Microsoft Azure: AZ-204 Certification

Cheat Sheet

Quick Bytes for you before the exam!

The information provided in cheat sheet is for educational purposes only; created in our efforts to help aspirants prepare for the Microsoft Azure Exam AZ-204 certification. Though references have been taken from Azure documentation, it's not intended as a substitute for the official docs. The document can be reused, reproduced, and printed in any form; ensure that appropriate sources are credited and required permissions are received.

Are you Ready for Microsoft Azure "AZ-204" Certification?

Self-assess yourself with

Whizlabs FREE TEST

750+ Hands-on-Labs

Hands-on Labs - AWS, GCP, Azure (Whizlabs)

Cloud Sandbox environments

Cloud Sandbox - AWS, Azure, GCP & Power BI

AZ-204 Whizcard Index

Containerized Solutions

Azure Container Registry (ACR): It is a service that allows you to build, store and manage container images and artifacts in a private registry for all types of container deployments. It is a managed registry service based on the open source Docker Registry 2.0.

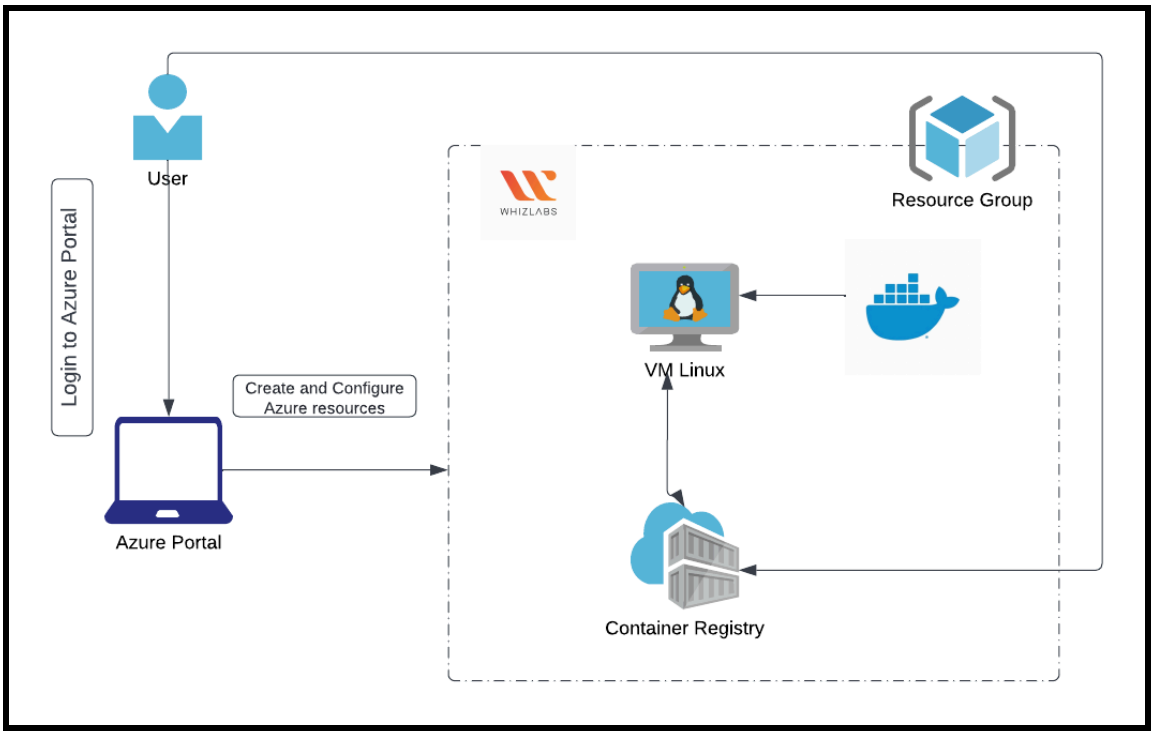
Azure Container Instances (ACI): It provides a fast and simple way to run a container on Azure, without having to manage any virtual machines and adopt a high-level service. It is a great solution for any scenario that can operate in discrete containers, including common applications, task automation, and build jobs.

Azure Container Apps (ACP) : It is a fully managed environment that lets you run microservices and containerized applications on a serverless platform. It lets you run microservices and containerized applications on a serverless platform running on top of Azure Kubernetes Service.

Azure Container Registry

What is Azure Container Registry?

- Docker is an open source containerization platform. It enables developers to package applications into containers—standardized executable components combining application source code with the operating system (OS) libraries and dependencies required to run that code in any environment.
- Containers simplify delivery of distributed applications, and have become increasingly popular as organizations shift to cloud-native development and hybrid multicloud environments.
- The Azure container registry is Microsoft's own hosting platform for Docker images.
- Azure Container Registry is a private registry service for building, storing, and managing container images and related artifacts. In this quickstart, you create an Azure container registry instance with the Azure portal. Then, use Docker commands to push a container image into the registry, and finally pull and run the image from your registry.
- Azure Container Registry is a multi-tenant service, where the data endpoint storage accounts are managed by the registry service. There are many benefits for managed storage, such as load balancing, contentious content splitting, multiple copies for higher concurrent content delivery.



What is Docker?

Docker is a platform designed to simplify the process of developing, deploying, and managing applications by utilizing containerization technology. To Simplift, Docker allows you to package an application and all its dependencies into a single, lightweight unit called a container. These containers can run consistently across various environments, from a developer's laptop to a production server, ensuring that an application behaves predictably and is easily scalable.

What is Docker Image?

Docker images are self-contained, immutable packages designed for efficient application deployment in containerized environments.

Key characteristics include: Portability, Layered Structure, Versioning, Reusability, and Security.

Overall, Docker images revolutionize application packaging and deployment, promoting consistency, scalability, and ease of management in containerized ecosystems.

Whizlabs Hands-on-labs:

- Azure Container Registry (whizlabs.com)
- Build Docker images and learn about container registries (whizlabs.com)

References links:

- Create a service connection and build and publish Docker images to Azure Container Registry - Azure Pipelines | Microsoft Learn
- Build and push Docker images to Azure Container Registry with Docker templates - Azure Pipelines | Microsoft Learn

What is Containerization?

Containerization is a way to package and run software applications, just like bare-metal or virtualized deployments. Containerization provides below features"

- **Isolation:** Containerization helps keep each application separate, like having its own box. This means that one app won't overlap with other application environment even though both of them are on the same hardware system.
- **Consistency:** Containers are like standardized boxes, ensuring that the application works the same way on different computers. This makes it easier for developers to build and test software because they know it will behave the same everywhere.
- **Efficiency:** Containers are lightweight and use resources efficiently. This makes it possible to run many containers on one computer, saving time and money.

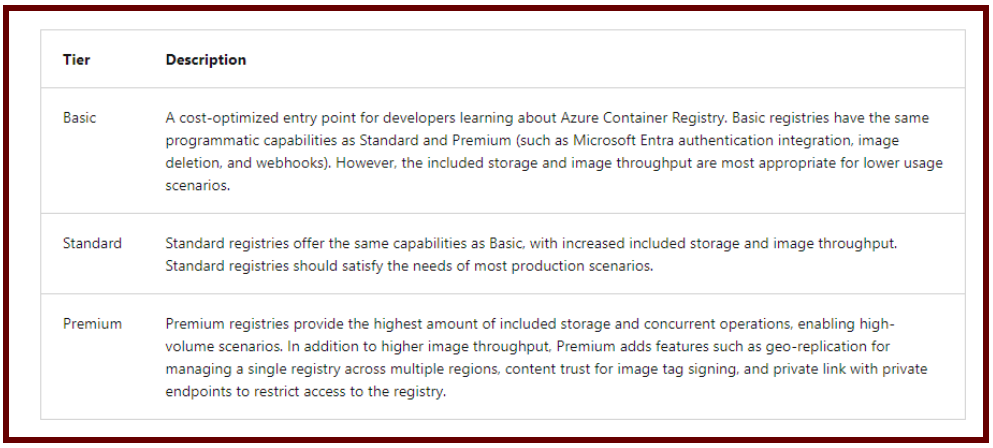
Usage of Azure Container Registry

Use the Azure Container Registry (ACR) service with your existing container development and deployment pipelines, or use Azure Container Registry tasks to create container images in Azure. Build on demand or fully automate builds with triggers like source code commits and base image updates.

Use cases: You can drag images from Azure Container Registry to various deployment targets:

- Scalable orchestration systems
- Azure services

Azure Container Registry is available in multiple service tiers. These tiers offer predictable pricing and multiple options to align to the capacity.



Tier	Description
Basic	A cost-optimized entry point for developers learning about Azure Container Registry. Basic registries have the same programmatic capabilities as Standard and Premium (such as Microsoft Entra authentication integration, image deletion, and webhooks). However, the included storage and image throughput are most appropriate for lower usage scenarios.
Standard	Standard registries offer the same capabilities as Basic, with increased included storage and image throughput. Standard registries should satisfy the needs of most production scenarios.
Premium	Premium registries provide the highest amount of included storage and concurrent operations, enabling high-volume scenarios. In addition to higher image throughput, Premium adds features such as geo-replication for managing a single registry across multiple regions, content trust for image tag signing, and private link with private endpoints to restrict access to the registry.

Source: Discover the Azure Container Registry - Training | Microsoft Learn

By the end of this module, you will be able to:

- Run the Azure Container Registry.
- Build and deploy a container image to an Azure Container instance using Azure Container Registry tasks.
- Replicate a container image to multiple Azure regions.

Whizlabs Hands-on-labs:

- Create a container registry by using a Bicep file (whizlabs.com)
- Create a geo-replicated container registry by using an ARM template (whizlabs.com)
- Build Docker images and learn about container registries (whizlabs.com)
- Artifact Registry vs Container Registry (whizlabs.com)

Create and manage container images for solutions:

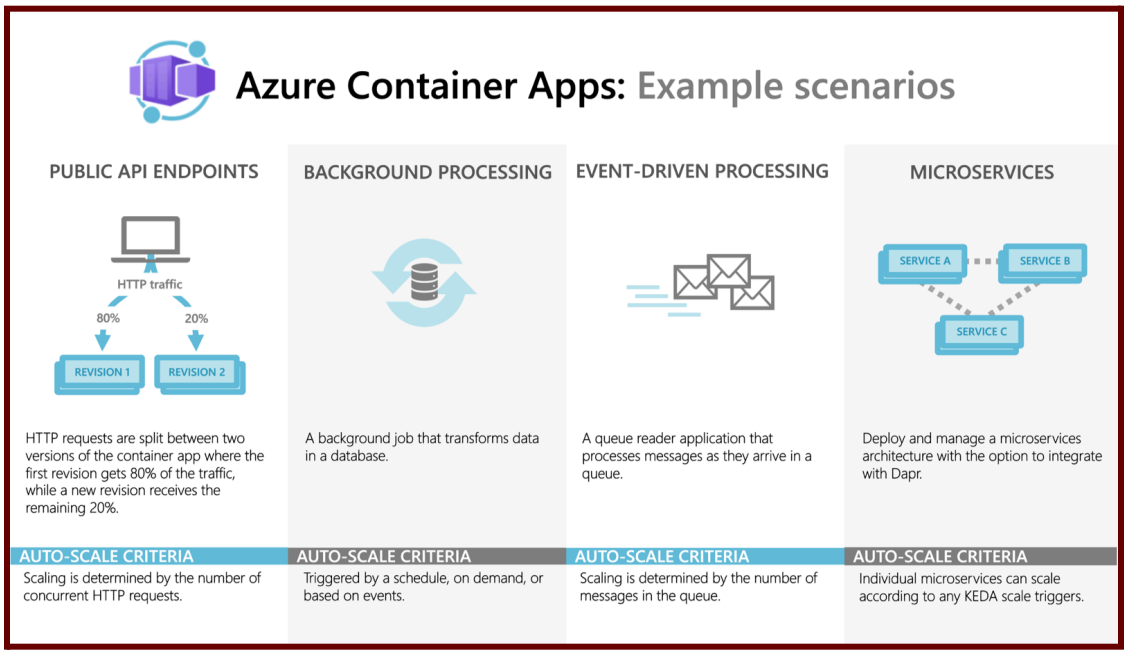
- Exercise: Build and run a container image by using Azure Container Registry Tasks
- Exercise - Deploy an Azure container registry - Training
- Exercise - Deploy images from Azure Container Registry - Training
- Exercise - Replicate a container image to different Azure regions - Training

Azure Container Apps

Azure Container Apps is a serverless platform that allows you to manage less infrastructure and save costs when running containerized applications. Instead of worrying about server configuration, container orchestration, and deployment details, Container Apps provides all the up-to-date server resources you need to keep your applications stable and secure.

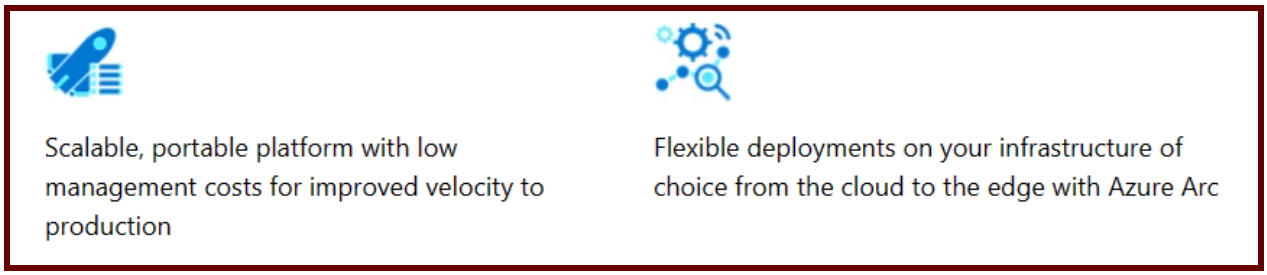
Common uses of Azure Container Apps are as follows:

- Deploying API endpoints
- Hosting background processing jobs
- Handling event-driven processing
- Running microservices



(Source: Azure Container Apps overview | Microsoft Learn)

Azure Container Apps lets you run microservices and containerized applications on a serverless platform. With container apps, you enjoy the benefits of running containers while leaving behind the worries of manually configuring cloud infrastructure and complex container orchestrators.



(Source: Azure Container Apps | Microsoft Azure**)**

Reference Links:

Azure Container Apps | Microsoft Azure
Quickstart: Deploy your first container app using the Azure portal | Microsoft Learn
Quickstart: Deploy your first container app with containerapp up | Microsoft Learn
Comparing Container Apps with other Azure container options | Microsoft Learn

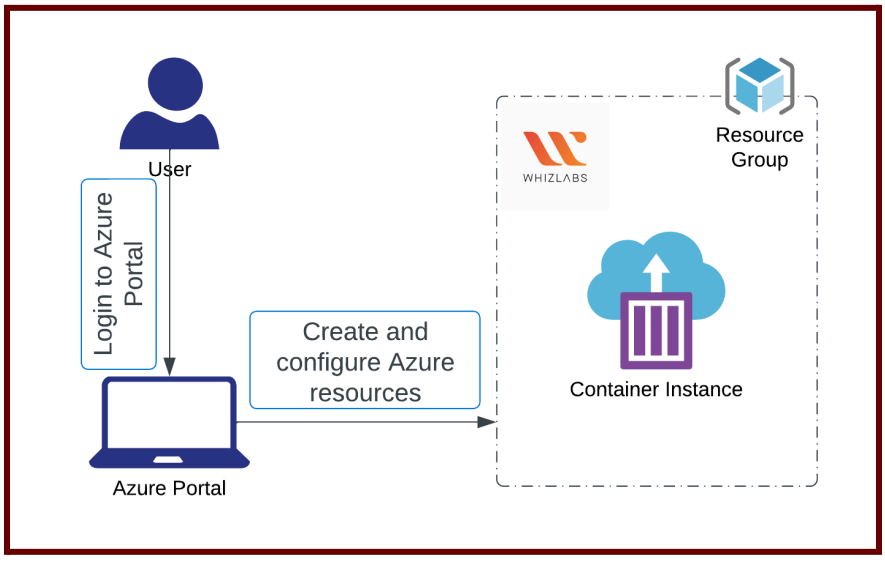
Azure Container Instances (ACI)

Containers are becoming the preferred way to package, deploy and manage cloud applications. Azure Container Instances provide a fast and simple way to run a container on Azure, without having to manage any virtual machines and adopt a high-level service.

Azure Container Instances is a great solution for any scenario that can operate in discrete containers, including common applications, task automation, and build jobs. For scenarios where you need full container orchestration, including service discovery, automatic scaling and coordinated application upgrades across multiple containers, we recommend Azure Kubernetes Service (AKS).

What are Azure Container instances?

- Azure Container Instances are used for Developing apps fast without managing virtual machines.
- By running our workloads in Azure Container Instances (ACI), we can focus on designing and building our applications instead of managing the infrastructure that runs them.
- A Container offers significant startup benefits over a Virtual Machine.
- It also supports executing a command in a running container by providing an interactive shell to help with application development and troubleshooting.
- Azure Container Instances can schedule both Windows and Linux containers with the same API.
- It even supports scheduling of multi-container groups that share a host machine, local network, storage and lifecycle. This enables you to combine your main application container with other supporting role containers.
- You pay based on what you need and get billed by the second, so you can fine-tune your spending based on actual need.



Use Azure Container Instances to run serverless Docker containers on Azure with simplicity and speed. When you don't need a full container orchestration platform like Azure Kubernetes Service, deploy the application to an on-demand container instance.

Whizlabs Hands-on-labs:

Deploy Azure Container Instances (whizlabs.com)
Create an Azure Container Instance with a public IP address using Terraform (whizlabs.com)
Deploying a container instance using Bicep (whizlabs.com)
Deploying a container instance using ARM template (whizlabs.com)

Reference Links:

Serverless containers in Azure - Azure Container Instances
Quickstart - Deploy Docker container to container instance - Portal
Run container images in Azure Container Instances - Training | Microsoft Learn
Run Docker containers with Azure Container Instances - Training | Microsoft Learn

Azure App Service & Web Apps

Implement Azure App Service Web Apps

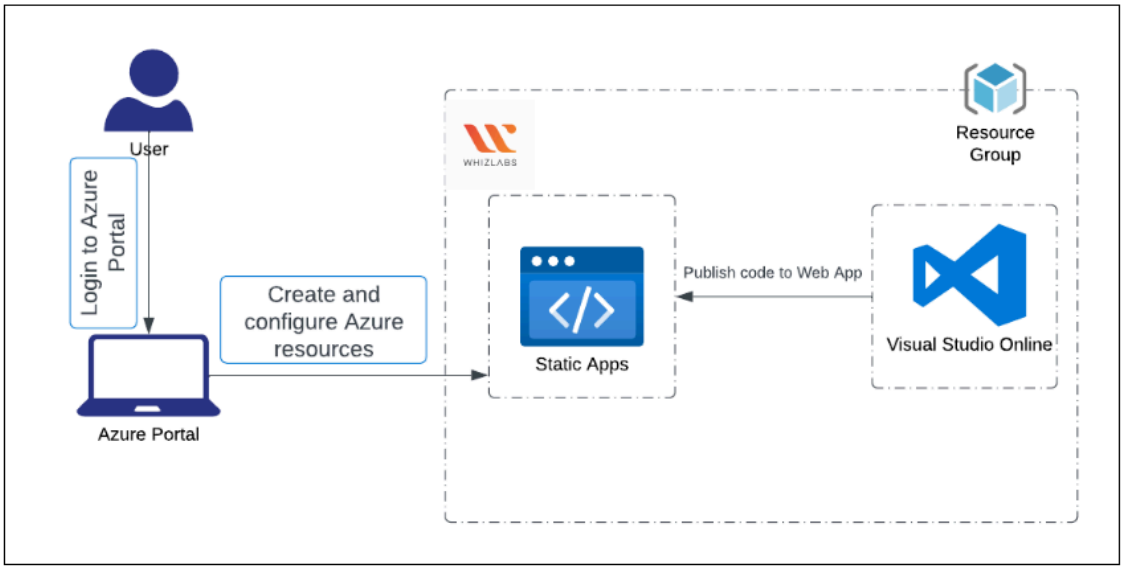
Azure App Service

Azure App Service is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends. You can develop in your favorite programming language or framework. Applications can be easily deployed and scaled in Windows and Linux-based environments.

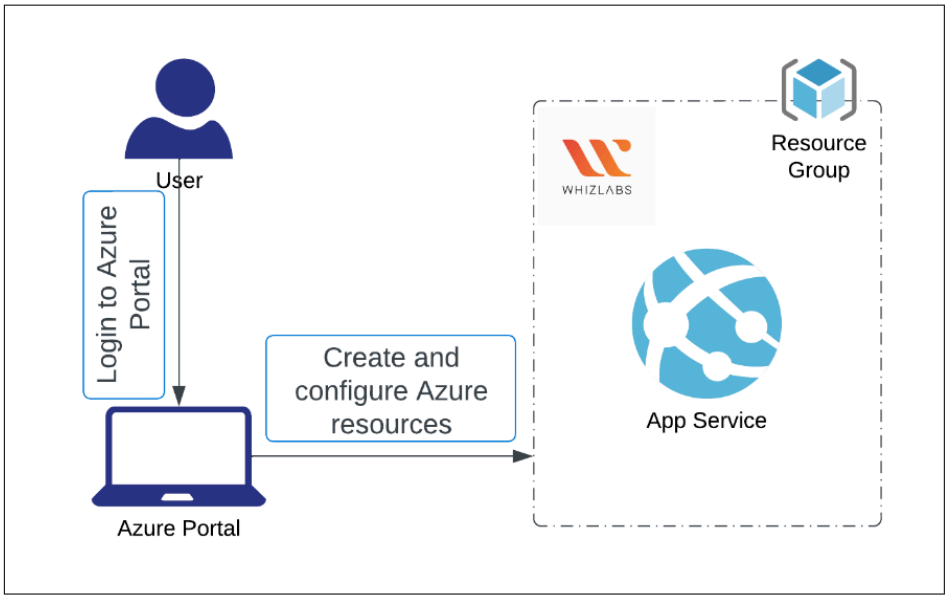
App Service not only adds the power of Microsoft Azure to your application, such as security, load balancing, autoscaling, and automated management. You can also take advantage of its DevOps capabilities, such as continuous deployment, package management, staging environments, custom domain, and TLS/SSL certificates from Azure DevOps, GitHub, Docker Hub, and other sources.

Why Azure app service and Web Apps?

Azure App Service is a platform-as-a-service (PaaS) offered on Microsoft Azure that enables developers to quickly build, deploy, and scale web, mobile, and API applications. Azure Web Apps is a specific type of Azure App Service that focuses on hosting web applications.



Azure Web Apps provides a fully managed platform for building and hosting web applications using popular programming languages such as .NET, Java, Node.js, Python, and PHP. It includes features like automatic scaling, load balancing, traffic management, continuous deployment, and monitoring. Azure App Service can host web apps natively on Linux for supported application stacks.



Key features :

- Security, Compliance, Multiple languages and frameworks
- Managed production environment, Containerization and Docker
- DevOps optimization and Serverless code
- Global scale with high availability, Connections to SaaS platforms and on-premises data
- Authentication and authorization with Application templates
- Visual Studio, API and mobile features

Advantages of Azure App Service

- You can manage the infrastructure and quickly scale up or down to meet demand.
- Azure takes care of all patching, monitoring and other operational tasks.
- You can There is a wide range of built-in DevOps capabilities.

Disadvantages of Azure App Service

- Less control over the infrastructure and it costs more than other hosting plans.
- App Service on Linux is not supported in the shared price range.

Advantages of Azure Web App

- Simple to develop, launch and maintain without having to stress about server administration.
- Your app is available everywhere with the help of Azure's global network,
- Web apps scale up or down as needed to meet demand.
- Due to the usage-based pricing structure, you only pay for what you actually use.
- Work with existing development tools like Visual Studio and GitHub.

Disadvantages of Azure Web App

- When the load is heavy, speed can occasionally be an issue.
- It’s only suitable for applications with small storage requirements.

Azure Web App or App Service, which should you choose?

The service you choose really depends on your wants and needs, as each has its advantages and disadvantages. If we had to choose one, Azure App Service would focus on Azure App Service as it supports more programming languages, development frameworks and features than Azure Web App.

- Azure App Service costs a bit more than Azure Web App. But depending on the additional features and benefits you get with Azure App Service; We think the extra cost is justified.

- Azure Web Apps is a specific type of Azure App Service that focuses primarily on hosting web apps, while Azure App Service is a broader category of PaaS offerings that includes Azure Web Apps and other related services.

Azure App Service plans

➔ **Shared:** The two base tiers, Free and Shared, run the app in the same Azure VM as other App Service apps, including other customers' apps. These arrays assign CPU quotas to each app running on shared resources, and the resources cannot scale.

➔ **Dedicated**: 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 scale, the more VM instances you have available for scale-out.

➔ **Isolated:** The Isolated and IsolatedV2 tiers run dedicated Azure VMs in dedicated Azure virtual networks. It provides network isolation on top of compute isolation for your apps. This provides maximum scale-out capabilities.

Whizlabs Hands-on-labs Links

- Azure app service (whizlabs.com)
- How to create a Web App | Azure Hands-on Lab (whizlabs.com)

References Links:

Implement Azure App Service web apps - Training | Microsoft Learn

1. Diagnostics Logging

When managing a web application, you should be ready for anything that can go wrong, from 500 errors to people informing you that your website is unavailable. You may troubleshoot your app with the intelligent and interactive App Service Diagnostics without having to configure anything. In the event that you have problems with your app, App Service Diagnostics identifies the issues and directs you to the appropriate resources for a quicker and easier troubleshooting process.

App Service apps can be debugged with the help of built-in diagnostics. You may add instrumentation to your application, enable diagnostic logging, and retrieve the data that Azure has logged in this course.

The logging kinds, supporting platforms, and locations for storing and accessing logs are displayed in the following table.

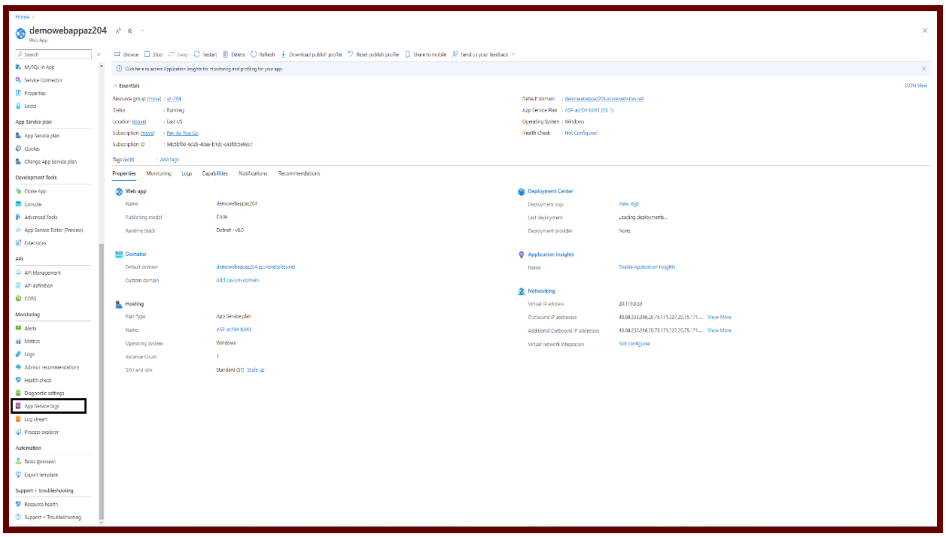
Type	Platform	Location
Application Logging	Windows, Linux	App Service file system and/or Azure Storage blobs
Web server Logging	Windows	App Service file system or Azure Storage blobs
Detailed error Logging	Windows	App Service file system
Failed request tracing	Windows	App Service file system
Deployment Logging	Windows, Linux	App Service file system

Type	Description
Application Logging	Record the messages that your application code produces. The web framework of your choice or your application code itself, utilizing the built-in logging mechanism of your language, produce the messages. One of the following categories—Critical, Error, Warning, Info, Debug, and Trace—is assigned to each message.
Web server Logging	The format of the W3C extended log file contains raw HTTP request data. Data such as the HTTP method, resource URI, client IP, client port, user agent, response code, and so forth are included in every log message.
Detailed error Logging	Copies of the error pages in .html format that were supposed to be forwarded to the client browser. App Service can store the error page each time an application error with HTTP code 400 or higher occurs, however detailed error pages shouldn't be delivered to clients in production for security reasons.
Failed request tracing	Comprehensive tracking data on unsuccessful requests, comprising a trail of the IIS components that processed the request and the duration of each component's processing. For every unsuccessful request, a folder containing the XML log file and the XSL stylesheet to read it with is generated.
Deployment Logging	Aids in figuring out why a deployment went wrong. There are no customizable settings for deployment logging; everything occurs automatically.

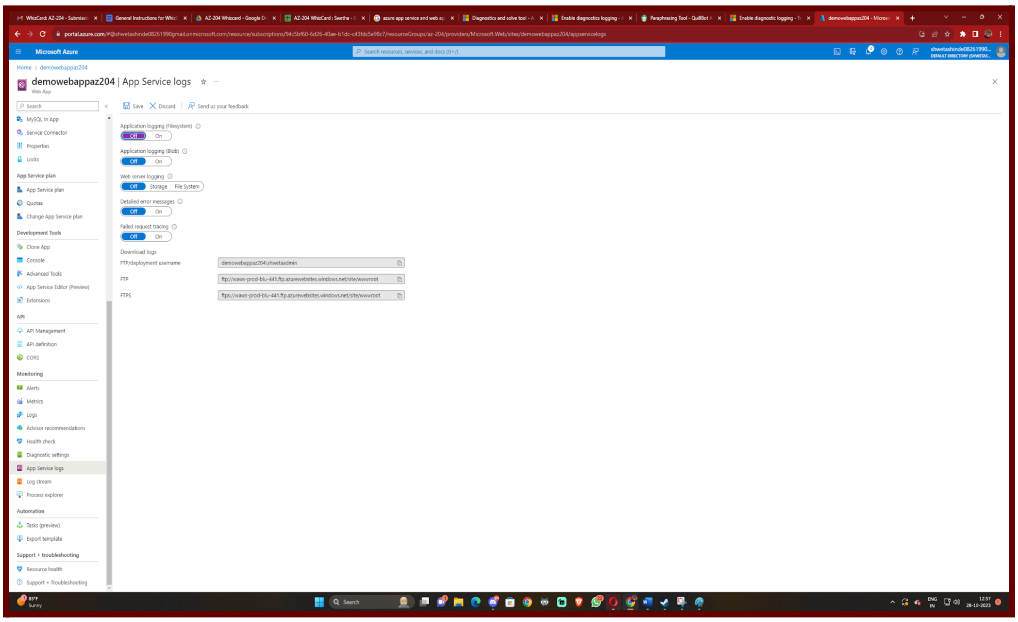
Below are the ways to enable application logging on different platform

1. Enable Application Logging (Windows)

A. Navigate to your app and select App Service logs in the Azure interface to enable application logging for Windows apps.



B. To enable either the Filesystem Application Logging or the Blob Application Logging, or both, select On. The Filesystem option is for short-term debugging and shuts down after 12 hours. For long-term logging, use the Blob option. To write logs to a blob storage container, you'll need one.



C. As indicated in the accompanying table, you can also adjust the Level of details recorded in log.

Level	Categories
Disabled	None
Error	Error, Critical
Warning	Warning, Error, Critical
Information	Info, Warning, Error, Critical
Verbose	All categories: trace, debug, info, warning, error, and critical

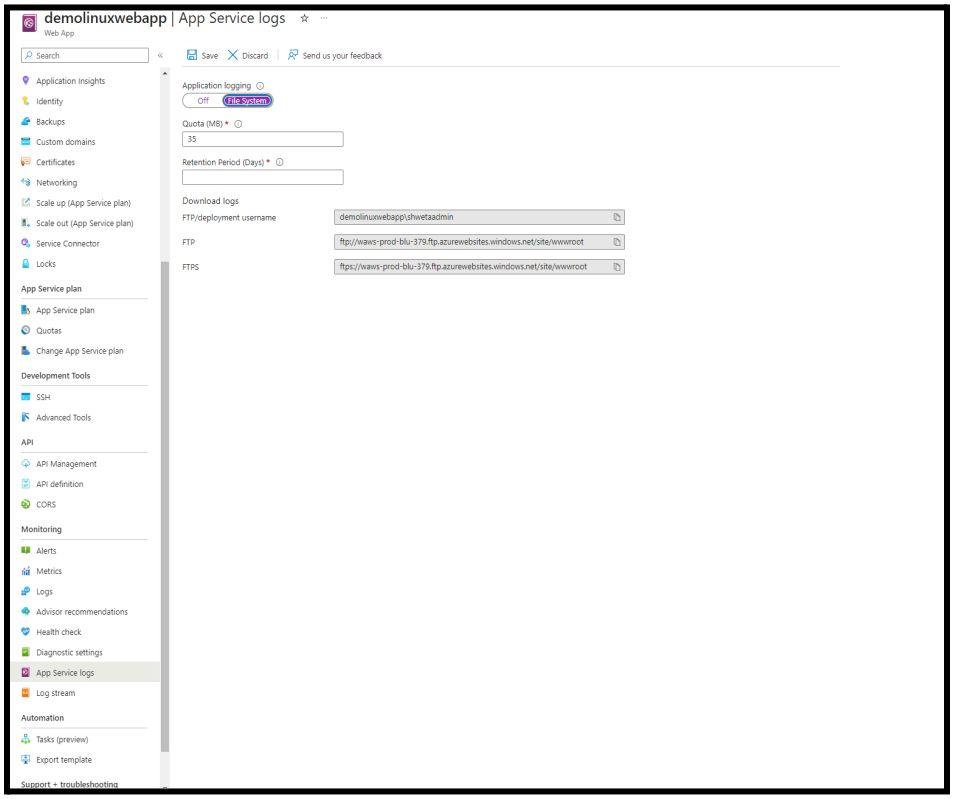
D. After all appropriate selection, click Save.

2. Enable Application Logging (Linux/Container)

A. Set the Application logging option to File System in App Service logs.

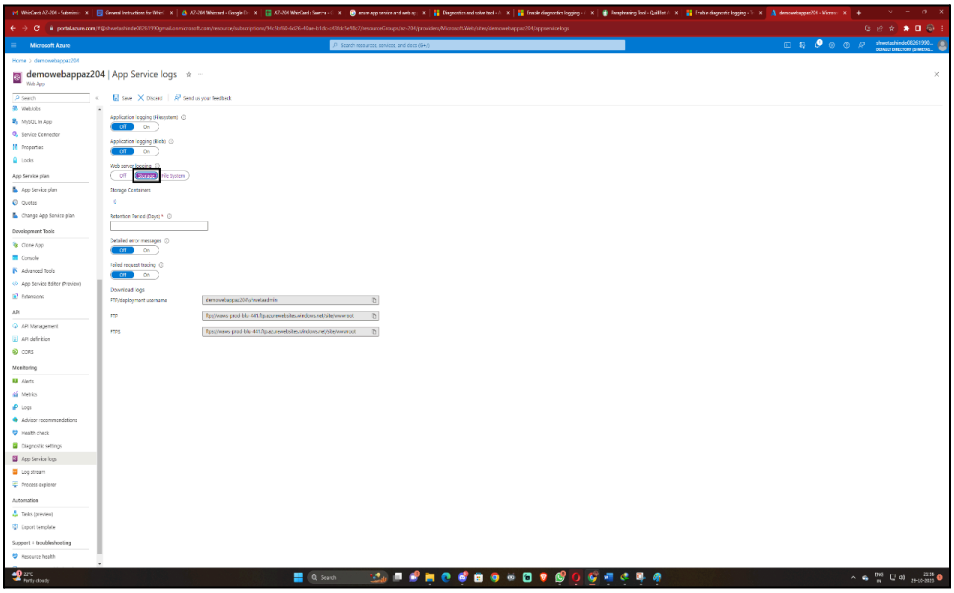
B. Set the disk quota for the application logs in Quota (MB). Enter the number of days that the logs should be kept under Retention Period (Days) as per your application need.

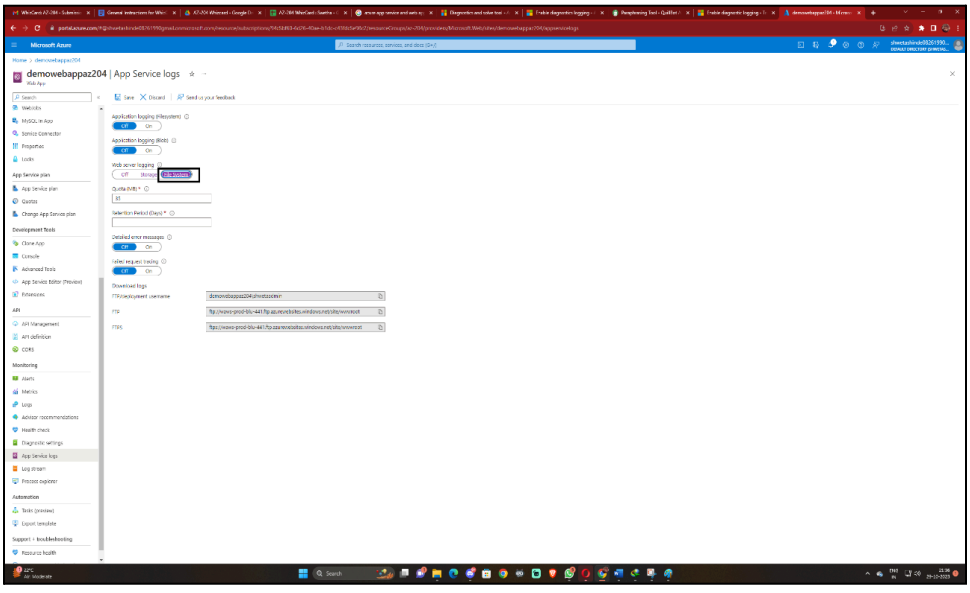
C. After you're done, choose **Save.**



3. Enable Web server Logging

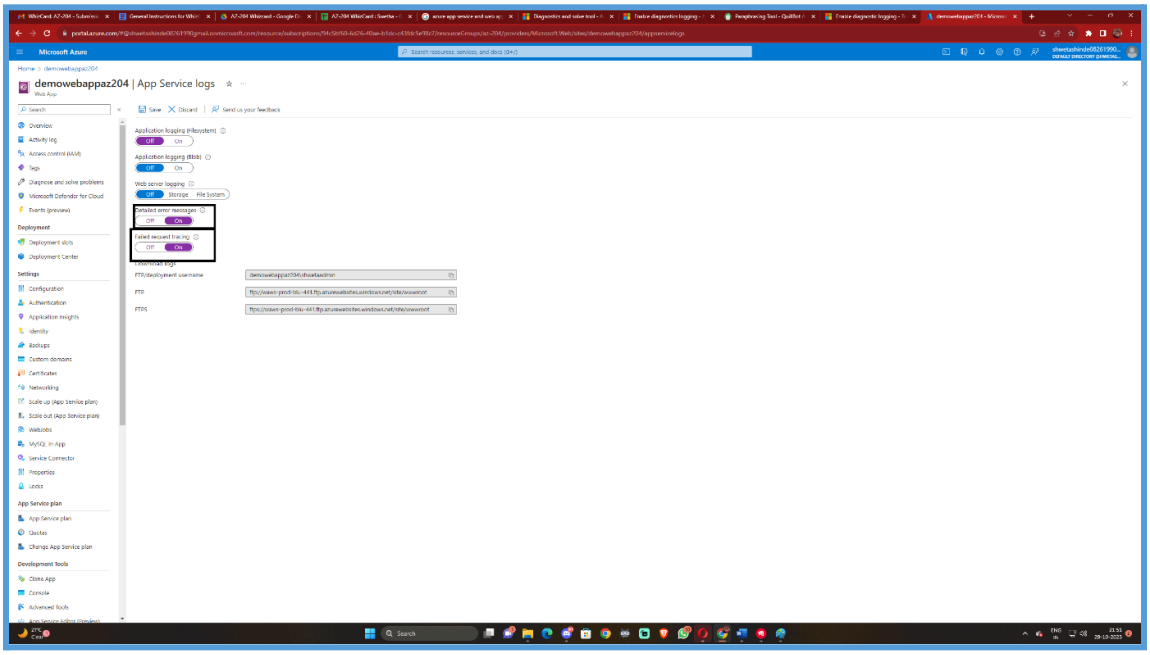
A. To store logs on blob storage for Web server logging, choose Storage; alternatively, choose File System to store logs on the App Service file system.
B. Enter the number of days that the logs should be kept under Retention Period (Days).
C. After you're done, choose Save.





Log Detailed Errors

- Navigate to your app and pick App Service logs in the Azure portal to record the error page or failed request tracing for Windows apps.
- After choosing on under Detailed Error Logging or Failed Request Tracing, choose Save.
- The Failed Request Tracing feature by default captures a log of requests that failed with HTTP status codes between 400 and 600. To specify custom rules, you can override the <traceFailedRequests> section in the web.config file.



Way to Add Log message in Code

To send log messages to the application logs, use the standard logging tools in your application code. For instance:

- ASP.NET Core makes use of Microsoft.Extensions.Logging by default.AzureAppServices is the log source.
- Logs can be sent to the application diagnostics log by Python applications using the OpenCensus library.
- The System. Diagnostics. Trace class allows ASP.NET applications to log information to the application diagnostics log. For instance:e.g., C# code - System. Diagnostics. Trace.TraceError("If you're seeing this, something bad happened");

Stream Logs

Make sure the log type you want is enabled before you start streaming real-time logs. App Service streams any data written to the console output or files with the extensions.txt,.log, or.htm that are kept in the D:\home\LogFiles directory.

Events in the stream may occur out of order as a result of some logging buffer types writing to the log file. For instance, the appropriate HTTP log entry for the page request may appear in the stream before an application log entry that happens when a user accesses a page.

- Azure portal: Go to your app and choose "Log stream" to start streaming logs.
- Azure CLI - Use the following command to stream logs in real time in Cloud Shell:

  az webapp log tail --name appname --resource-group myResourceGroup

- Local console: Install Azure CLI and log in to your account in order to stream logs in the local console. Once logged in, adhere to the Azure CLI instructions.

Access Log Files

A client tool compatible with Azure Storage is required if you set up the Azure Storage blobs option for a log type.

The simplest method for accessing logs kept in the App Service file system is to use the browser to download the ZIP file at:

- **https://<app-name>. scm.azurewebsites.net/api/logs/docker/zip is the URL for Linux and container apps.**
- **<app-name>. scm.azurewebsites.net/api/dump is the URL for Windows apps.**

Console output logs for the docker host and container are included in the ZIP file for Linux/container apps. One set of logs for each instance is contained in the ZIP file for a scaled-out application. These log files are the contents of the /home/Logfiles directory in the App Service file system.

2. Autoscaling

Definition - A cloud method or procedure known as autoscaling modifies available resources in response to the demand at any given time. Instead of scaling up and down, autoscaling scales in and out.

2.1 Autoscaling in Azure app service

Azure App Service's autoscaling keeps track of a web application's resource metrics while it's operating. When additional resources are needed to manage a growing demand, it recognizes those circumstances and makes sure they are accessible before the system overloads.

When the environment changes, autoscaling adjusts by adding or removing web servers and distributing the load among them. Autoscaling merely modifies the number of web servers; it has no influence on the CPU, memory, or storage capacity of the web servers that run the application.

2.2 Rules of Autoscaling

Autoscaling follows the principles you provide when making decisions. A rule defines a metric's threshold and, upon crossing it, initiates an auto scale event. Redistributing resources as the workload decreases is another benefit of autoscaling.

Carefully define your autoscaling rules. A Denial-of-Service assault, for instance, is likely to cause a significant spike in incoming traffic. It would be costly and ineffective to try to manage a DoS attack-induced spike in requests. These aren't legitimate requests, and handling them would be a mistake. Detecting and filtering requests made during an assault before they reach your service is a better way to handle the situation.

2.3 Factors while consider Autoscaling

Your services will be elastic thanks to autoscaling. For instance, you would anticipate a business app to have a spike in or drop out during the holidays. Fault tolerance and availability are enhanced by autoscaling. It can assist in ensuring that client requests to a service won't be turned down because an instance crashed due to overload or was unable to respond to the request promptly.

Adding or deleting web servers is how autoscaling operates. Autoscaling might not be a good strategy if your web apps receive requests that require a lot of resources. It could be required to manually scale up in some circumstances. For instance, depending on the instance size, a single request to a web application that requires extensive processing over a sizable dataset may use up all of the instance's processing and memory resources.

For managing long-term growth, autoscaling is not the ideal strategy. It's possible that your web application has a small user base at first but gains popularity over time. The overhead of autoscaling is related to resource monitoring and scaling event determination. If you can predict the rate in this case.

Another element is the quantity of instances of a service. Most of the time, you might anticipate simply running a few instances of a service. Nevertheless, whether autoscaling is turned on or not, your service is vulnerable to outages or unavailability in this scenario. As autoscaling spins up more instances, the fewer instances you have at first, the less capacity you have to accommodate a rising workload.

2.4 Identifying Factor

You can define the circumstances in which a web application should scale out and back in again with autoscaling. When autoscaling is done well, it can control expenses during periods of low demand while guaranteeing that there are enough resources to address high demand.

Autoscaling can be set up to determine when to scale in and out based on resource utilization and a variety of other criteria. Additionally, autoscaling can be set up to happen on a schedule.

You will learn how to define the parameters that can be utilized to auto scale a service in this unit.

- **App Service Plan with Autoscaling**

  One feature of the App Service Plan that the web application uses is autoscaling. Azure launches more instances of the hardware specified in the App Service Plan for the web application when it grows larger.

  An App Service Plan contains an instance limit in order to stop runaway autoscaling. Plans with higher cost tiers have a greater cap. This is the maximum number of instances that autoscaling can produce.

- **Autoscale circumstances**

  By establishing autoscale circumstances, you demonstrate how to autoscale. Azure offers two autoscaling options:

  ➔ Adjust the scale according to a measure, like the number of HTTP requests waiting to be processed or the length of the disk queue.
  ➔ Scale in accordance with a timetable to a given instance count. You can plan to scale out, for instance, on a specified day of the week or at a specific time of day. Additionally, you can choose an end date, after which the system scales back.

You can only scale out to a certain number of instances when you scale to a particular

instance count. Metric and schedule-based autoscaling can be combined in the same autoscale condition if you need to scale out gradually. Therefore, you may set up the system to scale out only during specific hours of the day in the event that the volume of HTTP requests exceeds a specified threshold.

To accommodate various timetables and measurements, you can establish several autoscale conditions. Azure scales your service automatically if any of these scenarios hold true. In addition, an App Service Plan provides a default condition that kicks in when all other requirements are met. This condition doesn't have a schedule; it's always active.

- **Autoscale rule metrics**

  You must define one or more autoscale rules in order to use autoscaling by metric. A measure to be tracked and the way autoscaling should react when it is above a certain threshold are both specified in an autoscale rule. For a web application, the metrics you may keep an eye on are:

  ➔ CPU As a Share. This measure shows how much CPU is being used in each instance. When a value is high, it indicates that instances are approaching CPU limits, which may result in delays when handling client requests.
  ➔ Memory As a Share. This measure records the application's memory occupancy for each and every iteration. A large number suggests that there may not be enough free RAM, which could lead to one or more instances failing.
  ➔ Queue length for disks. The number of pending I/O requests across all instances is represented by this metric. Disk contention may be present if the value is high.
  ➔ Http Queue Distance. This indicator displays the number of client requests that the web application is currently handling. Client requests may fail with HTTP 408 (Timeout) problems if this number is high.
  ➔ Info In. The total number of bytes received by all instances is this statistic.
  ➔ Information Out. The total quantity of bytes transferred by all instances is this statistic.

  For other Azure services, you may additionally scale according to metrics. For instance, if the web application handles requests from a Service Bus Queue, you may need to spin up additional web applications if an Azure Service Bus Queue has more items than can be held there for a certain amount of time.

- **Autoscale Actions**

  An autoscale rule has the ability to carry out an autoscale action upon determining thet a m etric has exceeded a threshold. Scale-out or scale-in autoscale actions are also possible.

