux.
  + We can also deploy/Use this container image and deploy this image to web app.
  + Now then what is difference between container instance and web app as we can use both to deploy container image.
    - The first difference is pricing in both resources and second is web app has more control options such as scaling, auto scaling and backup and application insights which container instance don’t have.

## Azure App Service Web Apps

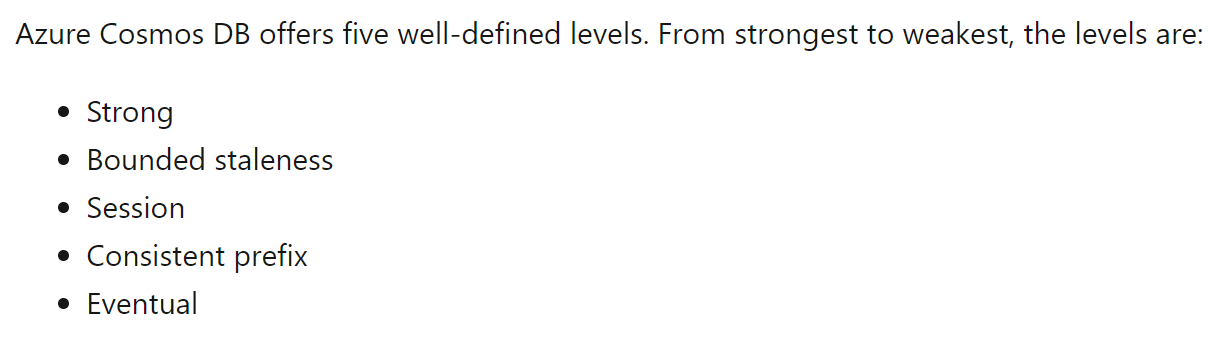
* Create an Azure App Service Web App
* Enable diagnostics logging
  + By default app service logging is off, you can turn on by going to app service log and turn on logging.
    - App service logging has following options:
      1. Application Logging (File Storage)
      2. Application Logging (Blob storage)
      3. Web Server Logging
      4. Detailed Error Log
      5. Failed request tracing
  + There is a second type of logging, you can go to the diagnostics setting.
  + With Azure diagnostic logs, you can view core analytics and save them into one or more destinations including: Azure Storage account. Log Analytics workspace. Azure Event Hubs. Following are logs we can get under diagnostic logs:
    - 
* Deploy code to a web app:
  + Deploy Directly or using pipeline:
    - You can deploy your code directly from visual studio code. Just go to publish by right clicking on solution and select resource and publish code.
    - One can also deploy code in web apps using pipeline to GitHub, azure devOps and git bucket.
  + Create Web app PowerShell:
    - 
* Configure web app settings including Secure Sockets Layer (SSL), API settings, and connection strings
  + Web app or anything in azure or in any project in visual studio code, have default application configuration in .config file. In web app, in application setting, any setting you add, will overwrite that default configuration.
  + SSL:
    - This are the setting, one can set under SSL.
    - SSL certificate is required, if we are using custom domain for website. If we are using .azureappservice.net, microsoft have default SSL certificate for this domain but if we are using custom domain, then we can purchase SSL certificate and upload it to azure under SSL tab in app service.
* Implement autoscaling
  + Autoscaling is only available for standard or higher plans.
  + Autoscaling enables a system to adjust the resources required to meet the varying demand from users, while controlling the cost associated with these resources.
  + Autoscaling enables you to specify the conditions under which a web app should be scaled out and back in again.
    - Autoscaling is like a smart system that automatically changes how much computer power it needs based on how many people are using it. This helps make sure it has enough power when lots of people are using it and saves money when fewer people are around.
  + Autoscaling can be triggered according to schedule or by assessing whether the system is running short on resources like:
    - CPU utilization grows
    - Memory occupancy increases
    - Incoming requests surging
  + Azure provides two options for autpscaling.
    - Scale based on a metric: such as the length of the disk queue or number of http requests awaiting processing
    - Scale to a specific instance count according to schedule.
  + Metrics for autoscale rules:
    - Autoscaling by metric requires that you define one or more autoscale rules. An autoscale rule specifies a metric to monitor, and how autoscaling should respond when this metric crosses a defined threshold.
      1. CPU Percentage
      2. Memory Percentage
      3. Desk Queue length
      4. Http queue length
      5. Data in
      6. Data out
    - Example

## Implement Azure Functions

* Create and configure an Azure Function App
  + ...
* Implement input and output bindings
  + Go to Integration and there you can modify input/output binding.
* Implement function triggers by using data operations, timers, and webhooks
  + Timer:
    - When creating timer trigger function app we need to provide recurrence timing we want function to run.
    - CRON format is followed for giving schedule.
      * CRON: SMHDMW
      * 
    - i.e. 0 \*/5 \* \* \* \* -> every 5 minutes
* Durable function:
  + To work with durable function we need dependancy installed in project to work with it.
  + For Node.js
    - Go to app service editor (priview)
    - Add package.json file with some default content like name and version.
    - Then exit the editor. Go to console and run npm install durable\_function.

# Develop for Azure storage (15–20%)

#### **Develop solutions that use Azure Cosmos DB**

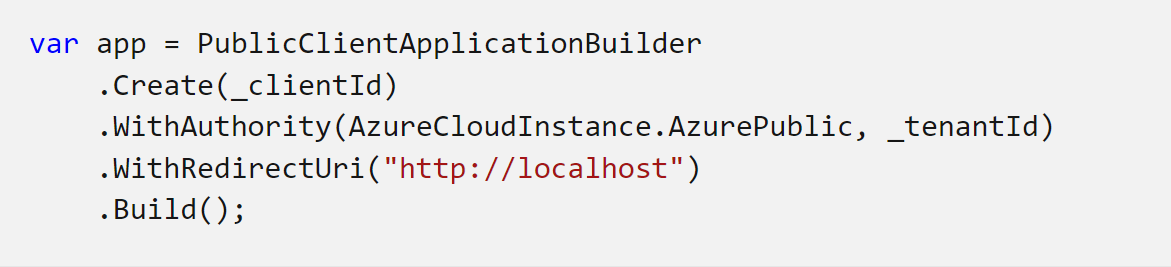
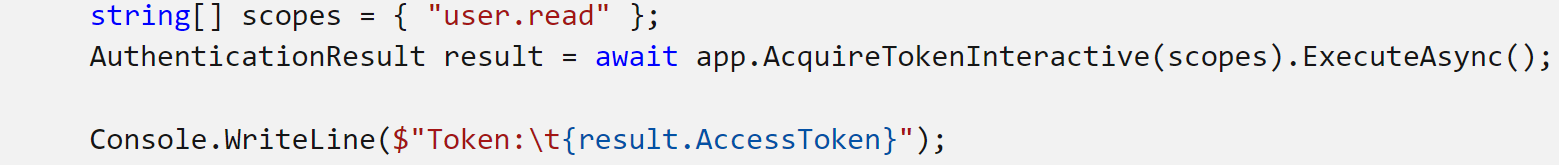
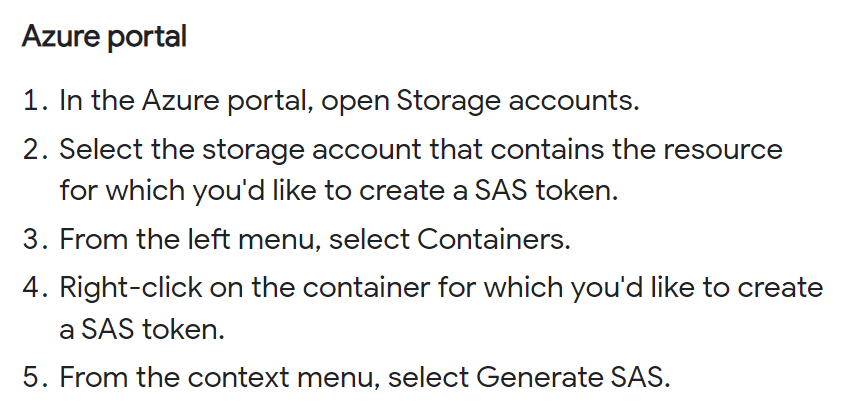
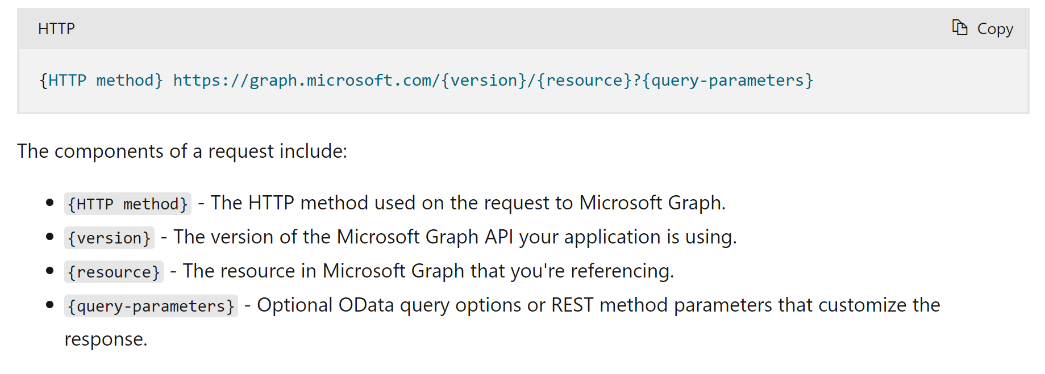
* Perform operations on containers and items by using the SDK.
  1. A container is a schema-agnostic container of items. Items in a container can have arbitrary schemas. For example, an item that represents a person and an item that represents an automobile can be placed in the same container. By default, all items that you add to a container are automatically indexed without requiring explicit index or schema management.
* Set the appropriate consistency level for operations
  1. 
  2. **Strong** is like if write in one region, it will available in other region in no time.
  3. **Bounded staleness** the lag of data between any two regions is always less than a specified amount (Either “K” versions or “T” time intervals, whichever is reached first) you need to set this K and T.
  4. **Session** After every write operation, the client receives an updated Session Token from the server. The client caches the tokens and sends them to the server for read operations in a specified region. If the replica against which the read operation is issued contains data for the specified token (or a more recent token), the requested data is returned. If the replica doesn't contain data for that session, the client retries the request against another replica in the region. If necessary, the client retries the read against extra available regions until data for the specified session token is retrieved.
  5. Consistent prefix : if u write a document, all of the content will update all together whatever time.
  6. Eventual : In Eventual consistency, the client issues read requests against any one of the four replicas in the specified region. This replica may be lagging and could return stale or no data. Eventual consistency is the weakest form of consistency because a client may read the values that are older than the ones it read in the past. Eventual consistency is ideal where the application doesn't require any ordering guarantees. Examples include count of Retweets, Likes, or non-threaded comments. The following graphic illustrates the eventual consistency with musical notes.
* Implement change feed notifications
  1. Change feed in Azure Cosmos DB is a persistent record of changes to a container in the order they occur. Change feed support in Azure Cosmos DB works by listening to an Azure Cosmos DB container for any changes. It then outputs the sorted list of documents that were changed in the order in which they were modified.
  2. There are two ways you can read from the change feed with a push model: Azure Functions Azure Cosmos DB triggers, and the change feed processor library.

#### **Develop solutions that use Azure Blob Storage**

* Set and retrieve properties and metadata
  1. Metadata are tags, we can set this tags using portal by going to properties or could do using visual studio code like this:-
  2. This is how we get meta data using getProperties and we can set meta data using SetMetadata();
  3. 
* Perform operations on data by using the appropriate SDK
* Implement storage policies and data lifecycle management
  1. We can enable backup for storage going to data protection under data management.
  2. Lifecycle management allows us to write rules to move blob from hot tier to cold tier or archive tier based on last access date.
* Implement static website hosting:
  1. We can search static web app from marketplace.
* Copy files between storage accounts and containers.
  1. We can use AzCopy in cloud Shell. There are many commands for different options like copy from google cloud, and all but if we want to copy with in azure we can use ‘azcopy copy ‘sourceURLWithSAS’ ‘destinationURLWithSAS’.
     1. SourceURL SAS must have read access.
     2. DestinationURL SAS must have write, add, read access.
  2. For Visual Studio:
     1. 