  A scale-in action lowers the instance count, whereas a scale-out action raises the

  number of instances.An operator (such as less than, greater than, equal to, and so on) is use d by an autoscale action to decide how to respond to the threshold.

  When performing scale-out activities, the larger than operator is usually used to compare th e metric value to the threshold.

  Scale-in operations typically use the less than operator to compare the metric value to the thr eshold.Alternatively, an autoscale action can establish a target instance count instead of incre asing or decreasing the total number of instances.

  The cool-down time of an autoscale action is measured in minutes. The scale rule won't be invoked once more throughout this time. In between autoscale occurrences, this will enable the system to stabilize. Keep in mind that starting and stopping instances take time, thus any metrics collected may not reveal any noteworthy changes for a few minutes. Five minutes is the bare minimum for the cool-down time.

- **Combining rules for autoscale**

  Multiple autoscale rules (e.g., a scale-out rule and the matching scale-in rule) can be contained in a single autoscale condition. On the other hand, there is no requirement that the autoscale rules in an autoscale condition be connected. In the same autoscale scenario, the following four rules could be defined:

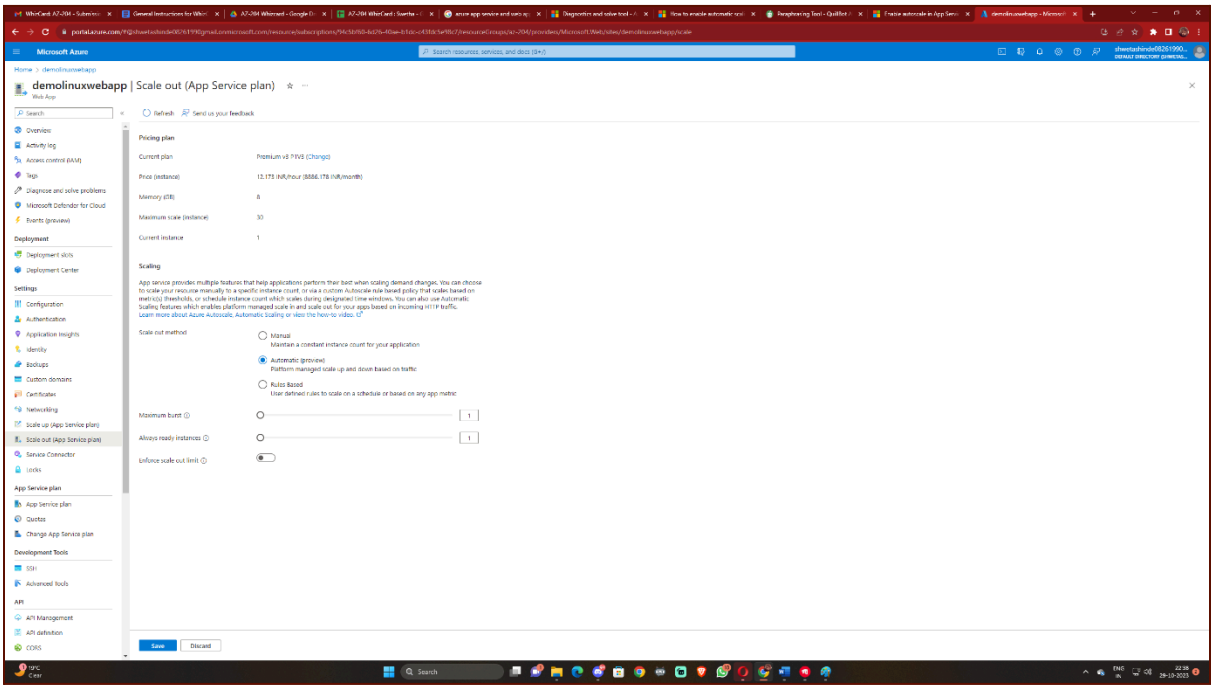
  A. Scale out by 1 if the length of the HTTP queue is more than 10.
  B. Scale out by one if the CPU use is more than 70%.
  C. Scale in by one if the HTTP queue length is zero.
  D. Scale in by one if the CPU use falls below 50%.

  The autoscale action is carried out when any of the scale-out rules—that is, when the CPU usage above 70% or the HTTP queue length reaches 10—are satisfied while deciding whether to scale out. Only when all scale-in requirements are satisfied—that is, when the CPU utilization falls below 50% and the HTTP queue length reduces to zero—does the autoscale action initiate while scaling in. You must describe the rules in distinct autoscale conditions if you need to scale in when only one of the scale-in rules is met.

2.5 Enable autoscale in App Service

i. Enable Autoscaling:

- Go to your App Service plan in the Azure portal and choose Scale out (App Service plan) in the Settings group in the left navigation pane to begin autoscaling.
- An App Service Plan only performs manual scaling by default. You can adjust your scale settings by using the condition groups that appear when you select Custom autoscale.
- The maximum number of instances that your App Service Plan may grow to in response to incoming HTTP requests is known as the maximum burst. You can set a maximum burst of up to 30 instances for Premium v2 and v3 subscriptions. The maximum burst must match the App Service Plan's chosen worker count, if not exceed it.
- Go to the web app's left menu and choose Scale out (App Service Plan) to enable automatic scaling. After changing the Maximum burst value and choosing Automatic (preview), click the Save button.

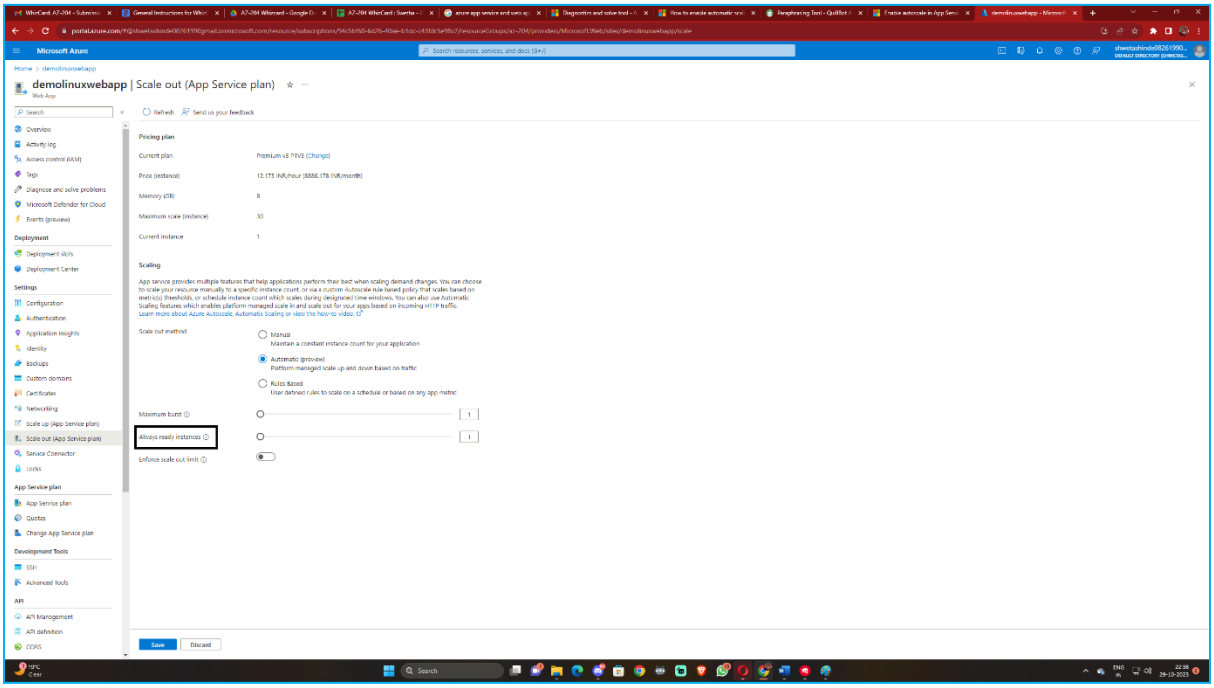


Limit the quantity of web application instances.

An app-level setting to set the minimum number of instances is called Always Ready Instances.

Up to the App Service Plan's chosen maximum burst, more instances are added if the load is greater than what the always-ready instances can manage.

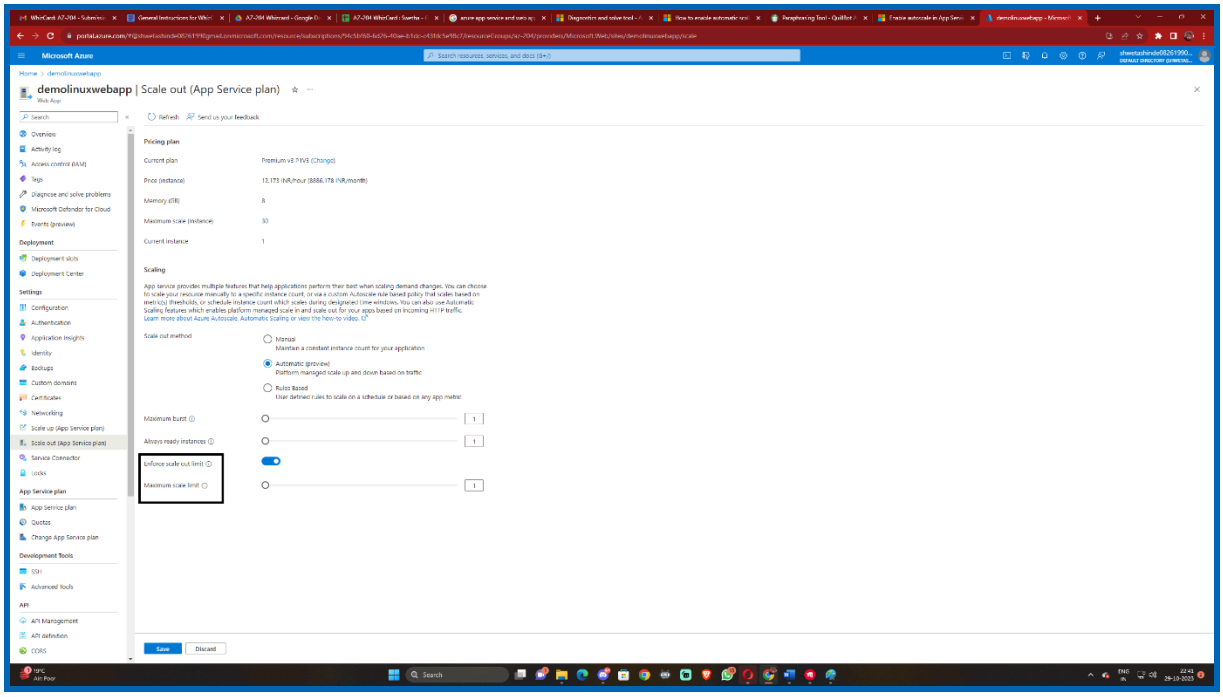
Go to the web app's left menu and choose Scale out (App Service Plan) to define the minimum number of instances. After making changes to the Always ready instances value, click Save.



Limit the number of instances of web apps.

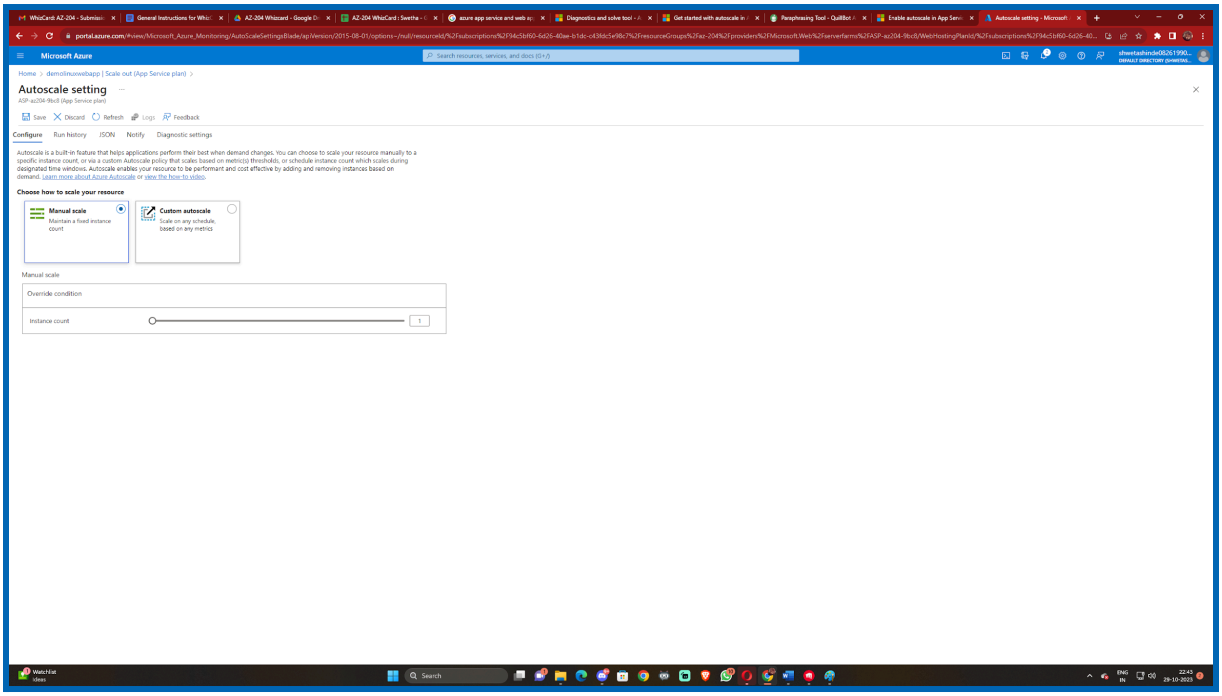
- The most instances to which a web application can scale is determined by its maximum scale limit. When a database or other downstream component has limited throughput, the maximum scale limit can be helpful. Between 1 and the maximum burst can be the per-app maximum.

- Go to the web app's left menu and choose Scale out (App Service Plan) to define the maximum number of instances. After updating the Maximum scale limit and choosing Enforce scale out restriction, click Save.



ii. Scale out with rules

An App Service Plan only performs manual scaling by default. You can adjust your scale settings by using the condition groups that appear when you select Custom autoscale.

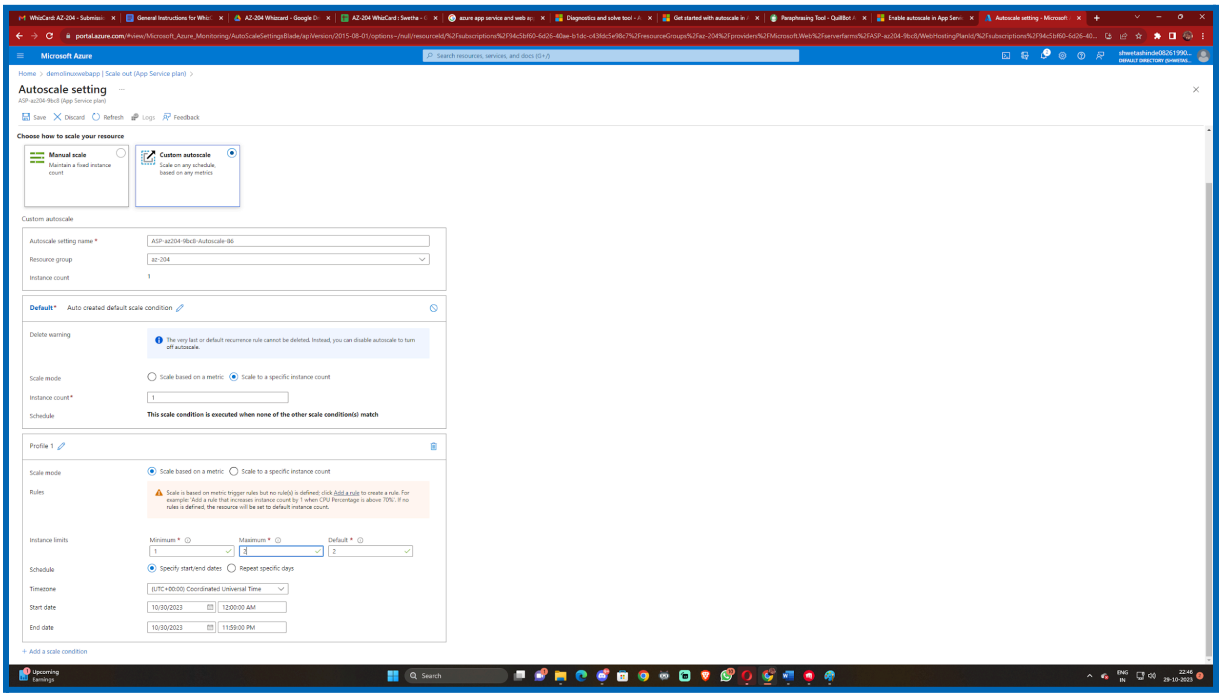


Include scale parameters

You can add and modify your own custom scale criteria to the automatically established default scale condition after you enable autoscaling. Keep in mind that every scale condition has two options: it can scale to a specified instance count or it can scale depending on a metric.

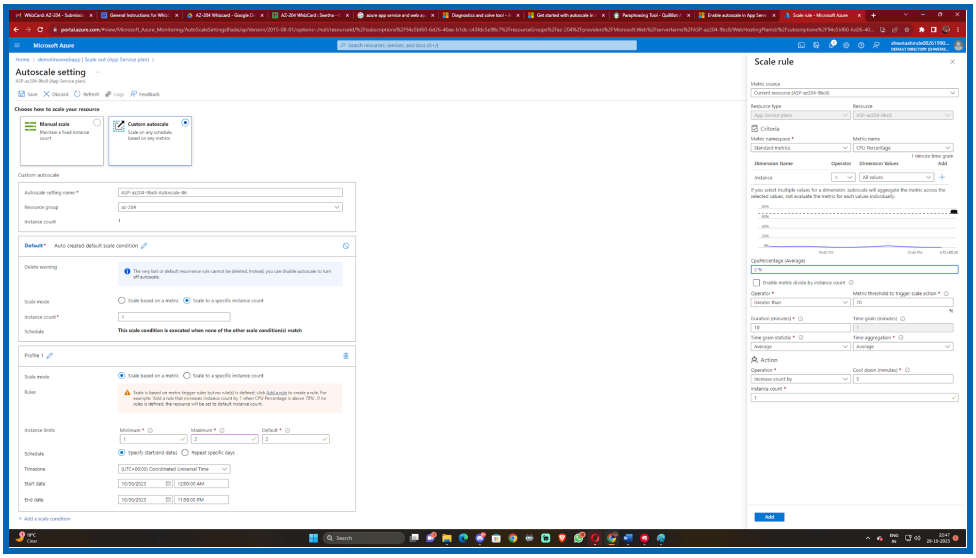
When none of the other scale conditions are in effect, the default scale condition is carried out.

It is also possible to set the minimum and maximum number of instances to be created using a metric-based scaling condition. The maximum quantity cannot go over the restrictions set forth in the pricing tier. A schedule specifying when the condition should be applied may also be included for any scale condition that differs from the default.



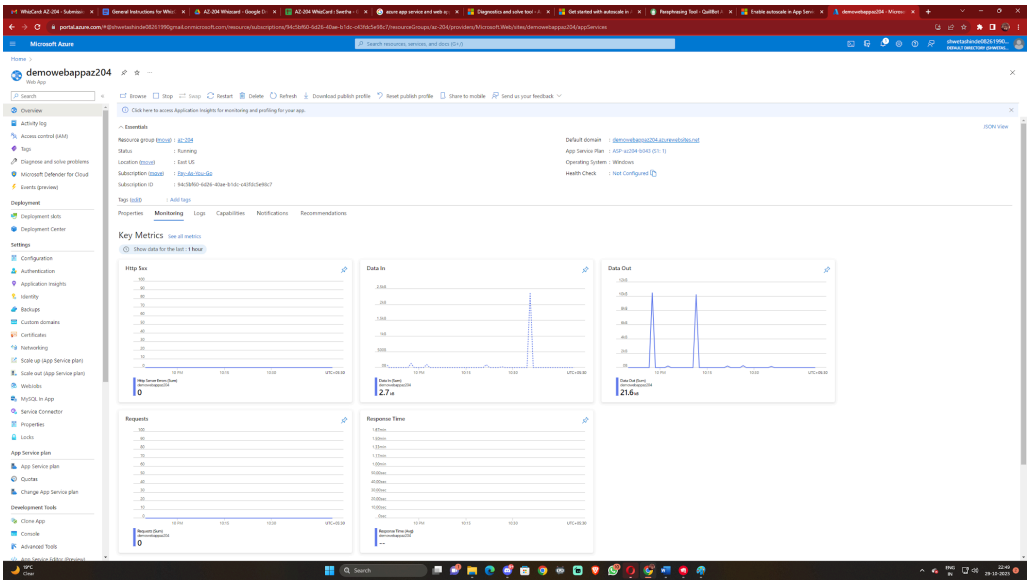
Make scale guidelines.

One or more scale rules are present in a metric-based scale condition. To add your own unique rules, click the Add a rule option. You define the criteria that indicate when a rule should trigger an autoscale action, and the autoscale action to be performed (scale out or scale in) using the metrics, aggregations, operators, and thresholds described earlier.



Observe the autoscaling process

The Run history chart in the Azure portal allows you to monitor when autoscaling has taken place. This graph illustrates the changes in the number of cases throughout time as well as the autoscale conditions that led to each variation.



2.6 Best Practices

i. When developing your autoscale rules, keep the following recommended practices in mind. Make sure there is a sufficient margin between the maximum and lowest readings and that they differ from each other.

No scale action can happen if your setup has minimum=two, maximum=two, and there are now two instances.Keep an adequate margin between the maximum and minimum instance counts, which are inclusive. Autoscale always scales between these limits.

ii. You can select Average, Minimum, Maximum, and Total as the metric to scale by for diagnostics metrics. The average statistic is the most used one.

iii. For each sort of measure, carefully select the thresholds.

Based on real-world scenarios, we advise carefully selecting distinct scale-out and scale-in criteria. The following instances, which have the same or comparable threshold values for out and in circumstances, are not recommended autoscale settings:

When thread count exceeds 600, increase instances by one count.
When Thread Count <= 600, reduce instances by one count.

Let's examine an illustration of what can cause a behavior that could appear unclear. Examine the following order.

1. Let's say there are two instances at start, and after that, each instance has an average of 655 threads.
2. When a third instance is added, autoscale scales out.
3. Assume next that there are 525 threads on average per instance.
4. Autoscale makes an attempt to predict the ultimate state if it scales in before actually doing so. As an illustration, the current instance count of 525 x 3 = 1,575 / 2 (the final instance count after scaling in) = 787.5 threads. This means that if the average thread count stays the same or even slightly decreases, autoscale would need to scale out again right away. But if it grew larger once again, the entire procedure would happen again, creating an endless cycle.

The goal of estimation during a scale-in is to prevent "flapping"—a condition in which actions related to scale-in and scale-out are continuously performed back and forth. When selecting the same scale-out and scale-in thresholds, bear this behavior in mind.

We advise selecting a sufficient buffer between the in and scale-out criteria. Take into consideration the following improved rule combination as an example.

When CPU% >= 80, increase instances by one count.
When CPU%<= 60, reduce the number of instances by one count.

In this instance - Let's say there are two occurrences at first - Autoscale stops adding a third instance if the average CPU% for all instances reaches 80 - Let's now assume that the CPU% gradually drops to 60.

The final state, if it were to scale-in, is estimated by Autoscale's scale-in rule. For instance, the ultimate number of instances when scaled in is equal to 180 / 2 (60 x 3), which is 90. Because it would have to scale out again right away, autoscale doesn't scale in. It does not scale in instead.

When autoscale checks again, the CPU keeps dropping below 50. It calculates once more: 50 x 3 instances = 150 / 2 instances = 75. Since this is less than the 80 scale-out criteria, the scaling in to 2 instances is successful.

iv. Scaling considerations when a profile has several rules configured
In certain situations, a profile may need to have more than one rule set.
When several rules are set, services employ the following set of autoscale rules.
If any rule is met during scale-out, autoscale executes. Every need must be fulfilled for autoscale on scale-in.

Assume you have the following four autoscale rules, for example:

- Reduce by 1 if CPU is less than 30%.
- Reduce by 1 if Memory is less than 50%.
- If CPU is more than 75%, reduce by 1.
- Scale out by 1 if Memory > 75%

Next, the following takes place:

- We scale out if the CPU is 76% and the memory is 50%.
- We scale out if the CPU is 50% and the memory is 76%.

Conversely, autoscale fails to scale-in when the CPU is at 25% and the RAM is at 51%. If the CPU is 29% and the memory is 49%, both of the scale-in rules would be satisfied, resulting in an automated scale-in.

v. Because autoscale adjusts your service to that count when metrics are unavailable, the default instance count is crucial. Choose a default instance count that is safe for your workloads as a result.

vi. If any of the following scenarios materialize, Autoscale adds a note to the Activity Log:
An autoscale scale operation is issued.
   - ❖ A scale action is successfully finished by the autoscale service.
   - ❖ The scale action is not taken by the autoscale service.
   - ❖ For autoscale services to decide on a scale, there are no available metrics.
   - ❖ Recuperated metrics are once again available for use in determining scale.
   - ❖ An Activity Log alert is another tool you may use to keep an eye on the autoscale engine's condition. The notifications tab on the autoscale setting allows you to set up email or webhook notifications in addition to activity log alerts to receive notifications for successful scale actions.

Azure App Service – Deployment of code

A. Deploying code to webapp

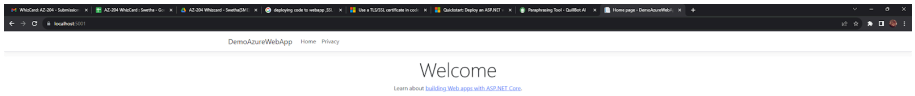
1. On your computer, open a terminal window and navigate to a working directory. Use the dotnet new webapp command to build a new.NET web application, and then navigate into the newly created app's directories.

```
C:\Users\wildc>dotnet new webapp -n DemoAzureWebApp --framework net7.0
The template "ASP.NET Core Web App" was created successfully.
This template contains technologies from parties other than Microsoft, see https://aka.ms/aspnetcore/7.0-third-party-not
ices for details.

Processing post-creation actions...
Restoring C:\Users\wildc\DemoAzureWebApp\DemoAzureWebApp.csproj:
  Determining projects to restore...
  Restored C:\Users\wildc\DemoAzureWebApp\DemoAzureWebApp.csproj (in 67 ms).
Restore succeeded.



C:\Users\wildc>cd DemoAzureWebApp
```

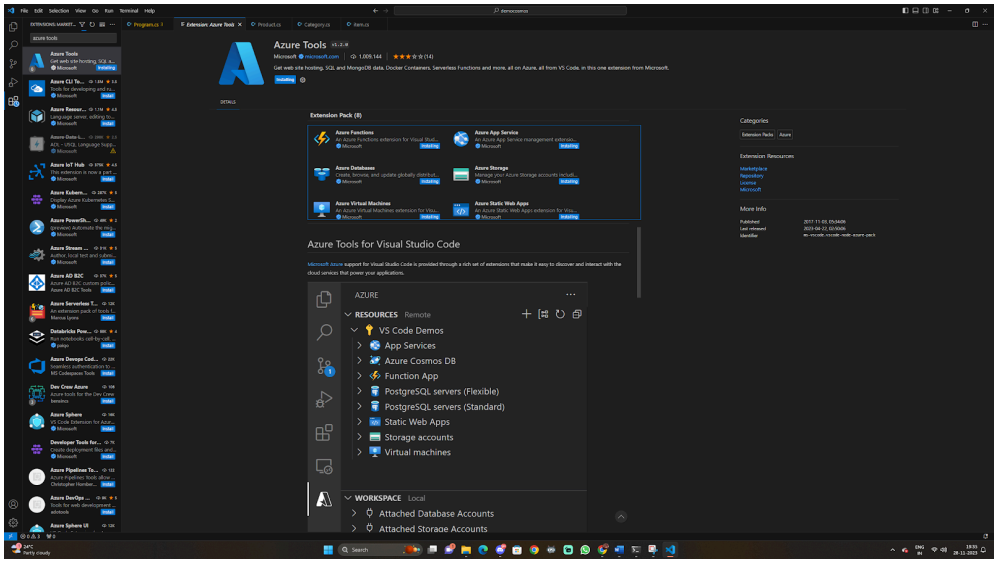
2. Use the dotnet run command to launch the application locally from within the same terminal session.

```
C:\Users\wildc\DemoAzureWebApp>dotnet run --urls=https://localhost:5001/
Building...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\wildc\DemoAzureWebApp
```

3. Open a web browser, and navigate to the app at https://localhost:5001.
   You see the template ASP.NET Core 7.0 web app displayed in the page.

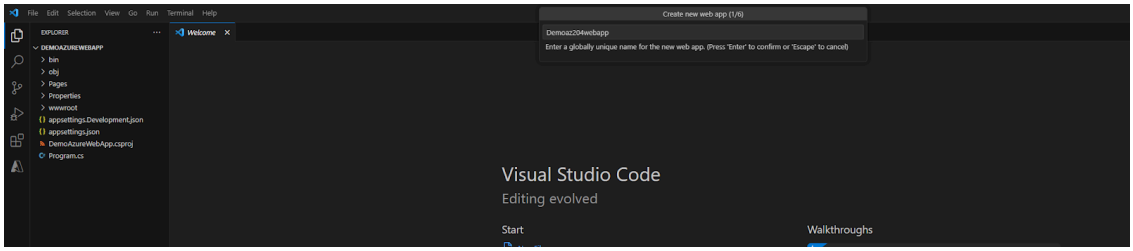


4. Go to extensions and install Azure tools in visual studio code.

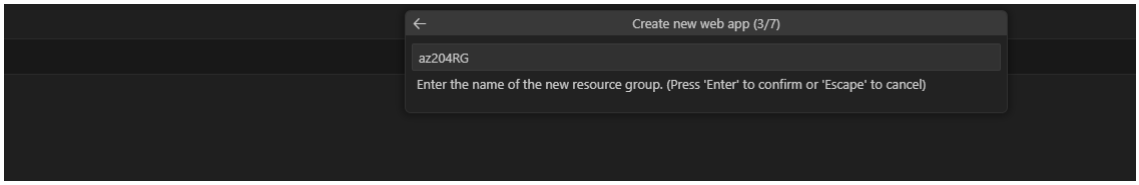


5. In Visual Studio Code, open the Command Palette by selecting View > Command Palette.
6. Search for and select "Azure App Service: Create New Web App (Advanced)".
7. Respond to the prompts as follows:

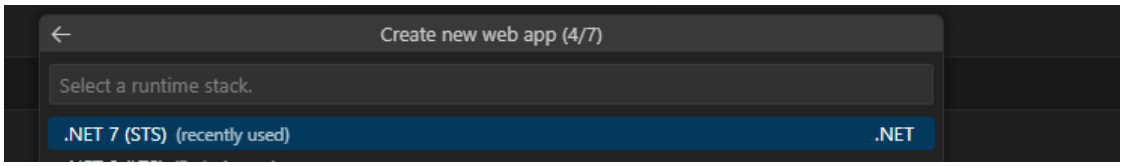
- If prompted, sign in to your Azure account.
- Select your Subscription.
- Select Create new Web App... Advanced.
- For Enter a globally unique name, use a name that's unique across all of Azure (valid characters are a-z, 0-9, and -). A good pattern is to use a combination of your company name and an app identifier.



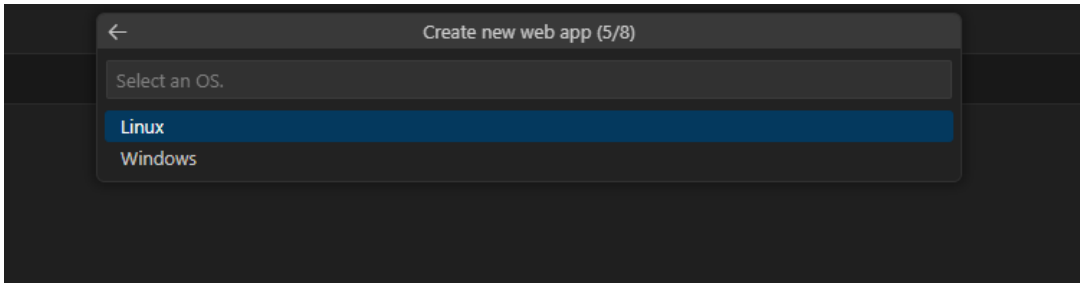
- Select Create new resource group and provide a name like az204RG.



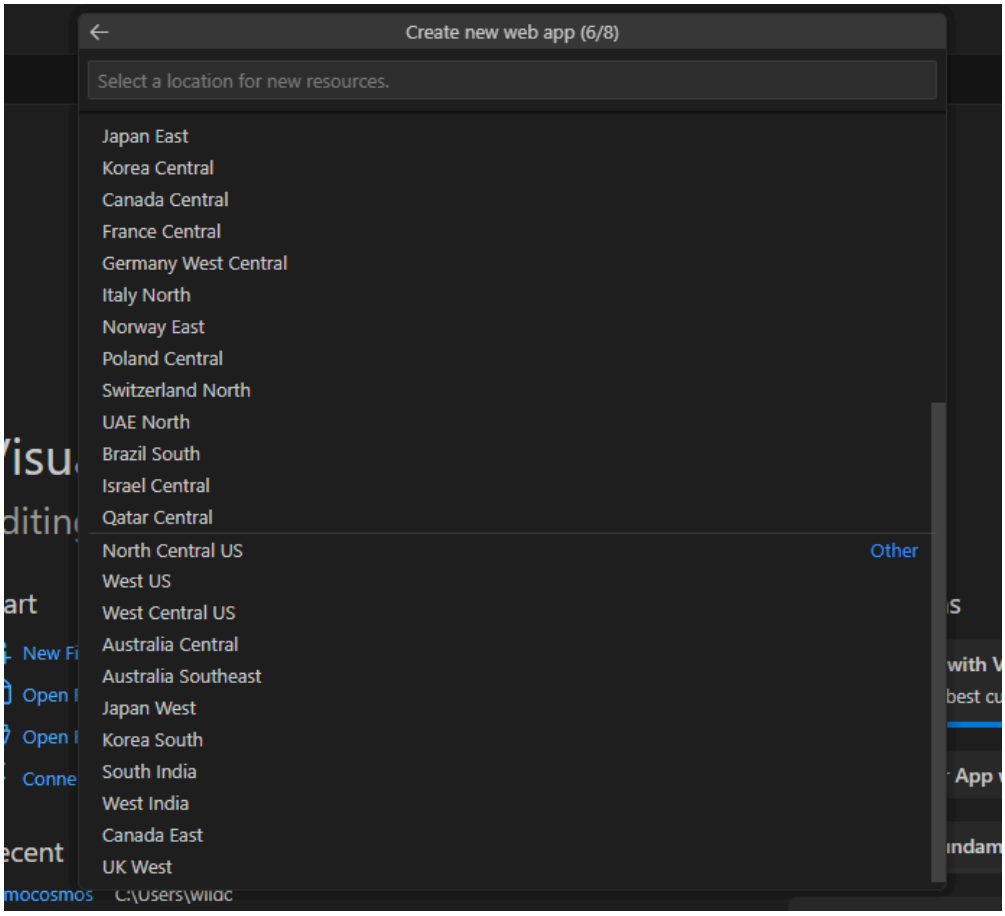
- When prompted to Select a runtime stack, select .NET 7 (STS).



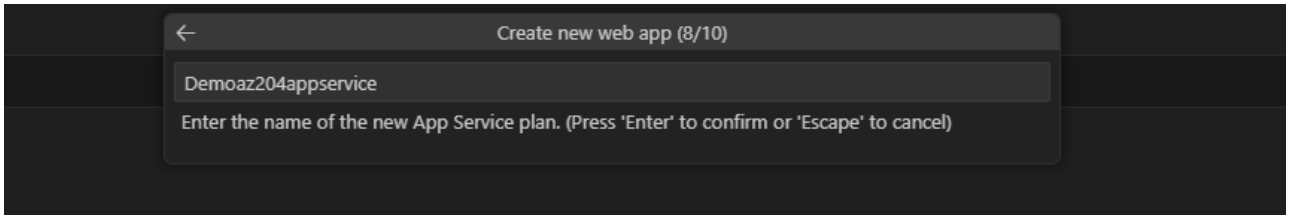
- Select an operating system (Windows or Linux).

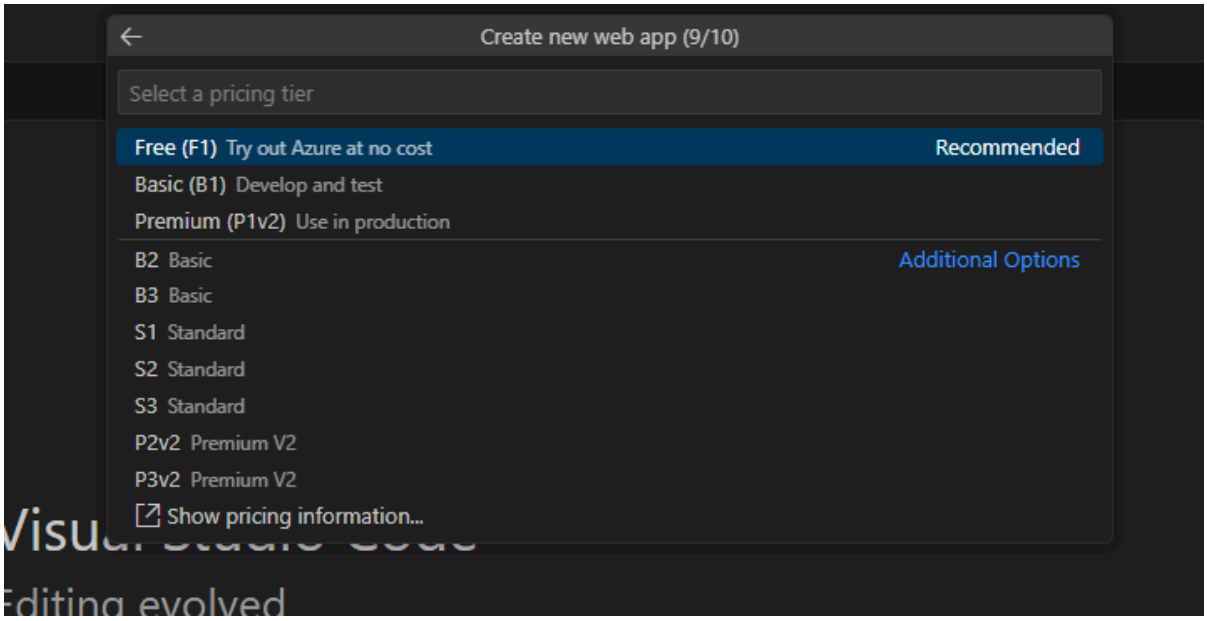


- Select a location near you.

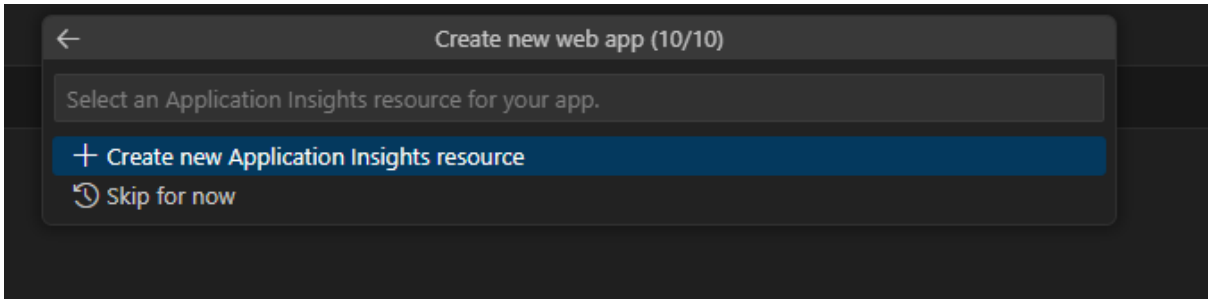


- Select Create a new App Service plan, provide a name, and select the F1 Free pricing tier.



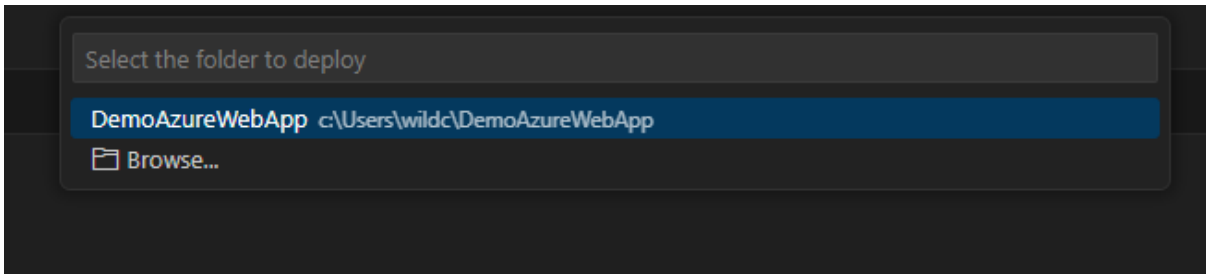


- Select Skip for now for the Application Insights resource.

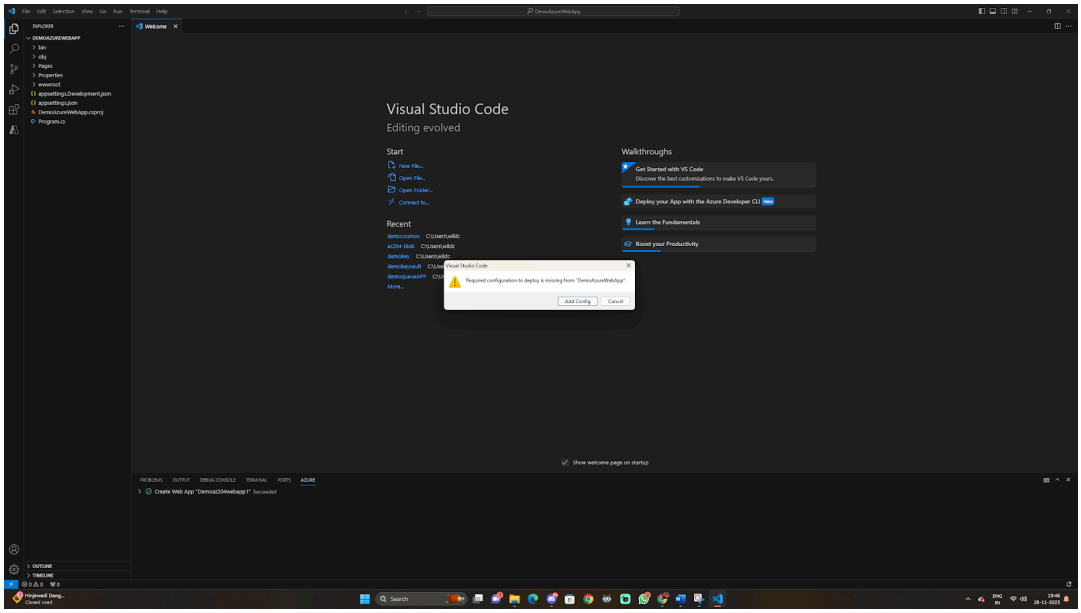


- When prompted, click Deploy.

- Select DemoAzureWebApp as the folder to deploy.

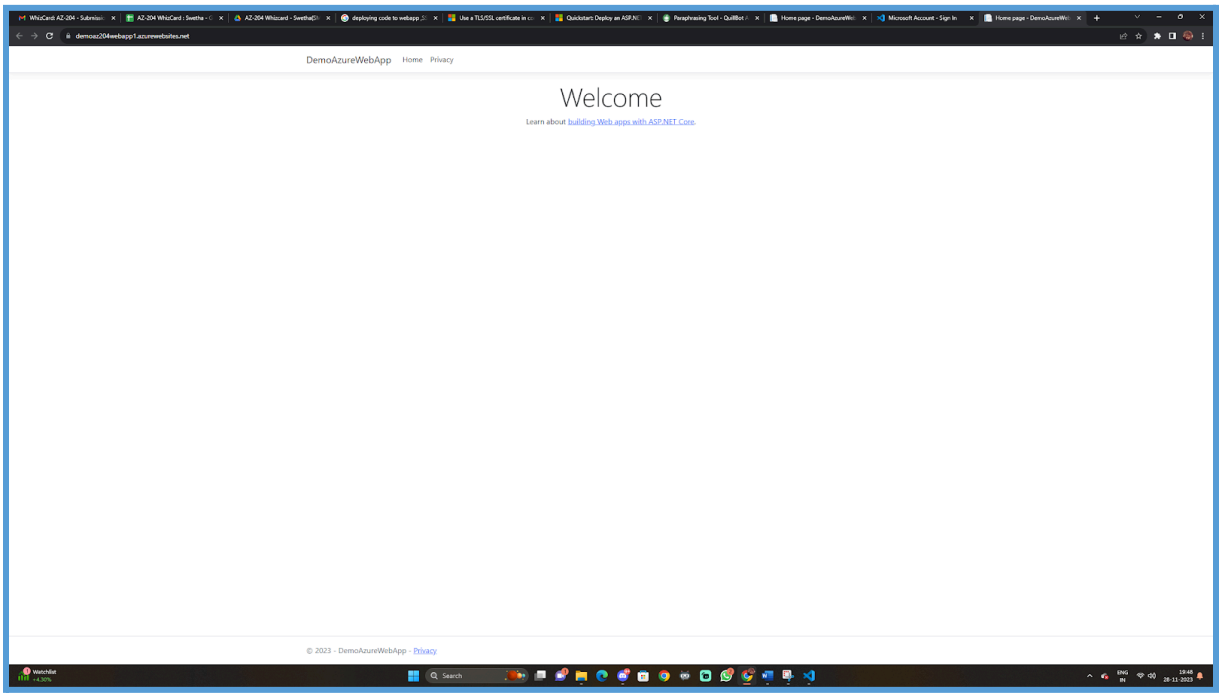


- Select Add Config when prompted.



8. Whenever you are in the "DemoAzureWebApp" workspace in Visual Studio Code, the App Service app will be deployed too automatically. Choose the Yes option from the dialog to make this happen.
9. When publishing completes, select Browse Website in the notification and select Open when prompted.

   You see the ASP.NET Core 7.0 web app displayed in the page.

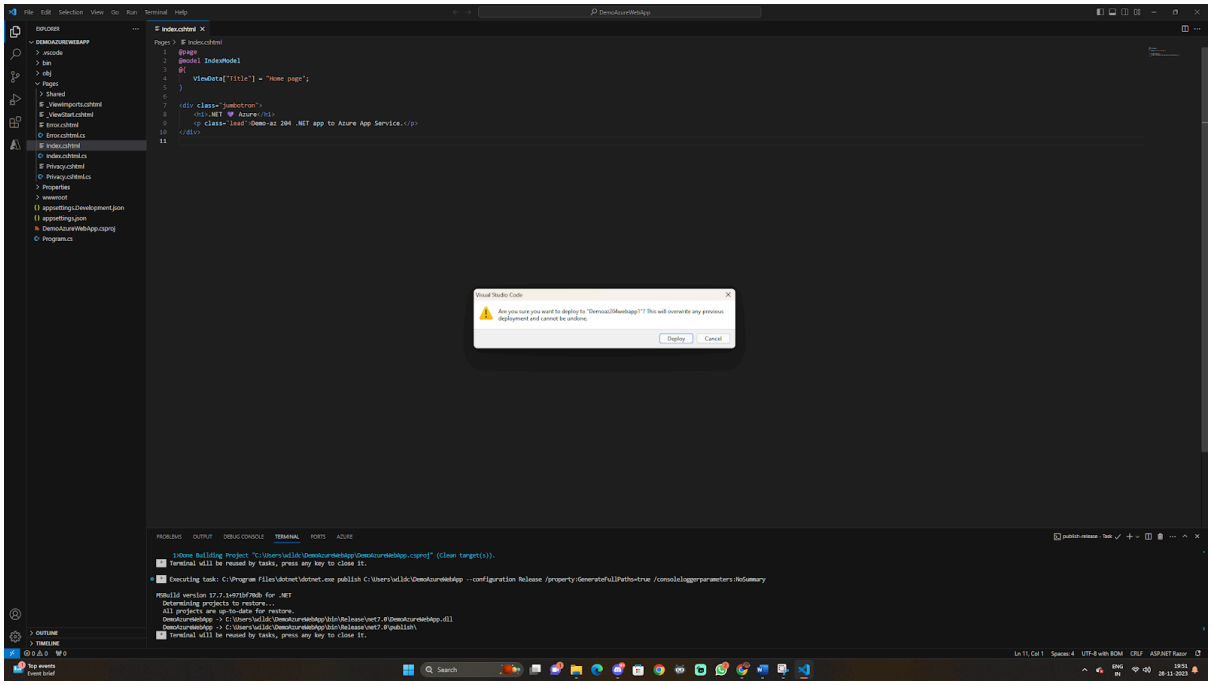


10. Open Index.cshtml.
11. Replace the first element with the following code:

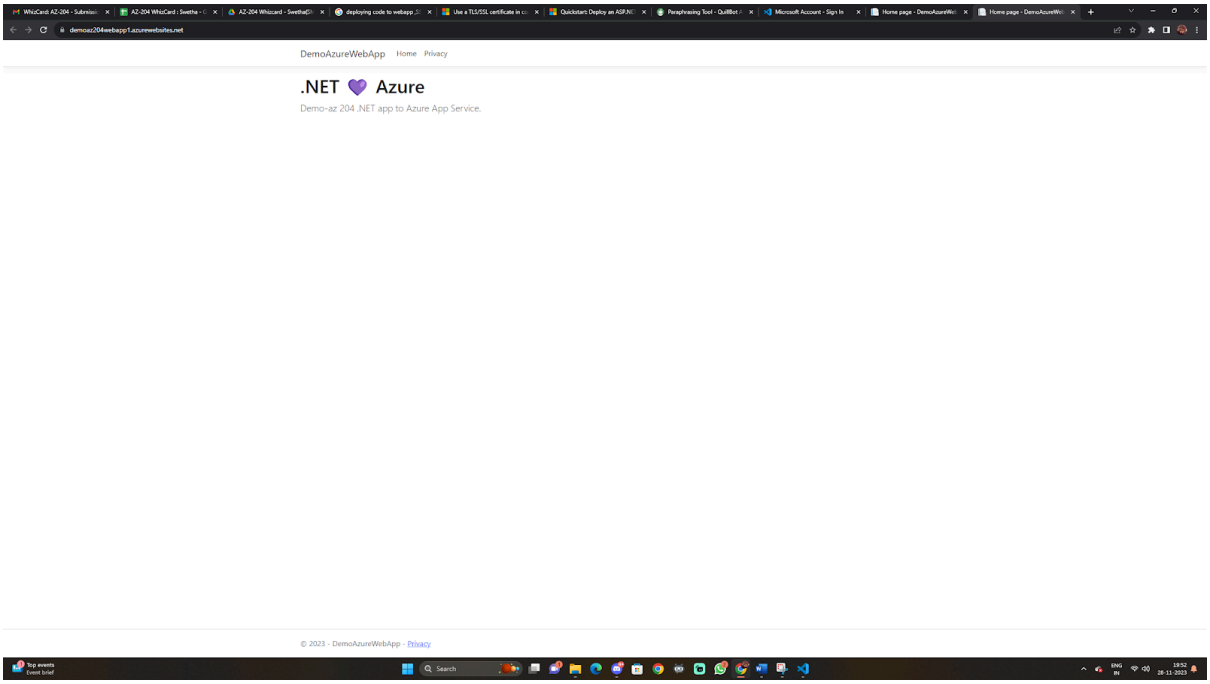
```
Pages > Index.cshtml
1   @page
2   @model IndexModel
3   @{
4       ViewData["Title"] = "Home page";
5   }
6
7   <div class="jumbotron">
8       <h1>.NET 💜 Azure</h1>
9       <p class="lead">Demo-az 204 .NET app to Azure App Service.</p>
10  </div>
11
```

Save your changes.

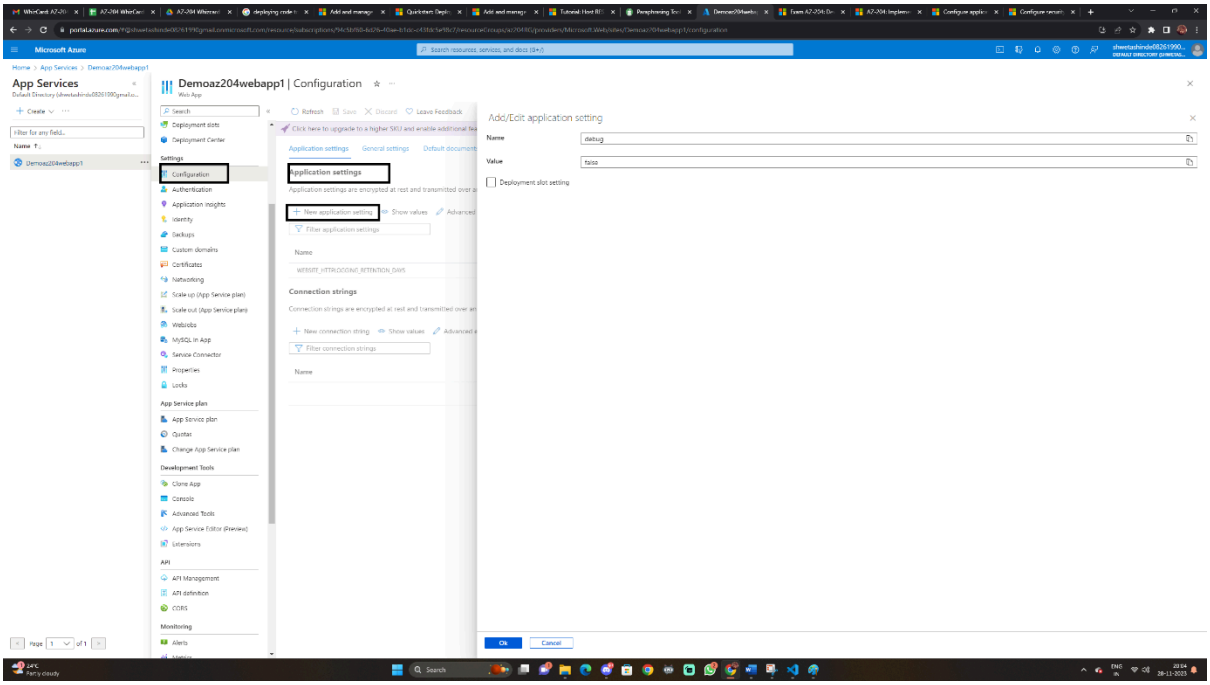
12. In Visual Studio Code, open the Command Palette, Ctrl+Shift+P.
13. Search for and select "Azure App Service: Deploy to Web App".
14. Select Deploy when prompted.



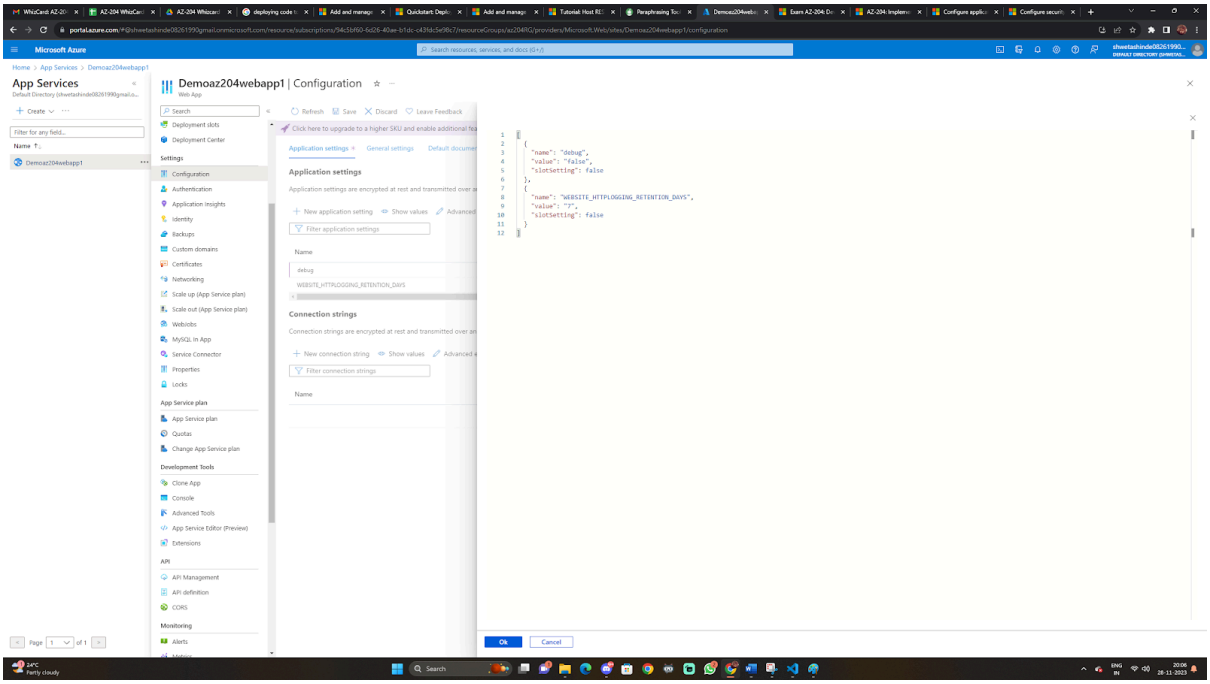
15. When publishing completes, select Browse Website in the notification and select Open when prompted.
16. You see the updated ASP.NET Core 7.0 web app displayed in the page.



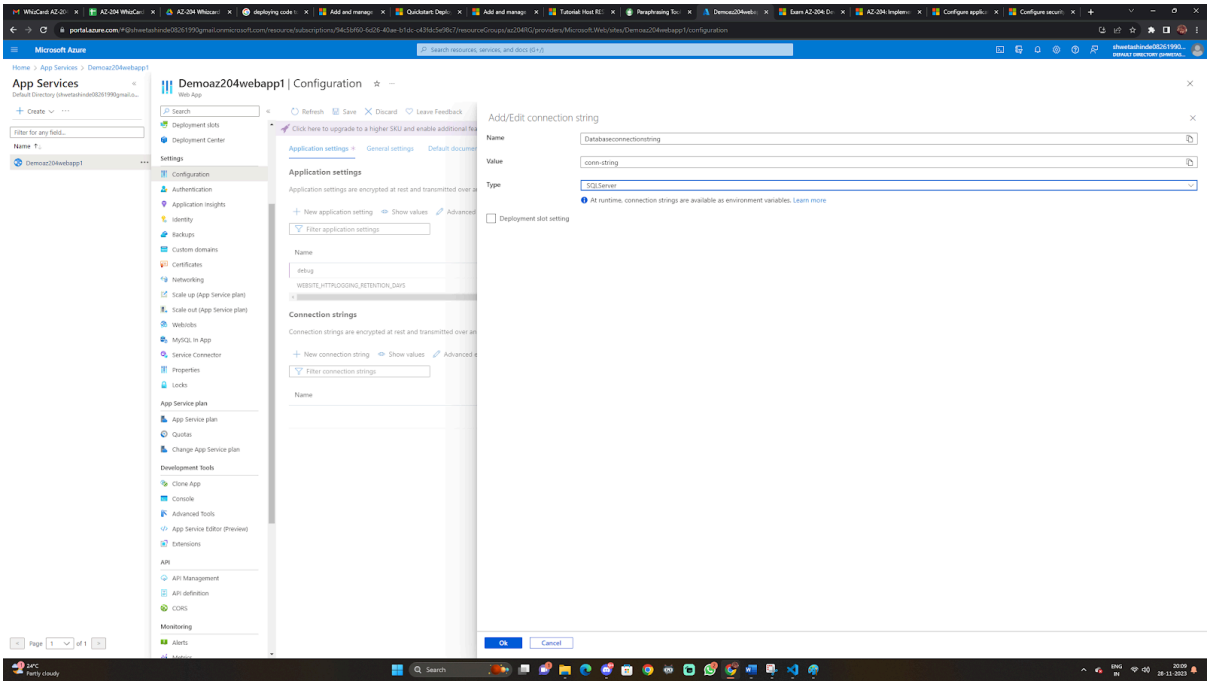
17. To add a new app setting, select the new application setting. If you're using deployment slots you can specify if your setting is swappable or not. In the dialog, you can stick the setting to the current slot.



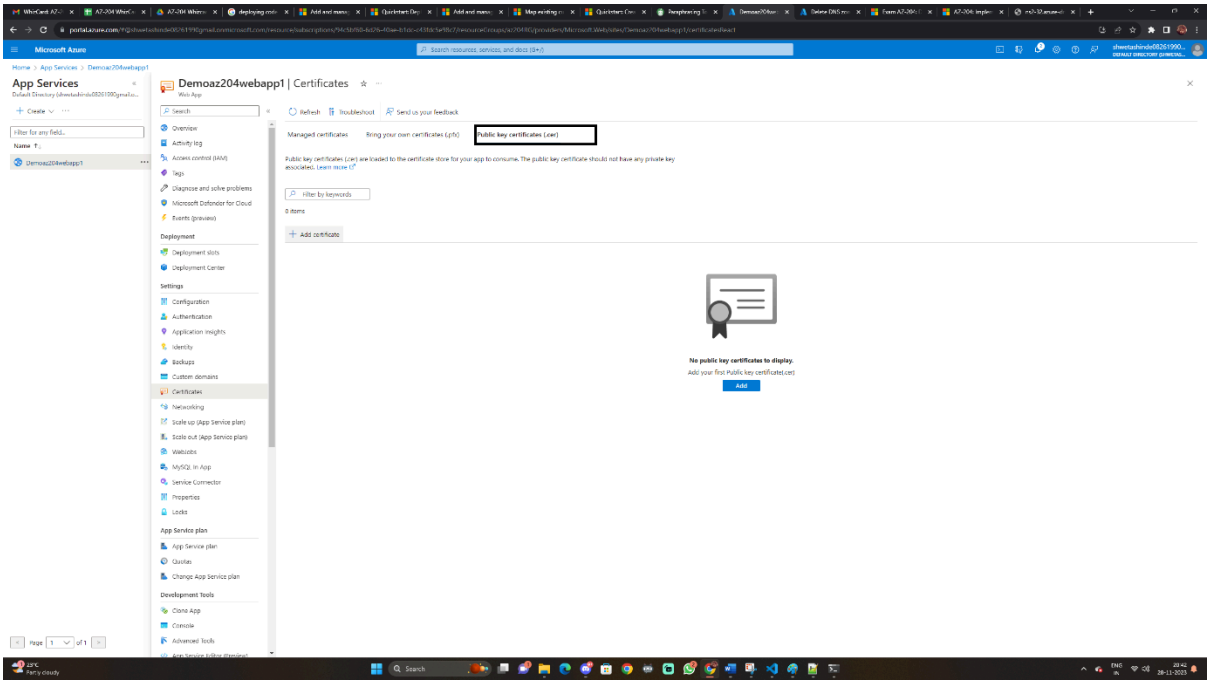
18. If you want to update application settings in bulk then click on Advanced edit. You can now edit the settings as per your need.



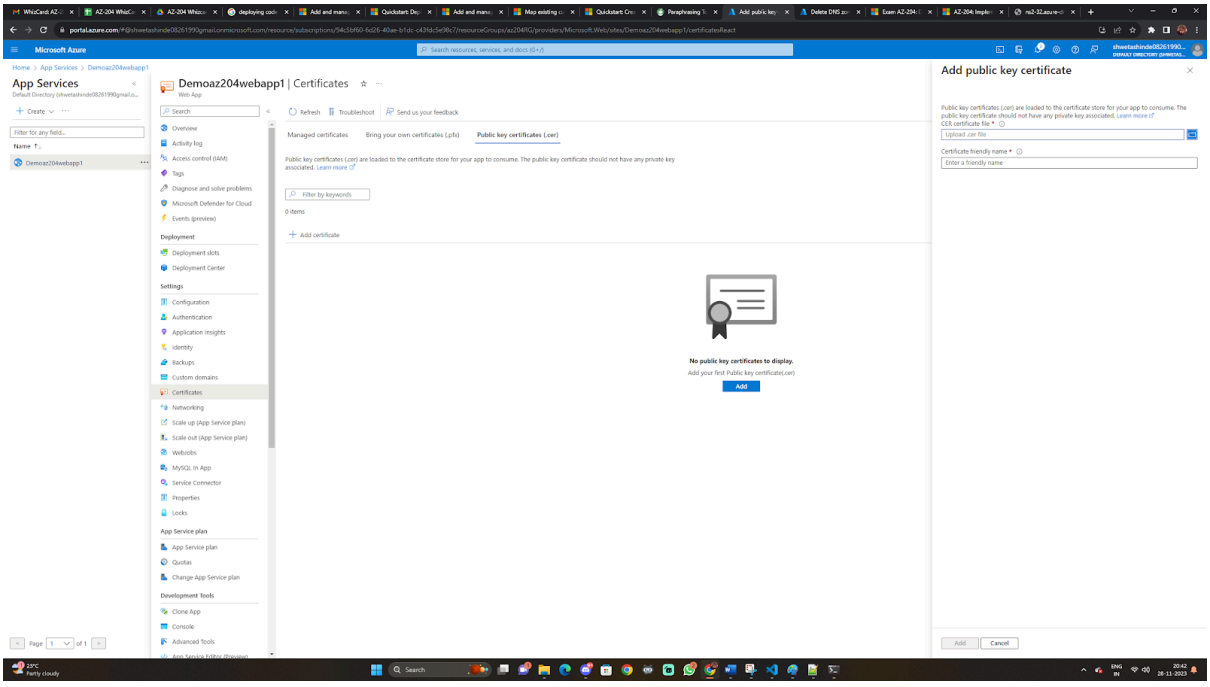
19. If you want to add a connection string for your app then if you are using a .NET application then you can specify your connection string in web.config or else for other languages you can add connection string by clicking on the new connection string button in the configuration window.



20. To use certificates for your webapp click on certificates under settings select Certificates > Public key certificates (.cer) > Add certificate.



21. Select your existing certificate if you have any and click on Add.



Azure function App

- Azure Functions are part of the Platform as a Service(PaaS) model in azure.
- Azure Functions is a serverless solution that allows you to write less code, manage less infrastructure, and save costs.
- Azure Functions is an event-driven, serverless compute platform that helps you develop more efficiently using the programming language of your choice.
- It supports the following languages.

C#	Java	JavaScript	TypeScript	PowerShell	Python

- It allows us to write less code, maintain less infrastructure while saving on cost.
- Azure provides all infrastructure and maintain in background for azure function app.
- Azure Functions supports triggers and bindings.
- Triggers are ways to start execution of your code,
- Bindings are ways to simplify coding for input and output data.
- Your functions can operate in an Azure execution environment thanks to a function app. It serves as the deployment and management unit for your functions as a result. A function app is made up of one or more distinct functions that are coordinated in their management, deployment, and scaling. A function app's functions all have the same deployment strategy, price structure, and runtime version. To arrange and manage your functions collectively, consider a function app.

Creating and configuring a Function APP

You can create and deploy the Function App through different methods, Main are.

1. Through Azure Portal
2. Through The Azure PowerShell/CLI
3. Through ARM Templates

When you create function app in the azure portal you must choose a hosting plan for your app.

★ Let's understand first what different hosting plans and differences between them.
★ There are different hosting plans available for Azure Functions (those are as below).

1. **Consumption plan** - This is the standard hosting package. You only pay for compute resources while your functions are running, and it scales automatically. Based on the volume of incoming events, instances of the Functions host are dynamically added and withdrawn.
2. **Premium plan** - Pre-warmed workers, which launch programs instantly after being idle, run on more powerful instances, and connect to virtual networks, automatically scale based on demand.

3. **Dedicated (App service) Dedicated plan.** - Use the standard App Service plan charges to execute your functions. Best suited for prolonged situations in which Durable Functions cannot be used.

Based on hosting plan following behaviors can be dictated:

- How your function app is scaled.
- The resources available to each function app instance.
- support for cutting-edge features including access to the Azure Virtual Network.

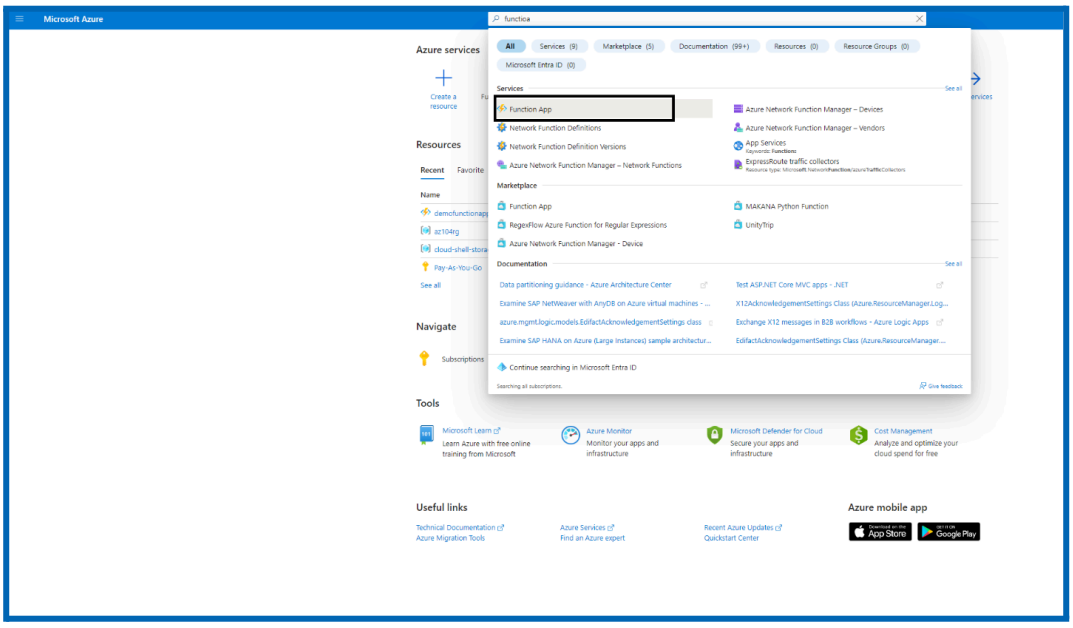
Hosting plans and scaling

The following table compares the scaling behaviors of the various hosting plans. Unless otherwise stated, maximum instances are stated on a per-function app (Consumption) or per-plan (Premium/Dedicated) basis.

Plan	Scale out	Max #Instances
Consumption plan	Event driven. Scale out automatically	Windows: 200, Linux: 100
Premium plan	Event driven. Scale out automatically	Windows: 100, Linux: 20-100
Dedicated plan	Manual/Auto Scale	10-20

Steps to cerate Function app through Azure portal.

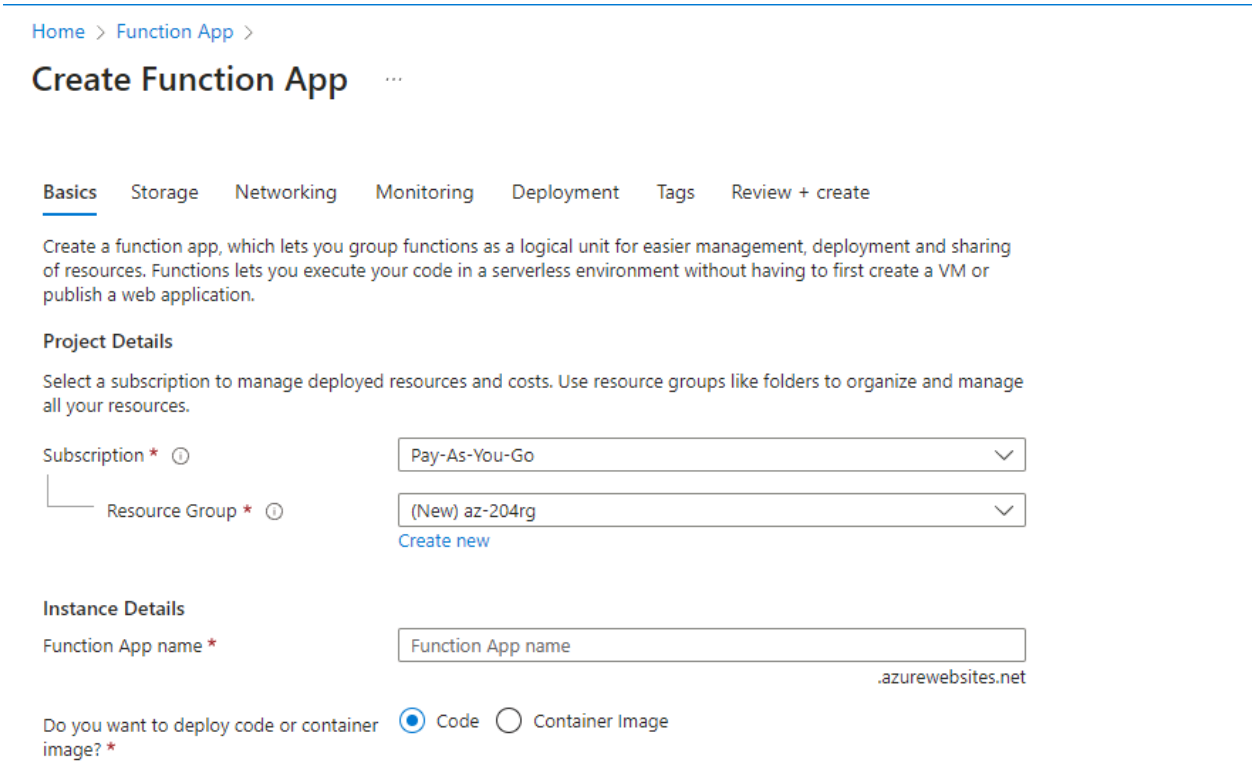
1. Search a function app in search box in azure portal.



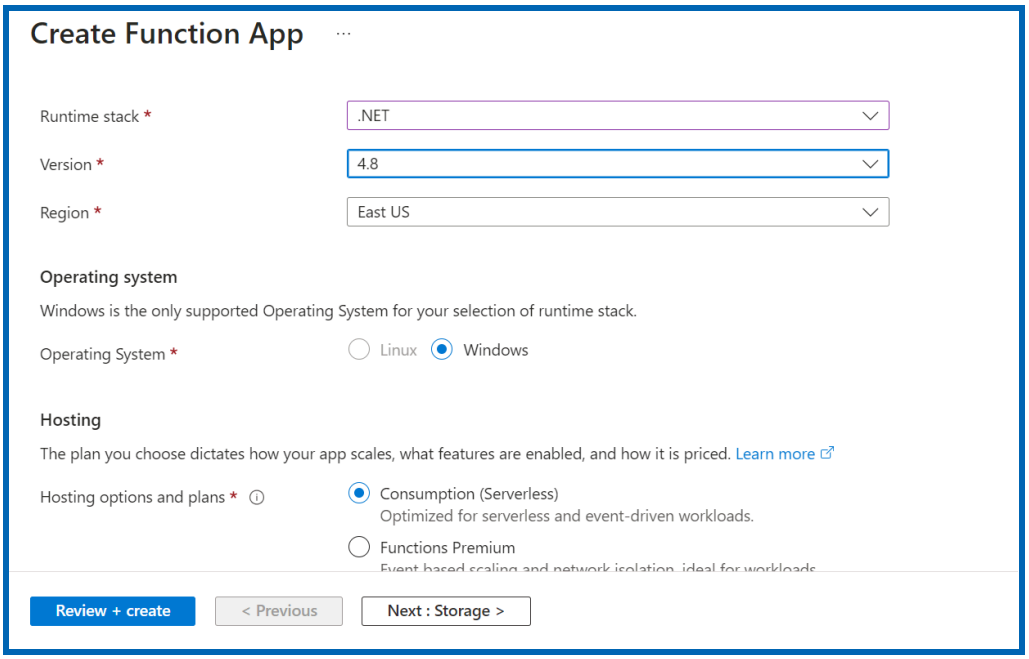
2. Click on it and then function app window will open, in that window click on create function app. The following window will open.

1. In this window select your subscription, select a resource group if already created of create a new one. Give your function meaningful name.

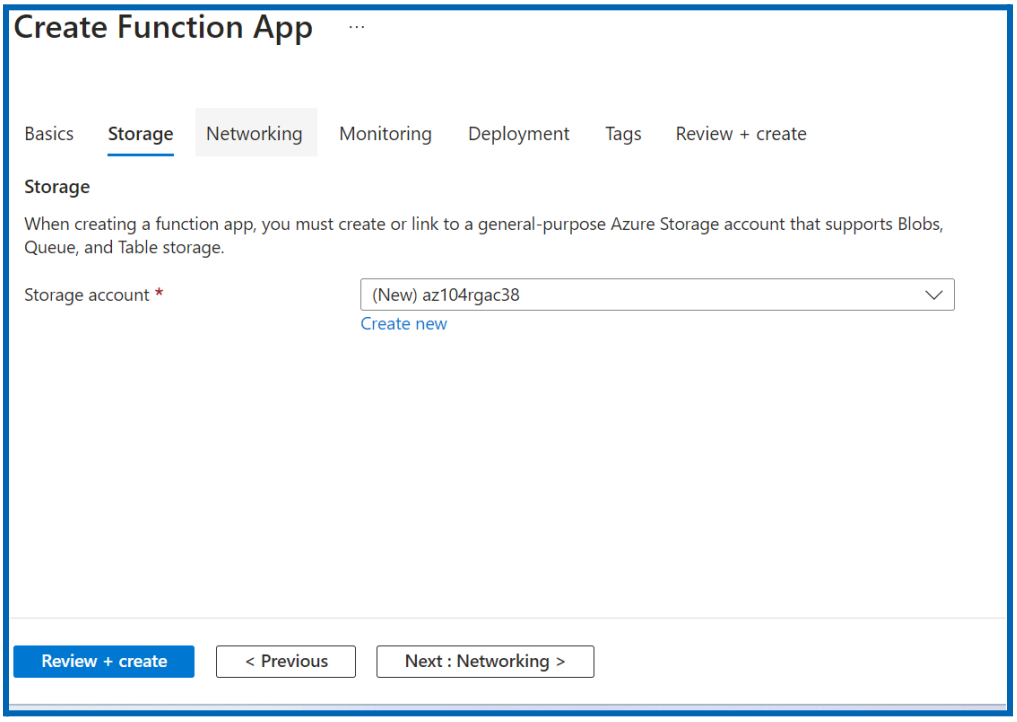
2. Next option we have is if we want to deploy code or container image. Based on your requirement select appropriate option.



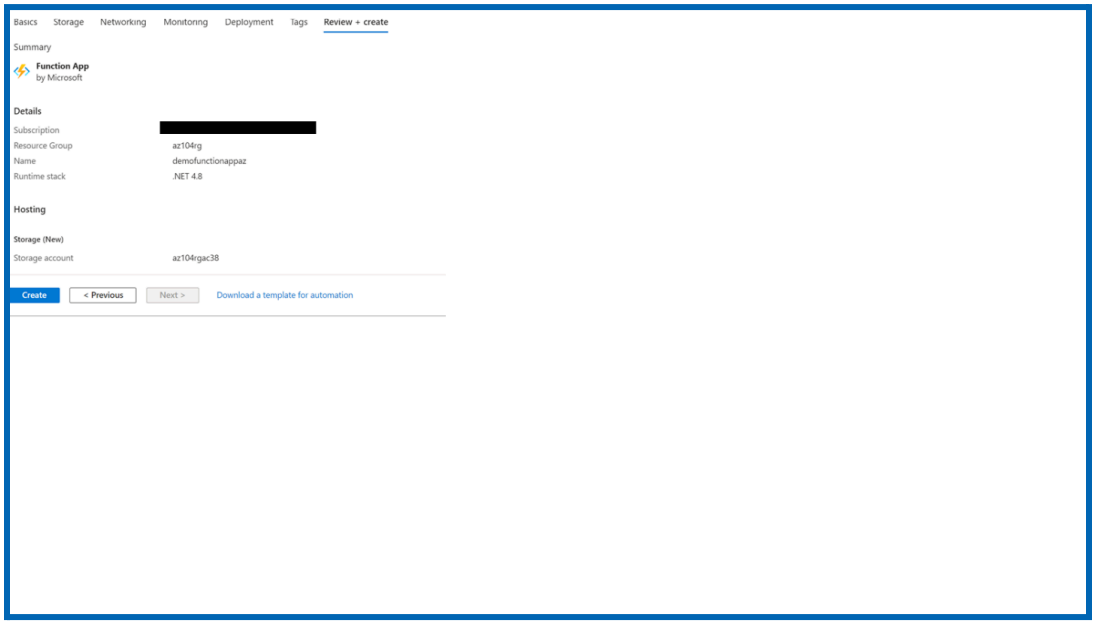
3. The next option is to select your coding language if you have selected code in the option and its runtime version. Select appropriate options. Based on your runtime stack choice operating system option will be enabled or disabled. For e.g. If you choose .NET as your runtime stack, then by default windows operating system will be choose as .NET can't operate on Linux machine.
4. Choose your hosting plan as per requirement and click on next.



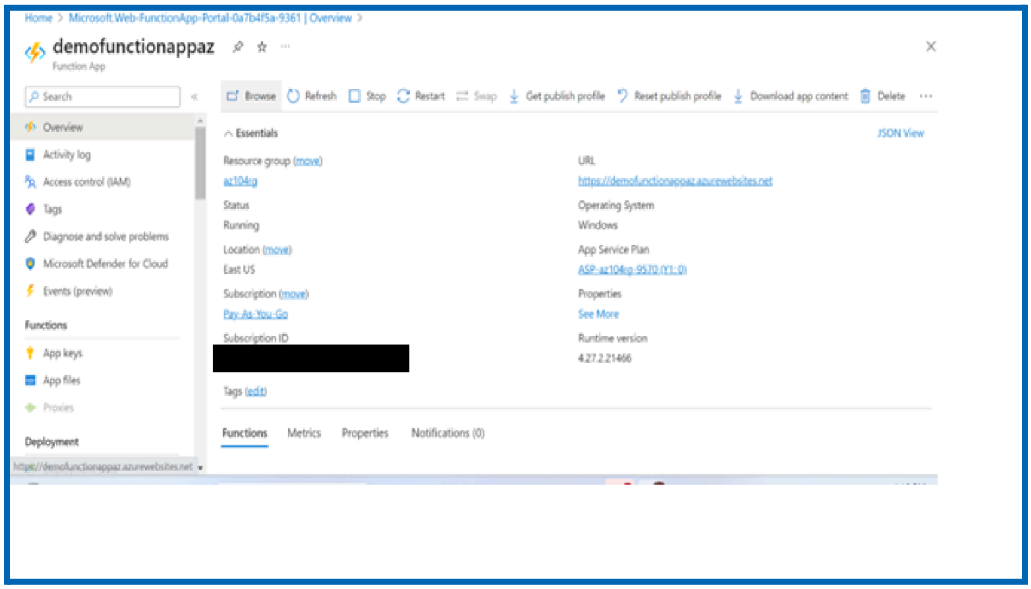
5. The next window is storage. select your storage account and click on next.
   NOTE: - On any plan, a function app requires a general Azure Storage account, which supports Azure Blob, Queue, Files, and Table storage. This is due to the fact that some storage accounts do not support queues and tables, despite the fact that functions rely on Azure Storage for tasks like managing triggers and logging function executions. The same storage account used by your function app can also be used by your triggers and bindings to store your application data Use a different storage account, though, for procedures that require a lot of storage.



6. select appropriate networking and monitoring option as per your requitement. If monitoring in required, then select yes or else no and go to deployment section.
7. If your function needs continuous deployment, then select that option or else leave default settings as it is.
8. Under tags create tags if necessary and then click on review + create. Review window will open. Please cross check your configuration once again in review window and click on cerate.



9. Once deployment is completed click on go to resource you will be navigated to your function app overview window.



Azure Function

A function runs as a result of triggers. It is necessary for a function to have exactly one trigger since it determines how the function is called. Data connected with triggers is frequently supplied as the function's payload.

An additional resource can be declaratively linked to a function by binding to it; bindings can be connected as input bindings, output bindings, or both. The function receives parameters that contain data from bindings.

You can combine various bindings to make it work for you. A function may have one or more input and/or output bindings; bindings are optional.

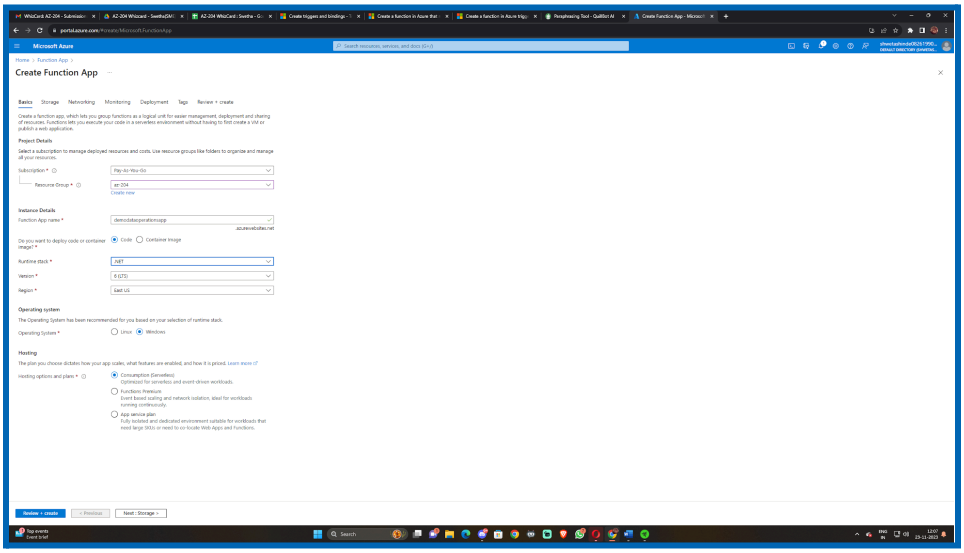
You can avoid hardcoding access to other services by using triggers and bindings. Function parameters are used to receive data (like the contents of a queue message) for your function. You submit data by utilizing the function's return value, for instance, to construct a queue message. In the function, every trigger and binding have a direction property. JSON data:

- The direction of triggers is always in and out.
- Input and output bindings use in and out
- A unique direction in out is supported by some bindings. Only the Advanced editor is accessible through the Integrate tab on the gateway if you utilize in out.

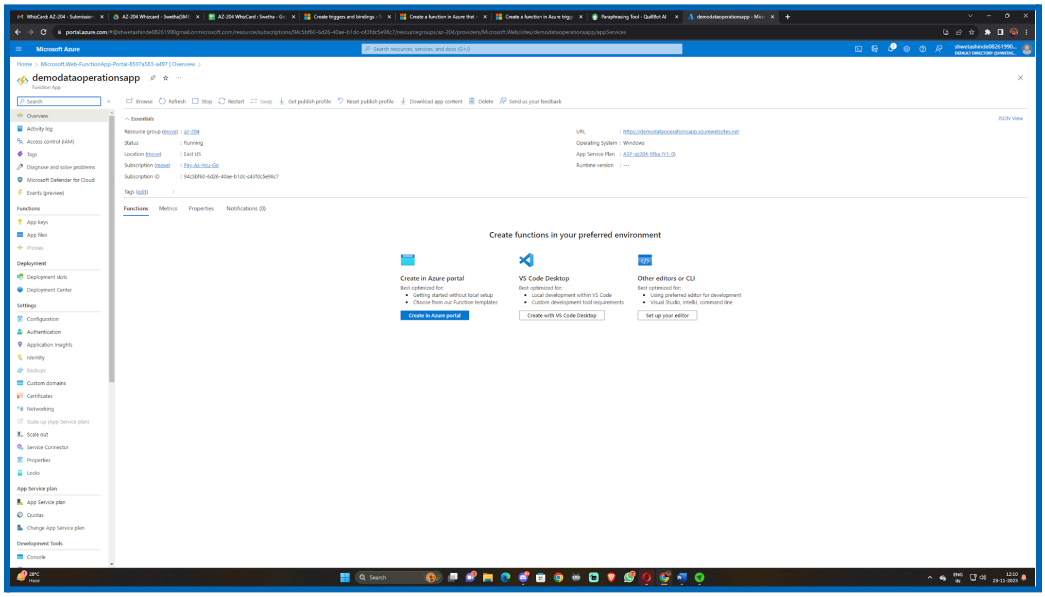
The direction is given in an attribute constructor or inferred from the parameter type when you use attributes in a class library to configure triggers and bindings.

A. Example of triggers and bindings using data Operations.

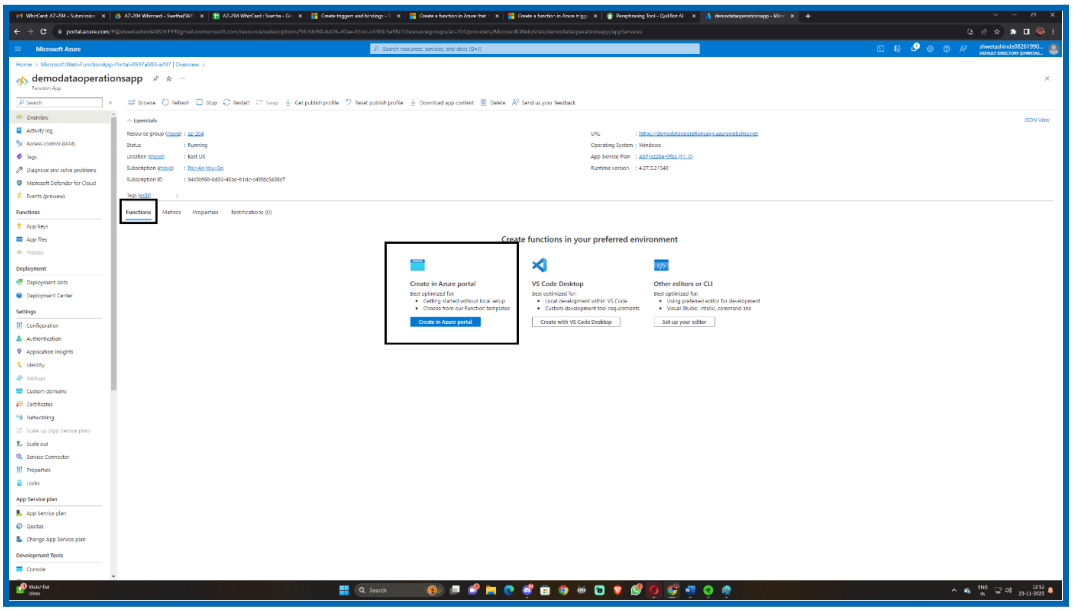
1. Choose Create a resource from the Azure portal menu or the Home page.
2. Select Compute > Function App from the New page.
3. Use the function app settings as per your choice on the Basics page:



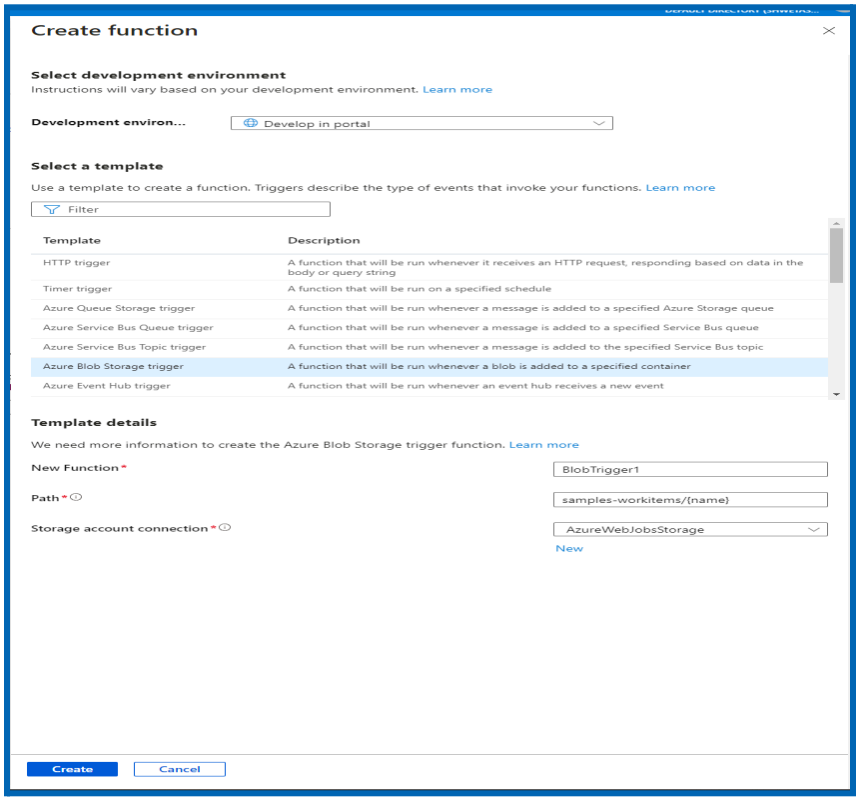
4. Accept the default settings to create a new Application Insight instance on the Monitoring tab and a new storage account on the Storage tab. Using an already-existing storage account or Application Insights instance is another option.
5. To examine the app configuration you selected, pick Review + Create. To provision and deploy the function app, select Create.
6. In order to examine your new function app, select Go to resource. You've successfully created your new function app.



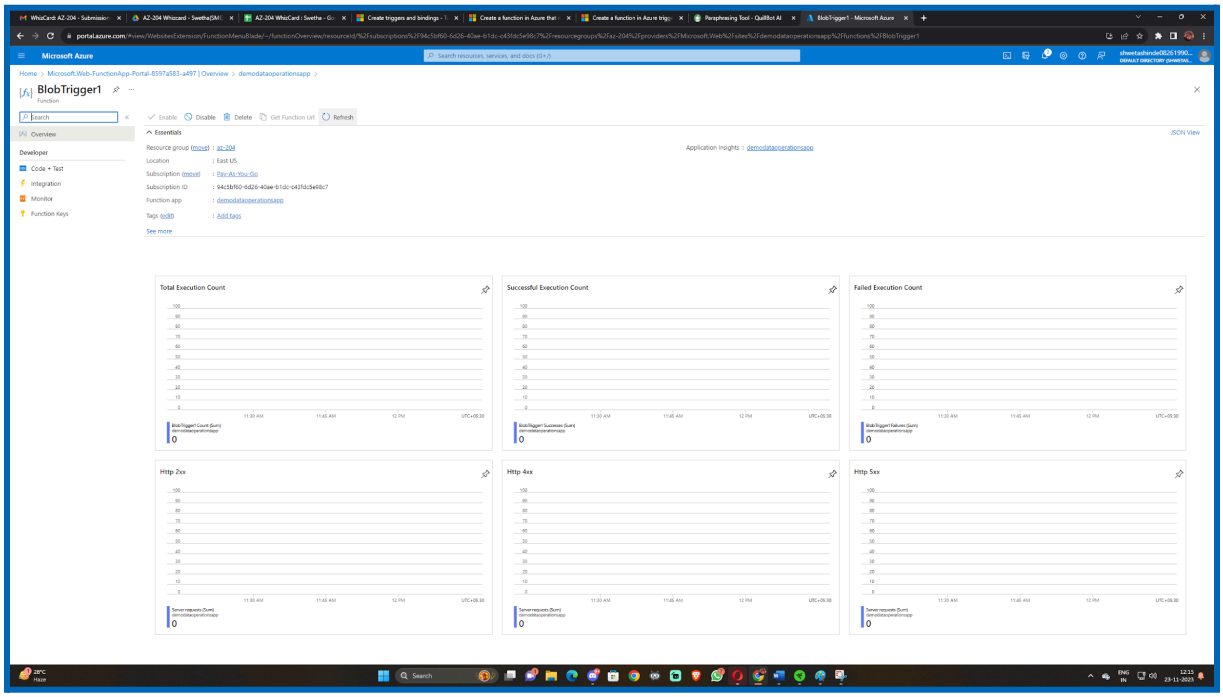
7. Next, you create a function in the new function app. Select create in azure portal from functions tab.