### **Implement Azure security (20–25%)**

#### **Implement user authentication and authorization**

* Authenticate and authorize users by using the Microsoft Identity platform
* To access resources secured by a Microsoft Entra tenant, the entity that requires access must be represented by a security principal. This is true for both users (user principal) and applications (service principal).
* The security principal defines the access policy and permissions for the user/application in the Microsoft Entra tenant. This enables core features such as authentication of the user/application during sign-in, and authorization during resource access.
* The application object is the *global* representation of your application for use across all tenants, and the service principal is the *local* representation for use in a specific tenant. The application object serves as the template from which common and default properties are *derived* for use in creating corresponding service principal objects.
* Authenticate and authorize users and apps by using Microsoft Entra ID.
* In OAuth 2.0, these types of permission sets are called *scopes*. They're also often referred to as *permissions*. In the Microsoft identity platform, a permission is represented as a string value. An app requests the permissions it needs by specifying the permission in the scope query parameter. Identity platform supports several well-defined [OpenID Connect scopes](https://learn.microsoft.com/en-us/azure/active-directory/develop/v2-permissions-and-consent#openid-connect-scopes) and resource-based permissions (each permission is indicated by appending the permission value to the resource's identifier or application ID URI). For example, the permission string https://graph.microsoft.com/Calendars.Read is used to request permission to read users calendars in Microsoft Graph.
* Registering an application with the Microsoft identity platform in Azure" refers to the process of creating an entry for your application within Azure Active Directory (Azure AD). This registration allows your application to integrate with Azure AD for authentication and authorization purposes.
* When you register an application, you provide Azure AD with information such as the application's name, type (e.g., Web app, Native app, API), redirect URIs (for web apps), client IDs, and optionally client secrets or certificates for authentication. Azure AD then assigns a unique identifier (client ID) to your application and may provide additional configuration settings depending on your application type.
* Once the application is registered, it can use Azure AD for user authentication and to access Microsoft Graph API or other secured resources. This registration process essentially establishes trust between your application and Azure AD, enabling secure communication and access control.
* **Main Topic:**
  + So if I want to build a application and wanted to use Microsoft Entra ID from authentication, the first thing I need to do is to register my app with Microsoft Entra ID by providing application name and redirect URL. Once app is register then while coding the application I need to initilize the application (there are two types First is **PublicClientApplication** and second is **ConfidencialClientApplication**) so for initilize application I need to use IPublicClientApplication and IConfidentialClientApplication which is fetched from Microsoft.Identity.Client NuGet Package.
  + Code used to initilize public Client Application:
    - ClientID is what we get back once we register out application to Entra ID.
  + Once user is authenticated using their account from Entra ID, this is how we can get token.
    - 
* Create and implement shared access signatures.
  1. 
  2. A shared access signature (SAS) is a URI that grants restricted access rights to Azure Storage resources. You can provide a shared access signature to clients that you want to grant delegate access to certain storage account resources.
  3. There are three types of SAS in azure.
     1. **User Delegated SAS**: secure by microsoft entra ID and can only access Blob storage.
     2. **Service SAS**: Secured by Account Key and can access Blob, queue, table and files.
     3. **Account SAS**: Secured by Account Key and can access one or more storage account and everything inside it.
  4. So, now If you want to use SAS you need two things:
     1. **Resource URL**
     2. **SAS Token:** Which was generated by developer. For Example
        + **SAS token:** sp=r&st=2020-01-20T11:42:32Z&se=2020-01-20T19:42:32Z&spr=https&sv=2019-02-02&sr=b&sig=SrW1HZ5Nb6MbRzTbXCaPm%2BJiSEn15tC91Y4umMPwVZs%3D
          - Sp(Specific Permission) (r=read, a=add, c=create, d=delete)
          - St (Start TimeStamp)
          - Se (End Timestamp)
          - Sr (Kind of storage) (b=blob)
* Implement solutions that interact with Microsoft Graph.
* The Microsoft Authentication Library (MSAL) can be used to provide secure access to **Microsoft Graph**, other Microsoft APIs, third-party web APIs, or your own web API.
* Microsoft Graph is like a virtual bridge that connects different Microsoft services, like Office 365, Azure, Outlook, Teams, and more. It lets them work together and share information easily.
  1. Imagine you have your calendar in Outlook and files stored in OneDrive. With Microsoft Graph, you can create an app that accesses both your calendar and files without needing separate logins for each service.
* Microsoft Graph is the gateway to data and intelligence in Microsoft 365. It provides a unified programmability model that you can use to access the tremendous amount of data in Microsoft 365, Windows 10, and Enterprise Mobility + Security.
  1. I can access this Microsoft 365 throught Microsoft graph REST API or Webhook API.
     1. REST API: I need to create a URL like this:
        + 

#### **Implement secure Azure solutions**

* Secure app configuration data by using App Configuration or Azure Key Vault
  1. **Azure Key Vault** is a cloud service for securely storing and accessing secrets. A secret is anything that you want to tightly control access to, such as API keys, passwords, certificates, or cryptographic keys.
* To do any operations with Key Vault, you first need to authenticate to it. There are three ways to authenticate to Key Vault:
* **Managed identities for Azure resources**: When you deploy an app on a virtual machine in Azure, you can assign an identity to your virtual machine that has access to Key Vault. You can also assign identities to other Azure resources. The benefit of this approach is that the app or service isn't managing the rotation of the first secret. Azure automatically rotates the service principal client secret associated with the identity. We recommend this approach as a best practice.
* **Service principal and certificate**: You can use a service principal and an associated certificate that has access to Key Vault. We don't recommend this approach because the application owner or developer must rotate the certificate.
* **Service principal and secret**: Although you can use a service principal and a secret to authenticate to Key Vault, we don't recommend it. It's hard to automatically rotate the bootstrap secret that's used to authenticate to Key Vault.
* Develop code that uses keys, secrets, and certificates stored in Azure Key Vault
* Implement Managed Identities for Azure resources.
  1. System-assigned managed identity is automatically created and managed by Azure for a specific Azure resource, such as an Azure Function App. This means that when you enable system-assigned managed identity for an Azure resource, Azure automatically creates an identity and manages its lifecycle along with the associated resource.
  2. On the other hand, user-assigned managed identity is created and managed separately from Azure resources. You can create a user-assigned managed identity as an independent Azure resource and then assign it to one or more Azure resources, such as Azure Virtual Machines or Azure App Services. This allows you to use the same identity across multiple Azure resources, promoting reusability and central management.
  3. In simple terms, system-assigned managed identity is tied directly to a specific Azure resource and managed by Azure, while user-assigned managed identity is a standalone identity that can be assigned to multiple Azure resources as needed.
  4. A client application can request managed identities for Azure resources app-only access token for accessing a given resource. The token is based on the managed identities for Azure resources service principal. The recommended method is to use the DefaultAzureCredential.