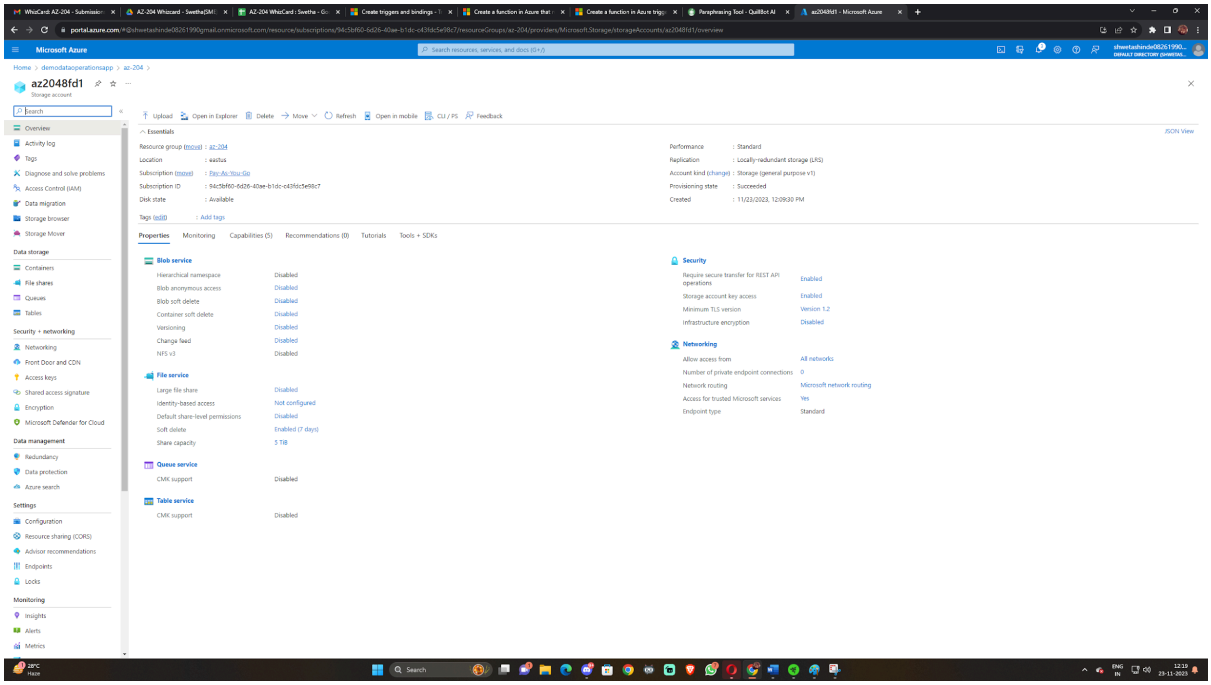
8. Choose the Azure Blob Storage trigger template.
9. Use the settings as specified in the image below .



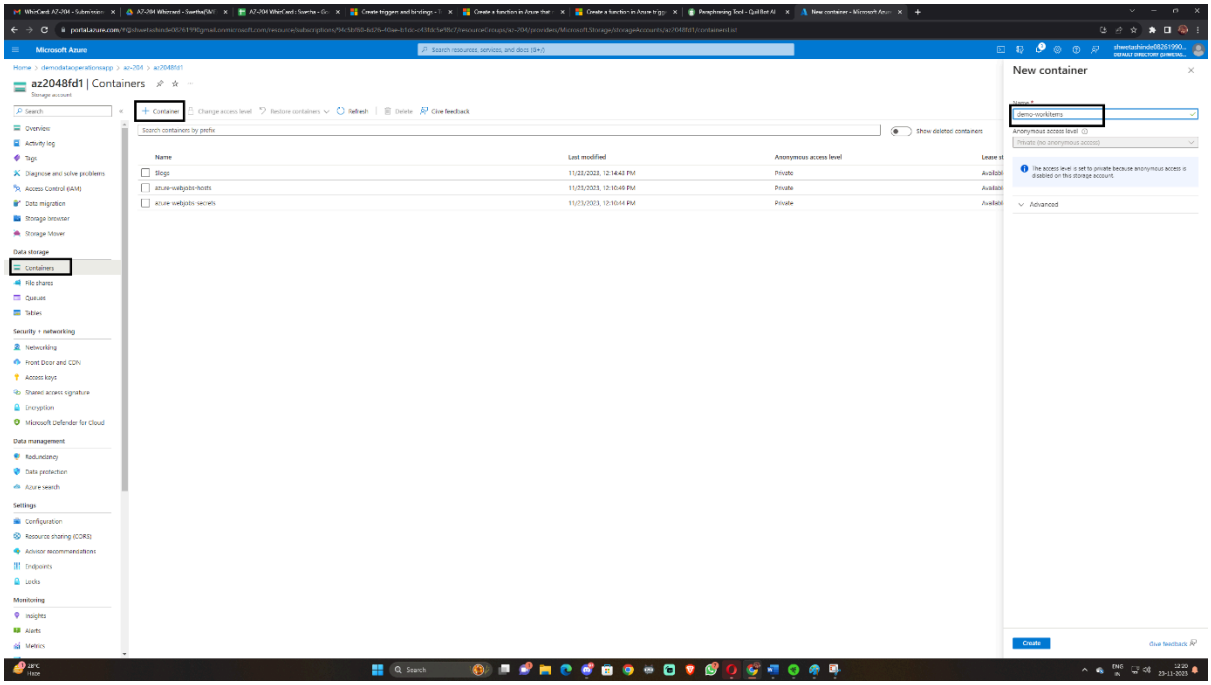
10. Select Create to create your function.



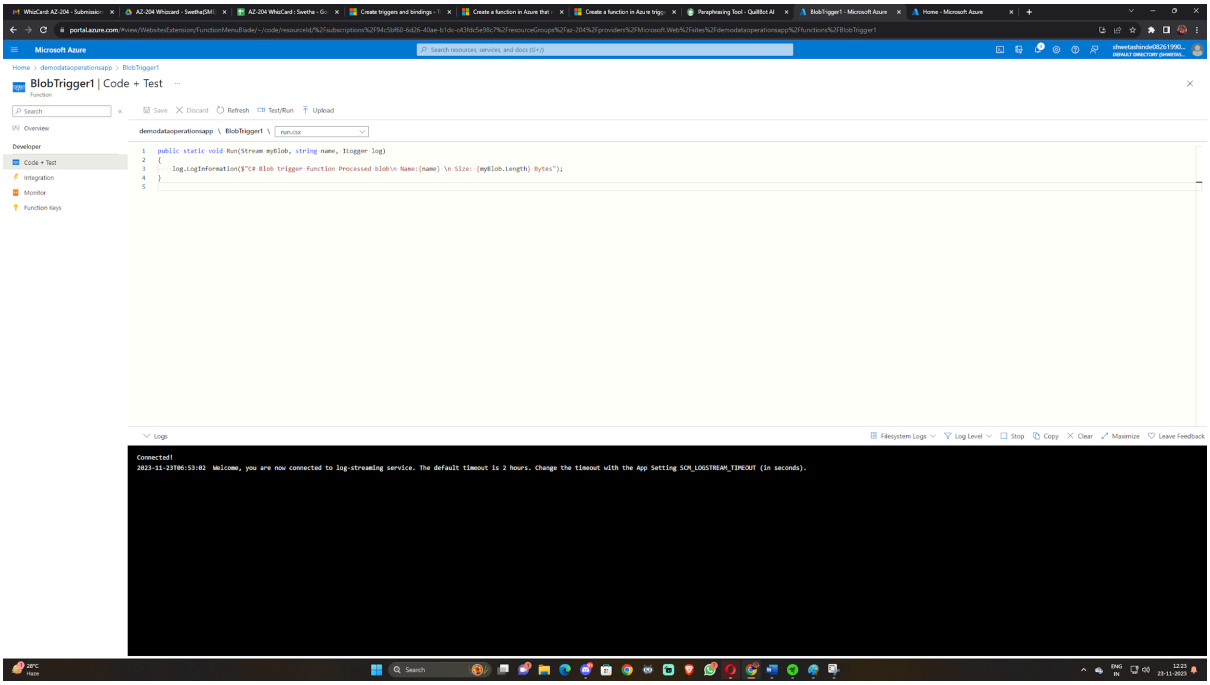
11. Next, create the demo-workitems container. Find your functions storage account and go to that storage account.



12. Choose Containers, and then choose + Container.
13. In the Name field, type demo-workitems, and then select Create.

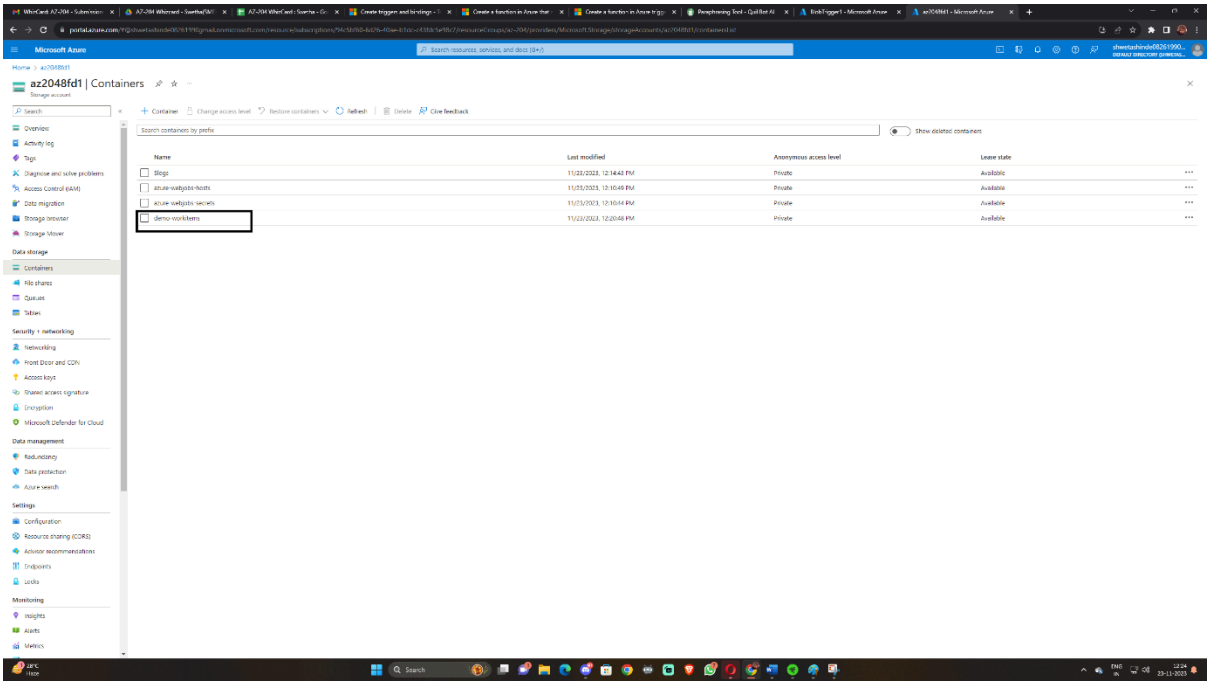


14. Now that you have a blob container, you can test the function by uploading a file to the container. Back in the Azure portal, browse to your function expand the Logs at the bottom of the page and make sure that log streaming isn't paused.

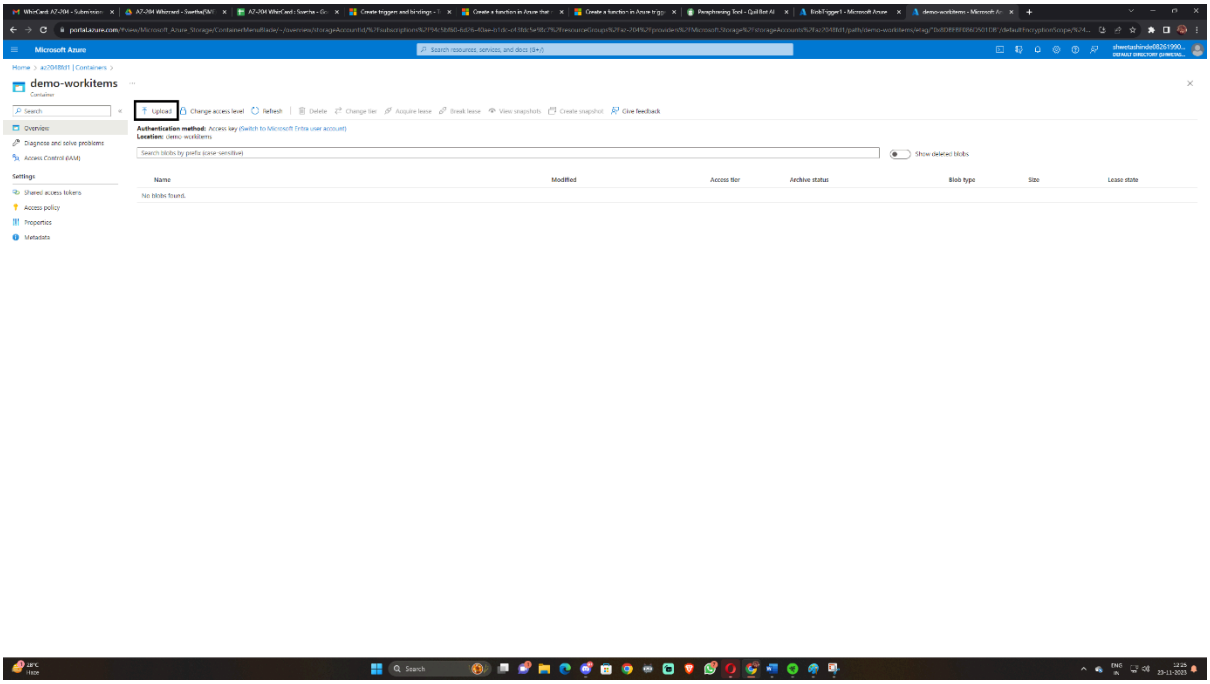


15. In a separate browser window, go to your resource group in the Azure portal, and select the storage account.

16. Select Containers, and then select the demo-workitems container.

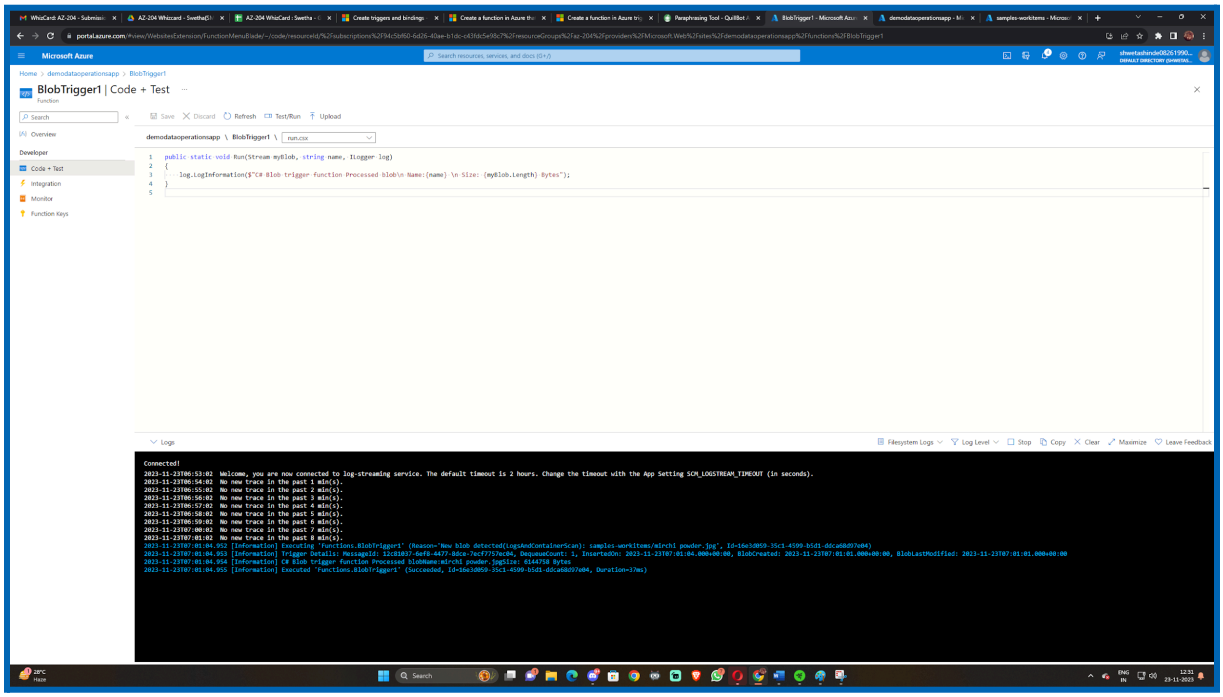


17. Select Upload, and then select the folder icon to choose a file to upload.



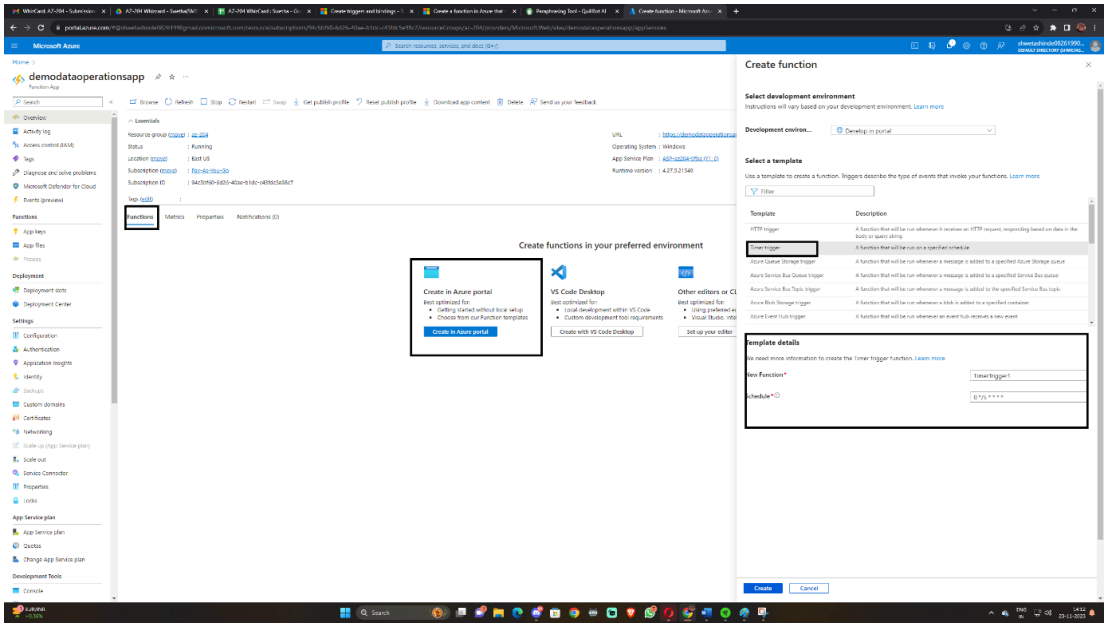
18. Browse to a file on your local computer, such as an image file, choose the file. Select Open and then Upload.

19. Go back to your function logs and verify that the blob has been read.

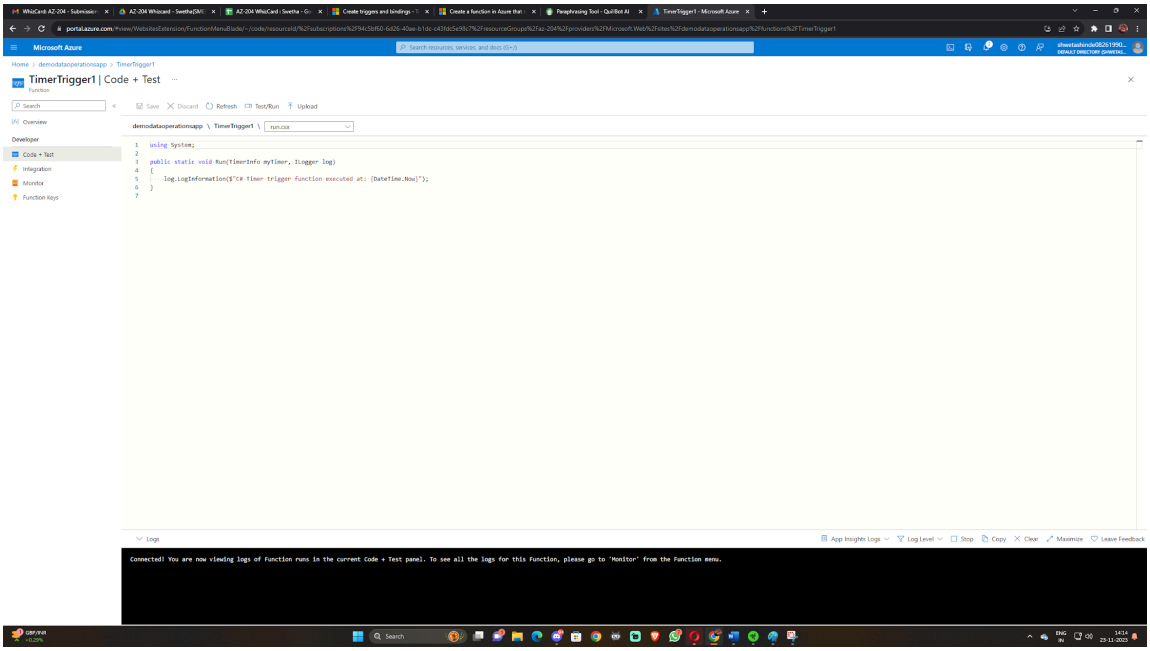


B. Example of triggers and bindings Timers.

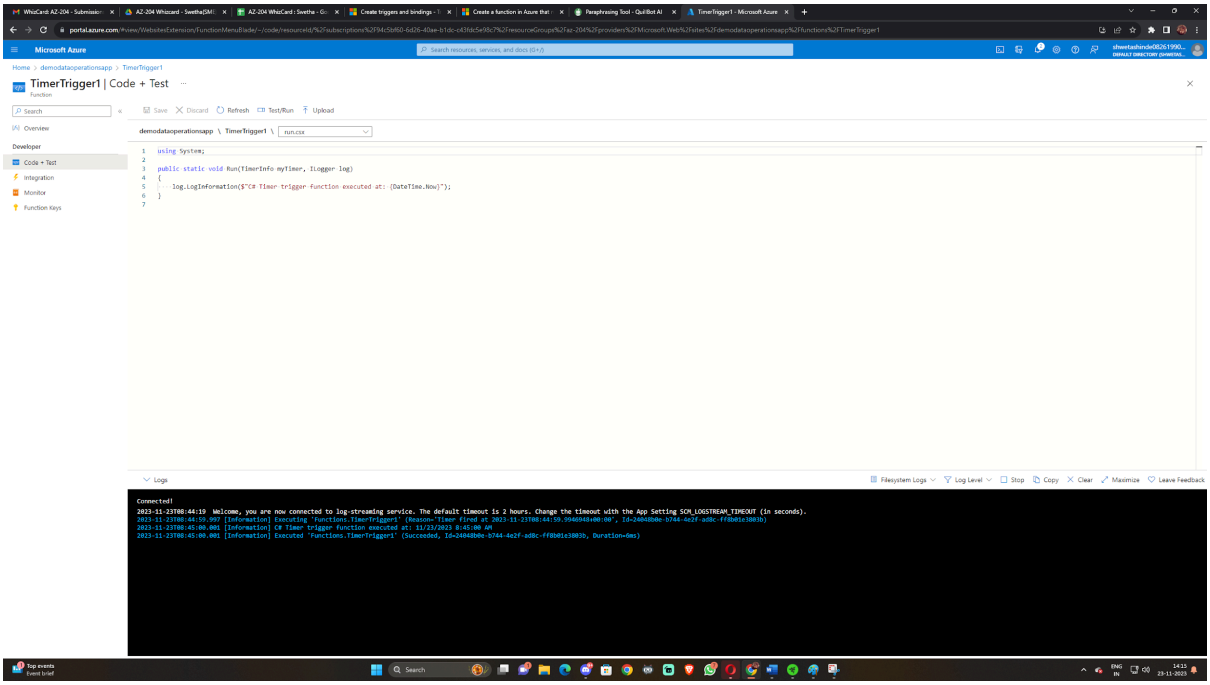
1. We will use the same function app created in previous example to create a timer function.
2. In your function app, select Functions, and then create in azure portal.
3. Select the Timer trigger template.
4. Configure the new trigger with the settings as specified in the image, and then select Create.



5. In your function, select Code + Test and expand the Logs.

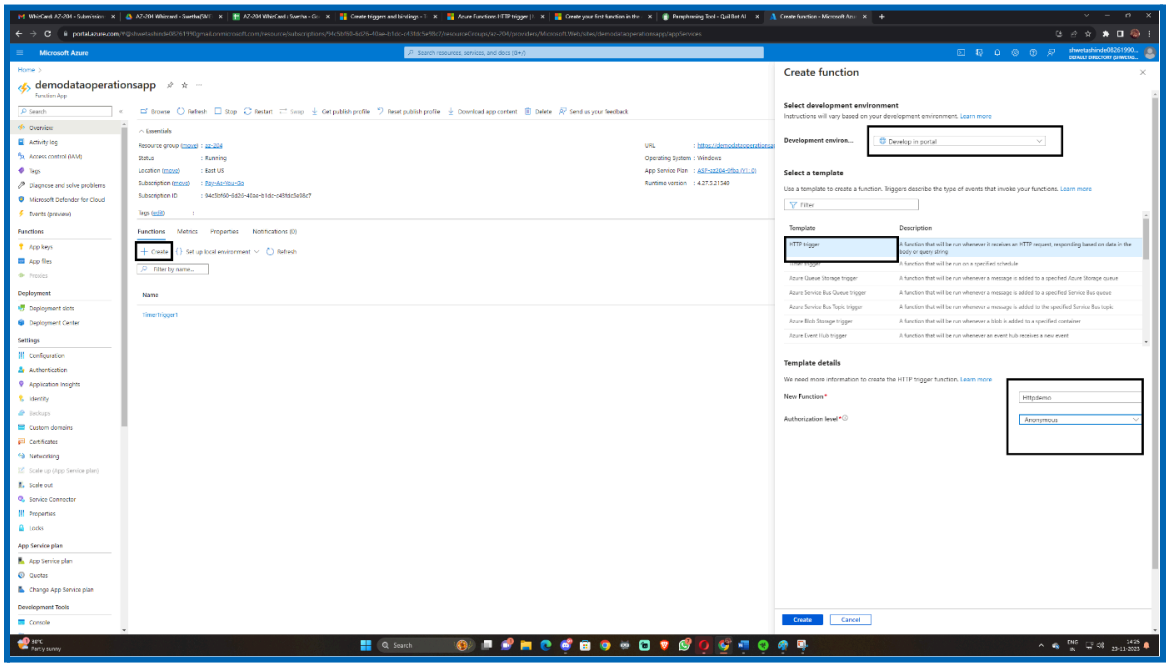


6. Verify execution by viewing the information written to the logs.

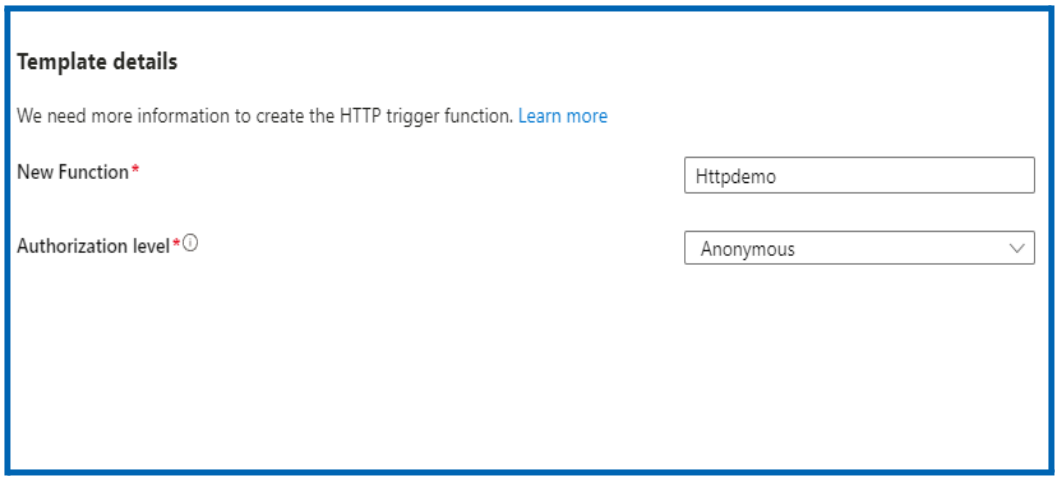


C. Example of triggers and bindings for webhooks.

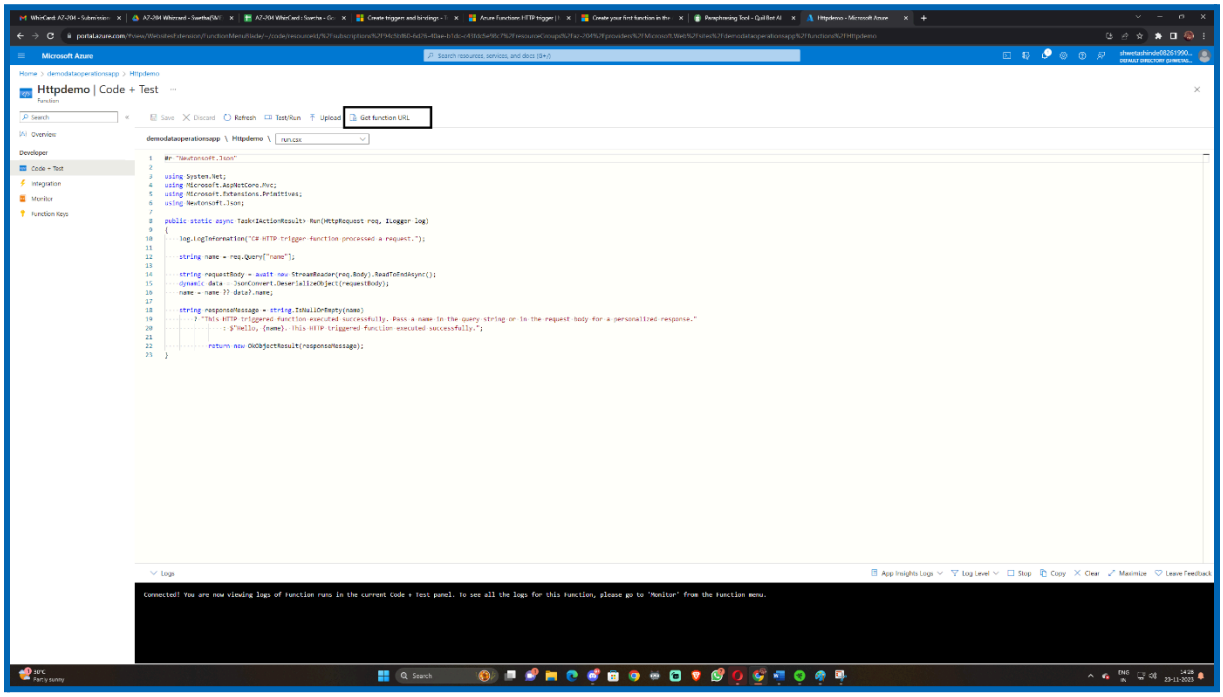
1. We will use the same function app created in previous example to create a timer function.
2. In your function app, select Functions, and then create in azure portal.
3. From the Create Function window, leave the Development environment property as Develop in portal, and then select the HTTP trigger template.



4. Under Template details use Httpdemo for New Function, select Anonymous from the Authorization level drop-down list, and then select Create. Azure creates the HTTP trigger function. Now, you can run the new function by sending an HTTP request.

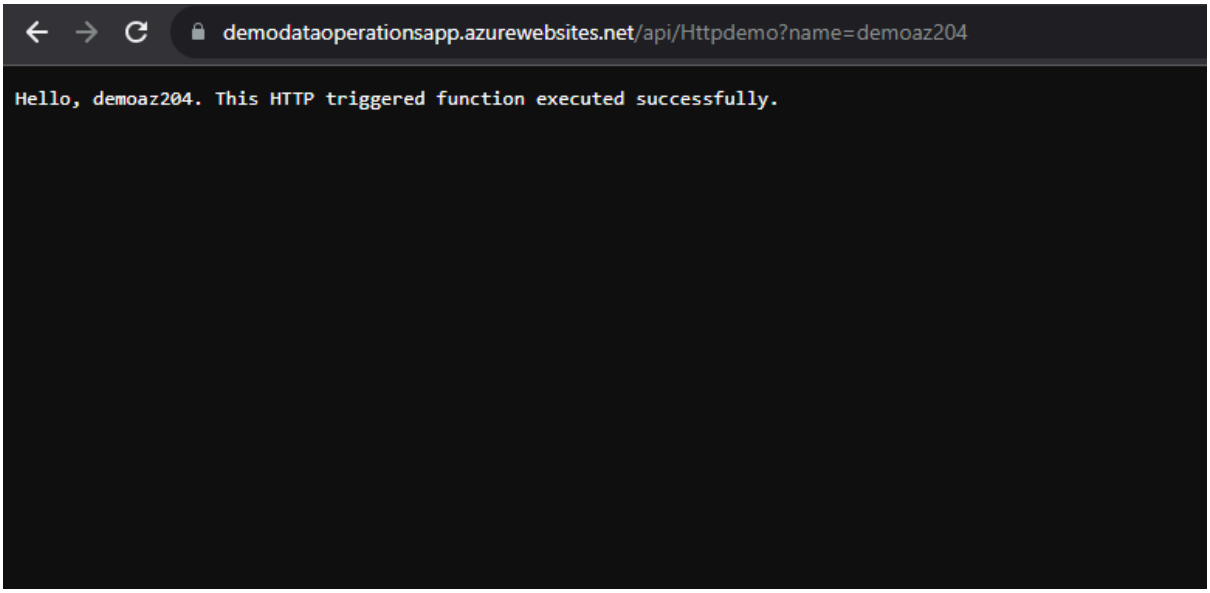


5. In your new HTTP trigger function, select Code + Test from the left menu, and then select Get function URL from the top menu.

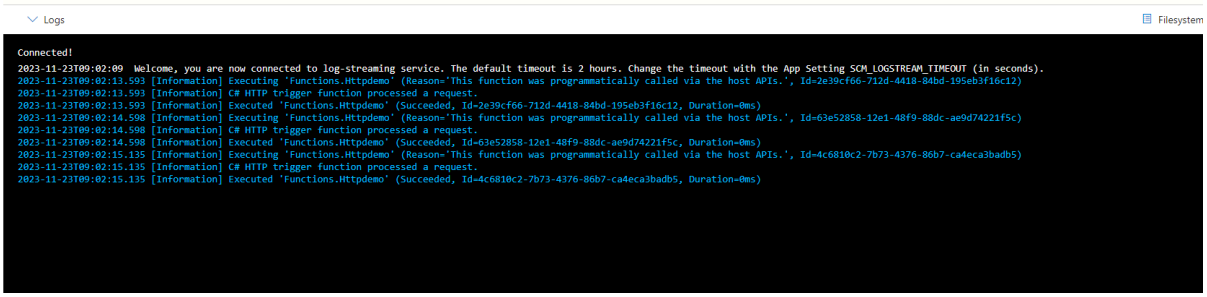


6. In the Get function URL dialog, select default from the drop-down list, and then select the Copy to clipboard icon.
7. Copy and paste the function URL to the address bar of your browser. at initiate the request, include the query string value?name=<your_name> at the end of this URL and hit Enter. Your query string value must be echoed back in a response message that the browser displays.

When defining the function, you chose Function above Anonymous access level if the request URL contained an access key (?code=...). Instead, you have to append &name=<your_name> in this situation.



8. Information about trace is published to the logs when your function executes. Expand the Logs arrow at the bottom of the Code + Test page on the gateway to view the trace output. To view the trace output written to the logs, call your function once again.



9. Once all the things done, please clean up your resources by deleting created functionapp, functions.

Azure Cosmos DB

Perform operations on containers and items

A fully managed NoSQL database, Azure Cosmos DB is intended to offer low latency, elastic throughput scaling, well-defined semantics for data consistency, and high availability.

Your databases can be set up to be globally dispersed and accessible across all Azure regions. Locate the data close to your consumers' locations to reduce latency. The locations of your users and the application's global reach will determine which regions are necessary.

You can add or remove regions linked to your account with Azure Cosmos DB at any moment. To add or remove a region, your application doesn't need to be stopped or redeployed.

Benefits of Global Distribution

Every area allows writes and reads thanks to its unique multi-master replication technology. Additionally, the multi-master capacity permits:

- elastic write and read scalability that never ends.
- Global read and write availability is 99.999%.
- At the 99th percentile, guaranteed reads and writes in less than 10 milliseconds are provided.

All of the regions you selected for your database can be read from and written to by your application in almost real-time. Data replication between regions is internally managed by Azure Cosmos DB, with consistency level assurances up to the level you've chosen.

A database is more readily available when it is operated at several different locations across the globe. Application requests are automatically handled by other locations in case one area is unavailable. For multi-region databases, Azure Cosmos DB provides 99.999% read and write availability.

Elements in Cosmos DB Account

At the core of scalability is an Azure Cosmos DB container. On a container, you can essentially have infinite storage and provisioned throughput (RU/s). To elastically scale your provided throughput and storage, Azure Cosmos DB transparently splits your container using the logical partition key that you supply.

With an Azure subscription, you can currently establish up to 50 Azure Cosmos DB accounts (this is a soft restriction that can be expanded via support request). You may manage the data in

your Azure subscription account by creating databases, containers, and things after you create an account.

1. Azure Cosmos DB databases - Under your account, you are able to build one or more Azure Cosmos DB databases. A namespace is comparable to a database. The management unit for a collection of Azure Cosmos DB containers is called a database.
2. Azure Cosmos DB containers - The unit of scalability for both provisioned throughput and storage is an Azure Cosmos DB container. A container is copied over several areas after being horizontally partitioned. Based on the partition key, the objects you add to the container are automatically organized into logical partitions that are dispersed over physical partitions. Every physical partition on a container has an equal amount of throughput.

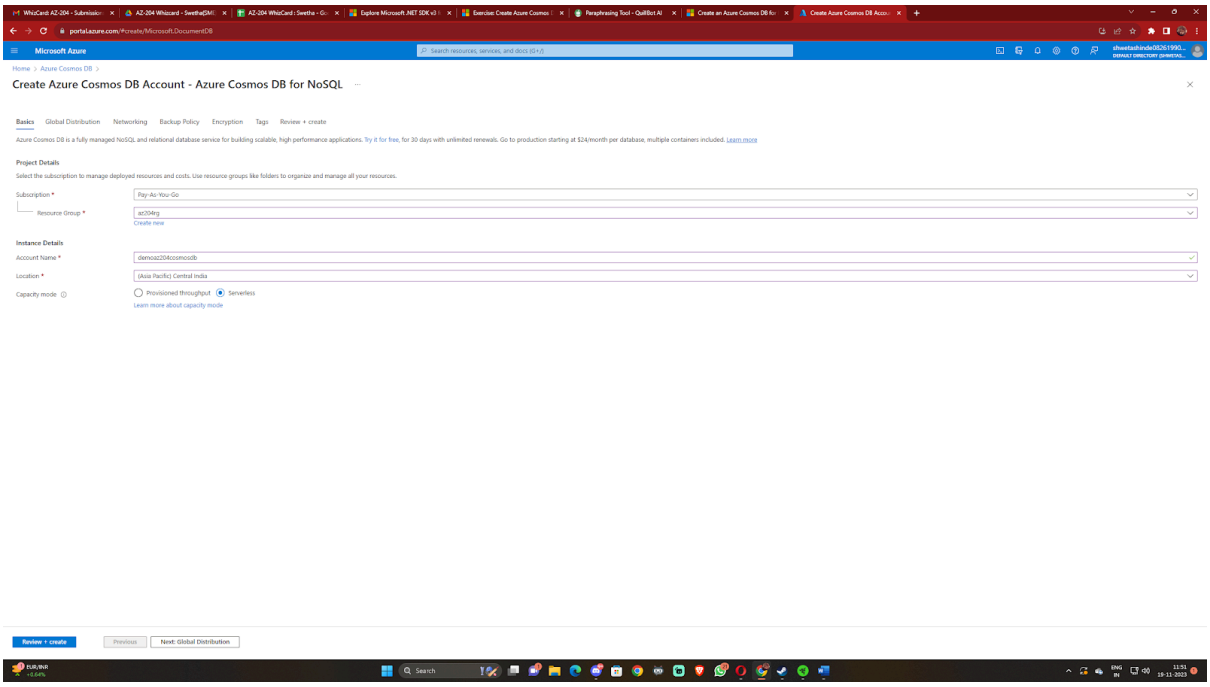
   One of the following modes is available for configuring throughput when creating a container:
   - Dedicated provisioned throughput mode: A container's allotted throughput is guaranteed by the SLAs and is only used by that container.
   - Shared provisioned throughput mode: With the exception of containers that are configured with dedicated supplied throughput, these containers share the provisioned throughput with the other containers in the same database. Stated differently, each of the "shared throughput" containers shares the database's provisioned throughput.

Implementation

1. First, we need to create an Azure Cosmos DB Account.
2. Log in to the Azure portal.
3. From the Azure portal navigation pane, select + Create a resource.
4. Search for Azure Cosmos DB, then select Create/Azure Cosmos DB to get started.
5. Which API best suits your workload? page, select Create in the Azure Cosmos DB for NoSQL box.
6. In the Create Azure Cosmos DB Account - Azure Cosmos DB for NoSQL page, enter the basic settings for the new Azure Cosmos DB account.
   - Subscription: Select the subscription you want to use.
   - Resource Group: Select Create new if you don’t have one or choose already create one eg.az204rg.
   - Account Name: Enter a unique name to identify your Azure Cosmos account. The name can only contain lowercase letters, numbers, and the hyphen (-) character. It must be between 3-31 characters in length.

- Location: Use the location that is closest to your users to give them the fastest access to the data.
- Capacity mode: Select Serverless.

7. Choose Review + Create.



8. After checking the account settings, choose Create. A few minutes are needed to create the account. Await the portal page's appearance. The deployment has ended for you.
9. Choose To access the Azure Cosmos DB account page, navigate to resource.
10. Now we will create .NET application to do interaction with created azure Cosmos DB Using the console template and the dotnet new command, create a new.NET application.

```
C:\Users\wildc>dotnet new console -n democosmos
The template "Console App" was created successfully.

Processing post-creation actions...
Restoring C:\Users\wildc\democosmos\democosmos.csproj:
  Determining projects to restore...
  Restored C:\Users\wildc\democosmos\democosmos.csproj (in 80 ms).
Restore succeeded.
```

11. Import the Microsoft.Azure.Cosmos NuGet package using the dotnet add package command.

```
C:\Users\wildc>dotnet add package Microsoft.Azure.Cosmos
  Determining projects to restore...
  Writing C:\Users\wildc\AppData\Local\Temp\tmp665F.tmp
info : X.509 certificate chain validation will use the default trust store selected by .NET for code signing.
info : X.509 certificate chain validation will use the default trust store selected by .NET for timestamping.
info : Adding PackageReference for package 'Microsoft.Azure.Cosmos' into project 'C:\Users\wildc\wildc.csproj'.
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/microsoft.azure.cosmos/index.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/microsoft.azure.cosmos/index.json 989ms
info : Restoring packages for C:\Users\wildc\wildc.csproj...
info :   GET https://api.nuget.org/v3-flatcontainer/microsoft.azure.cosmos/index.json
info :   OK https://api.nuget.org/v3-flatcontainer/microsoft.azure.cosmos/index.json 963ms
info :   GET https://api.nuget.org/v3-flatcontainer/microsoft.azure.cosmos/3.36.0/microsoft.azure.cosmos.3.36.0.nupkg
info :   OK https://api.nuget.org/v3-flatcontainer/microsoft.azure.cosmos/3.36.0/microsoft.azure.cosmos.3.36.0.nupkg 16m
s
info :   GET https://api.nuget.org/v3-flatcontainer/azure.core/index.json
info :   GET https://api.nuget.org/v3-flatcontainer/microsoft.bcl.hashcode/index.json
```

12. Build the project with the dotnet build command.

```
C:\Users\wildc\democosmos>dotnet build
MSBuild version 17.7.1+971bf70db for .NET
  Determining projects to restore...
  All projects are up-to-date for restore.
  democosmos -> C:\Users\wildc\democosmos\bin\Debug\net7.0\democosmos.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:01.75
```

Make an instance of the CosmosClient class in order to establish a connection to the Azure Cosmos DB NoSQL API. The first step in carrying out any database activity is to take this class. Using the CosmosClient class, there are three main methods to establish a connection to an API for a NoSQL account:

- Establish a connection to a NoSQL endpoint API and read/write a key.
- Use an API to establish a NoSQL connection string.
- Make a Microsoft Entra ID connection.

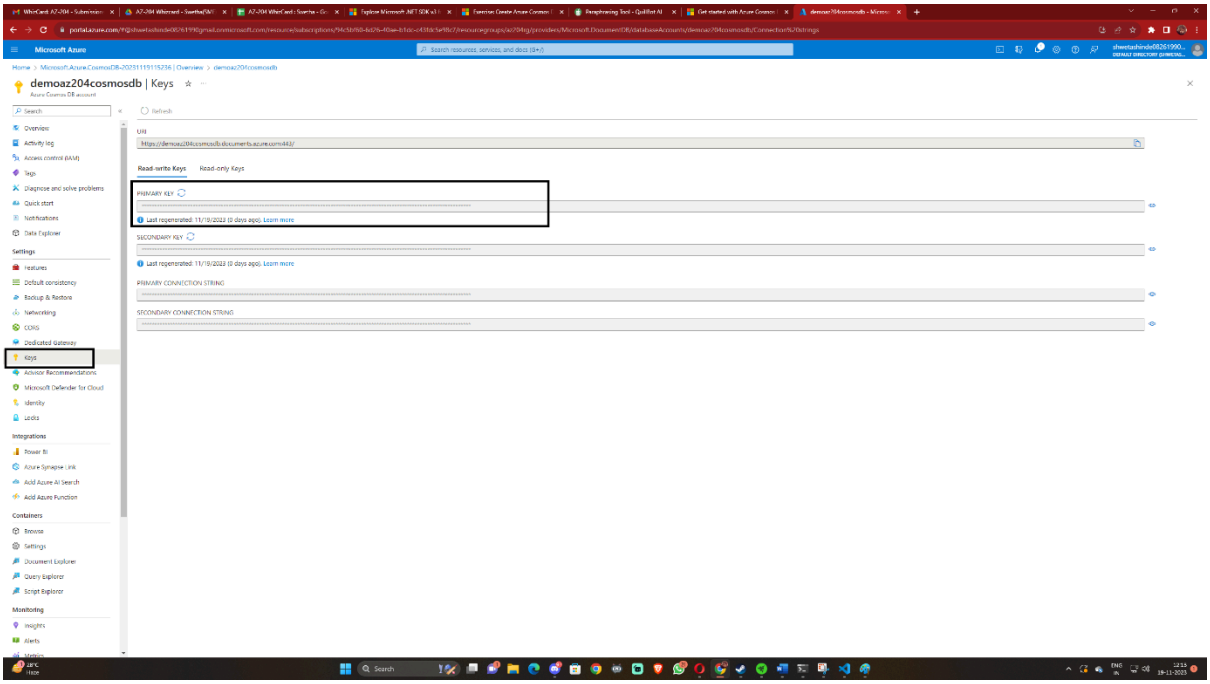
Connect with an endpoint and key

Two parameters are present in the most popular CosmosClient constructor:

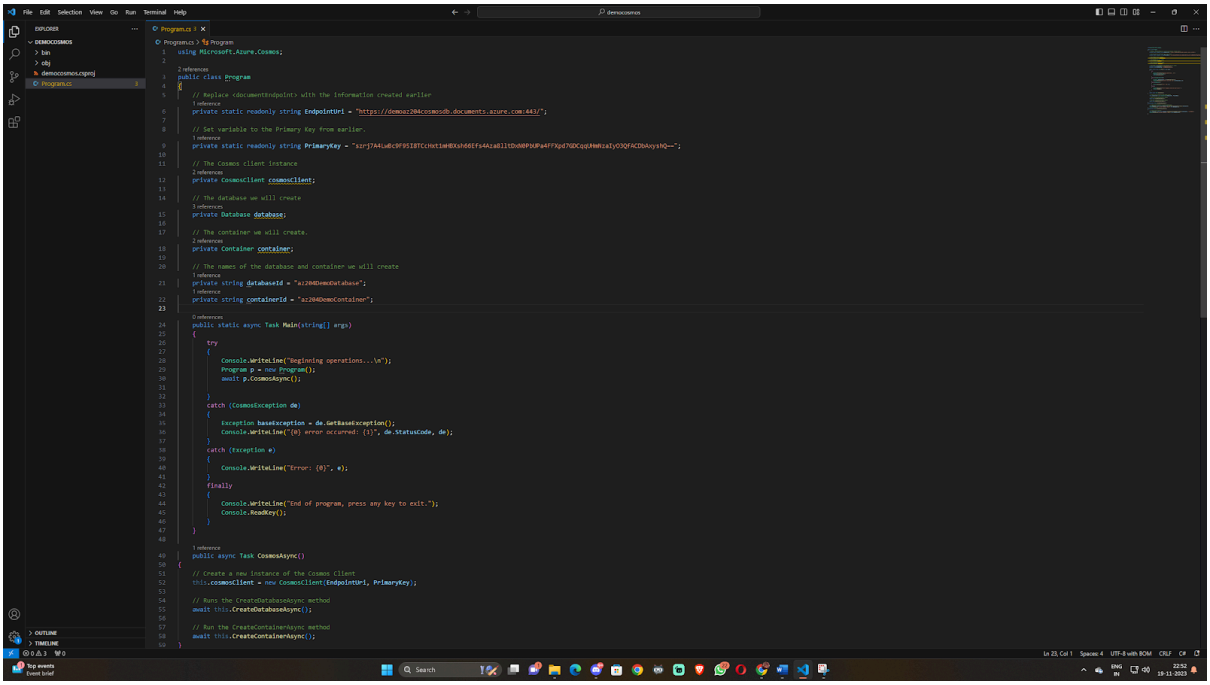
1. accountEndpoint: API for NoSQL endpoint to use for all requests.
2. authKeyOrResourceToken: Account key or resource token to use when authenticating

Retrieve your account endpoint and key

13. Open the Azure portal and log in.
14. Go to the Azure Cosmos DB for NoSQL account page that is currently there.
15. Click the Keys menu item from the Azure Cosmos DB for NoSQL account page.
16. Note the contents of the fields labelled "PRIMARY KEY" and "URI." These values will be used in a later phase.



17. Replace any existing code with the using Microsoft.Azure.Cosmos statement in the Program.cs file.
18. After the using statement, insert the following bit of code. The code snippet incorporates some error checking along with constants and variables into the class. Make sure you update the EndpointUri and PrimaryKey placeholder values in accordance with the instructions provided in the code comments.



19. Establish a new asynchronous task named CosmosAsync underneath the Main function. It adds code to call the methods you add later to establish a database and a container, as well as instantiates our new CosmosClient.

```
1 reference
public async Task CosmosAsync()
{
    // Create a new instance of the Cosmos Client
    this.cosmosClient = new CosmosClient(EndpointUri, PrimaryKey);

    // Runs the CreateDatabaseAsync method
    await this.CreateDatabaseAsync();

    // Run the CreateContainerAsync method
    await this.CreateContainerAsync();
}
```

20. Write the CreateDatabaseAsync method after the CosmosAsync method. CreateDatabaseAsync creates a new database with ID az204DemoDatabase if it doesn't already exist.

```
1 reference
private async Task CreateDatabaseAsync()
{
    // Create a new database using the cosmosClient
    this.database = await this.cosmosClient.CreateDatabaseIfNotExistsAsync(databaseId);
    Console.WriteLine("Created Database: {0}\n", this.database.Id);
}
```

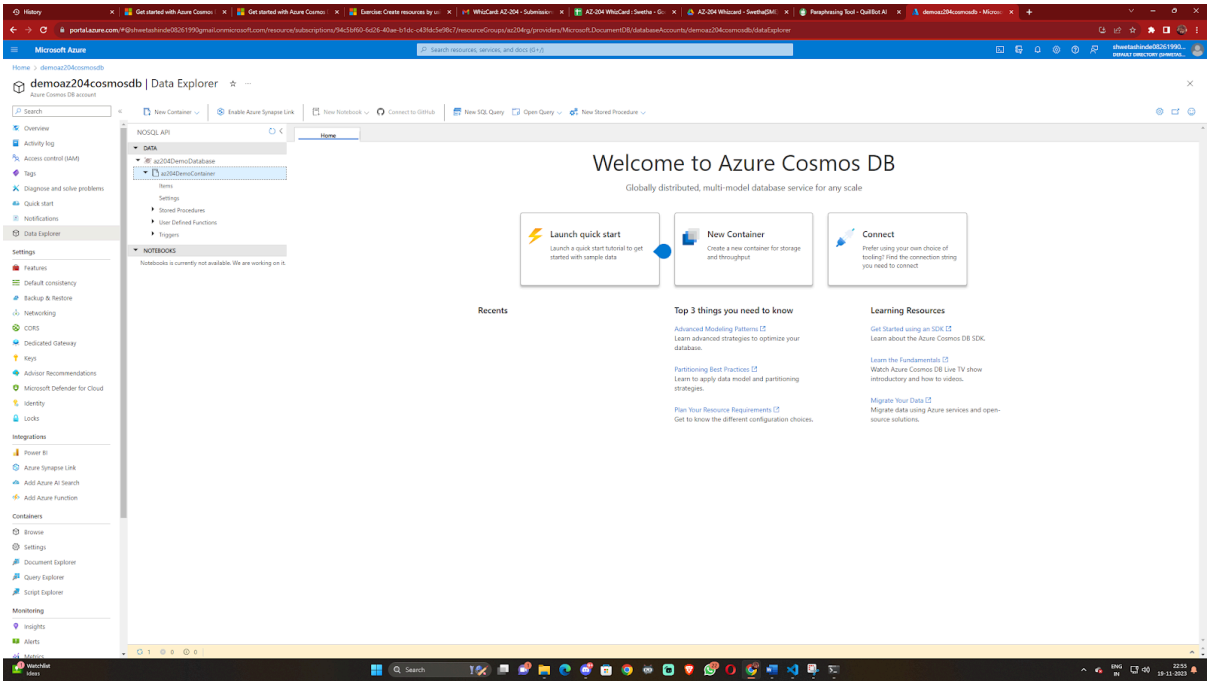
21. Copy and paste the CreateContainerAsync method below the CreateDatabaseAsync method.

```
private async Task CreateContainerAsync()
{
    // Create a new container
    this.container = await this.database.CreateContainerIfNotExistsAsync(containerId, "/LastName");
    Console.WriteLine("Created Container: {0}\n", this.container.Id);
}
```

22. After saving your work, execute the dotnet build command in a Visual Studio Code terminal to see if there are any issues. Run the dotnet run command if the build is successful. The following messages are shown on the console.

```
C:\Users\wildc\democosmos>dotnet run
Beginning operations...

Created Database: az204DemoDatabase

Created Container: az204DemoContainer

End of program, press any key to exit.
```

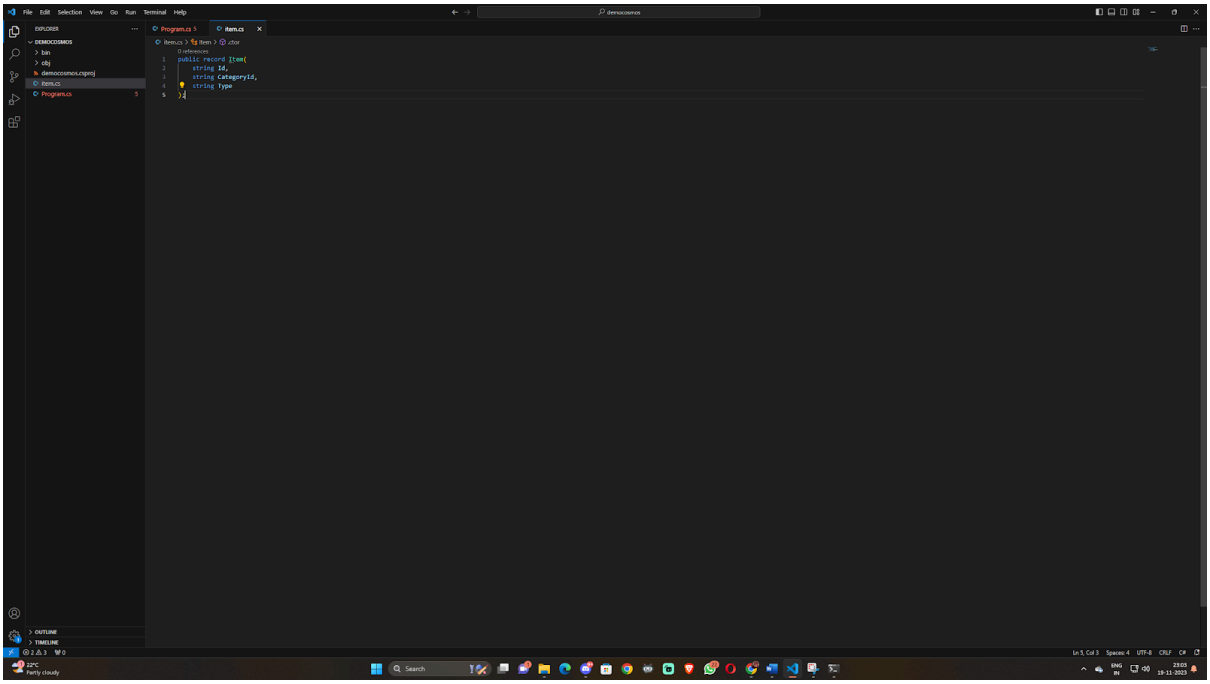
23. Verify the creation of database and container in azure portal.



24. Now we will create an item in the created container. Using Visual Studio Code, create a new file named Item.cs. Then, open the file in the editor.

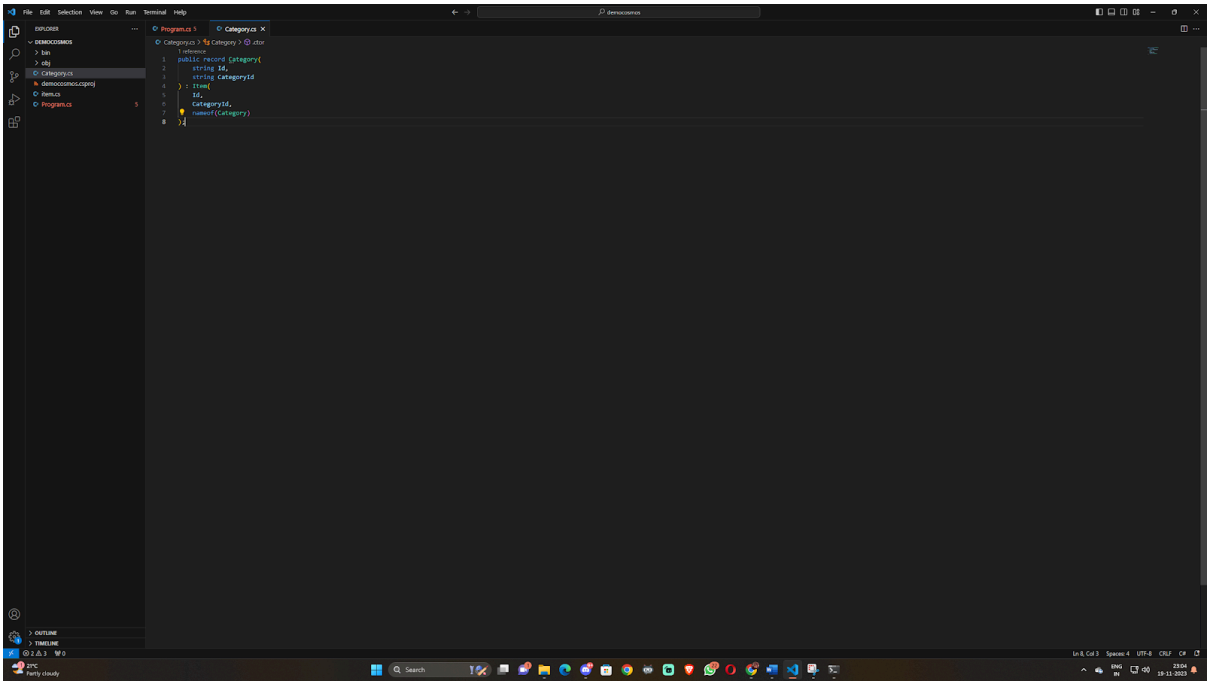
    Create a base record type named Item that carries the three properties you want to use in all items for this container: id, categoryId, and type.

25. Save the Item.cs file. Close the Item.cs file.



26. Create another new file named Category.cs. Open this file in the editor now.

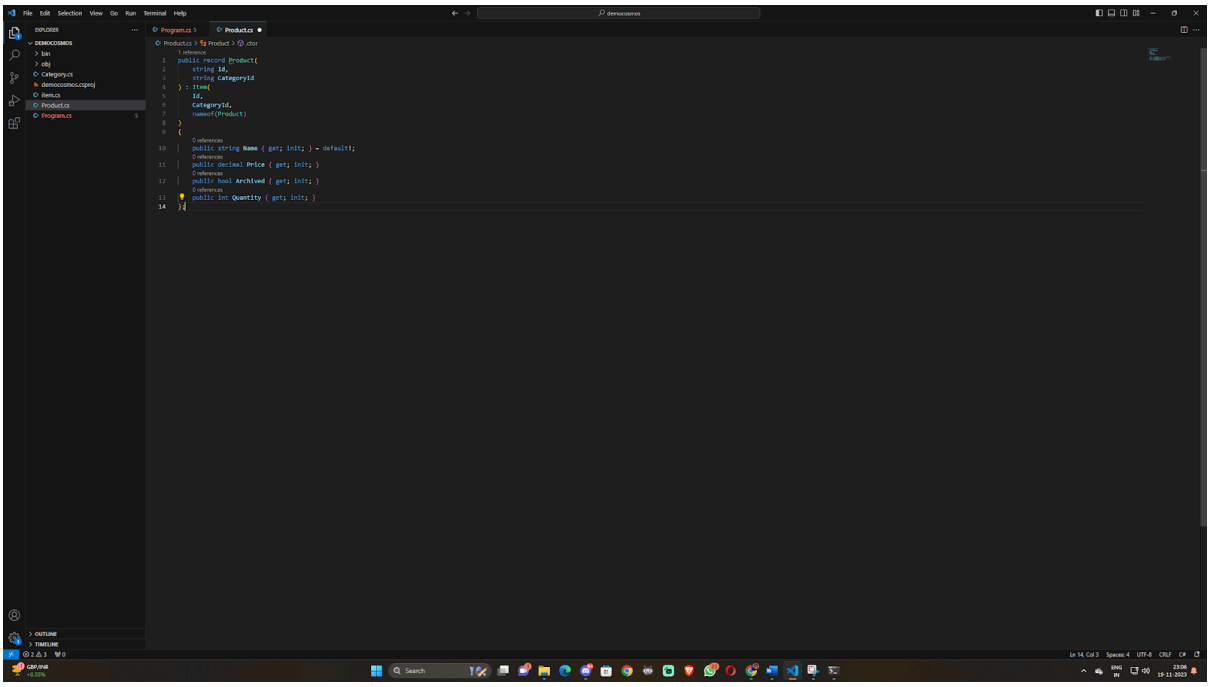
27. Create a new type named Category that inherits from the Item type. Ensure the type passes its values to the base implementation, and set the type variable to output the name of the Category type.



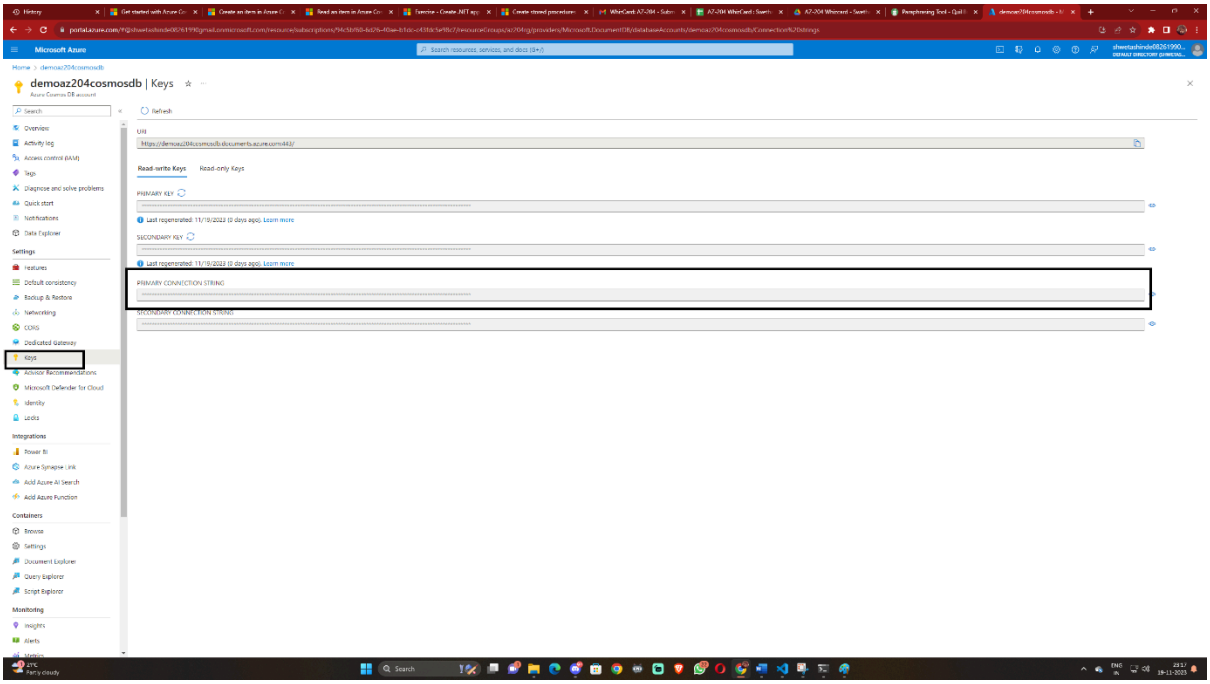
28. Save the Category.cs file. Close the Category.cs file.

29. Finally, create one last file named Product.cs. Open this file in the editor too.

30. Create a new type named Product that inherits from Item and adds a few new properties: name, price, archived, and quantity.



31. Save the Product.cs file. Close the Product.cs file.
32. To get the access to our container and database copy the connection string from keys in Azure Cosmos DB storage.



33. Open program.cs file and write below code in it.

```
Program.cs
using Microsoft.Azure.Cosmos;
using Microsoft.Azure.Cosmos.Fluent;
using Microsoft.Azure.Cosmos.Linq;



const string connectionString = "AccountEndpoint=https://demoaz204cosmosdb.documents.azure.com:443/;AccountKey=szrj7A4LwBc9F95I8TCcHxt1mHBXsh66Efs4Aza8lltDxN0PbUPa4FFXpd7GDCqqUHmNzaIyO3QfACDbAxyshQ==;";

Console.WriteLine($"[Connection string]:\t{connectionString}");

CosmosSerializationOptions serializerOptions = new()
{
    PropertyNamingPolicy = CosmosPropertyNamingPolicy.CamelCase
};

using CosmosClient client = new CosmosClientBuilder(connectionString)
    .WithSerializerOptions(serializerOptions)
    .Build();

Console.WriteLine("[Client ready]");

```

34. In the program.cs file now create a new individual item.

```
Category book = new(
    Id: "1",
    CategoryId: "fiction-book"
);
```

35. Create a new PartitionKey instance using the same value as the categoryId property for the Category instance you created earlier.

```
PartitionKey fictionKey = new("fiction-book");
```

36. Use the UpsertItemAsync method to create or replace the item passing in an object for the item to create and a partition key value.

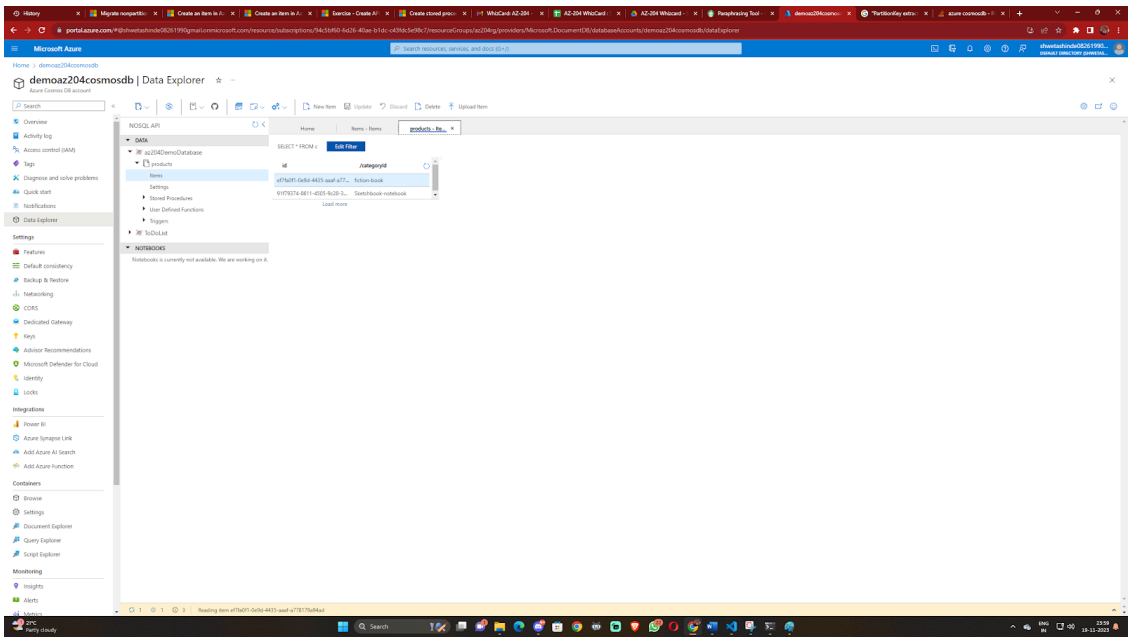
```
Category result = await container.UpsertItemAsync(books, fictionKey);
```

37. Print various properties of response to the console including: The unique identifier of the underlying item, the type of the underlying item, and the request charge in RUs.

```
Console.WriteLine($"[New item created]:\t{result.Id}\t(Type: {result.Type})");
```

38. Save the code and run the code through the console. New item will be created in container.

```
C:\Users\wildc\democosmos>dotnet run
C:\Users\wildc\democosmos\Program.cs(20,26): warning CS8618: Non-nullable field 'cosmosClient' must contain a non-null
value when exiting constructor. Consider declaring the field as nullable. [C:\Users\wildc\democosmos\democosmos.csproj]
C:\Users\wildc\democosmos\Program.cs(23,22): warning CS8618: Non-nullable field 'database' must contain a non-null valu
e when exiting constructor. Consider declaring the field as nullable. [C:\Users\wildc\democosmos\democosmos.csproj]
C:\Users\wildc\democosmos\Program.cs(26,23): warning CS8618: Non-nullable field 'container' must contain a non-null val
ue when exiting constructor. Consider declaring the field as nullable. [C:\Users\wildc\democosmos\democosmos.csproj]
C:\Users\wildc\democosmos\Program.cs(30,20): warning CS0414: The field 'Program.containerId' is assigned but its value
is never used [C:\Users\wildc\democosmos\democosmos.csproj]
Beginning operations...

Created Database: az204DemoDatabase

Created Container: products

[New item created]:     ef7fa0f1-0e9d-4435-aaaf-a778179a94ad    (Type: Category)
[New item created]:     91f79374-8611-4505-9c28-3bbbf1aa7df7    (Type: Category)        (RUs: 6.29)
End of program, press any key to exit.
```



39. Create a new PartitionKey instance for Sketchbook-notebook.

```
PartitionKey readKey = new("Sketchbook-notebook");
```

40. Use Container.ReadItemAsync to point read a specific item by using the id property and the partition key value.

```
ItemResponse<Category> readResponse = await container.ReadItemAsync<Category>(
    id: "91f79374-8611-4505-9c28-3bbbf1aa7df7",
    partitionKey: readKey
);
```

41. Get your serialized generic type using the Resource property of the ItemResponse class.

```
Category readItem = readResponse.Resource;
```

42. Output the unique identifier and request charge for the point read operation.

```
Console.WriteLine($"[Point read item]:\t{readItem.id}\t(RUs: {readResponse.RequestCharge})");
```

43. Run the code and you will get below kind of result in the command prompt window.

```
C:\Users\wildc\democosmos>dotnet run
C:\Users\wildc\democosmos\Program.cs(20,26): warning CS8618: Non-nullable field 'cosmosClient' must contain a non-null
value when exiting constructor. Consider declaring the field as nullable. [C:\Users\wildc\democosmos\democosmos.csproj]
C:\Users\wildc\democosmos\Program.cs(23,22): warning CS8618: Non-nullable field 'database' must contain a non-null valu
e when exiting constructor. Consider declaring the field as nullable. [C:\Users\wildc\democosmos\democosmos.csproj]
C:\Users\wildc\democosmos\Program.cs(26,23): warning CS8618: Non-nullable field 'container' must contain a non-null val
ue when exiting constructor. Consider declaring the field as nullable. [C:\Users\wildc\democosmos\democosmos.csproj]
C:\Users\wildc\democosmos\Program.cs(30,20): warning CS0414: The field 'Program.containerId' is assigned but its value
is never used [C:\Users\wildc\democosmos\democosmos.csproj]
Beginning operations...

Created Database: az204DemoDatabase

Created Container: products

[Point read item]:      91f79374-8611-4505-9c28-3bbbf1aa7df7    (RUs: 1)
```

Azure Cosmos DB

Consistency Level & Change Feed Notifications

Consistency levels

Instead of seeing data consistency as two extremes, Azure Cosmos DB views it as a spectrum of options. There are several consistency options all over the spectrum, with eventual consistency and strong consistency being at opposite ends. These options enable developers to make fine-grained decisions and trade-offs regarding performance and high availability.

Azure Cosmos DB offers five well-defined levels. From strongest to weakest, the levels are:

- Strong
- Bounded staleness
- Session
- Consistent prefix
- Eventual

Each level provides availability and performance trade-offs. The following image shows the different consistency levels as a spectrum.

Image ref : https://learn.microsoft.com/en-us/azure/cosmos-db/consistency-levels

Regardless of the region from which the reads and writes are delivered, the number of regions linked to your Azure Cosmos DB account, or whether your account is set up with a single or multiple write region, the consistency levels are region-agnostic and guaranteed for all operations.

A single read operation scoped within a logical partition or a partition-key range is covered by read consistency. Either a stored process or a remote client can initiate the read action.

Default consistency level

On your Azure Cosmos DB account, you can change the default consistency level at any time. All Azure Cosmos DB databases and containers associated with your account are subject to the default consistency level that you have set. By default, every read and query made against a database or container uses the set consistency level.

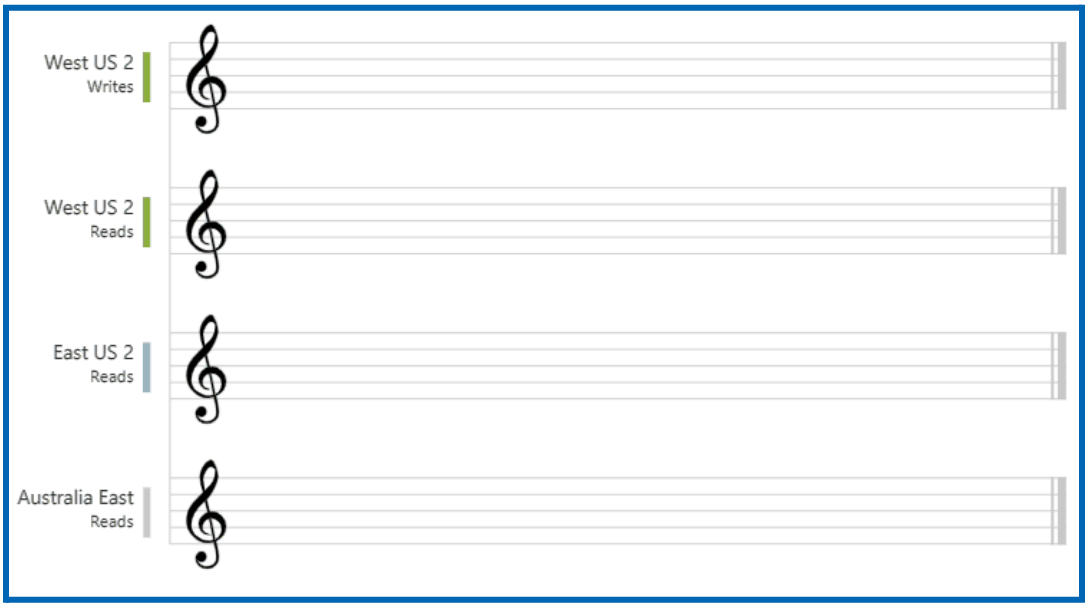
A single read operation scoped inside a logical partition is covered by read consistency. Either a stored process or a remote client can initiate the read action.

Guarantees associated with consistency levels

Azure Cosmos DB guarantees that 100 percent of read requests meet the consistency guarantee for the consistency level chosen.

The following sections provide a description of the semantics of the five consistency levels.

1. **Strong consistency:** Strong consistency provides a guarantee of linearizability. Concurrently fulfilling requests is referred to as linearizability. It is assured that the readings will return an item's most recent committed version. An incomplete or uncommitted write is never seen by the client. It is always assured that users will read the most recent commit.

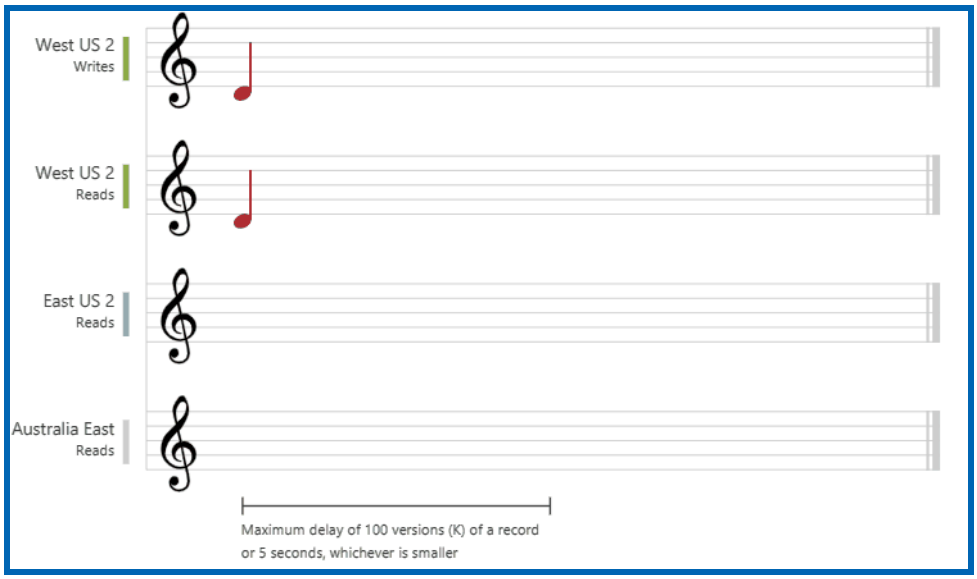


(Image ref: https://learn.microsoft.com/en-us/azure/cosmos-db/consistency-levels)

2. **Bounded staleness consistency:** The reads are guaranteed to respect the consistent-prefix guarantee in constrained staleness consistency. At most "K" versions (also known as "updates") of an item or a "T" time interval could cause the reads to lag behind the writes; whichever happens first will take precedence. Put differently, there are two configuration options for "staleness" when you select constrained staleness:

    - How many variants (K) there are of the item
    - It's possible for the time interval (T) readings to trail the writes.

    The minimum value of K and T for a single region account is 10 write operations, or 5 seconds. The minimum value of K and T for multi-region accounts is 100,000 write operations, or 300 seconds.



(Image ref: https://learn.microsoft.com/en-us/azure/cosmos-db/consistency-levels)

3. **Session consistency:** Reads are guaranteed to respect the consistent-prefix, monotonic reads, monotonic writes, read-your-writes, and write-follows-reads guarantees in session consistency inside a single client session. This presupposes that there is only one "writer" session or that numerous writers share the session token.

Image ref: https://learn.microsoft.com/en-us/azure/cosmos-db/consistency-levels

4. **Consistent prefix consistency:** Updates conducted as a single document writes eventually see consistency in consistent prefix. Updates committed as part of a batch within a transaction are returned in a manner consistent with the original transaction. Write actions are always displayed concurrently in a transaction involving many documents.

Assume that within transactions T1 and T2, two write operations are carried out on documents Doc 1 and Doc 2. The user never sees "Doc 1 v1 and Doc 2 v2" or "Doc 1 v2 and Doc 2 v1" for the same read or query action when the client does a read in any replica. Instead, they see "Doc 1 v1 and Doc 2 v1" or "Doc 1 v2 and Doc 2 v2".

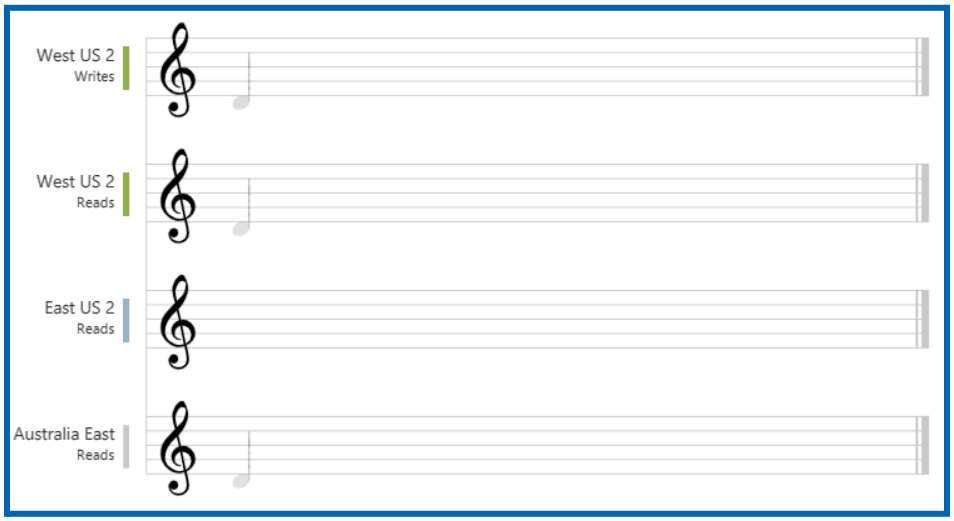


Image ref: https://learn.microsoft.com/en-us/azure/cosmos-db/consistency-levels

5. **Eventual consistency**: For reads, there is no ordering guarantee in eventual consistency. If no more writes occur, the replicas finally converge.

   The weakest type of consistency is eventual consistency since a client might read data that are more recent than those it has previously read. When there are no ordering guarantees needed for the application, eventual consistency is ideal. Retweets, Likes, and non threaded comments are a few examples.

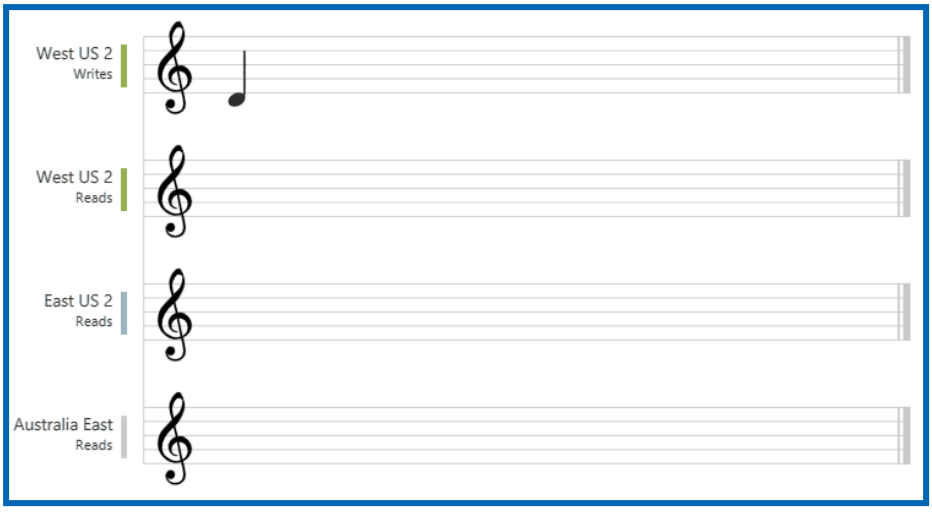


Image ref: https://learn.microsoft.com/en-us/azure/cosmos-db/consistency-levels

Change feed notification

A persistent record of changes to a container in the order they happen is called a change feed in Azure Cosmos DB. Azure Cosmos DB's change feed support operates by monitoring an Azure Cosmos DB container for any modifications. The amended documents are then sorted and output in the order that they were modified.

The output can be split among one or more consumers for processing in parallel, and the persistent modifications can be handled progressively and asynchronously.

Changing feed for Azure Cosmos DB reading

You have two options for interacting with the Azure Cosmos DB change feed: pull or push. In a push paradigm, the client with business logic for processing the job is pushed by the change feed processor. The change feed processor, however, manages the complexity of looking for work and maintaining the status of the most recent work that has been processed.

The client must retrieve the work from the server when using a pull model. In this scenario, the client not only provides business logic for processing work but also manages load balancing among numerous clients processing work in parallel, handles failures, and stores information for the last performed work.

Reading change feed with a push model

Azure Functions Azure Cosmos DB triggers and the change feed processor library are the two methods available to you when reading from the change feed using a push model. These two methods of reading the change feed are similar since Azure Functions utilizes the change feed processor in the background. Consider Azure Functions not as a whole new method of reading the change feed, but as merely a hosting platform for the change feed processor.

Azure Functions automatically parallelizes change processing across the partitions of your container by utilizing the change feed processor in the background.

1. **Azure Functions:** Small reactive Azure Functions can be created and set to run automatically in response to any new event in the change feed of your Azure Cosmos DB container. You may leverage the scalable and dependable event detection features of the Change Feed Processor without having to manage any worker infrastructure by using the Azure Functions trigger for Azure Cosmos DB.

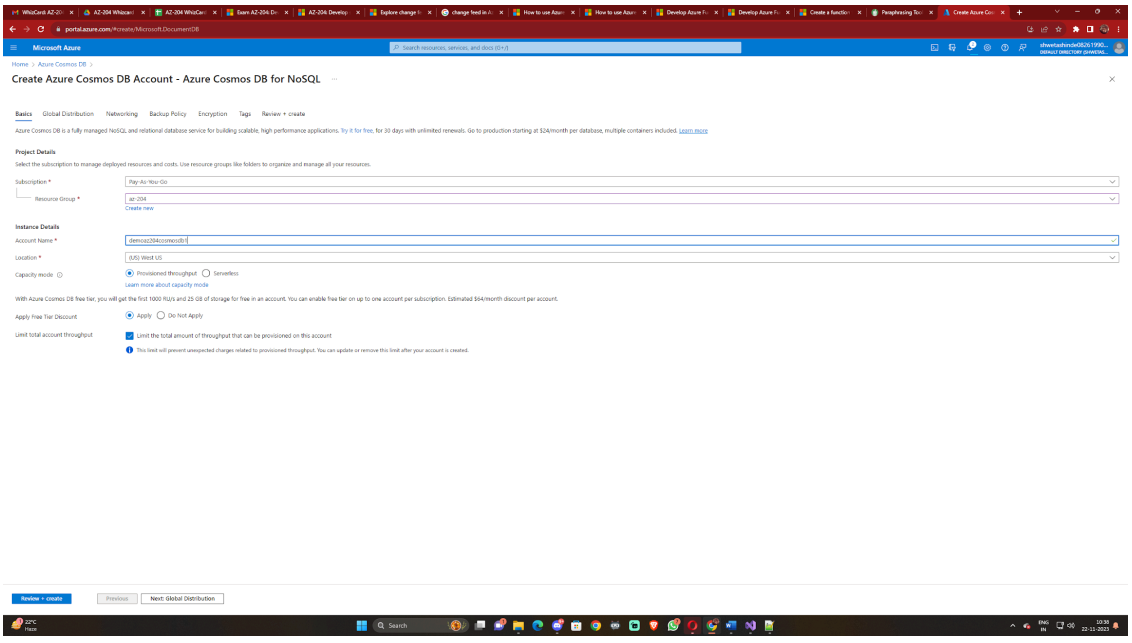
2. **Change feed processor:** The Java V4 and.NET V3 SDKs for Azure Cosmos DB include the change feed processor. It efficiently spreads the event processing among several consumers and streamlines the process of reading the change feed.

The change feed processor implementation consists of four key parts:

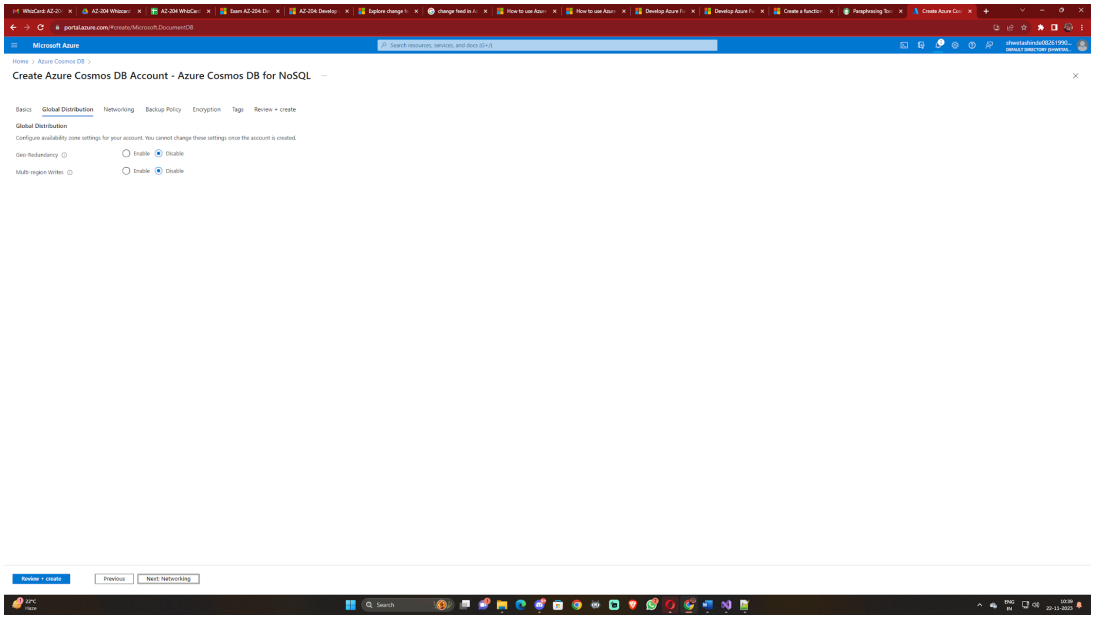
- **The monitored container:** The data used to construct the change feed is stored in the monitored container. The container's change feed reflects any inserts and updates made to the monitored container.
- **The lease container:** In addition to coordinating the processing of the change feed across several employees, the leasing container serves as a state store. The leased container may be kept in a different account or in the same account as the monitored container.
- **The compute instance:** The change feed processor is hosted by a compute instance, which is used to detect changes. It could be represented by a virtual machine (VM), a Kubernetes pod, an Azure App Service instance, or a real physical machine, depending on the platform. Throughout this text, the instance name is related to its unique identity.
- **The delegate:** The code that specifies what you, the developer, intend to happen to each batch of changes that the change feed processor reads is known as the delegate.

Change feed notification with Azure function

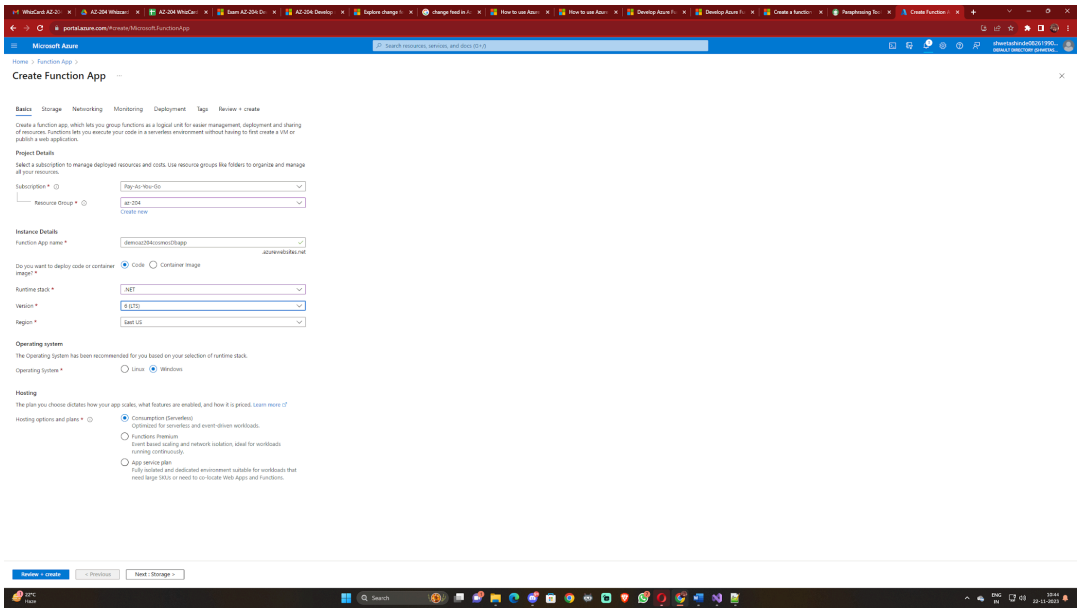
1. Choose Create a resource from the Azure portal menu or the Home page.
2. Look up Azure Cosmos Database. Choose Azure Cosmos DB under Create.
3. Choose the Create option in the Azure Cosmos DB for NoSQL section of the Create an Azure Cosmos DB account page.
4. Enter the new Azure Cosmos DB account's basic information on the Create Azure Cosmos DB Account page.