### **Monitor, troubleshoot, and optimize Azure solutions (15–20%)**

#### **Implement caching for solutions**

* Configure cache and expiration policies for Azure Cache for Redis
* Implement secure and optimized application cache patterns including data sizing, connections, encryption, and expiration
* Implement Azure Content Delivery Network endpoints and profiles

#### **Troubleshoot solutions by using Application Insights**

* Configure an app or service to use Application Insights
* Monitor and analyze metrics, logs, and traces
* Implement Application Insights web tests and alerts

### **Connect to and consume Azure services and third-party services (15–20%)**

#### **Implement API Management**

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

* Create an Azure API Management instance:
  1. Azure API is made up of 3 main components:
     1. API gateway:
        + This is act as middle men between api request and back end logic.
        + Accepts API calls and routes them to appropriate backends
        + Verifies API keys and other credentials presented with requests
        + Enforces usage quotas and rate limits
        + Transforms requests and responses specified in policy statements
        + Caches responses to improve response latency and minimize the load on backend services
        + Emits logs, metrics, and traces for monitoring, reporting, and troubleshooting
     2. The **management plane** is the administrative interface where you set up your API program. Use it to:
        + Provision and configure API Management service settings
        + Define or import API schema
        + Package APIs into products
        + Set up policies like quotas or transformations on the APIs
        + Get insights from analytics
        + Manage users
     3. The **Developer portal** is an automatically generated, fully customizable website with the documentation of your APIs. Using the developer portal, developers can:
        + Read API documentation
        + Call an API via the interactive console
        + Create an account and subscribe to get API keys
        + Access analytics on their own usage
        + Download API definitions
        + Manage API keys
* Create and document APIs
* Configure access to APIs.
* Applications must include a valid key in all HTTP requests when they make calls to API endpoints that are protected by a subscription. Keys can be passed in the request header, or as a query string in the URL.
* The default header name is **Ocp-Apim-Subscription-Key**, and the default query string is **subscription-key**.
* Client certificates are signed to ensure that they are not tampered with. When a partner sends you a certificate, verify that it comes from them and not an imposter. There are two common ways to verify a certificate:
  1. Check who issued the certificate. If the issuer was a certificate authority that you trust, you can use the certificate. You can configure the trusted certificate authorities in the Azure portal to automate this process.
  2. If the certificate is issued by the partner, verify that it came from them. For example, if they deliver the certificate in person, you can be sure of its authenticity. These are known as self-signed certificates.
* Implement policies for APIs
* Policies are a collection of statements that are executed sequentially on the request or response of an API. Popular statements include format conversion from XML to JSON and call rate limiting to restrict the number of incoming calls from a developer, and many other policies are available.
* Policy expressions can be used as attribute values or text values in any of the API Management policies, unless the policy specifies otherwise. Some policies such as the Control flow and Set variable policies are based on policy expressions.
* Policies can be applied at different scopes, depending on your needs: global (all APIs), a product, a specific API, or an API operation.
* he policy definition is a simple XML document that describes a sequence of inbound and outbound statements. The XML can be edited directly in the definition window.
* The configuration is divided into inbound, backend, outbound, and on-error. The series of specified policy statements is executed in order for a request and a response.

#### **Develop event-based solutions**

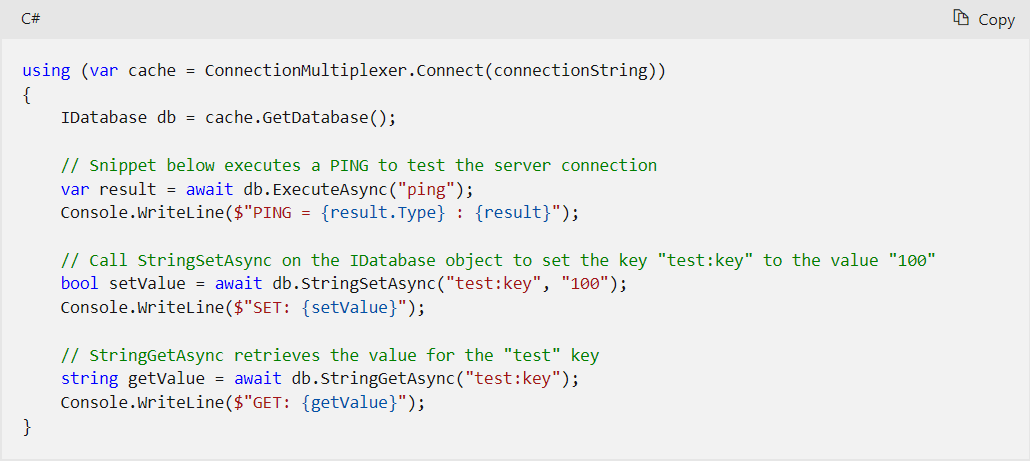
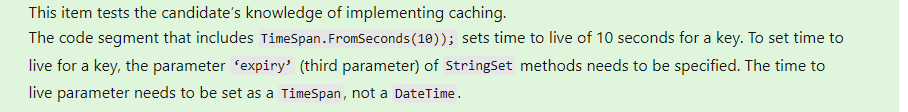
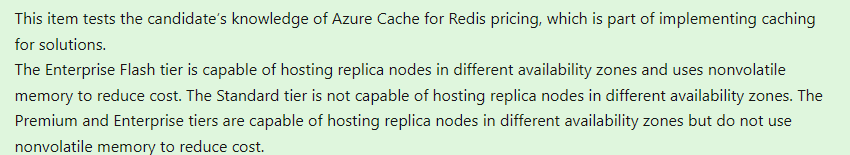
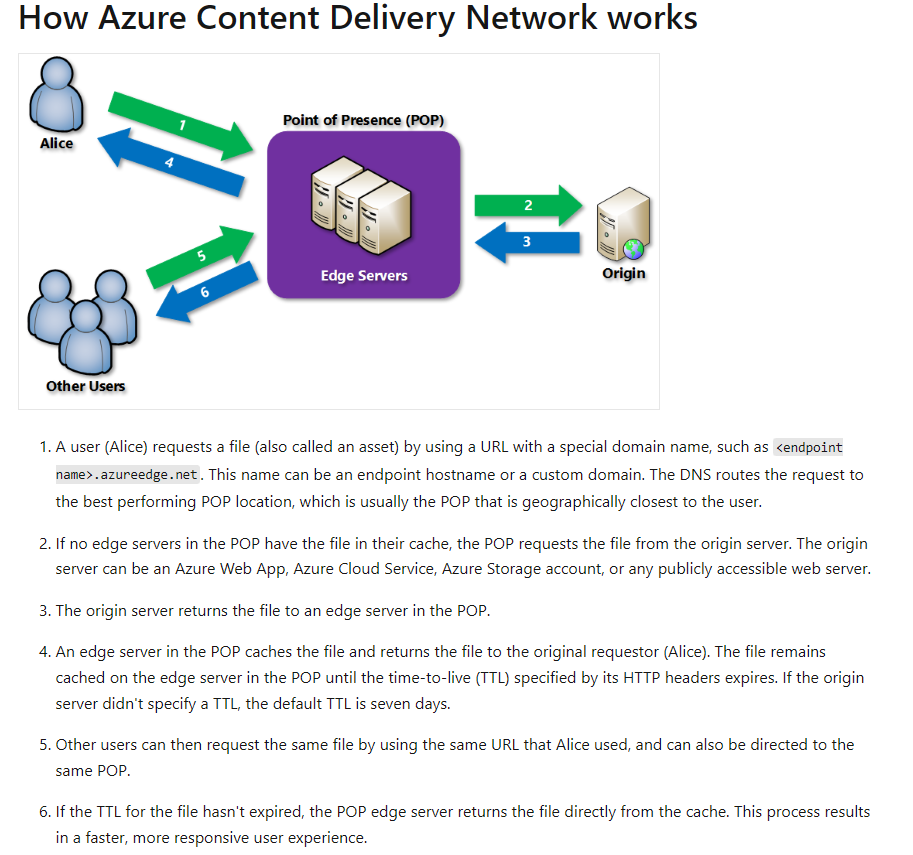
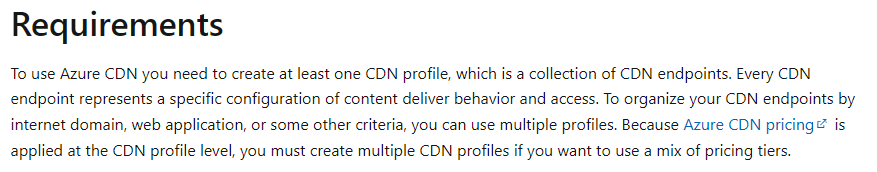
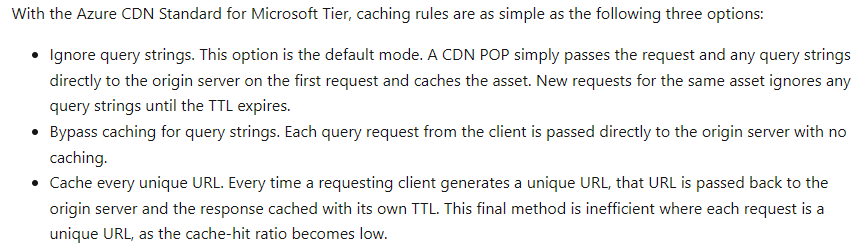
* Implement solutions that use Azure Event Grid
* Concepts in Event Grid:
  1. **Event:** An event is the smallest amount of information that fully describes something that happened in the system. Every event has common information like: source of the event, time the event took place, and unique identifier.
  2. **Event Source**: An event source is where the event happens.
  3. **Topic**: The Event Grid topic provides an endpoint where the source sends events. The publisher creates the Event Grid topic, and decides whether an event source needs one topic or more than one topic. A topic is used for a collection of related events.
  4. **Event Subscription**: A subscription tells Event Grid which events on a topic you're interested in receiving. When creating the subscription, you provide an endpoint for handling the event. You can filter the events that are sent to the endpoint. You can filter by event type, or subject pattern.
  5. **Event Handler**: From an Event Grid perspective, an event handler is the place where the event is sent. The handler takes some further action to process the event.
* Event JSON:
  1. I.e. {  
      "id": "7d6e05cf-b1a0-44c3-b0f5-ab76bdeea6ca",  
      "subject": "New Employee: Alexandre Doyon",  
      "data": {  
      "FullName": "Alexandre Doyon",  
      "Address": "456 College Street, Bow, WA 98107"  
      },  
      "eventType": "Employees.Registration.New",  
      "dataVersion": "1.0",  
      "metadataVersion": "1",  
      "eventTime": "2024-03-21T01:07:40.7171763Z",  
      "topic": "/subscriptions/808cfbe9-b20b-42fb-8f10-aa9705840679/resourceGroups/P02-Harshil-Alert\_group/providers/Microsoft.EventGrid/topics/hrtopicHarshil"  
     }
  2. Required Key value pair: Subject, EventTyle, eventTime, ID
* Implement solutions that use Azure Event Hub
* **Event Hubs**: Scalable event ingestion service for real-time data processing in Azure.
* **Event Hub Namespace**: Logical container for organizing and managing Event Hubs in Azure.
* **Event Hubs Consumers**: Applications or services that process event data from Event Hubs.
* **Partitions**: Units of scalability and parallelism for storing and processing event data in Event Hubs.
* **Event Publishers**: Sources that generate and send events to Event Hubs for ingestion.
* **Consumer Groups**: Logical groups enabling independent consumption and tracking of event streams.
* **Capture**: Feature to automatically store streaming data from Event Hubs into Azure storage.
* **Azure Monitor Integration**: Integration enabling monitoring and diagnostics for Event Hubs in Azure.