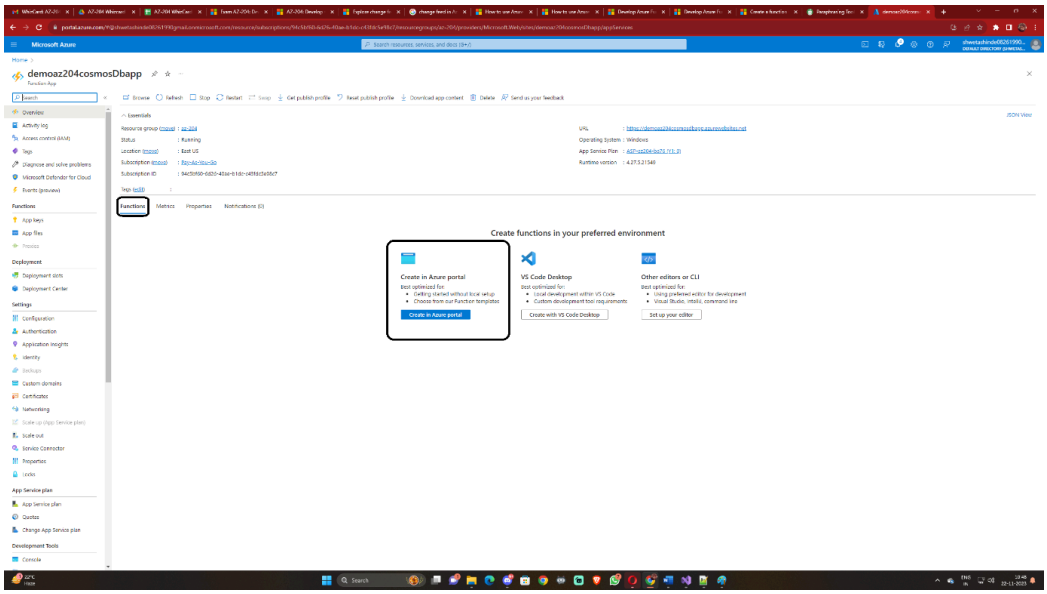
5. Set up these details under the Global Distribution tab. For this QuickStart, you can leave the default values as is:
    - Geo-Redundancy: Disable
    - Multi-region Writes: Disable
    - Availability Zones: Disable



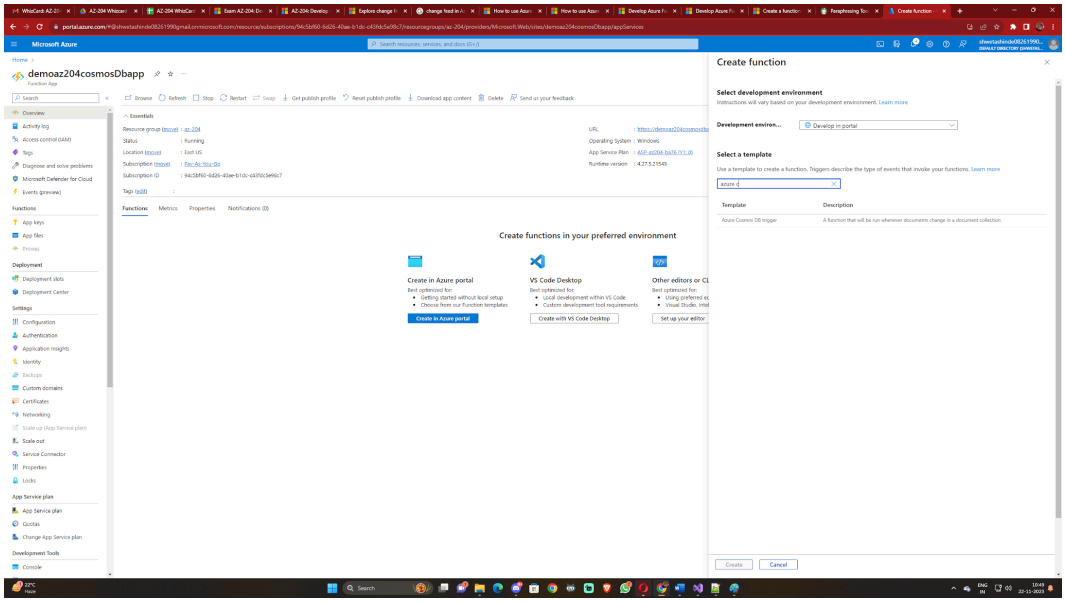
6. Select Review + create.
7. Review the account settings, and then select Create. It takes a few minutes to create the account. Wait for the portal page to display Your deployment is complete.
8. Select Go to resource to go to the Azure Cosmos DB account page.
9. Now we will create a function app for that From the Azure portal menu or the Home page, select Create a resource.
10. In the New page, select Compute > Function App.
11. On the Basics page, use the function app settings as specified below:
    A. Subscription: Your subscription
    B. Resource Group: az204ResourceGroup
    C. Function App name: Globally unique name
    D. Do you want to deploy code or container image? Code
    E. Runtime stack
    F. Preferred language:
    G. Version:
    H. Operating system: Windows
    I. Hosting options and plans: Consumption (Serverless)



12. Accept the default settings to create a new Application Insight instance on the Monitoring tab and a new storage account on the Storage tab. Using an already-existing storage account or Application Insights instance is another option.
13. Select Review + create to review the app configuration you chose, and then select Create to provision and deploy the function app.
14. Select Go to resource to view your new function app. Next, you create a function in the new function app.
15. In your function app, select Functions from the tab in overview section, and then select create in azure portal option.

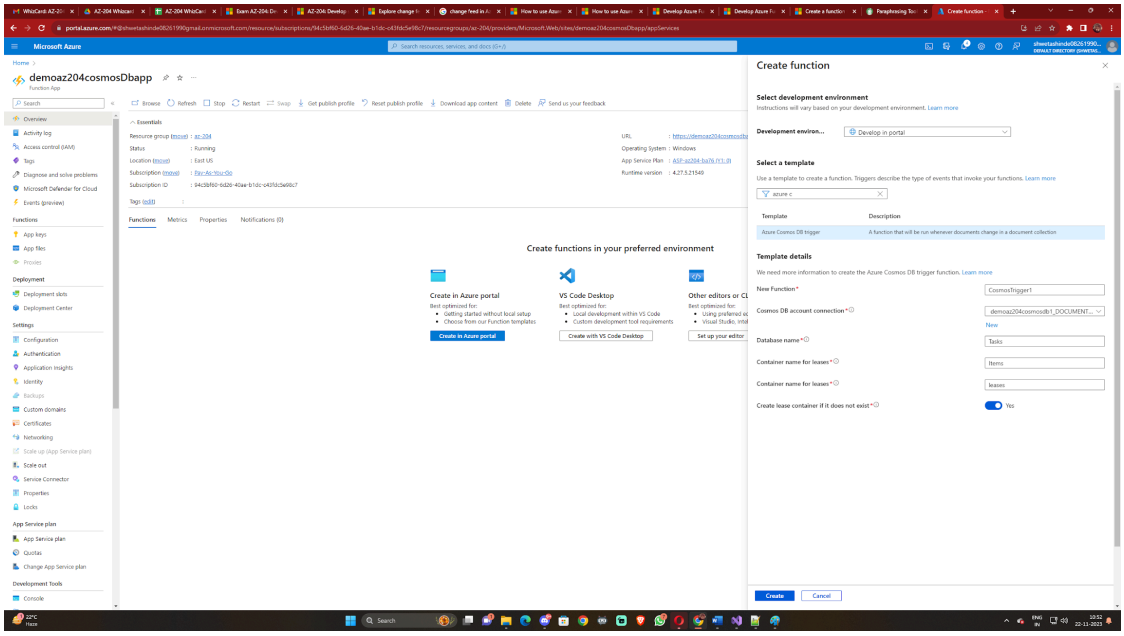


16. On the New Function page, enter cosmos in the search field and then choose the Azure Cosmos DB trigger template.



17. Configure the new trigger with the settings as specified below:

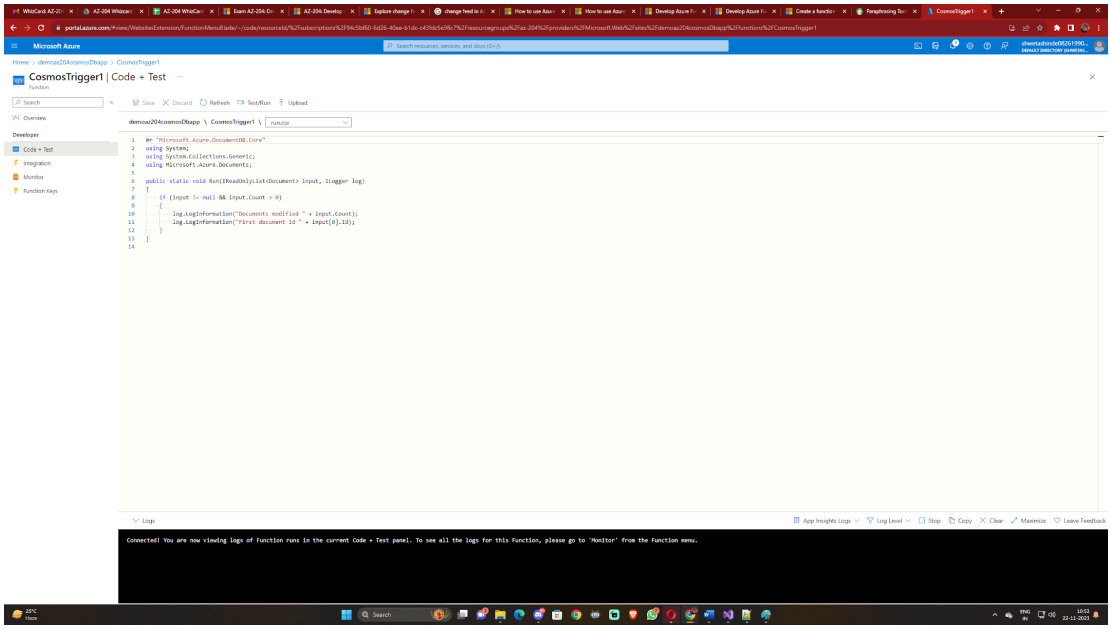
    1. New function: Accept the default name
    2. Azure Cosmos DB account connection: Accept the default new name
    3. Database name: Tasks
    4. Collection name: Items
    5. Collection name for leases: leases
    6. Create lease collection if it does not exist: Yes



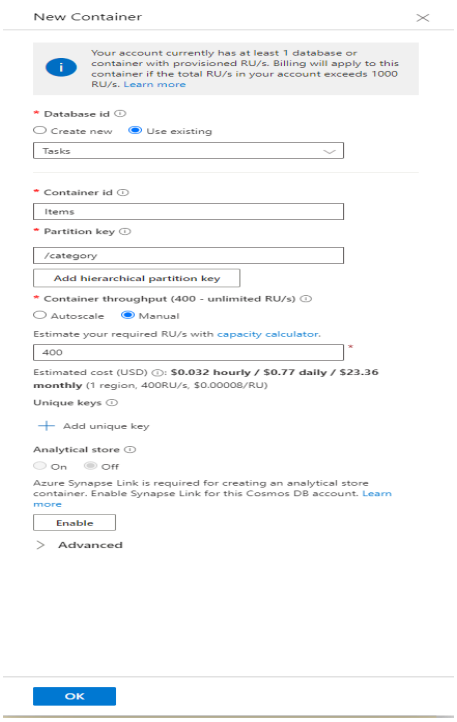
18. Select Create Function.

    Azure creates the Azure Cosmos DB trigger function.

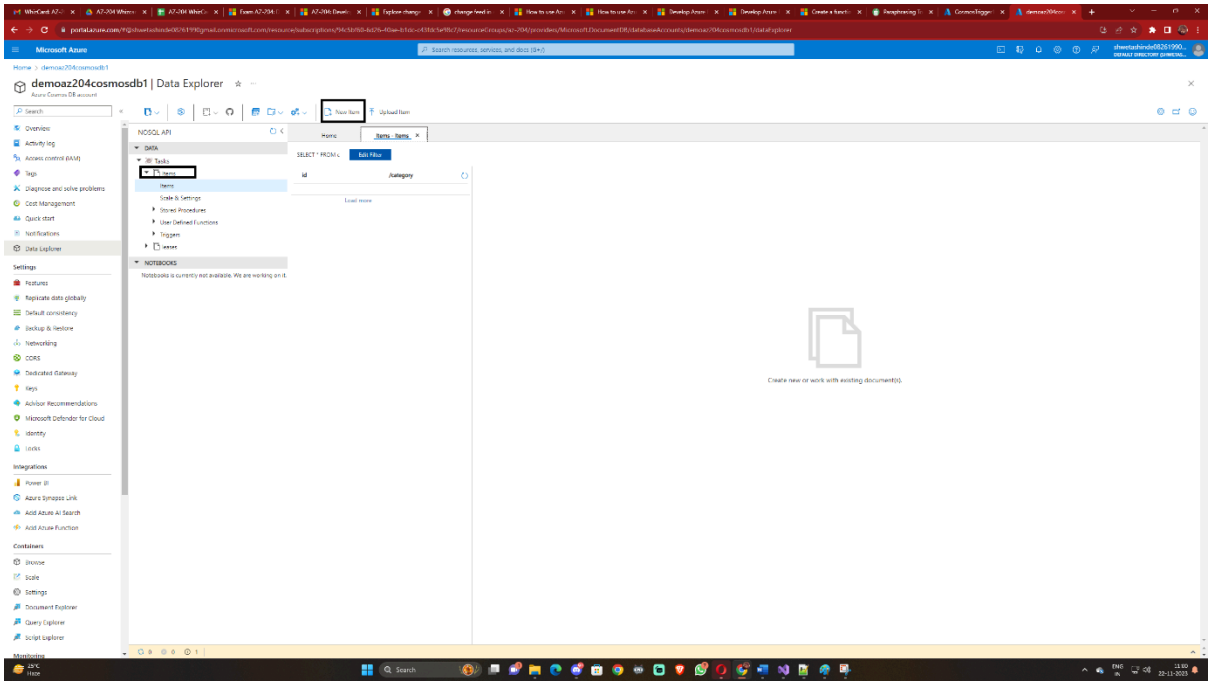
19. To display the template-based function code, select Code + Test.
This function template writes the number of documents and the first document ID to the logs.



20. Now we will create a container. Open a second instance of the Azure portal in a new tab in the browser.
21. Search azure Cosmos DB and go to your azure Cosmos DB account and then click on Data Explorer.
22. Under No SQL API, choose Tasks database and select New Container.
23. In Add Container, use the settings shown in the table below the image.



24. Click on Ok. After the container specified in the function binding exists, you can test the function by adding items to this new container.
25. Now we can test the function. Expand the new Items container in Data Explorer, choose Items, then select New Item.



26. Replace the contents of the new item with the following content, then choose Save.

```
{
   "id": "task1",
   "category": "general",
   "description": "some task"
}
```

27. Switch to the first browser tab that contains your function in the portal. Expand the function logs and verify that the new document has triggered the function. See that the task1 document ID value is written to the logs.

Azure Blob Storage

Set and retrieve properties and metadata

Microsoft's object storage solution for the cloud is called blob storage. Massively unstructured data storage is the specialty of blob storage. Data that doesn't follow a certain data model or specification, such text or binary data, is referred to as unstructured data.

Blob storage is intended for use in:

- delivering documents or images straight to a browser.
- putting files in storage for later access.
- audio and video streaming.
- logging data onto files.
- archiving, disaster recovery, and backup and restore data storage.
- storing data so that it can be analysed by an Azure or on-premises service.

Types of storage Accounts

Standard and premium storage accounts are available from Azure Storage in two different performance levels. Every performance level has a different price structure and provides a different set of features.

Standard: For the majority of Azure Storage scenarios, this basic general-purpose v2 account is advised.

Premium: By utilizing solid-state drives, Premium accounts provide improved performance. Block blobs, page blobs, and file shares are the three account types you can select from when creating a premium account.

The storage account types that Microsoft suggests for the majority of scenarios involving Blob storage are listed in the following table.

Storage Account Type	Performance tier	Usage
General-purpose v2	Standard	standard type of storage account for queues, tables, blobs, and file shares. Suggested for the majority of situations utilizing Blob Storage or any other Azure Storage service.
Block blob	Premium	Type of premium storage account for appending and blocking blobs. Suggested for situations requiring consistently low storage delay, fast transaction speeds, or smaller objects

Page blobs	Premium	Premium storage account type for page blobs only.

Access Tiers for block blob

Depending on usage patterns, Azure Storage offers various solutions for accessing block blob data. Azure Storage has access tiers that are each tailored to a specific data consumption pattern. You can save your block blob data as cheaply as possible by choosing the appropriate access tier.

Access tiers that are offered are:

- The **Hot** Access tier is designed to provide frequent access to storage account objects. The Hot tier offers the lowest access charges but the highest storage expenses. By default, new storage accounts are formed in the hot tier.
- **Cool Tier:** Large volumes of data that are rarely accessed and kept for at least 30 days are best served by the Cool access tier. In comparison to the Hot tier, the **Cool** tier has greater access prices and lower storage expenses.
- **Cold Tier -** An online tier optimized for storing data that is rarely accessed or modified, but still requires fast retrieval. Data in cold tier should be stored for at least 90 days. A cold tire has lower storage costs and higher access costs compared to a cool tire.
- **Archive Tier:** The only tier accessible for individual block blobs is **Archive**. Data that can withstand several hours of retrieval delay and will stay in the archive layer for at least 180 days is best suited for the archive tier. Although accessing data in the archive tier is more expensive than accessing data in the hot or cool tiers, it is the most economical option for keeping data.

Blob storage Resource Types

Blob storage offers three types of resources:

- The storage account.
- A container in the storage account
- A blob in a container

1. Storage Accounts

Your data in Azure has its own namespace thanks to a storage account. Your distinct account name appears in the address of each object you store in Azure Storage. The base address for the items in your storage account is made up of the account name and the Azure Storage blob endpoint.

For instance, the default URL for Blob storage is this if your storage account is called demostorageaccount: http://demostorageaccount.blob.core.windows.net

2. Containers

A container is a file system directory that holds a collection of blobs. An infinite number of containers may be included in a storage account, and an infinite number of blobs may be stored in a container.

Since a container name is a component of the unique URI (Uniform Resource Identifier) that is used to address the container or its blobs, it needs to be a valid DNS name. When naming a container, abide by following guidelines:

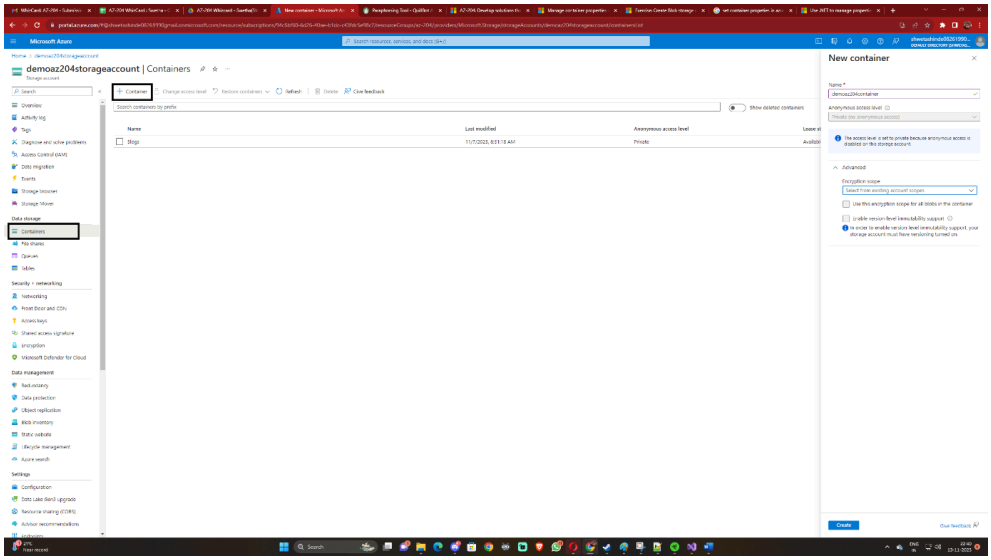
- The length of a container name might range from 3 to 63 characters.
- Container names can only contain lowercase letters, digits, and the dash (-) character. They must begin with a letter or number.
- Container names cannot contain two dash characters or more in a row.

3. Blobs

- Binary and text data are stored in block blobs. Block blobs consist of separate data blocks that are manageable. Block blobs are capable of holding up to 190.7 TiB.
- Similar to block blobs, append blobs are composed of blocks but are tailored for add operations. Append blobs are perfect in situations like virtual machine data logging.
- Random access files up to 8 TB in size are stored in page blobs. Page blobs act as drives for Azure virtual machines and store virtual hard drive (VHD) files.

Creating a container

1. Select the storage account in which you want to create a container. If you don’t have existing storage account then create new storage account as to create a container storage account is must
2. Go to container under data storage and click on container to create a new container



3. Click on create after putting appropriate name for container.
4. In the created container upload one text file with some content in it.

Container Properties

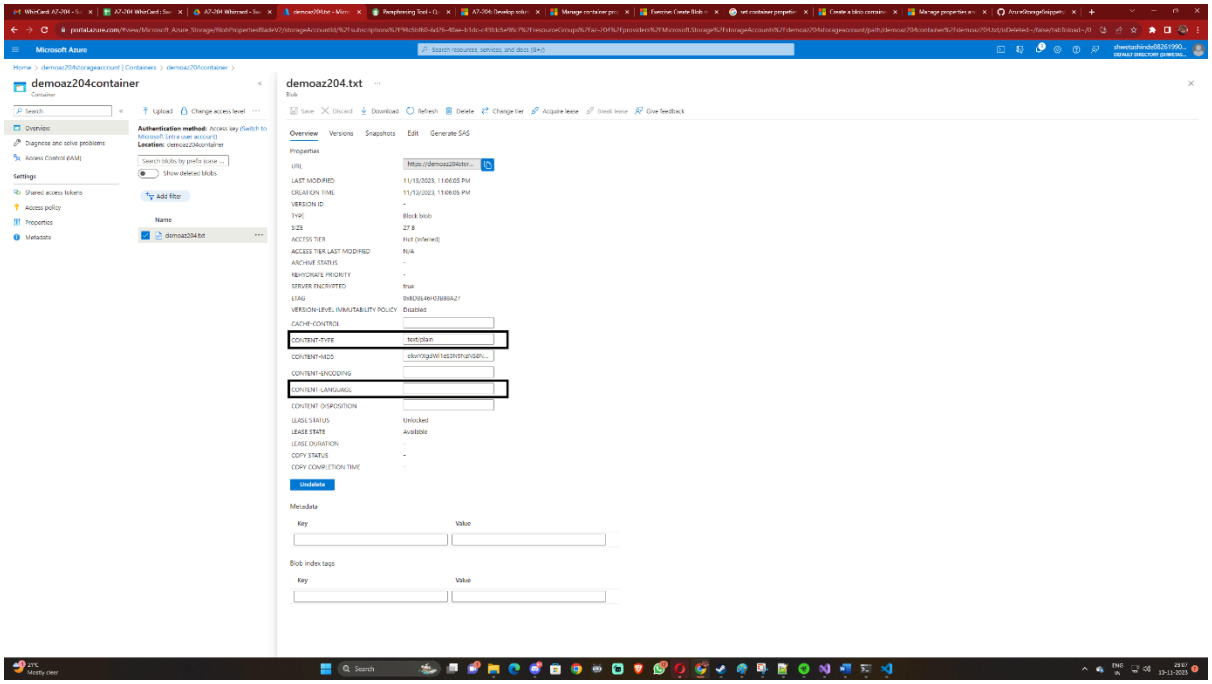
System properties: Every Blob storage resource has system attributes. While some are read-only, others can be set or read. Some system properties under the hood map to specific standard HTTP headers. These properties are kept up to date for you by the Azure Storage client library for.NET.

Setting and Retrieving container properties

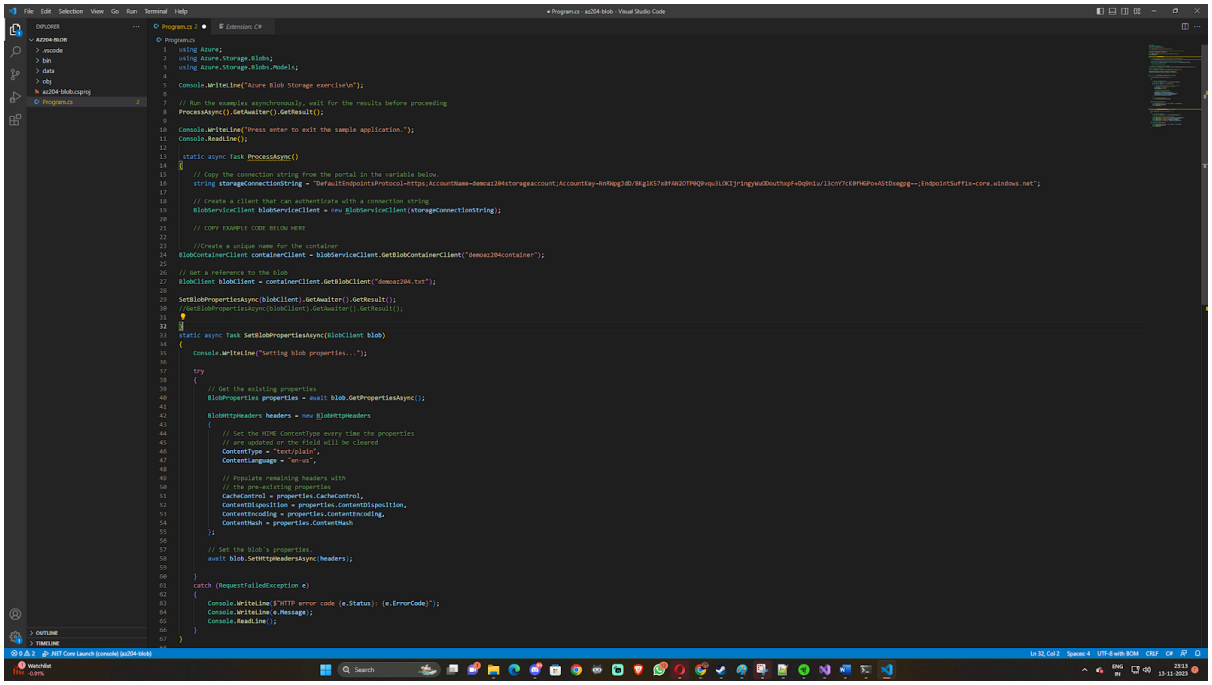
Call one of the BlobContainerClient class's methods to obtain container properties:

- GetProperties
- GetPropertiesAsync

Below are the original properties of blob in container.

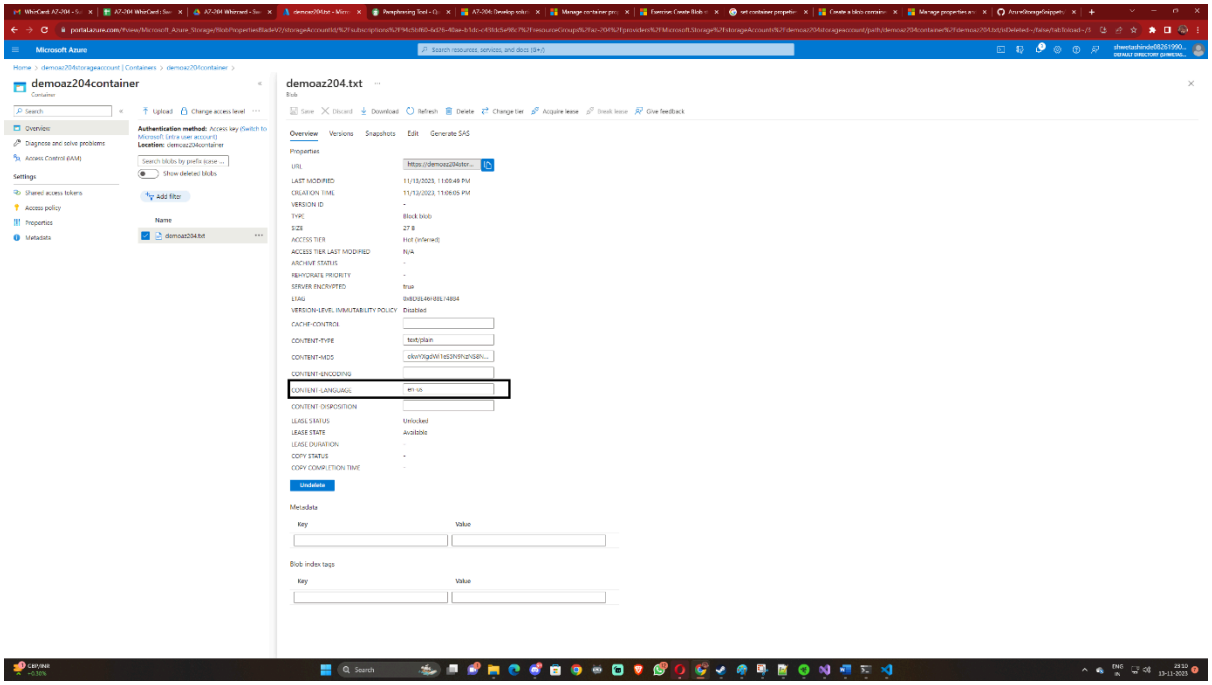


The code example below shows how to set the container properties. We are going to set content-type and content language properties of the blob.



```
using Azure;
using Azure.Storage.Blobs;
using Azure.Storage.Blobs.Models;

Console.WriteLine("Azure Blob Storage exercise\n");

// Run the examples asynchronously, wait for the results before proceeding
ProcessAsync().GetAwaiter().GetResult();

Console.WriteLine("Press enter to exit the sample application.");
Console.ReadLine();

static async Task ProcessAsync()
{
    // Copy the connection string from the portal in the variable below.
    string storageConnectionString = "DefaultEndpointsProtocol=https;AccountName=demoaz204storageaccount;AccountKey=HnRWgqJdD/BKgiKS7x8f4A92OTP6QkvquSLCKIjrngyWaODouthxpF+dqHm1u/i3cnY7cK0fHGPo4AtDxegpg==;EndpointSuffix=core.windows.net";

    // Create a client that can authenticate with a connection string
    BlobServiceClient blobServiceClient = new BlobServiceClient(storageConnectionString);

    // COPY EXAMPLE CODE BELOW HERE

    //Create a unique name for the container
BlobContainerClient containerClient = blobServiceClient.GetBlobContainerClient("demoaz204container");

// Get a reference to the blob
BlobClient blobClient = containerClient.GetBlobClient("demoaz204.txt");

SetBlobPropertiesAsync(blobClient).GetAwaiter().GetResult();
//GetBlobPropertiesAsync(blobClient).GetAwaiter().GetResult();

}
static async Task SetBlobPropertiesAsync(BlobClient blob)
{
    Console.WriteLine("Setting blob properties...");

    try
    {
        // Get the existing properties
        BlobProperties properties = await blob.GetPropertiesAsync();

        BlobHttpHeaders headers = new BlobHttpHeaders
        {
            // Set the MIME ContentType every time the properties
            // are updated or the field will be cleared
            ContentType = "text/plain",
            ContentLanguage = "en-us",

            // Populate remaining headers with
            // the pre-existing properties
            CacheControl = properties.CacheControl,
            ContentDisposition = properties.ContentDisposition,
            ContentEncoding = properties.ContentEncoding,
            ContentHash = properties.ContentHash
        };

        // Set the blob's properties.
        await blob.SetHttpHeadersAsync(headers);

    }
    catch (RequestFailedException e)
    {
        Console.WriteLine($"HTTP error code {e.Status}: {e.ErrorCode}");
        Console.WriteLine(e.Message);
        Console.ReadLine();
    }
}
```

```
C:\Users\wildc\az204-blob>dotnet run
C:\Users\wildc\az204-blob\Program.cs(14,20): warning CS1998: This async method lacks 'await' operators and will run syn
chronously. Consider using the 'await' operator to await non-blocking API calls, or 'await Task.Run(...)' to do CPU-bou
nd work on a background thread. [C:\Users\wildc\az204-blob\az204-blob.csproj]
Azure Blob Storage exercise

Setting blob properties...
Press enter to exit the sample application.
```



The code example that follows retrieves the system properties of a container and outputs the values of those properties to a console window:

```
static async Task GetBlobPropertiesAsync(BlobClient blob)
{
    Console.WriteLine("Getting blob properties...");
    try
    {
        // Get the blob properties
        BlobProperties properties = await blob.GetPropertiesAsync();

        // Display some of the blob's property values
        Console.WriteLine($" ContentLanguage: {properties.ContentLanguage}");
        Console.WriteLine($" ContentType: {properties.ContentType}");
        Console.WriteLine($" CreatedOn: {properties.CreatedOn}");
        Console.WriteLine($" LastModified: {properties.LastModified}");
    }
    catch (RequestFailedException e)
    {
        Console.WriteLine($"HTTP error code {e.Status}: {e.ErrorCode}");
        Console.WriteLine(e.Message);
        Console.ReadLine();
    }
}
```

```
C:\Users\wildc\az204-blob>dotnet run
C:\Users\wildc\az204-blob\Program.cs(14,20): warning CS1998: This async method lacks 'await' operators and will run syn
chronously. Consider using the 'await' operator to await non-blocking API calls, or 'await Task.Run(...)' to do CPU-bou
nd work on a background thread. [C:\Users\wildc\az204-blob\az204-blob.csproj]
C:\Users\wildc\az204-blob\Program.cs(36,19): warning CS8321: The local function 'SetBlobPropertiesAsync' is declared bu
t never used [C:\Users\wildc\az204-blob\az204-blob.csproj]
Azure Blob Storage exercise

Getting blob properties...
 ContentLanguage: en-us
 ContentType: text/plain
 CreatedOn: 13-11-2023 17:36:05 +00:00
 LastModified: 13-11-2023 17:41:46 +00:00
Press enter to exit the sample application.
```

Set and Retrieve metadata

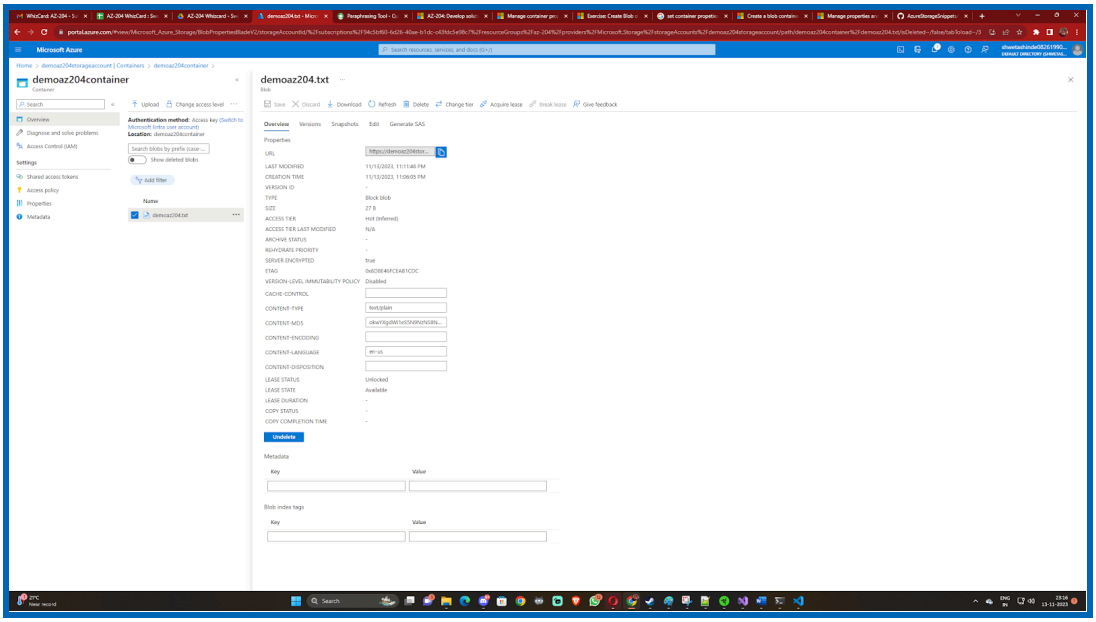
One or more name-value pairs that you choose for a Blob storage resource make up user-defined metadata. Extra values can be stored with the resource by using metadata. The metadata values are purely for your use and have no bearing on the functionality of the resource.

A blob or container resource can have one or more name-value pairs representing metadata. Add name-value pairs to an IDictionary object, then use one of the BlobContainerClient class's methods to write the values in order to set metadata:

- SetMetadata
- SetMetadataAsync

Your metadata name needs to follow the C# identifier naming guidelines. While metadata names are case-insensitive when set or read, they retain the case in which they were produced. Blob storage concatenates the two values and returns HTTP response code 200 (OK) if two or more metadata headers with the same name are submitted for a resource.

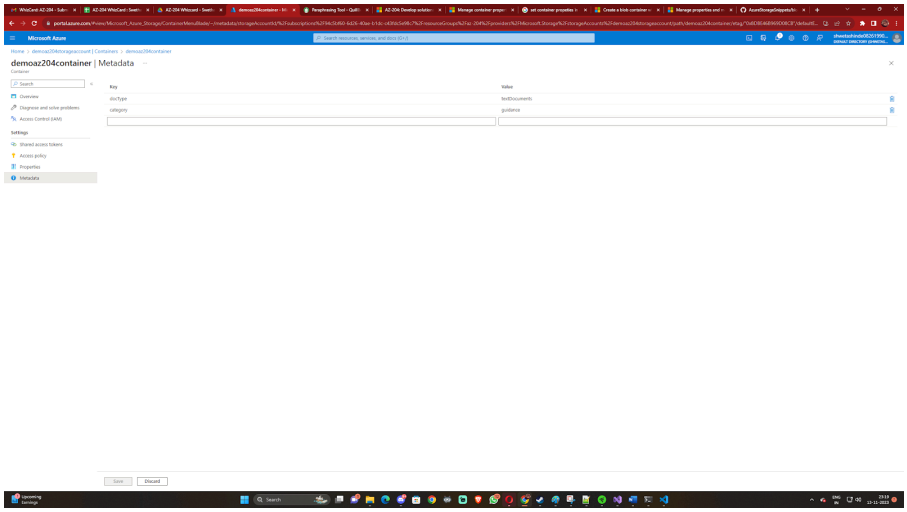
Before setting metadata



The below example shows how to set metadata

```
static async Task AddContainerMetadataAsync(BlobContainerClient container)
{
    Console.WriteLine("Setting blob Metadata...");
    try
    {
        IDictionary<string, string> metadata =
            new Dictionary<string, string>();

        // Add some metadata to the container.
        metadata.Add("docType", "textDocuments");
        metadata.Add("category", "guidance");

        // Set the container's metadata.
        await container.SetMetadataAsync(metadata);
    }
    catch (RequestFailedException e)
    {
        Console.WriteLine($"HTTP error code {e.Status}: {e.ErrorCode}");
        Console.WriteLine(e.Message);
        Console.ReadLine();
    }
}
```

After setting metadata



The GetProperties and GetPropertiesAsync methods are used to retrieve metadata in addition to properties as shown earlier.

The below example shows how to get the metadata

```
static async Task ReadContainerMetadataAsync(BlobContainerClient container)
{
    try
    {
        var properties = await container.GetPropertiesAsync();

        // Enumerate the container's metadata.
        Console.WriteLine("Container metadata:");
        foreach (var metadataItem in properties.Value.Metadata)
        {
            Console.WriteLine($"\tKey: {metadataItem.Key}");
            Console.WriteLine($"\tValue: {metadataItem.Value}");
        }
    }
    catch (RequestFailedException e)
    {
        Console.WriteLine($"HTTP error code {e.Status}: {e.ErrorCode}");
        Console.WriteLine(e.Message);
        Console.ReadLine();
    }
}
```

```
C:\Users\wildc\az204-blob>dotnet run
C:\Users\wildc\az204-blob\Program.cs(13,20): warning CS1998: This async method lacks 'await' operators and will run synchronously. Consider using the 'await' operator to await non-blocking API calls, or 'await Task.Run(...)' to do CPU-bound work on a background thread. [C:\Users\wildc\az204-blob\az204-blob.csproj]
Azure Blob Storage exercise

Container metadata:
        Key: docType
        Value: textDocuments
        Key: category
        Value: guidance
Press enter to exit the sample application.
```

Azure Blob storage

Storage Policies, Data Lifecycle Management(DLM), and Static Site Hosting

A. Stored Access Policy

On the server side, a stored access policy offers an extra degree of control over service-level shared access signatures (SASs). Creating a stored access policy allows shared access signatures to be grouped together and gives signatures constrained by the policy additional constraints.

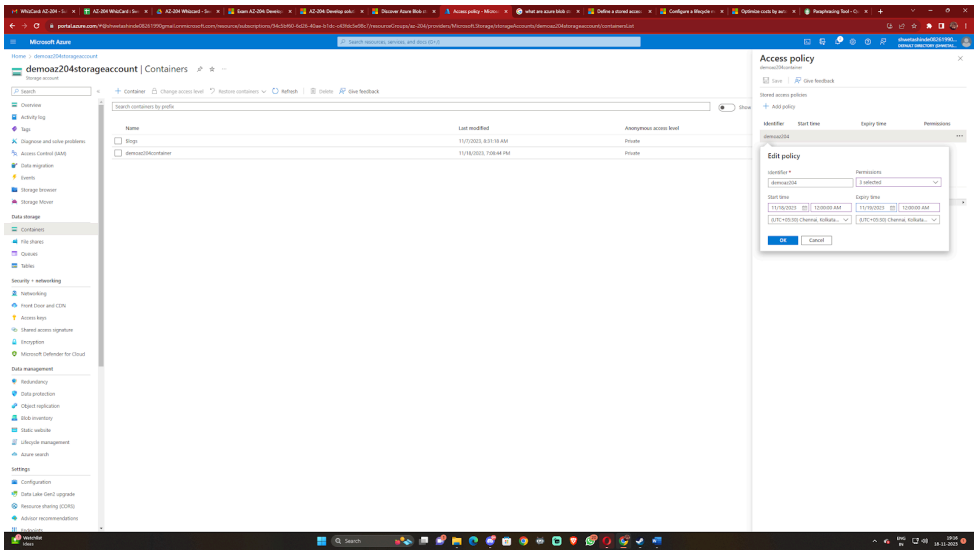
A stored access policy can be used to modify a signature's start time, expiration time, or permissions. Once a signature has been issued, it can also be revoked using a stored access policy.

The following storage resources support stored access policies:

- Blob containers
- File shares
- Queues
- Tables

Create or modify policy

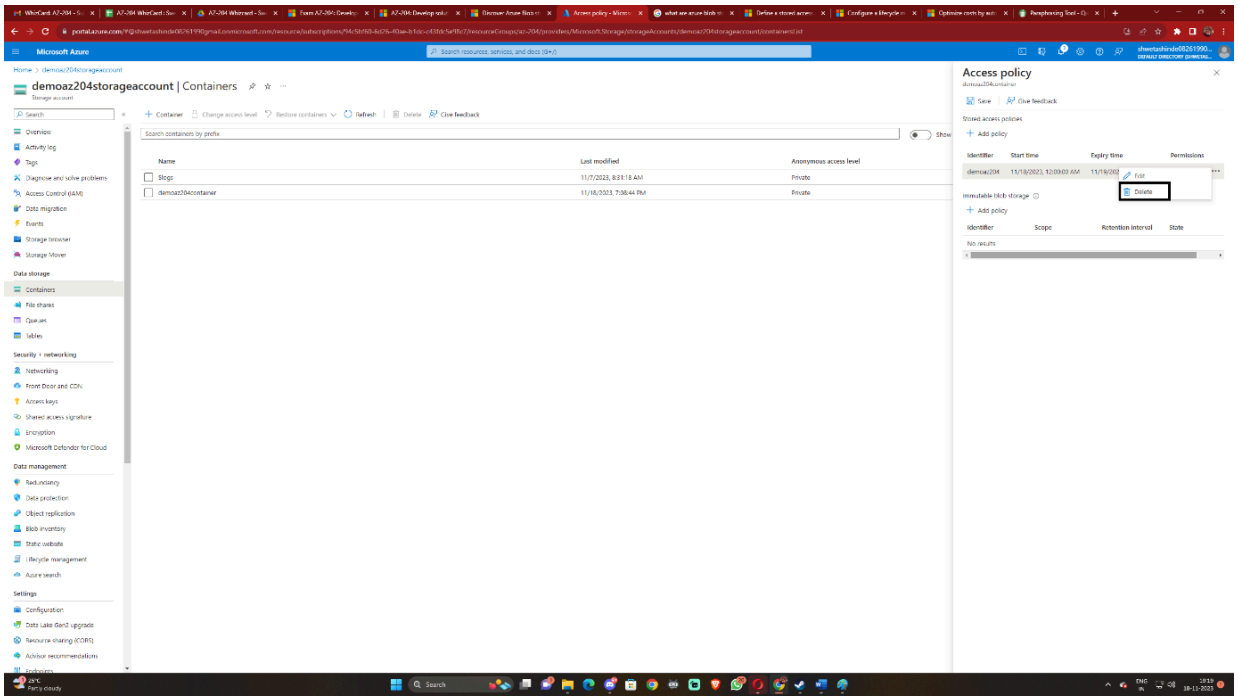
Call the Set ACL action for the resource (see Set Container ACL, Set Queue ACL, Set Table ACL, or Set Share ACL) with a request body that contains the access policy's terms in order to establish or modify a stored access policy. A distinct signed identification of your choosing, up to 64 characters long, is included in the request's body. The following optional access policy parameters are also included in the request body:



A queue, share, table, or container can have up to five access policies specified at once. One access policy is associated with each SignedIdentifier field, which has its own Id field. Setting more than five access policies at once results in a 400 status code (Bad Request) being returned by the service.

Revoke stored Access Policy

A stored access policy can be revoked by deleting it, altering its signed identifier, or setting its expiration period to a previous value. Any relationships between the stored access policy and any current signatures are broken by changing the signed identifier. Any related signatures expire when the expiry time is changed to a previous value. Any shared access signatures linked to the stored access policy will be instantly impacted if it is deleted or modified.



B. Data Lifecycle Management

Every data set has a different lifespan. People frequently access some data early in their lifespan. However, as data ages, the demand for access decreases significantly. Once saved on the cloud, certain data is rarely accessed and remains dormant. While some data sets are actively viewed and altered during their lifespan, others expire days or months after they are created.

You can store blob object data as cheaply as possible with Azure storage's various access tiers. There are several access tiers available:

- Hot: designed with frequent access to data in mind.
- Cool: designed to hold data that is kept for at least 30 days and is not commonly accessed.
- Cold tier: designed to hold data that is kept for at least ninety days and is not commonly accessed. In comparison to the cool tier, the cold tier has greater access fees and lower storage expenses.
- Archive: designed to store data with variable latency requirements, on the scale of hours, for a minimum of 180 days and infrequently accessed data.

The various access tiers are subject to the following considerations:

- Blobs can have their access tier changed either during or after upload.
- At the account level, you can only set the hot and cool access tiers. Only at the blob level is it possible to establish the archive access tier.
- Although the availability of data in the cold access tier is marginally lower than that of hot data, it nevertheless has excellent durability, retrieval latency, and throughput qualities that are comparable.
- The archival access tier's data is kept offline. The lowest storage prices are provided by the archive tier, but access charges and latency are also the greatest.
- All redundancy choices are supported by the hot and cool tiers. Only RA-GRS, GRS, and LRS are supported on the archive tier.
- Limits on data storage are determined per account, not per access tier. You have the option to exhaust your entire limit in one tier or throughout the course of all three tiers.

Managing the Data Lifecycle

A robust, rule-based policy for General Purpose v2 and Blob storage accounts is provided by Azure Blob storage lifecycle management. When your data reaches the end of its lifecycle, use the policy to expire it or move it to the proper access tiers. The policy for lifecycle management allows you to:

- To balance performance and cost, move blobs to a colder storage layer (hot to cool, hot to archive, or cool to archive).
- Once a blob's lifecycle is over, delete it.
- Establish rules that will be executed once daily at the storage account level.
- Use prefixes as filters to apply rules to containers or a subset of blobs.

Imagine a situation in which data is accessed frequently in the first two weeks of its lifecycle and then infrequently thereafter. After the initial month, there is hardly any access to the data set. Hot storage works well in the early stages of this scenario. The best storage for sporadic access is cold storage. After the data has aged for more than a month, archive storage is the best tier choice. You can construct the least priced storage choices for your needs by modifying storage

tiers based on the data's age. Lifecycle management policy rules are offered to migrate aging data to cooler tiers in order to accomplish this transition.

Lifecycle Policies:

A JSON document containing a set of rules is called a lifecycle management policy. A filter set and an action set are included in every rule definition found in a policy. The filter set restricts the items in a container or their names to which rule actions can be applied. The filtered set of objects is subject to the tier or delete actions by the action set:

JSON:

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

A policy is a collection of rules(At least one rule is required in a policy. You can define up to 100 rules in a policy.):

Each rule within the policy has several parameters:

Parameter Name	Type	Note	Required
name	String	The maximum number of alphanumeric characters in a rule name is 256. Case affects the rule name. It needs to be distinct inside a policy.	True
enabled	Boolean	An optional Boolean that permits the temporary disabling of a rule. If it isn't set, the default value is true.	False
type	An Enum value	The current valid type is Lifecycle.	True
definition	An object that defines the lifecycle rule	Each definition is made up of a filter set and an action set.	True

A filter set and an action set are included in every rule specification. The filter set restricts the items in a container or their names to which rule actions can be applied. The filtered set of items is subject to the tier or delete actions by the action set.

The example rule that follows filters the account to perform actions on objects that are present in democontainer and start with foo.