#### **Develop message-based solutions**

* Implement solutions that use Azure Service Bus
* Service Bus offers two message receive modes: **Receive and delete**, or **Peek lock**.
  1. Receive and delete mode marks the message as consumed once it's received, returning it to the application. This simple model is suitable for applications that can handle missing messages in case of failure.
  2. Peek lock mode is a two-stage process. First, the message is locked to prevent other consumers from receiving it, then it's returned to the application. After processing, the application completes the second stage to mark the message as consumed. If the application fails to process the message, it can be unlocked and made available again.
* **Topics and Subscriptions**:
  1. Unlike queues, topics and subscriptions allow one message to be processed by multiple consumers in a publish and subscribe pattern. This is useful for distributing messages to a large audience. Subscriptions can filter messages based on specific criteria.
  2. Publishers send messages to a topic, and subscribers receive copies of those messages based on subscription filters. Creating topics and subscriptions is similar to queues, and messages can be sent and received using various programming languages.
* Messages in Service Bus carry both payload and metadata. Metadata, presented as key-value pairs, describes the payload and provides handling instructions for Service Bus and applications. Sometimes, the metadata alone is enough to convey the sender's information to the receivers, leaving the payload empty.
* **Routing Patterns**:
  1. Simple request/reply: Publisher sends a message into a queue, expects a reply, and owns a queue to receive replies.
  2. Multicast request/reply: Publisher sends a message into a topic, and multiple subscribers may respond.
  3. Multiplexing: Enables multiplexing of streams of related messages through a single queue or subscription based on SessionId values.
  4. Multiplexed request/reply: Allows several publishers to share a reply queue, facilitating multiplexed replies.
* Implement solutions that use Azure Queue Storage queues.
* Message size is upto 64kb.

### **Monitor, troubleshoot, and optimize Azure solutions (15–20%)**

#### **Implement caching for solutions**

* Configure cache and expiration policies for Azure Cache for Redis.
  1. Caching is a common technique that aims to improve the performance and scalability of a system. It does this by temporarily copying frequently accessed data to fast storage that's located close to the application. If this fast data storage is located closer to the application than the original source, then caching can significantly improve response times for client applications by serving data more quickly.
  2. Redis has a command-line tool for interacting with an Azure Cache for Redis as a client. The tool is available for Windows platforms by downloading the [Redis command-line tools for Windows](https://github.com/MSOpenTech/redis/releases/)
* Expiration Policies:
  1. Caching is important because it allows us to store commonly used values in memory. However, we also need a way to expire values when they're stale. In Redis this is done by applying a time to live (TTL) to a key.
  2. When the TTL elapses, the key is automatically deleted, exactly as if the DEL command were issued. Here are some notes on TTL expirations.
     + Expirations can be set using seconds or milliseconds precision.
     + The expire time resolution is always 1 millisecond.
     + Information about expires are replicated and persisted on disk, the time virtually passes when your Redis server remains stopped (this means that Redis saves the date when a key expires).
       - set counter 100 OK > expire counter 5 (integer) 1 > get counter 100 ... wait ... > get counter (nil).
* Implement secure and optimized application cache patterns including data sizing, connections, encryption, and expiration
  1. Connection and Encryption:
     1. Recall that we use the host address, port number, and an access key to connect to a Redis server. Azure also offers a connection string for some Redis clients that bundles this data together into a single string. It looks something like the following (with the cache-name and password-here fields filled in with real values):
     2. [cache-name].redis.cache.windows.net:6380,password=[password-here],ssl=True,abortConnect=False
     3. You can pass this string to **StackExchange.Redis** to create a connection the server.
     4. Notice that there are two more parameters at the end:
        + **ssl** - ensures that communication is encrypted.
        + **abortConnection** - allows a connection to be created even if the server is unavailable at that moment.
  2. 
  3. 
  4. 
  5. 
* Implement Azure Content Delivery Network endpoints and profiles.
  1. A content delivery network (CDN) is a distributed network of servers that can efficiently deliver web content to users. CDNs store cached content on edge servers in point-of-presence (POP) locations that are close to end users, to minimize latency.
  2. Azure Content Delivery Network (CDN) offers developers a global solution for rapidly delivering high-bandwidth content to users by caching their content at strategically placed physical nodes across the world.
  3. 
  4. 
  5. 

#### **Troubleshoot solutions by using Application Insights**

* Configure an app or service to use Application Insights.
  1. Application Insights log-based metrics let you analyze the health of your monitored apps, create powerful dashboards, and configure alerts. There are two kinds of metrics:
     1. **Log-based metrics** behind the scene are translated into [Kusto queries](https://learn.microsoft.com/en-us/azure/kusto/query/) from stored events.
        + Developers can use the SDK to send events manually by writing code, or they can let the system automatically collect events. All these events are stored as logs in the Application Insights backend. The Application Insights tool in the Azure portal helps analyze and diagnose issues using these logs.
        + Keeping a record of all events in logs is very helpful for analysis and diagnosis. For instance, you can find out how many times a specific URL was requested and by how many different users. You can also see detailed information like errors and actions taken during a user's session. This information helps improve understanding of how an application is used and its health.
        + However, it might not be practical for applications that generate a lot of data to store all events. In such cases, Application Insights uses methods like sampling and filtering to reduce the number of events stored. Unfortunately, this can make the metrics less accurate because they rely on all events being available for analysis.
     2. **Standard metrics** are stored as pre-aggregated time series.
* Monitor and analyze metrics, logs, and traces
* Implement Application Insights web tests and alerts
  1. You can create up to 100 availability tests per Application Insights resource, and there are three types of availability tests:
     1. [URL ping test (classic)](https://learn.microsoft.com/en-us/azure/azure-monitor/app/monitor-web-app-availability): You can create this test through the portal to validate whether an endpoint is responding and measure performance associated with that response. You can also set custom success criteria coupled with more advanced features, like parsing dependent requests and allowing for retries.
     2. [Standard test (Preview)](https://learn.microsoft.com/en-us/azure/azure-monitor/app/availability-standard-tests): This single request test is similar to the URL ping test. It includes SSL certificate validity, proactive lifetime check, HTTP request verb (for example GET, HEAD, or POST), custom headers, and custom data associated with your HTTP request.
     3. [Custom TrackAvailability test](https://learn.microsoft.com/en-us/azure/azure-monitor/app/availability-azure-functions): If you decide to create a custom application to run availability tests, you can use the [TrackAvailability()](https://learn.microsoft.com/en-us/dotnet/api/microsoft.applicationinsights.telemetryclient.trackavailability) method to send the results to Application Insights.

# **Azure CLI Commands**

* **Az group list**: The az group list command in the Azure Command-Line Interface (CLI) is used to list all resource groups in a subscription.
  + --query: used to only get selected item from resource group object.
    - i.e. az group list –query “[].{id:name}” Only get ID from all resource group.
      * "[].{id:name}": This JMESPath query expression specifies the transformation to be applied to the output. Let's break it down:
      * []: This indicates that we are working with an array.
      * {id:name}: This is an object projection. It creates a new object for each item in the array with a property named "id" that contains the value of the "name" property from the original object.
* **az webapp up -g $resourceGroup -n $appName –html**
  + The az webapp up command in the Azure Command-Line Interface (CLI) is used to create and deploy a basic web application to Azure App Service.
* Create a new Resource Group PowerShell:
  + New-AzResourceGroup -Name RG01 -Location "South Central US"
* Createa new Resource Group CLI:
  + az group create -l westus -n MyResourceGroup
* Create a new app service plan PowerShell:
  + New-AzAppServicePlan -ResourceGroupName "Default-Web-WestUS" -Name "ContosoASP" -Location "West US" -Tier "Basic" -NumberofWorkers 2 -WorkerSize "Small"
* Create a new app service plan CLI:
  + az appservice plan create -g MyResourceGroup -n MyPlan
* Create **Service Bus Namespace**:
  + az servicebus namespace create \
  + --resource-group az204-svcbus-rg \
  + --name $myNameSpaceName \
  + --location $myLocation
* Create **Service Bus Queue**:
  + az servicebus queue create --resource-group az204-svcbus-rg \
  + --namespace-name $myNameSpaceName \
  + --name az204-queue
* Create **Redis Instance**:
  + 

## Deploy resource using **ARM template**.

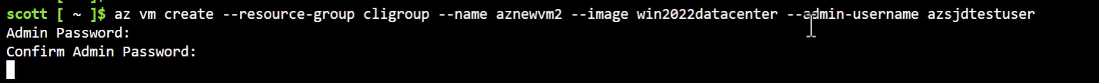
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"

$location = Read-Host -Prompt "Enter the location (i.e. centralus)"