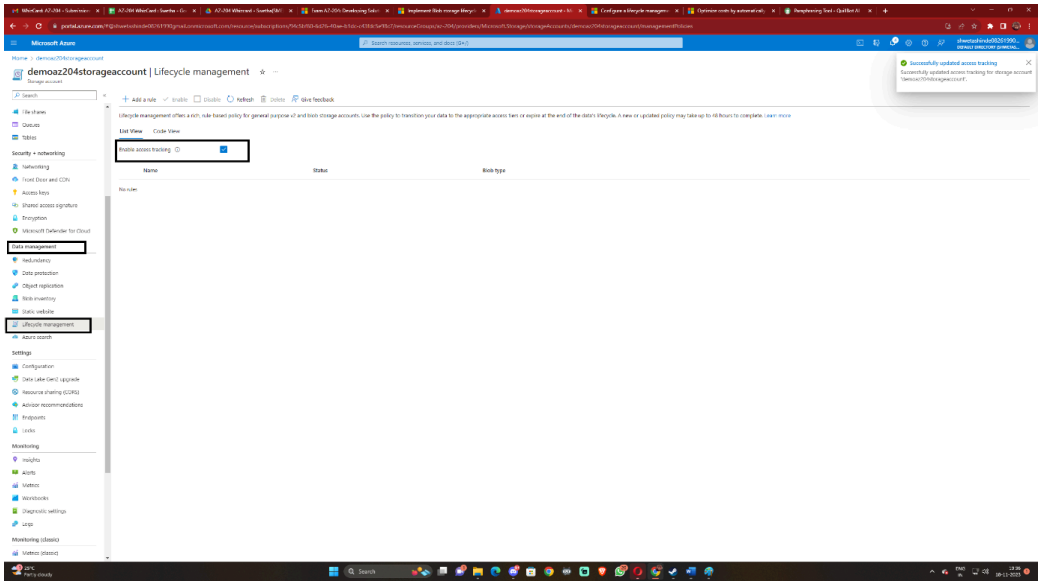
- Cool tier to tier blob thirty days following the most recent alteration
- Tier blob to be archived 90 days following the most recent update
- 2,555 days (seven years) after the last modification, delete the blob.
- 90 days after the snapshot is created, remove the blob snapshots.

```
{
 "rules": [
  {
   "name": "ruleFoo",
   "enabled": true,
   "type": "Lifecycle",
   "definition": {
    "filters": {
     "blobTypes": [ "blockBlob" ],
     "prefixMatch": [ " democontainer /foo" ]
    },
```

```
      "actions": {
        "baseBlob": {
          "tierToCool": { "daysAfterModificationGreaterThan": 30 },
          "tierToArchive": { "daysAfterModificationGreaterThan": 90 },
          "delete": { "daysAfterModificationGreaterThan": 2555 }
        },
        "snapshot": {
          "delete": { "daysAfterCreationGreaterThan": 90 }
        }
      }
    }
  }
]
}
```

Implementation

1. Use the Azure portal to enable last access time tracking by doing the following steps:
2. Go into the Azure portal and select your storage account.
3. Choose Lifecycle management from the Data management section.
4. Select the "Enable access tracking" checkbox.



5. Using the Azure portal, PowerShell, Azure CLI, or an Azure Resource Manager template, you can add, modify, or remove a lifecycle management policy.

   Using the Azure portal, there are two methods for adding a policy.

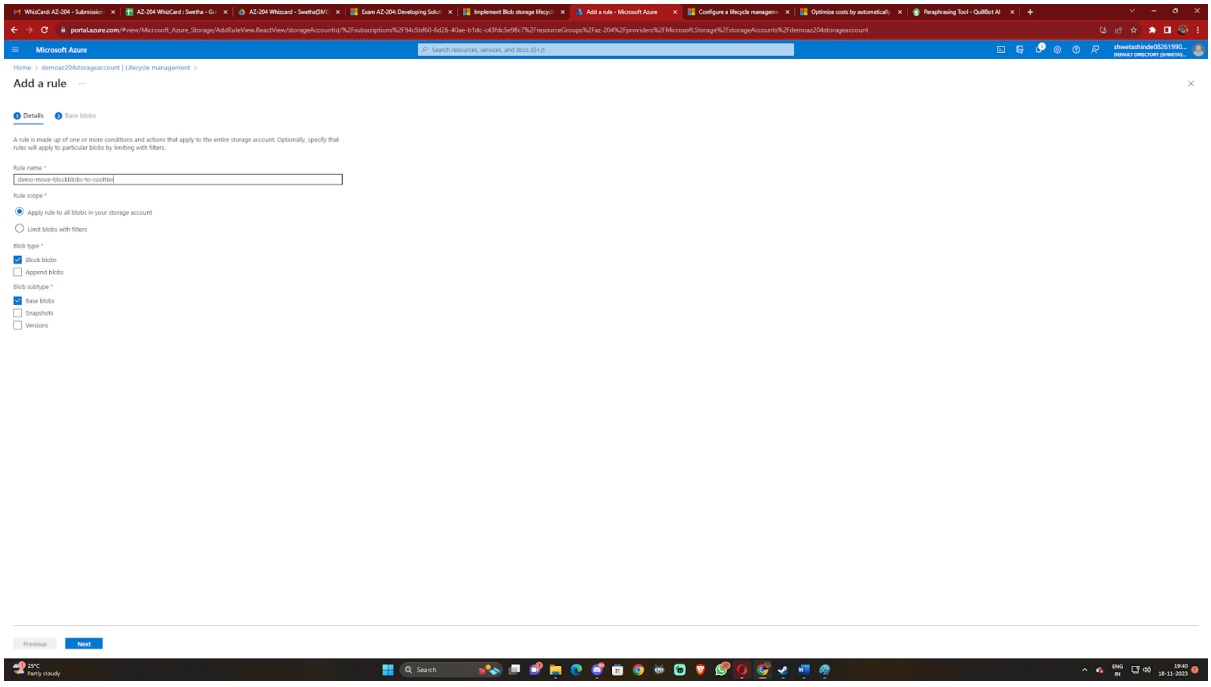
   Code view / List view

6. List View:
   Go to your storage account in the Azure interface.

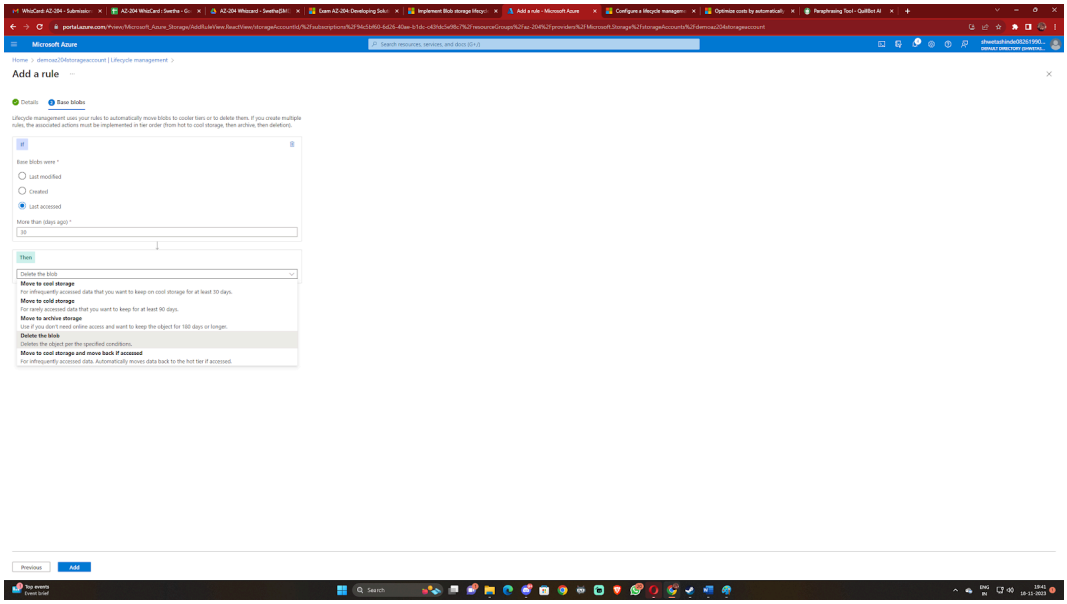
7. To see or modify lifecycle management policies, select Lifecycle Management under Data management.

8. Choose the tab for List View.

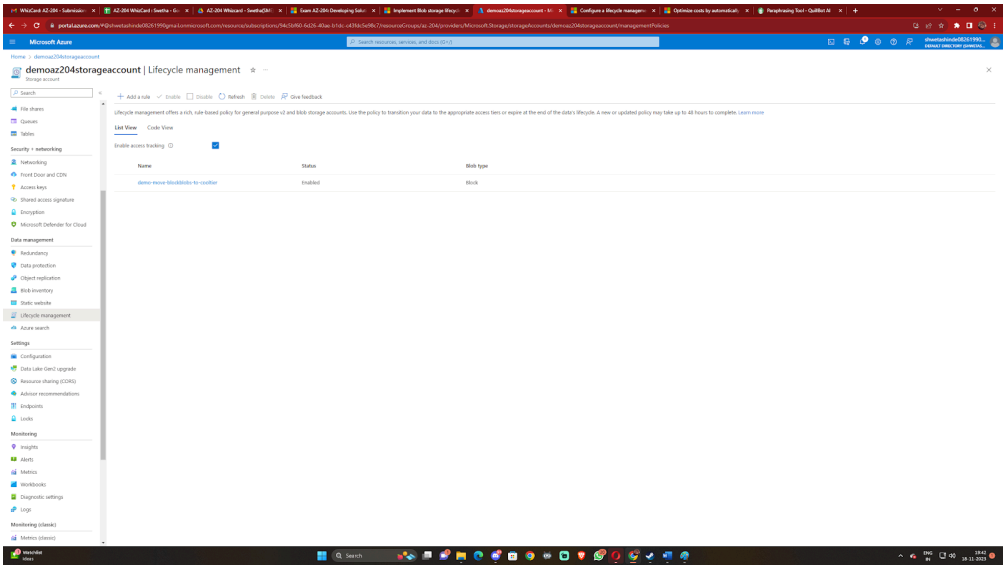
9. Choose On the Details form, add a rule and give it a name. The settings for the Rule scope, Blob type, and Blob subtype can also be changed. The scope is configured to filter blobs in the example that follows. The Filter set tab is added as a result of this.



10. Select Base blobs to set the conditions for your rule. In the following example, blobs are moved to cool storage if they haven't been modified for 30 days.



11. Click on Add. Your policy will get added.



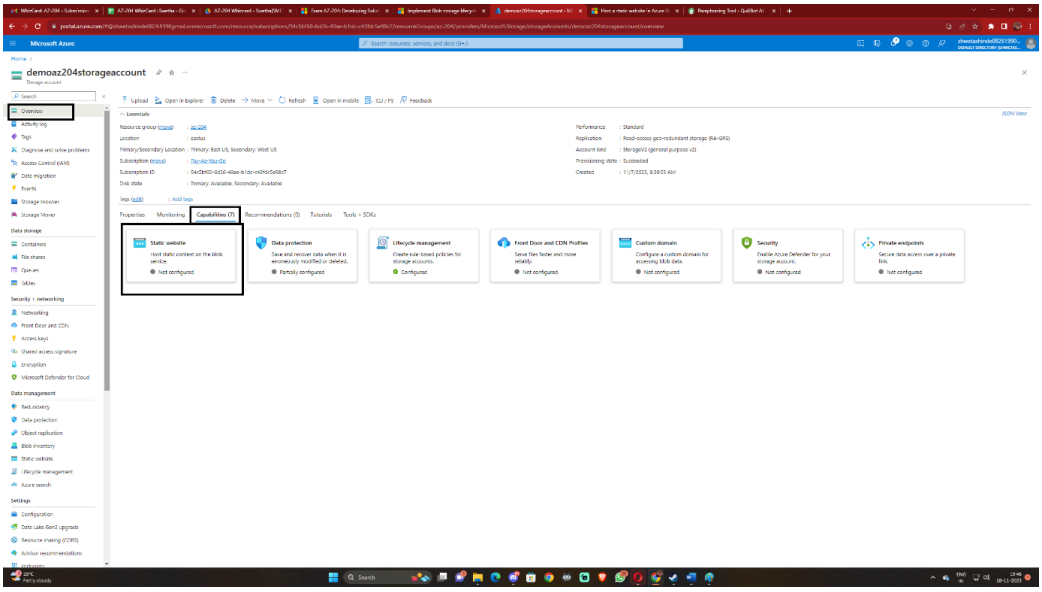
12. Code View:
    Go to your storage account in the Azure interface.
13. To see or modify lifecycle management policies, select Lifecycle Management under Data management.
14. Choose the tab for Code View. You can specify a lifecycle management policy in JSON on this tab.

The subsequent example A block blob whose name starts with log is moved to the cool tier by a JSON lifetime policy if it hasn't been updated in more than 15 days.

```
{
  "rules": [
    {
      "enabled": true,
      "name": "move-to-cool",
      "type": "Lifecycle",
      "definition": {
        "actions": {
          "baseBlob": {
            "tierToCool": {
              "daysAfterModificationGreaterThan": 15
            }
          }
        },
        "filters": {
          "blobTypes": [
            "blockBlob"
          ],
          "prefixMatch": [
            "sample-container/log"
          ]
        }
      }
    }
  ]
}
```

C. Static Website hosting

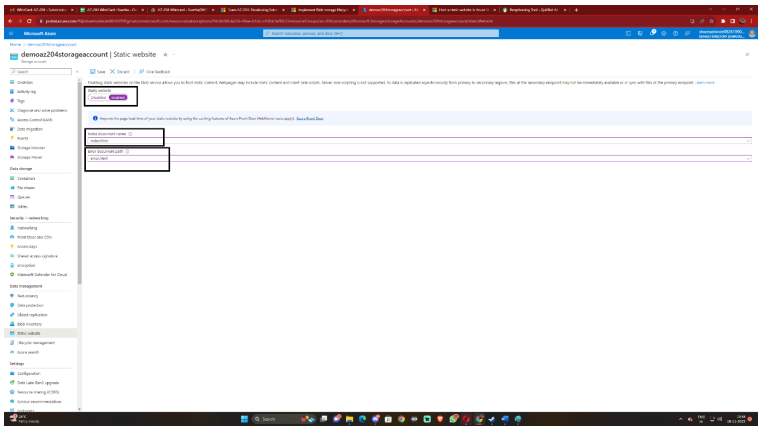
1. To begin, log in to the Azure portal.
2. Find your storage account and click it to bring up the Overview pane for that account.
3. Click the Capabilities tab in the Overview window. To view the static website's configuration page, pick Static website next.



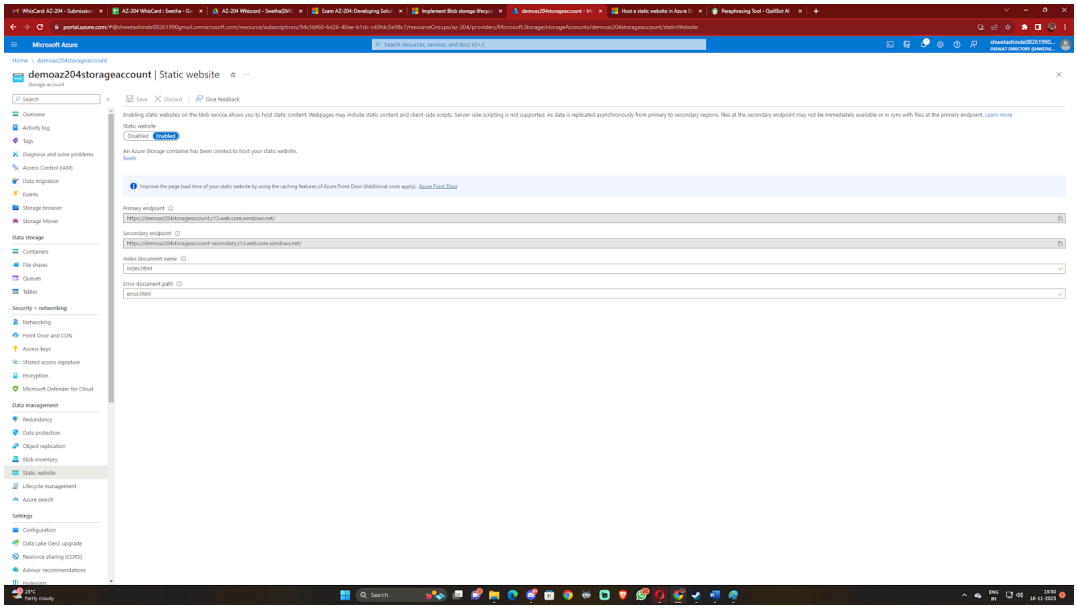
4. To allow hosting of static websites for the storage account, select Enabled.
5. Please provide the default index page (index.html, for example) in the Index document name box. When a person visits the root of your static website, the default index page appears.
6. Enter the default error page (404.html, for example) in the Error document path field.

   When a user tries to access a page that doesn't exist on your static website, the default error page is shown.

7. To complete configuring the static site, click Save.

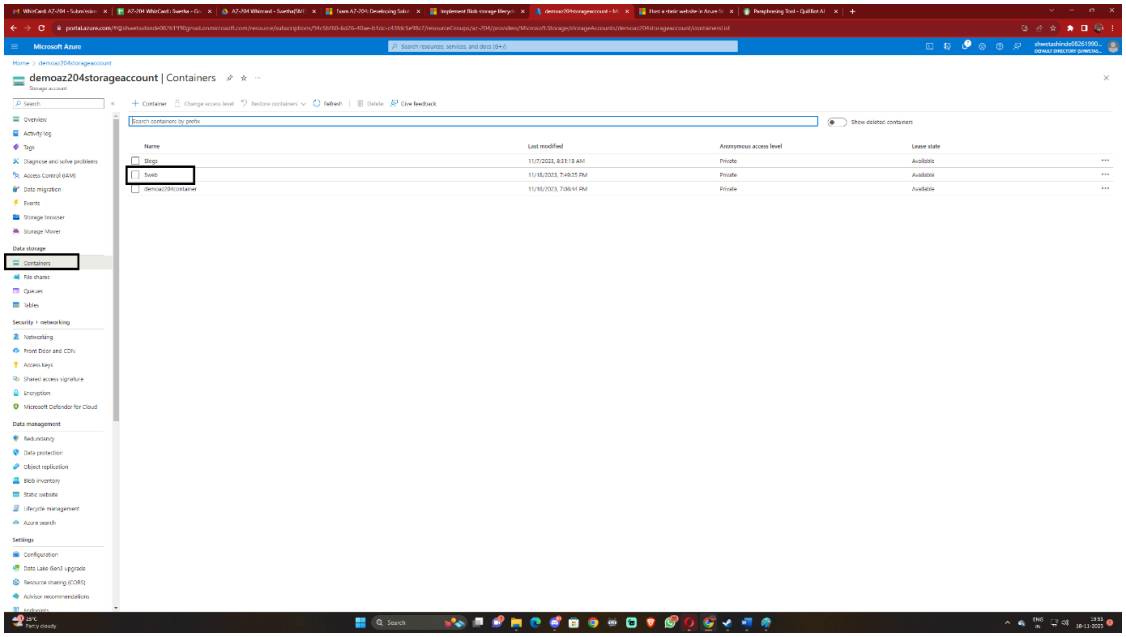


8. A notification of confirmation appears. The Overview pane displays other configuration details along with your static website endpoints.

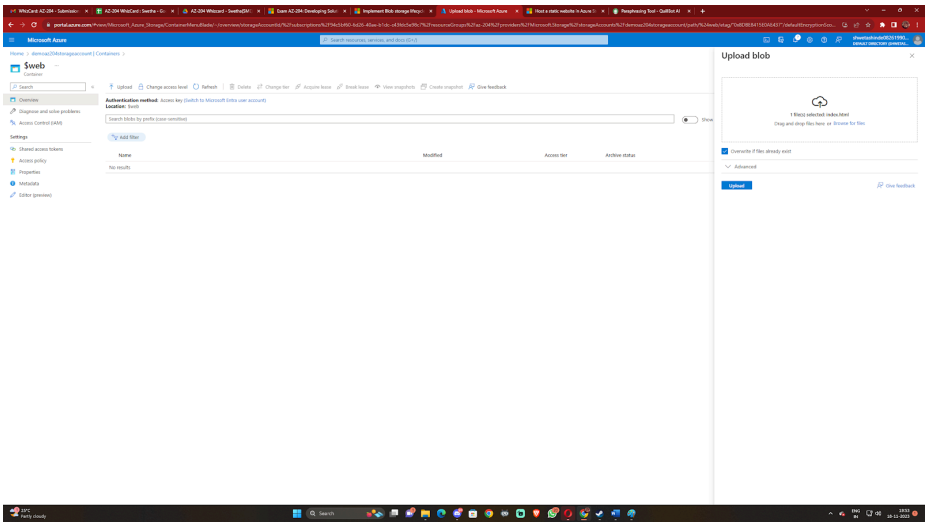


9. Go to the storage account that houses your static website in the Azure interface. To view the list of containers, select Containers from the left navigation pane.

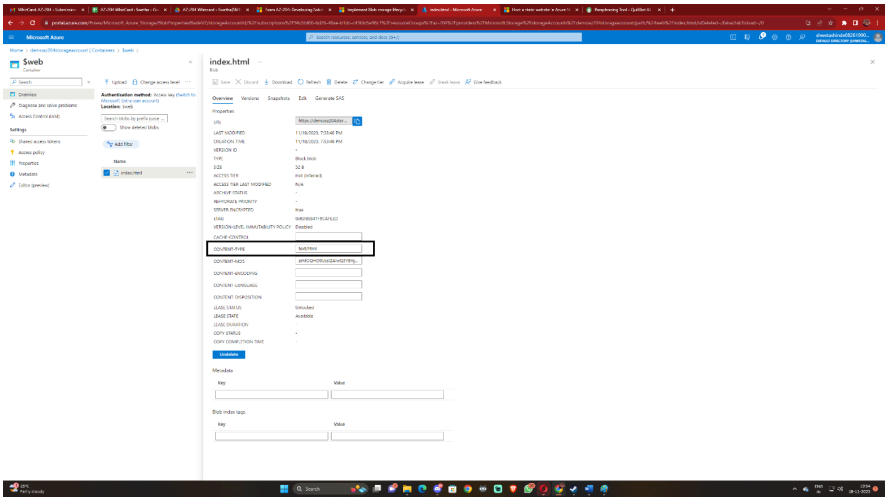
10. To access the Overview pane for a container, choose the $web container in the Containers pane.



11. To access the Upload blob window, click the Upload symbol in the Overview pane. Next, to launch the file browser, pick the Files option in the Upload blob pane. To fill in the Files area, navigate to the file you wish to upload, select it, and then click Open. Choose whether to check the Overwrite if files already exist.



12. Make sure that the file's content type is set to text/html if you want the browser to see its contents. To confirm this, enter the Overview pane by selecting the name of the blob you uploaded in the previous step. Make that the CONTENT-TYPE property field has the value set.



13. Select the URL from the static website from your storage account overview pane and check if it shows your contents.

User Authentication and Authorization

There are various ways through which we can implement authentication and authorization in azure. Below are some of the techniques:

A. Microsoft Identity Platform

The Microsoft Identity Platform is a cloud identity service that lets you build applications where your users and customers can sign in using their Microsoft identities or social accounts.

The Microsoft identity platform for developers is a set of tools that includes authentication service, open-source libraries, and application management tools.

The Microsoft identity platform facilitates the development of applications that users and customers may access using their social media accounts or Microsoft identities. It also grants permission to access your own or Microsoft APIs, such as Microsoft Graph.

The Microsoft identity platform is made up of various parts:

- **OAuth 2.0 and OpenID Connect standard-compliant authentication service :** Developers can authenticate several identity types via OAuth 2.0 and OpenID Connect, which are standards-compliant authentication services. These identity kinds include:
    - Accounts for work or school that are provided by Microsoft Entra ID
    - Individual Microsoft accounts such as Xbox, Skype, and Outlook.com
    - Social or local customer accounts that are provided Microsoft Entra External ID

- **Open-source libraries:** Support for additional standards-compliant libraries as well as Microsoft Authentication Libraries (MSAL)
- **Application management portal**: An Azure portal registration and configuration process, in addition to other Azure administration features.
- **Application configuration API and PowerShell**: To automate your DevOps processes, programmatically configure your applications using PowerShell and the Microsoft Graph API.
- **Microsoft identity platform endpoint**: OIDC certification is held by the Microsoft identity platform endpoint. It is compatible with any standards-compliant library, including the

Microsoft Authentication Libraries (MSAL). It adheres to industry standards by implementing scopes that are readable by humans.

- **Developer content**:Technical documentation that includes code samples, API references, tutorials, quickstarts, and how-to guides.

The Microsoft identity platform allows developers to integrate contemporary identity and security advancements such as Conditional Access, Step-Up authentication, and password less authentication. Applications that are integrated with the Microsoft identity platform automatically utilize this innovation, so you don't have to implement it yourself.

B. Microsoft Entra ID

Microsoft Entra ID is a cloud-based identity and access management service that enables your employees to access external resources. Example resources include Microsoft 365, the Azure portal, and thousands of other SaaS applications.

An application needs to be registered with a Microsoft Entra tenant to assign Identity and Access Management functions to Microsoft Entra ID. An identity configuration for your application that enables integration with Microsoft Entra ID is created when you register it with Microsoft Entra ID. when you register it through the Azure portal. You can select whether an application is:

- **Single tenant:** Only Accessible in your tenant
- **Multi-tenant:** accessible in another tenant

An application object (the globally unique instance of the app) and a service principal object are automatically established in your home tenancy whenever you register an application using the portal. Additionally, your app has an ID that is globally unique (the app or client ID). Then, to make your app function, you can add secrets, certificates, and scopes to the portal. You can also alter the app's branding in the sign-in dialog and do much more.

Application Object

An application on Microsoft Entra is defined by a single application object. The Microsoft Entra tenant (sometimes referred to as the application's "home" tenant) is where the application object is housed. A service principle object or more is created using an application object as a model or blueprint. Every tenant where the application is used has a service principal created in them. The application object includes some static properties that are applied to all new service principals, or application instances, much like a class in object-oriented programming.

Three parts of an application are described by the application object: resources that the application may require access to, actions that the application may take, and how the service can issue tokens to access the application.

Service Principal Object

The entity requiring access must be represented by a security principle in order to access resources protected by a Microsoft Entra tenant. This is true for apps (service principal) as well as users (user principal).

The Microsoft Entra tenant's application or user's permissions and access policies are specified by the security principle. This makes it possible for essential functions like resource access authorization and user/application authentication to take place during sign-in.

There are three types of service principal:

- **Application**: The local representation, or application instance, of a global application object within a single tenant or directory is this kind of service principal. Every tenant where the application is used creates a service principal that points to the globally unique app object. The service principal object specifies who can access the app, what resources it can access, and what the app can truly accomplish within the particular tenant.
- **Managed Identity:** To represent a managed identity, this kind of service principal is employed. Applications can leverage managed identities to establish a connection to resources that enable Microsoft Entra authentication. In your tenancy, a service principal representing a managed identity is established when it is enabled. Access and permissions can be given to service principals that represent managed identities, but they cannot be directly updated or changed.
- **Legacy**: An app developed using legacy experiences or prior to the introduction of app registrations is represented by this kind of service principle. Although a legacy service principal lacks an accompanying app registration, it can include features such as reply URLs, credentials, and service principal names that can be edited by an authorized user. Only the tenant in which it was initially created may use the service principal.

The service principal is the local representation of your application for use in a particular tenant, whereas the application object is the global representation for usage across all tenants. To create comparable service principal objects, common and default properties are obtained from the application object, which acts as a template.

There are two types of relationships:

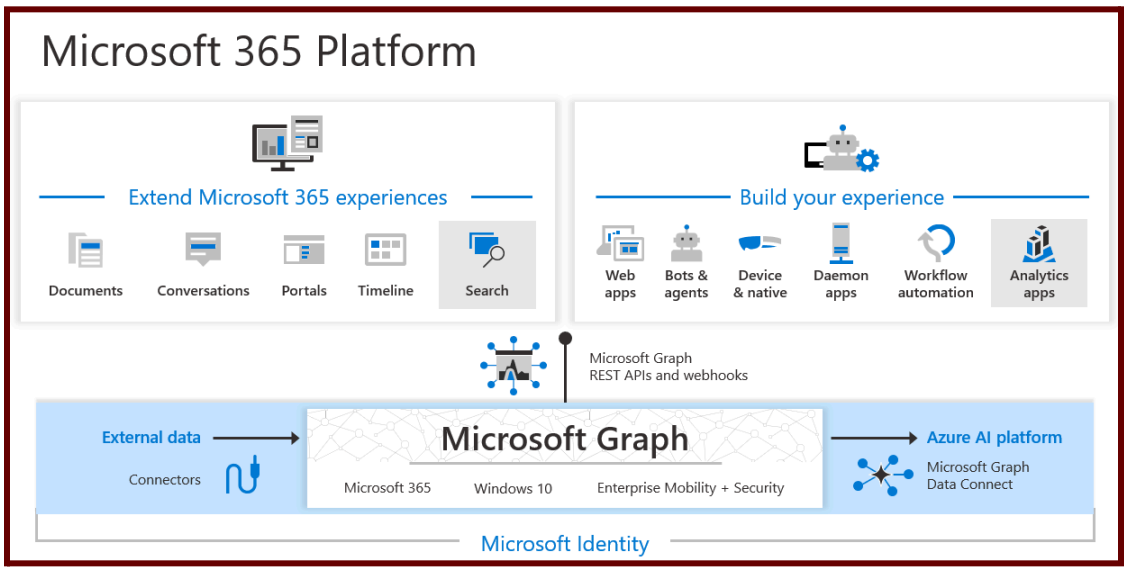
- one to many with the corresponding service primary object(s) and
- one to one with the software application.

In order for the application to establish an identity for sign-in and/or access to resources that

the tenant has secured, a service principal needs to be generated for each tenant. An application for a single tenant only has one service principal, which is the home tenant, and is produced and approved for usage at the time of application registration. In addition, a service principal is established in each tenant of a multi-tenant application when a user has granted permission for its use.

C. Microsoft Graph

The entry point to data and intelligence in Microsoft 365 is Microsoft Graph. You can leverage the uniform programmability paradigm it offers to access the massive amounts of data found in Windows 10, Microsoft 365, and Enterprise Mobility + Security.



Ref - learn.microsoft.com

Three key elements of the Microsoft 365 platform make data access and flow easier:

- There is just one endpoint available for the Microsoft Graph API: https://graph.microsoft.com. To access the endpoint, you can utilize SDKs or REST APIs. A robust suite of services that control user and device identification, access, compliance, security, and aid in shielding enterprises from data loss or leakage are also included in Microsoft Graph.

- In order to improve Microsoft 365 experiences like Microsoft Search, Microsoft Graph connectors operate in the incoming direction, sending data from outside the Microsoft cloud into Microsoft Graph services and applications. Many popular data sources, including Box, Google Drive, Jira, and Salesforce, have connectors available.

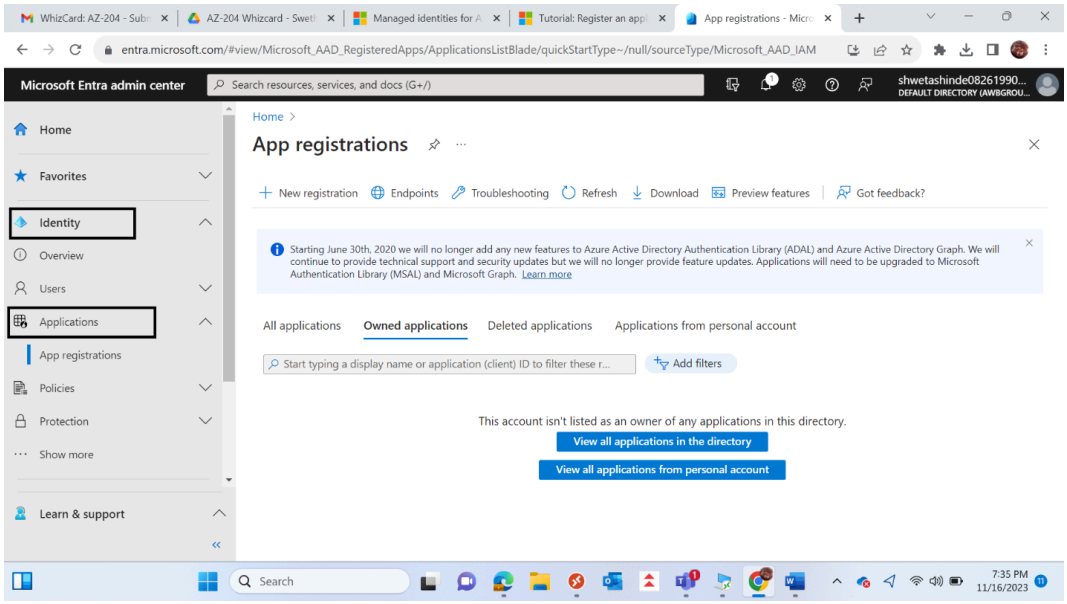
- A collection of tools called Microsoft Graph Data Connect makes it easier to deliver Microsoft Graph data to well-known Azure data stores in a safe and scalable manner. You can utilize the Azure development tools to create intelligent applications by using the cached data as data sources.

Implementation

Microsoft Entra ID needs to be informed about the application you develop in order for it to communicate with the Microsoft identity platform. This guide explains how to register an application on the Azure portal within a tenant.

Register the application and record identifiers.

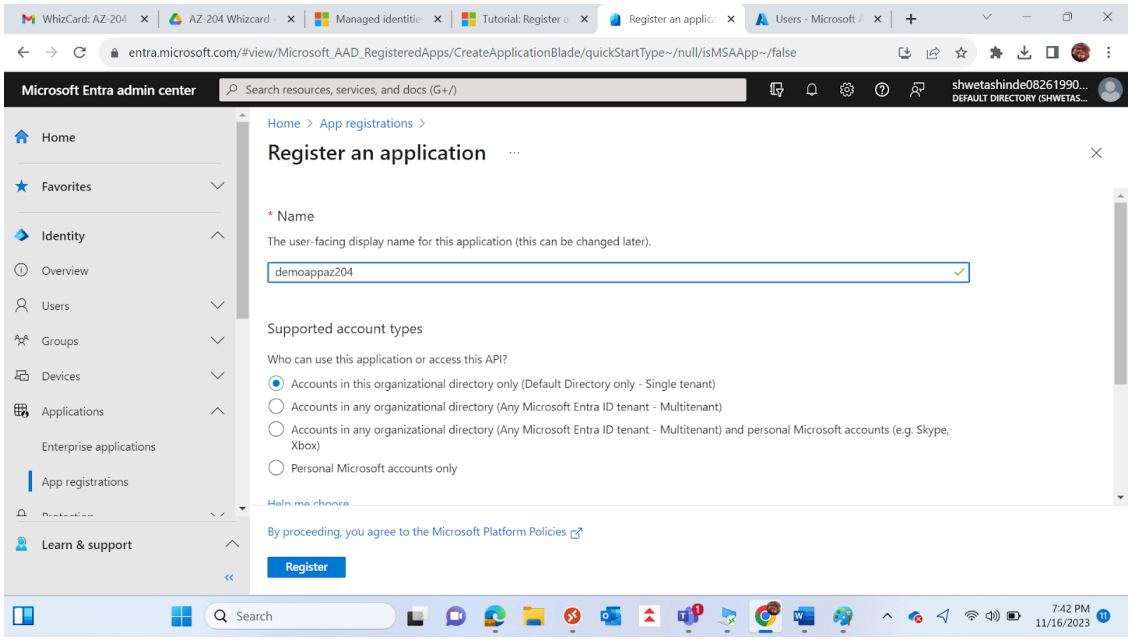
1. Sign into Microsoft Entra admin center with any of the role (Application developer/Application Administrator/ Cloud application administrator).
2. If you have access to more than one tenant, select the tenant from the **Directories + Subscriptions** menu using the Settings icon in the top menu before registering the application.
3. Go to **Identity** > **Applications** > **Application registrations**.



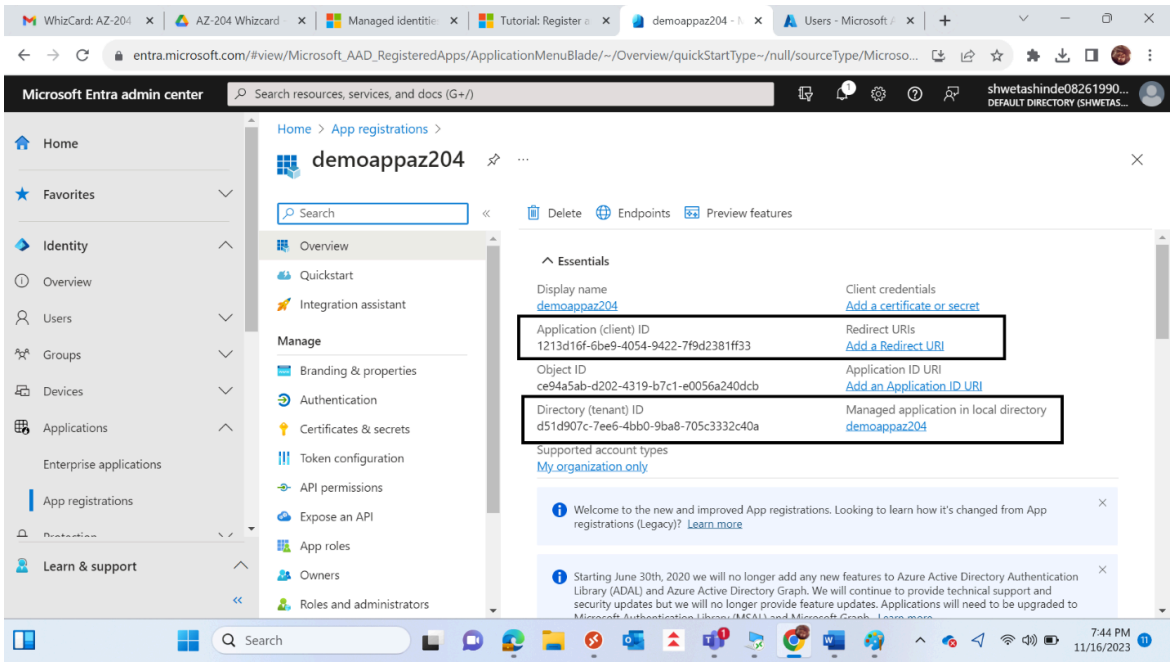
4. Select **New Registration**.
5. Enter a Name for your application eg.Myfirstwebapp/demowebapp
6. Choose Accounts in this organizational directory only under Supported account types. To learn more about the various account kinds, click the Help me pick menu item.

- Later on, the Redirect URI (optional) will be specified.

7. Select **Register**.

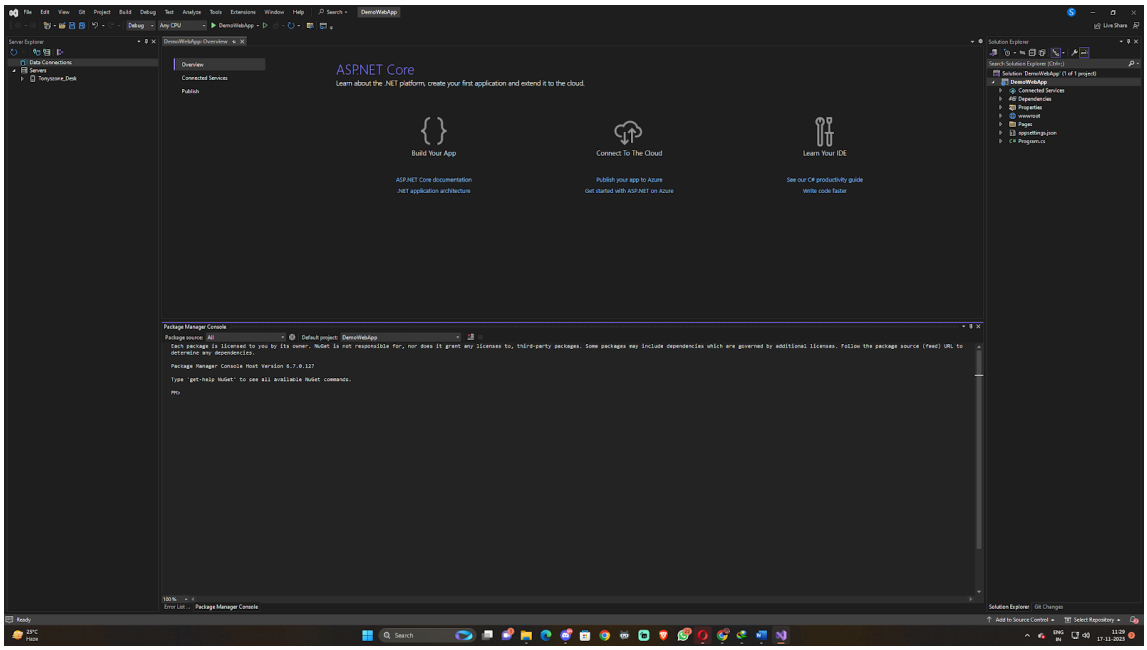


8. The application's **Overview** pane is displayed when registration is complete. Record the **Directory (tenant) ID** and the **Application (client) ID** to be used in your application source code. – need to rephrase from this point



Create an application for authentication.

1) Open visual studio and click on a new project.
2) Search for and choose the ASP.NET Core Web App template, and then select Next.
3) Enter a name for the project, such as DemoWebApp.
4) Choose a location for the project or accept the default option, and then select Next.
5) Accept the default for the Framework, Authentication type, and Configure for HTTPS. Authentication type can be set to none as this tutorial will cover this process.
6) Select create

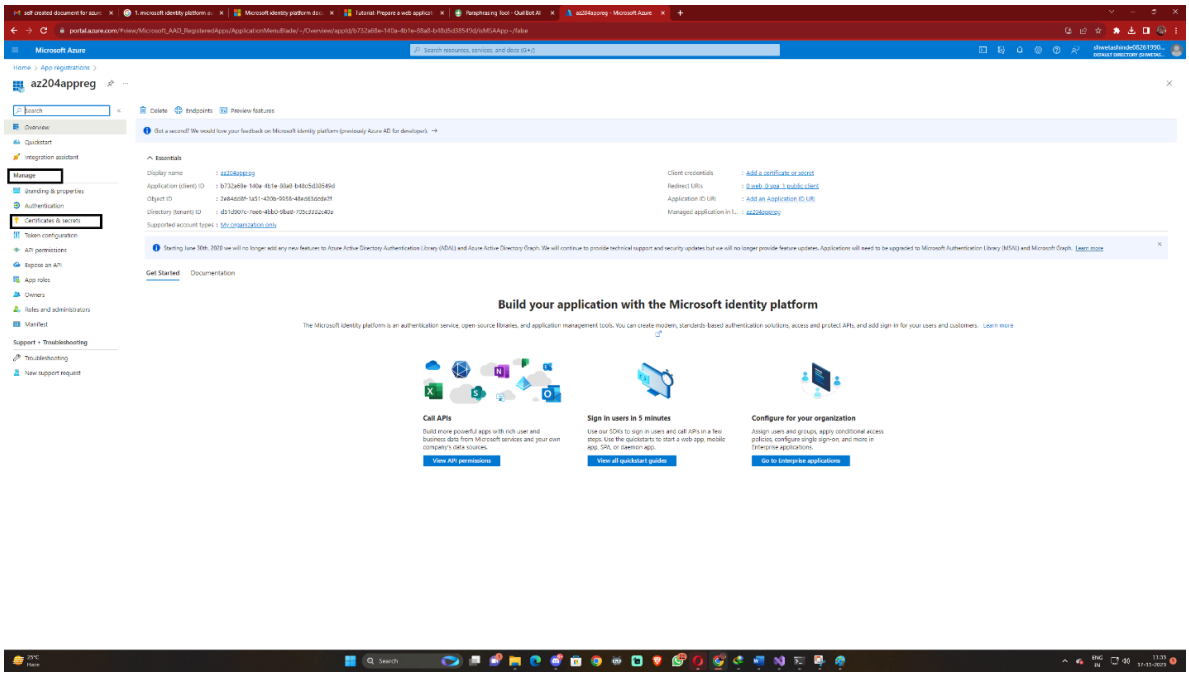


7) One recommended method for safeguarding client-server communication is to utilize certificates.

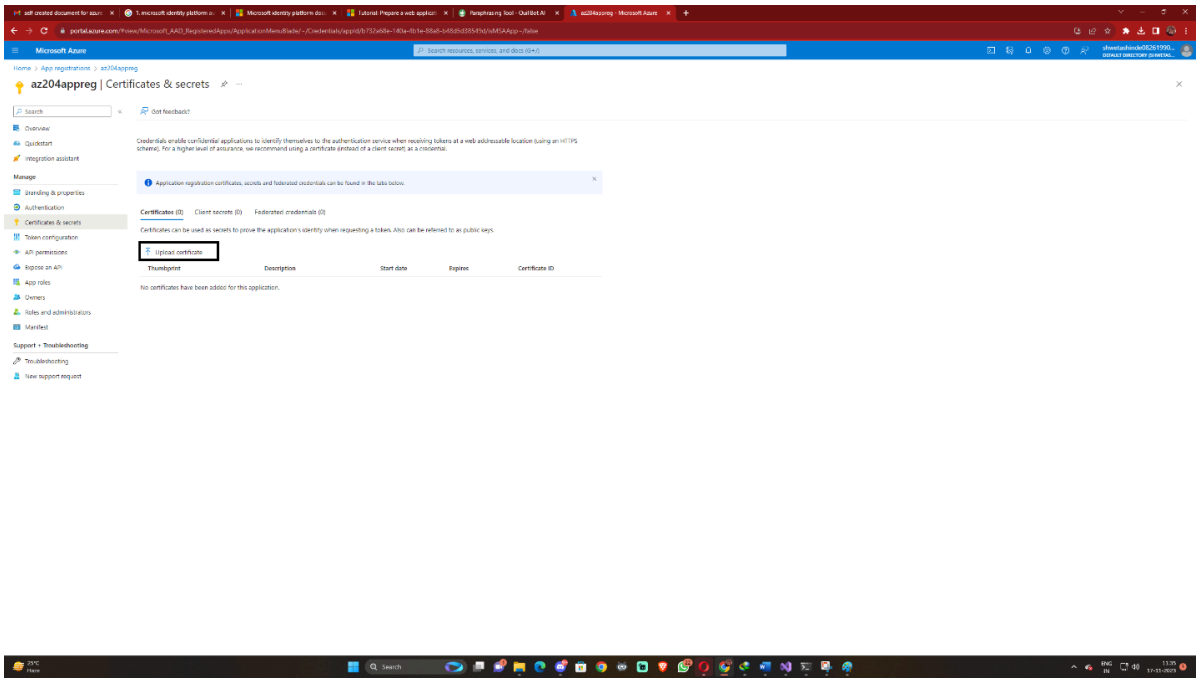
   Select Tools > Command Line > Developer Command Prompt.
8) Enter the following command to create a new self-signed certificate:

dotnet dev-certs https -ep ./certificate.crt –trust

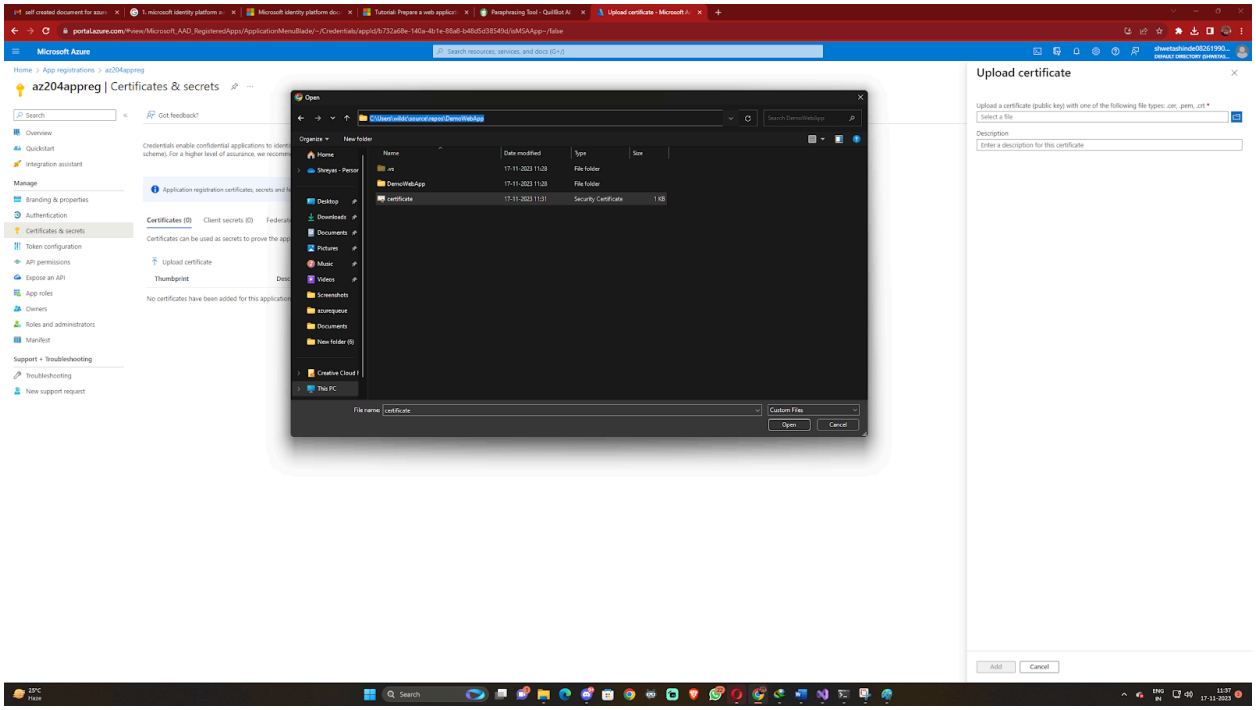
9) Starting from the Overview page of the app created earlier, under Manage, select Certificates & secrets and select the Certificates (0) tab.