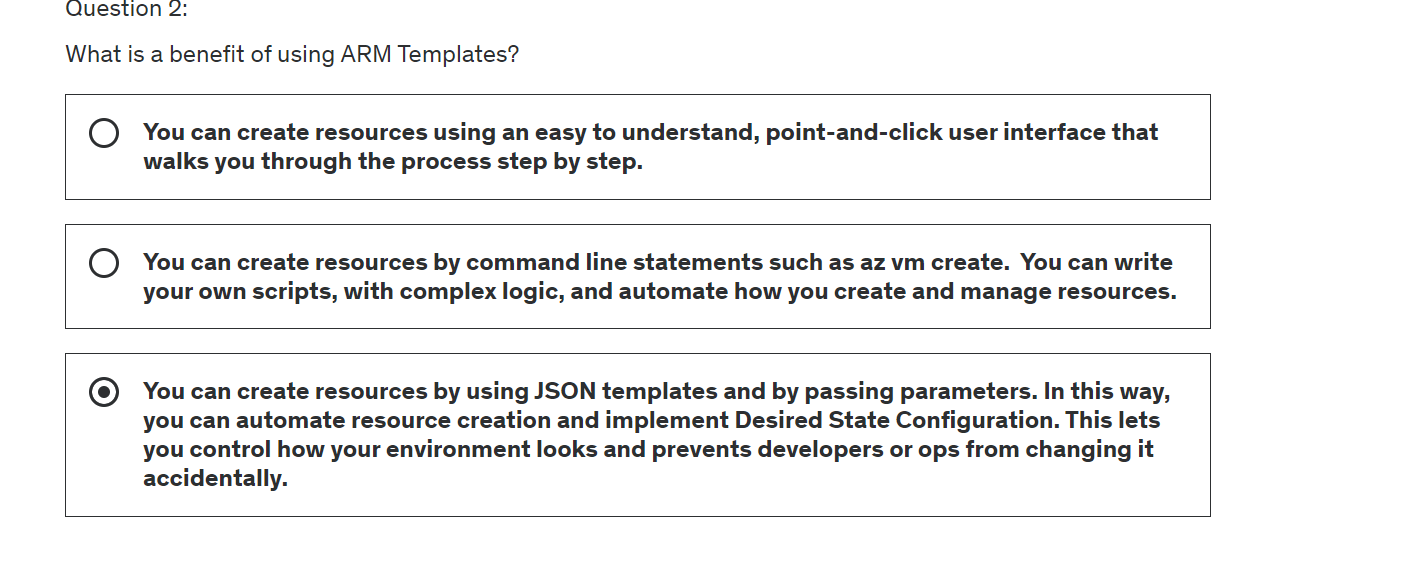
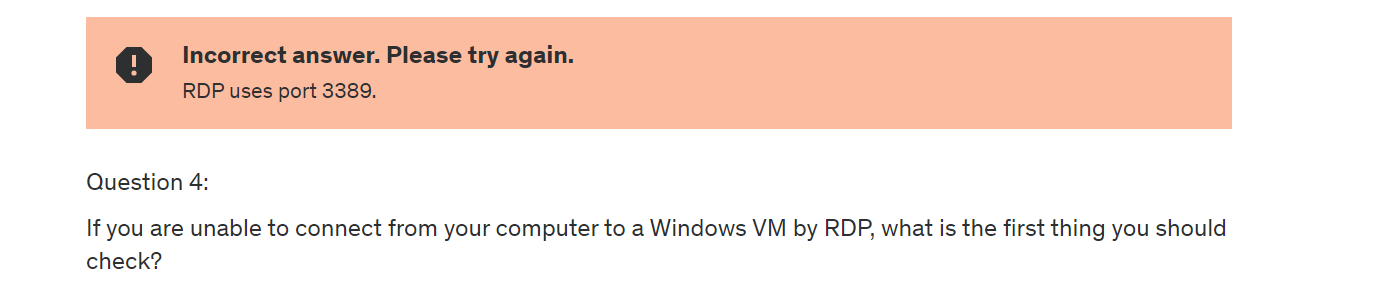
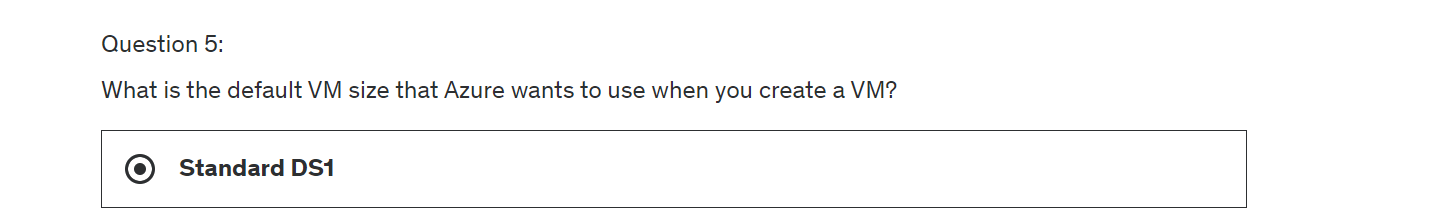
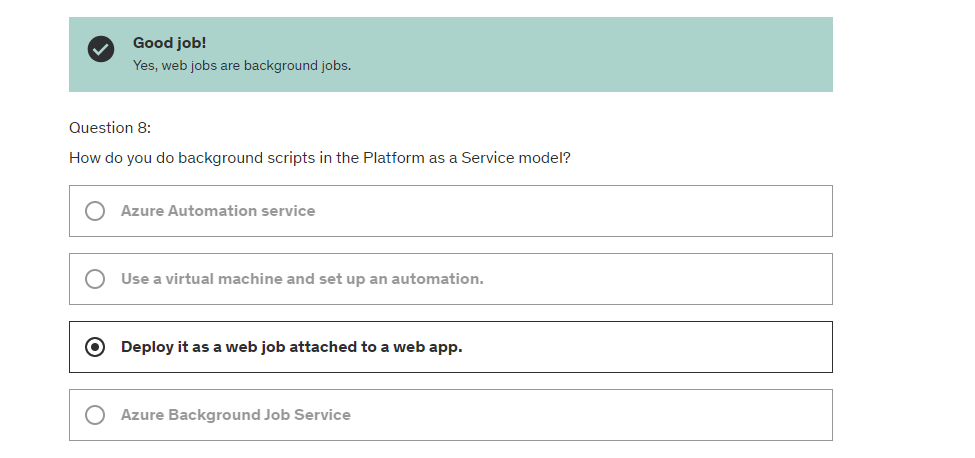
$templateUri = "https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/quickstarts/microsoft.storage/storage-account-create/azuredeploy.json"

**New-AzResourceGroupDeployment -ResourceGroupName $resourceGroupName -TemplateUri $templateUri -Location $location**

In BASH:



Incorrects:

1. The Azure Functions and Web Jobs Tools extension is included in Visual Studio and it does not require installation.
2. You plan on setting up continuous deployment to maintain source control on your functions code. Which of the following deployment sources are a viable option in Azure?
   1. OneDrive
   2. Bitbucket
   3. GitHub
3. You are developing an application named Application1 which uses an Azure Function. The source code for Application1 is in a GitHub repository under the master branch. You need to create your function and configure it to deploy code from a GitHub repository.
   1. az functionapp create --name $functionAppName \
   2. --storage-account $storageName \
   3. --consumption-plan-location $region \
   4. --resource-group myResourceGroup \
   5. --deployment-source-url $gitrepo \
   6. --deployment-source-branch master \
   7. --functions-version 2
4. 
5. 
6. 
7. 

Practice this:

1. <https://learn.microsoft.com/en-gb/azure/app-service/tutorial-dotnetcore-sqldb-app?pivots=platform-linux>
2. <https://github.com/MicrosoftLearning/AZ-204-DevelopingSolutionsforMicrosoftAzure>
3. <https://learn.microsoft.com/en-us/training/modules/create-serverless-logic-with-azure-functions/3-create-an-azure-functions-app-in-the-azure-portal?pivots=javascript>

Read:

* <https://learn.microsoft.com/en-us/azure/storage/common/storage-redundancy>

Pratice this today:

1. <https://learn.microsoft.com/en-us/training/modules/implement-managed-identities/5-acquire-access-token>
2. Develop function app that store connection string locally in json file and publish this function app on portal and store connection string to key vault, configuration and use managed Identity to access connection string.
   1. <https://learn.microsoft.com/en-us/training/modules/implement-azure-app-configuration/2-app-configuration-overview>
3. Pratice customer managed key in app configuration.
   1. <https://learn.microsoft.com/en-us/training/modules/implement-azure-app-configuration/5-secure-app-configuration-data>
4. Explore policy in api management.
   1. <https://learn.microsoft.com/en-us/training/modules/explore-api-management/5-create-advanced-policies>
5. Play with API management service on portal
   1. <https://learn.microsoft.com/en-us/training/modules/explore-api-management/8-exercise-import-api>

Practice Exercise: <https://learn.microsoft.com/en-us/credentials/certifications/azure-developer/practice/assessment?assessment-type=practice&assessmentId=35&practice-assessment-type=certification>