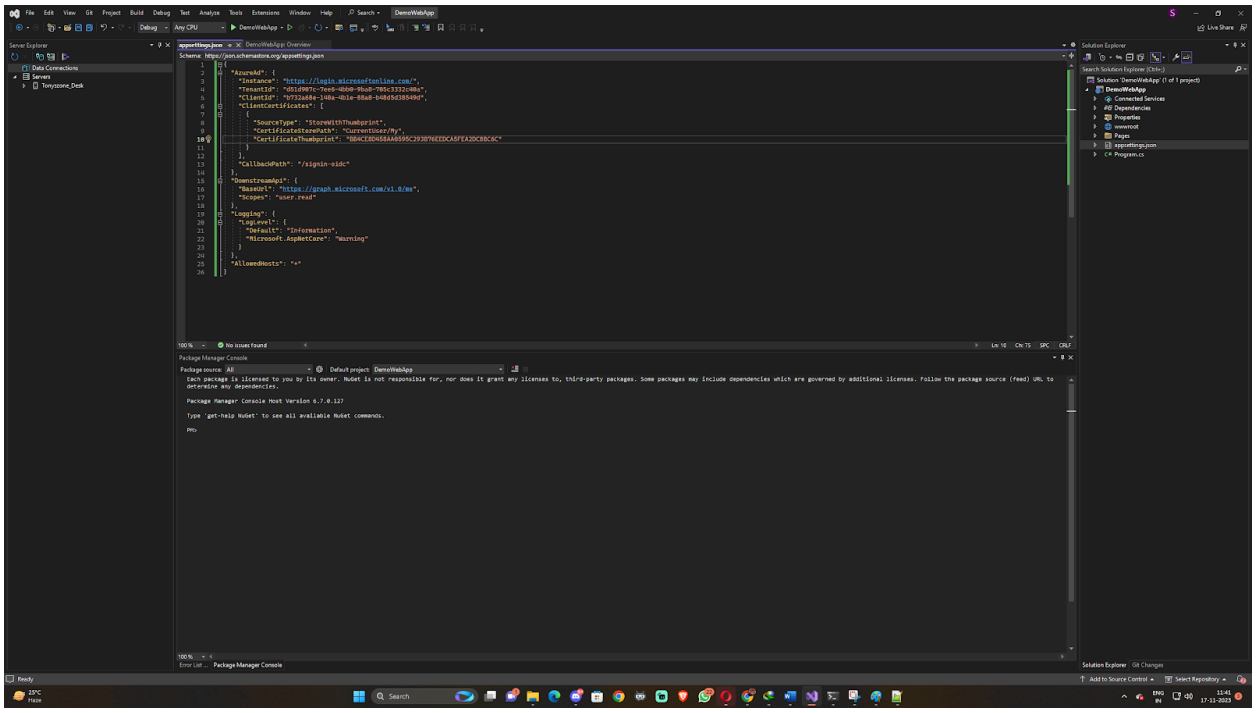
10) Click on the upload certificate.



11) Select the folder icon, then browse for and select the certificate that was previously created. The certificate will be available in your app folder which we created through visual studio.



12) Enter a description for the certificate and select Add.
13) Save the Thumbprint value, which will be used in the next step.
14) In your visual studio, open appsettings.json and replace the file contents with the following snippet:

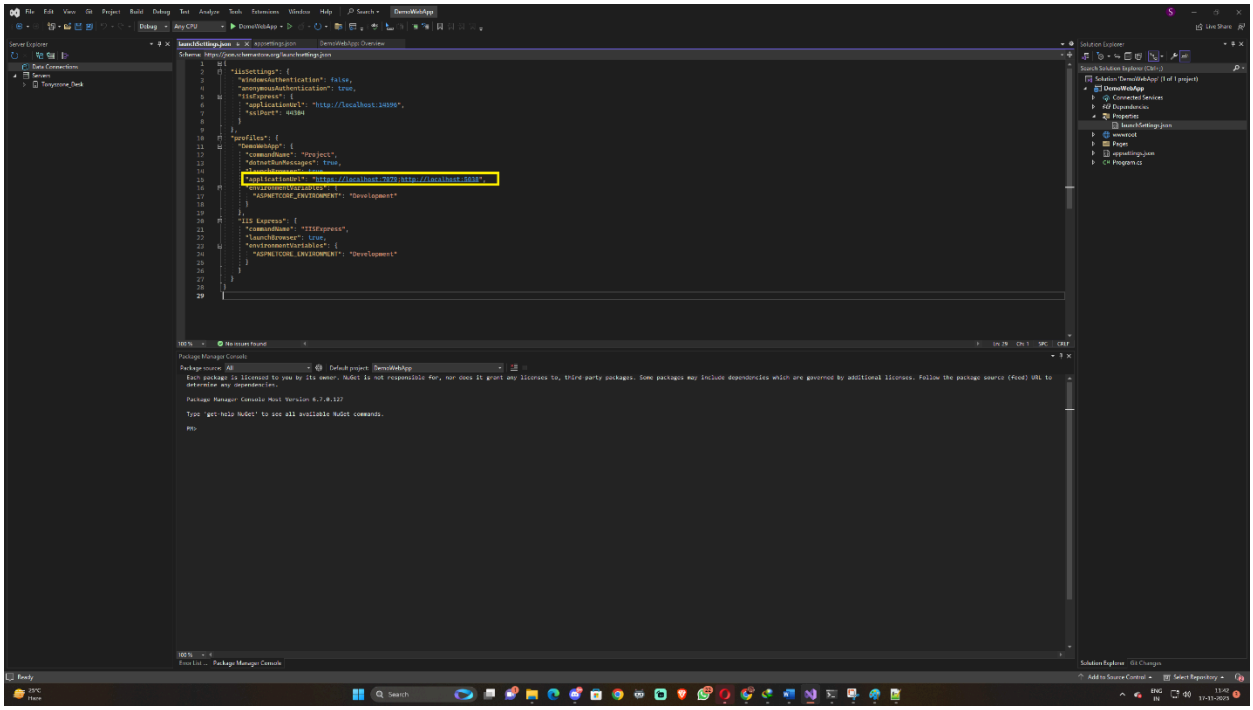


Code –

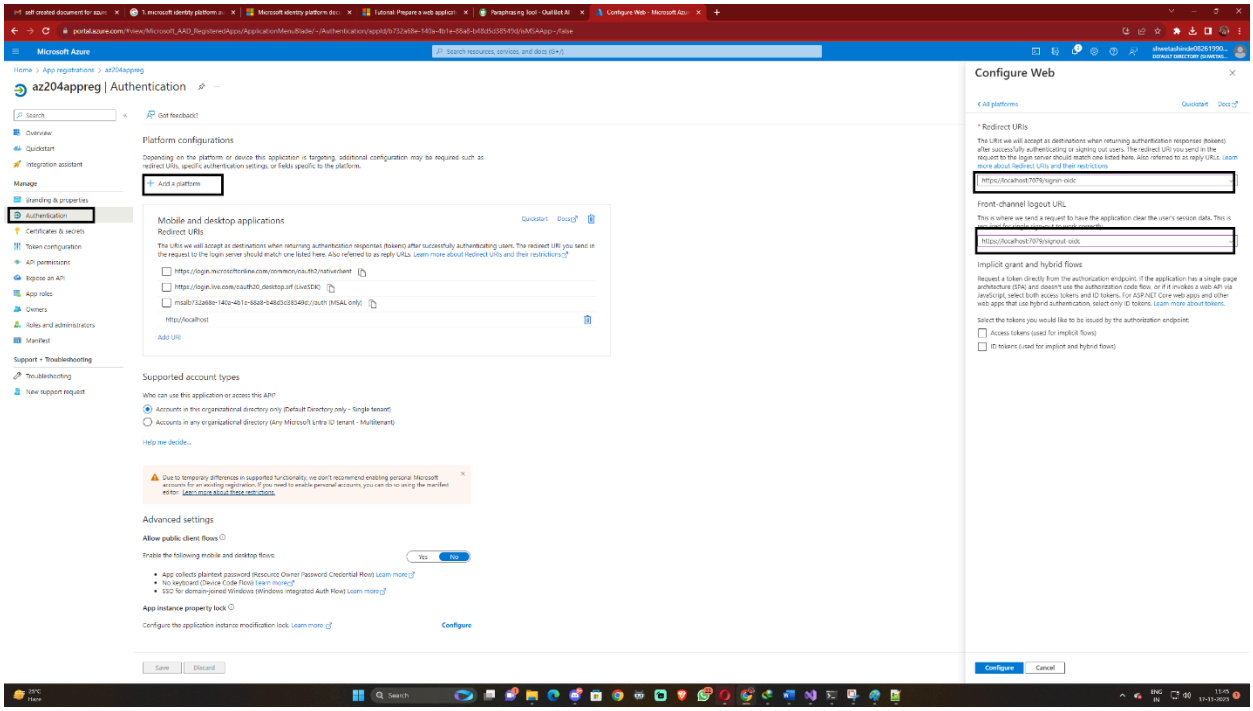
```
{
 "AzureAd": {
  "Instance": "https://login.microsoftonline.com/",
  "TenantId": "Enter the tenant ID obtained from the Azure portal",
  "ClientId": "Enter the client ID obtained from the Azure portal",
  "ClientCertificates": [
   {
    "SourceType": "StoreWithThumbprint",
    "CertificateStorePath": "CurrentUser/My",
    "CertificateThumbprint": "Enter the certificate thumbprint obtained from the Azure portal"
   }
  ],
  "CallbackPath": "/signin-oidc"
 },
 "DownstreamApi": {
  "BaseUrl": "https://graph.microsoft.com/v1.0/me",
  "Scopes": "user.read"
 },
 "Logging": {
  "LogLevel": {
   "Default": "Information",
   "Microsoft.AspNetCore": "Warning"
  }
 },
 "AllowedHosts": "*"
}
```

And then save the file

15) In the Properties folder, open the launch Settings.json file.
16) Find and save the https value applicationURI within launchSettings.json, for example https://localhost:{port}. This URL will be used when defining the Redirect URI.



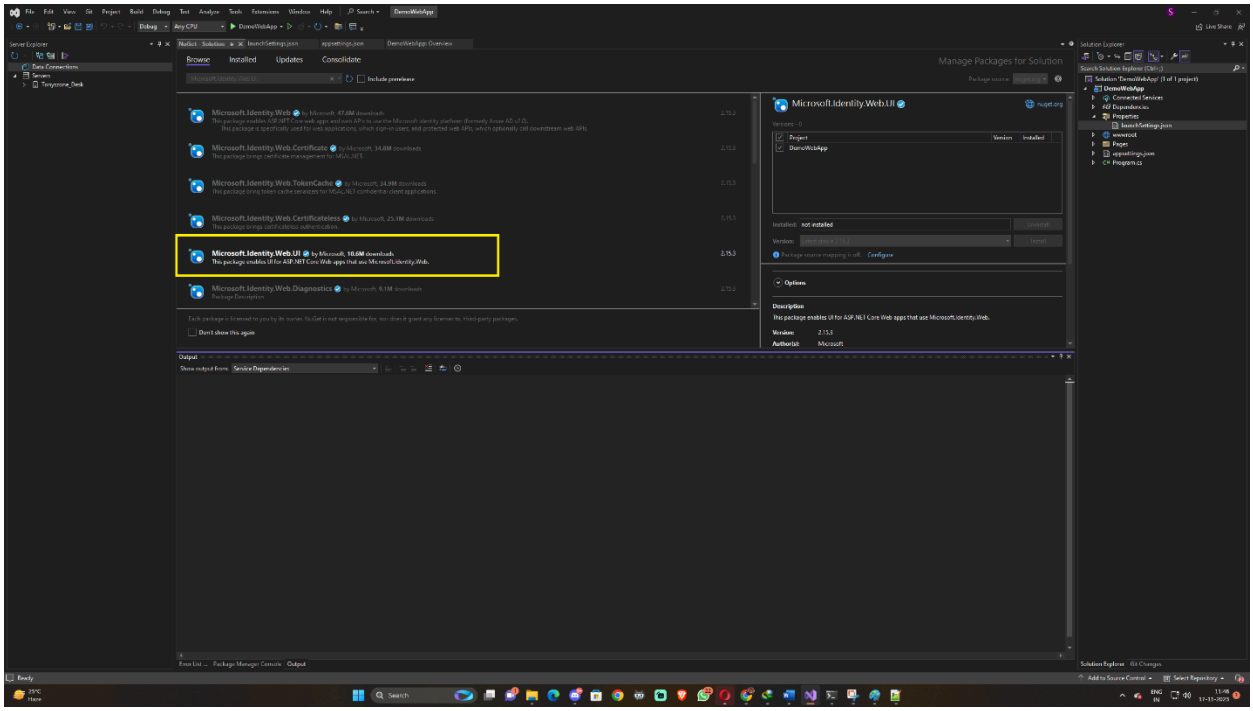
17) In the Azure portal, under Manage, select App registrations, and then select the application that was previously created.
18) In the left menu, under Manage, select Authentication.
19) In Platform configurations, select Add a platform, and then select Web.



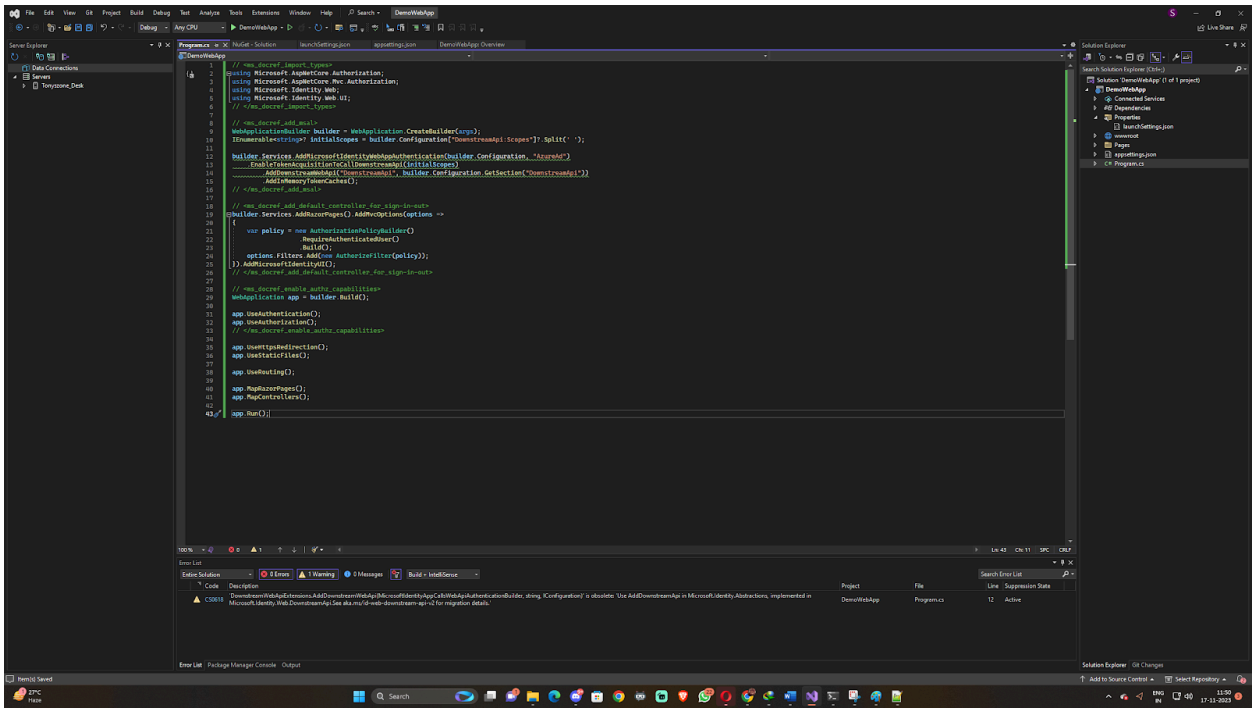
20) Click on configure.
21) Identity related NuGet packages must be installed in the project for authentication of users to be enabled.

In the top menu of Visual Studio, select Tools > NuGet Package Manager > Manage NuGet Packages for Solution.

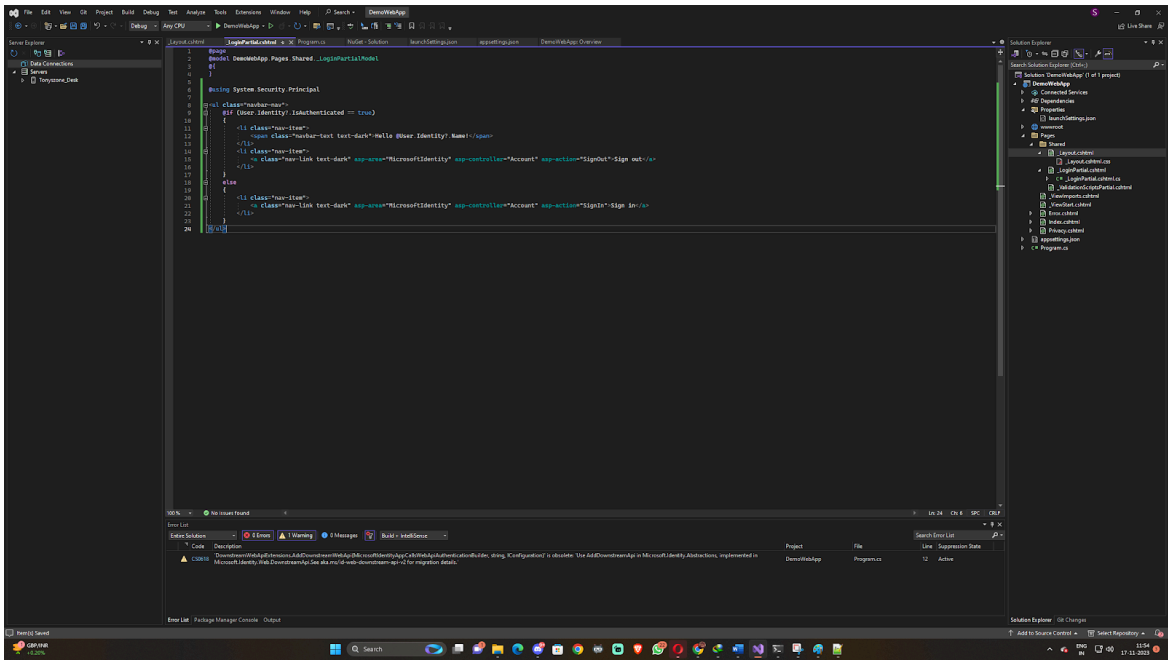
22) With the Browse tab selected, search for and select Microsoft.Identity.Web.UI. Select the Project checkbox, and then select Install and once prompted click on I Agree.



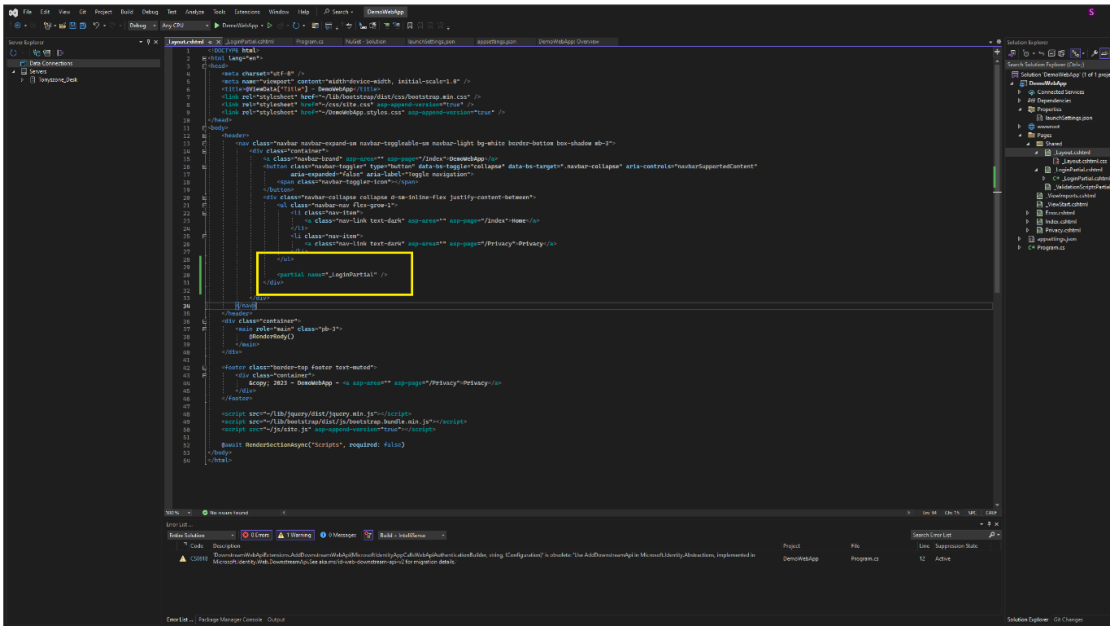
23) Open Program.cs and replace the entire file contents with the following code:



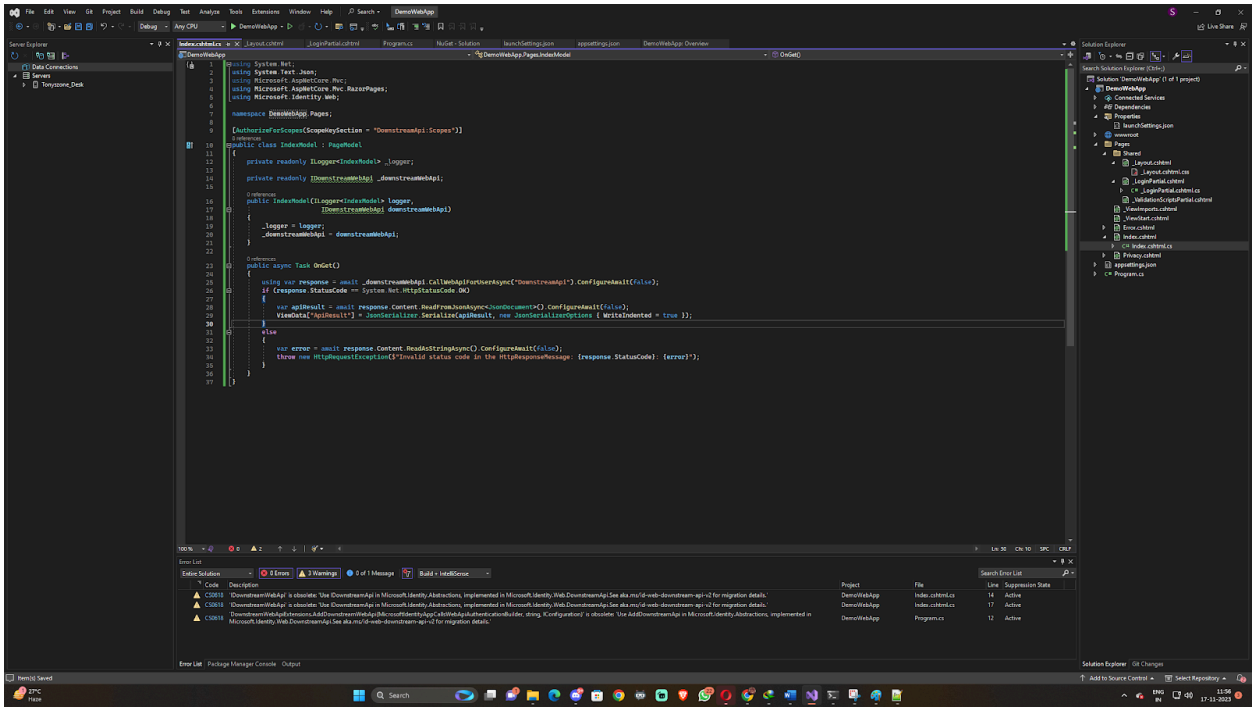
24) Expand Pages, right-click Shared, and then select Add > Razor page.
25) Select Razor Page - Empty, and then select Add.
26) Enter _LoginPartial.cshtml for the name, and then select Add.
27) Open _LoginPartial.cshtml and add the following code for adding the sign in and sign out experience:



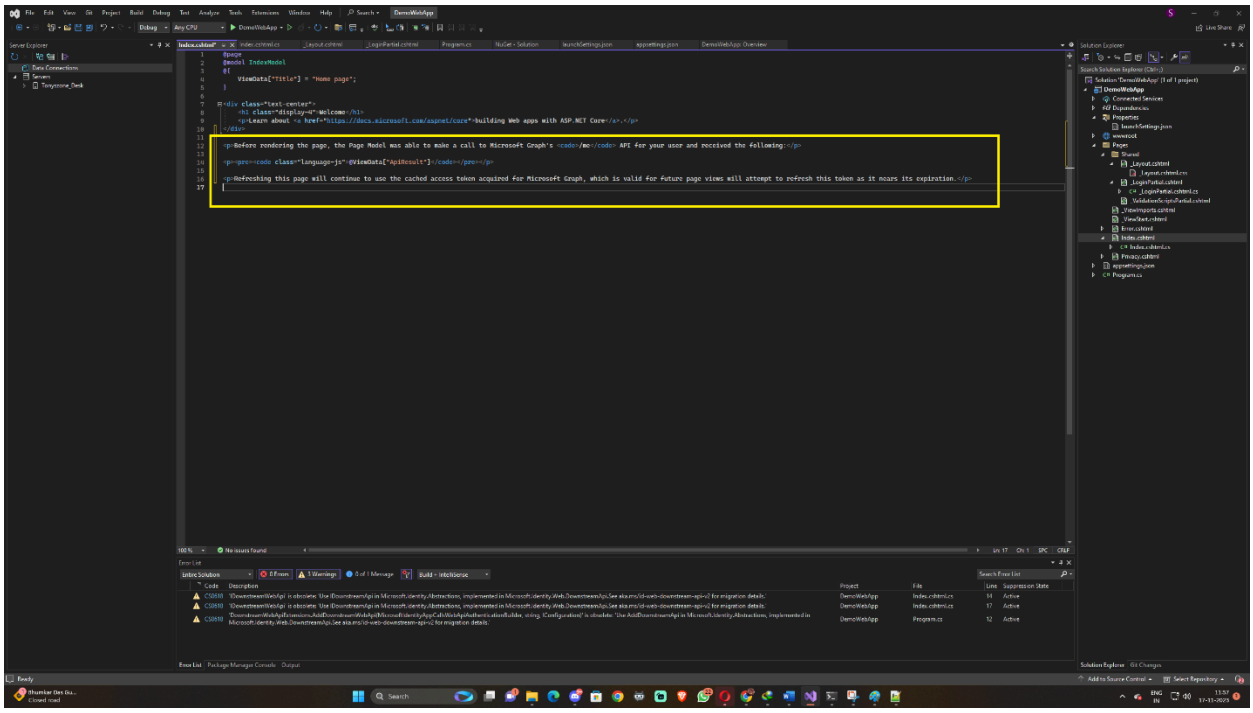
28) Open _Layout.cshtml and add a reference to _LoginPartial created in the previous step. This single line should be placed between and :



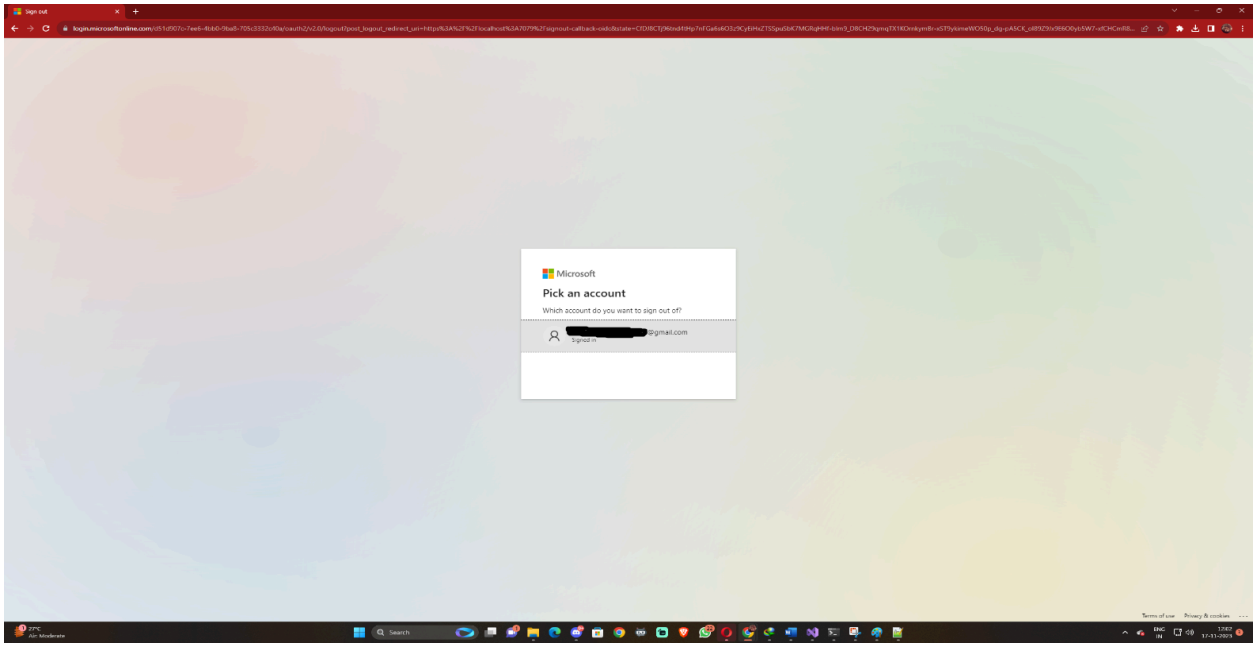
29) Under Pages, open the Index.cshtml.cs file and replace the entire contents of the file with the following snippet. Check that the project namespace matches your project name.



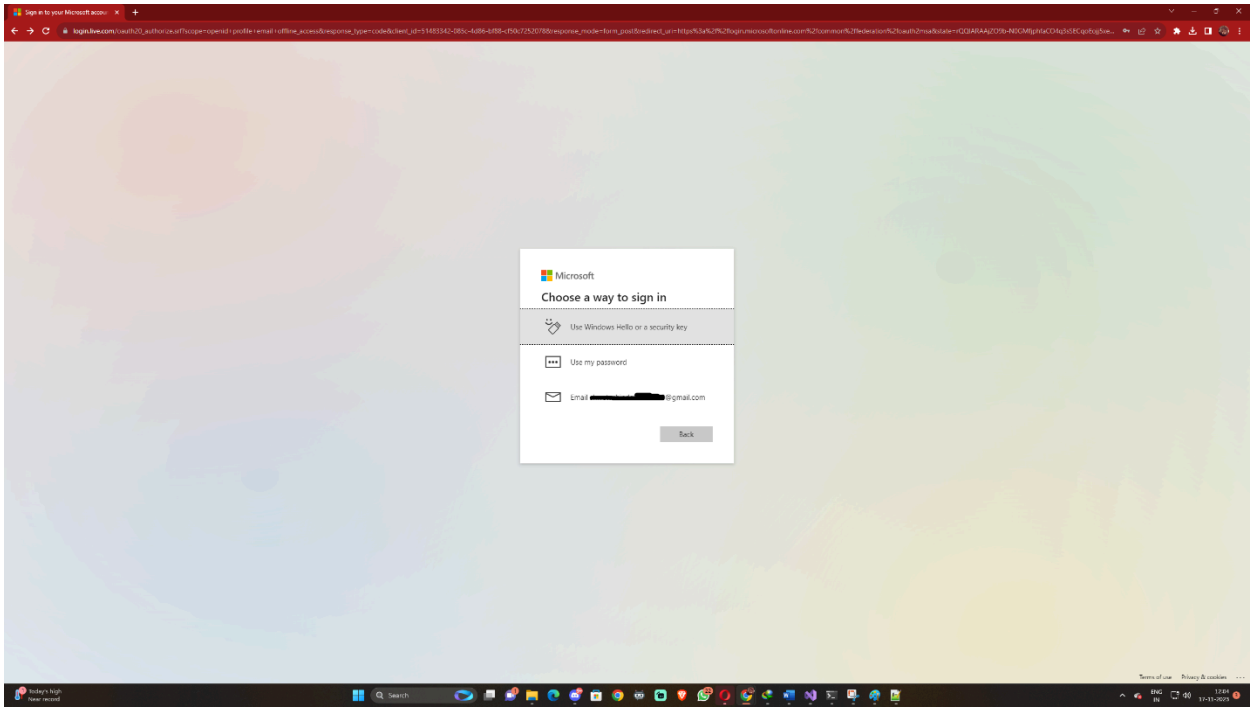
30) Open Index.cshtml and add the following code to the bottom of the file. This will handle how the information received from the API is displayed:



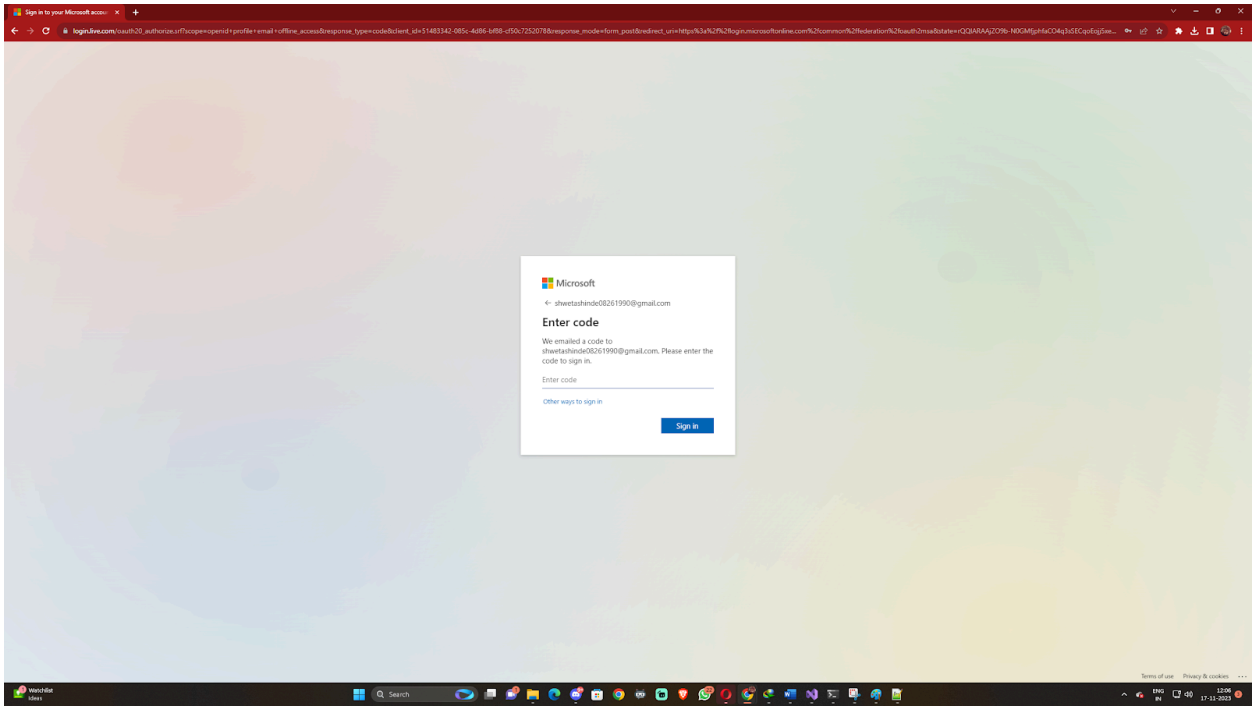
31) Run the application. After the sign in window appears, select the account in which to sign in with. Ensure the account matches the criteria of the app registration.



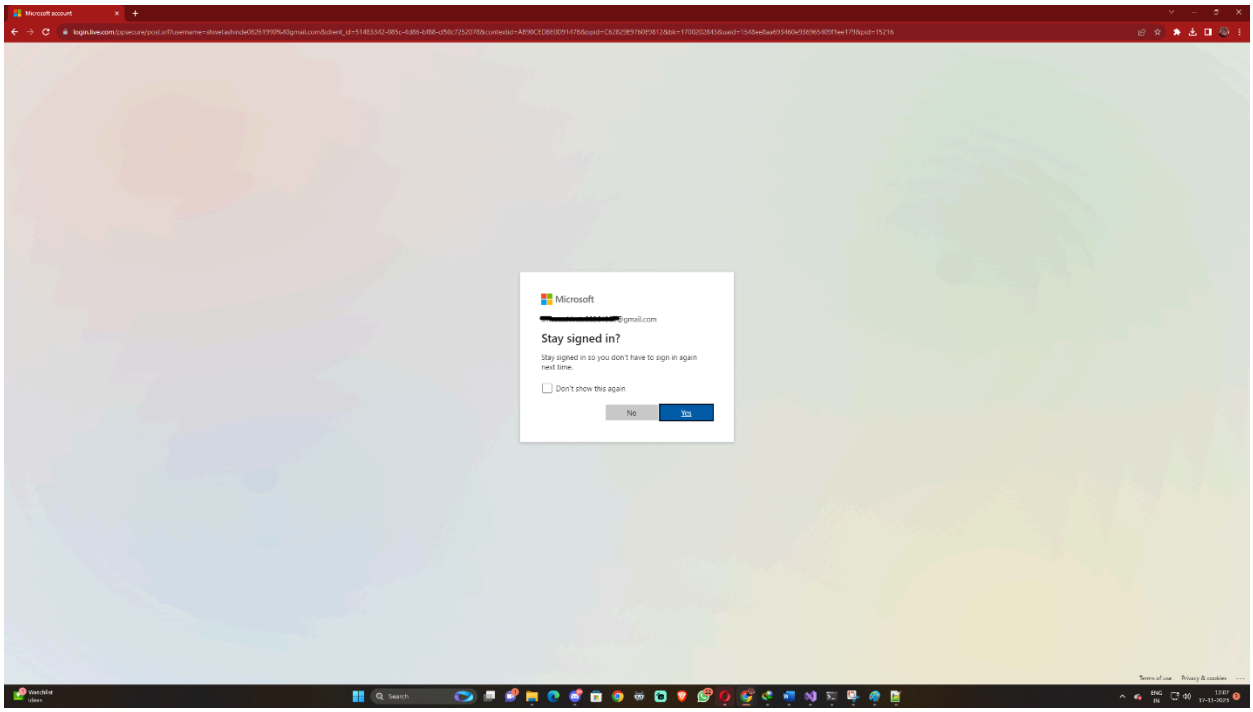
32) Upon selecting the account, a second window appears indicating that a code will be sent to your email address. Select Send code, and check your email inbox.



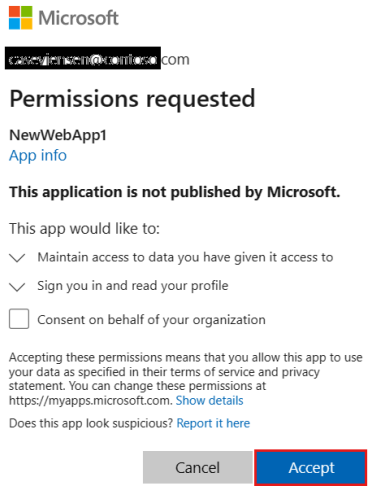
33) Open the email from the sender Microsoft account team, and enter the 7-digit single-use code. Once entered, select Sign in.



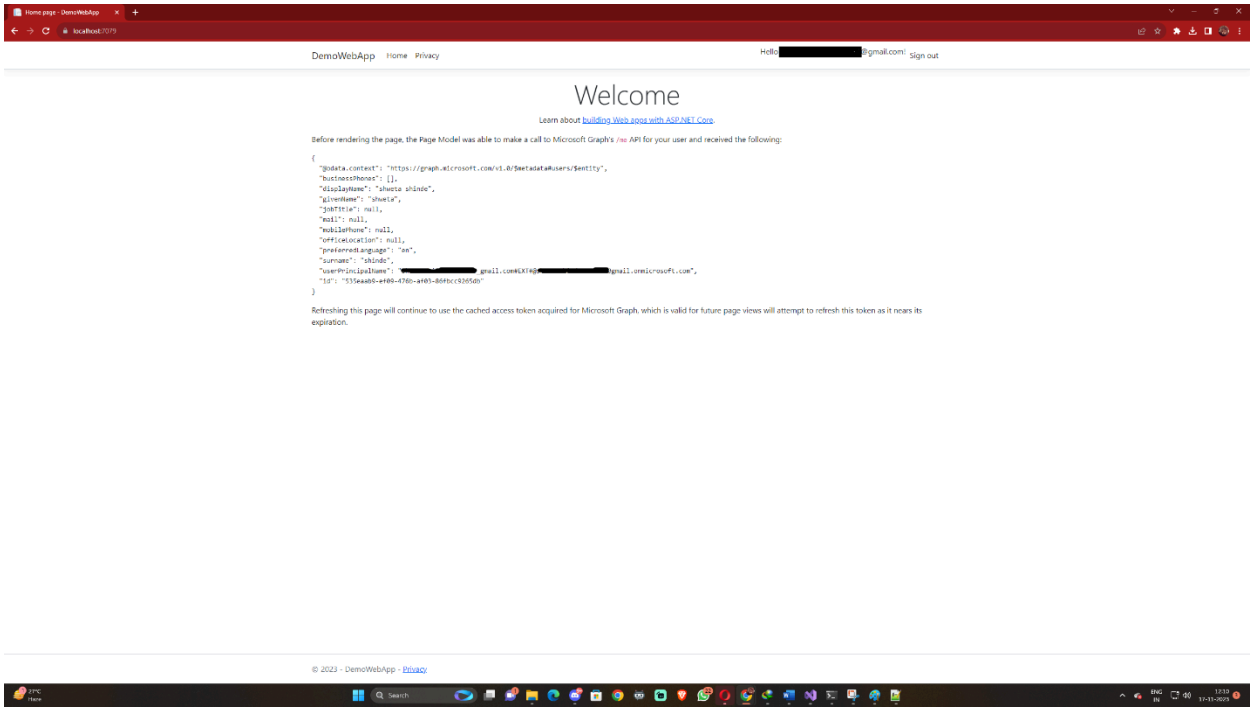
34) For Stay signed in, you can select either No or Yes.



35) The app will ask for permission to sign in and access data. Select Accept to continue.



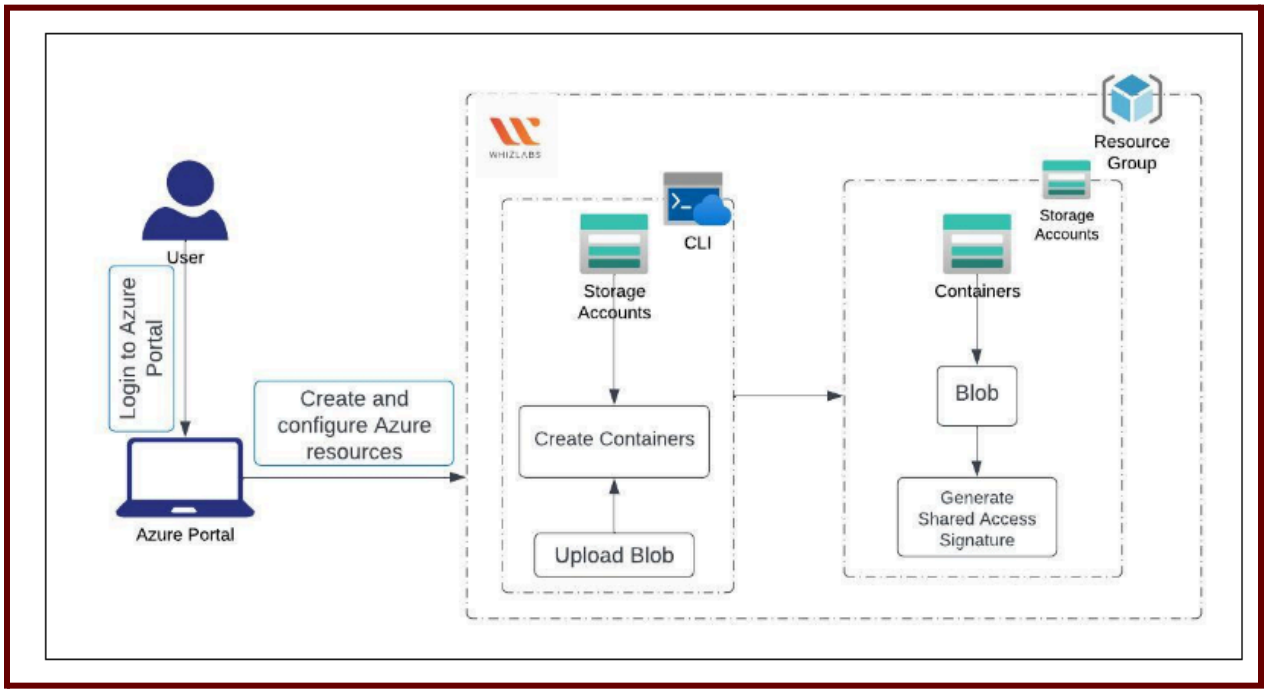
36) The web app now displays profile data acquired from the Microsoft Graph API.



Shared Access Signatures (SAS)

A **Shared Access Signature (SAS)** is a signed URI that refers to one or more storage resources and contains a token containing unique query parameters. The token indicates how the resource can be accessed by the client.

Shared Access Signature (SAS) provides secure delegated access to resources in your storage account. With SAS, you have granular control over how a client can access your data. For example: What resources the client may access, What permissions they have to those resources, and How long the SAS is valid.



A signature, one of the query parameters, is constructed from the SAS parameters and signed with the key used to create the SAS. This signature is used by Azure Storage to authenticate access to the storage resource.

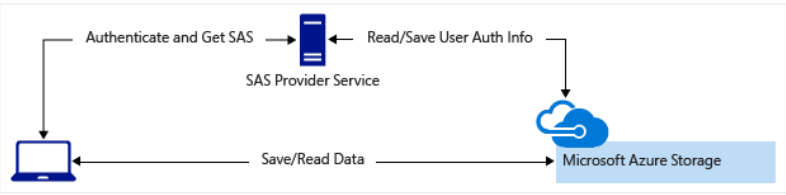
When to use a shared access signature

Use SAS to provide secure access to resources in your storage account to any client that does not have permissions to those resources.

A common scenario where SAS is useful is as a service where users read and write their own data to your storage account.
In the scenario where a storage account stores user data, there are two common design patterns:

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

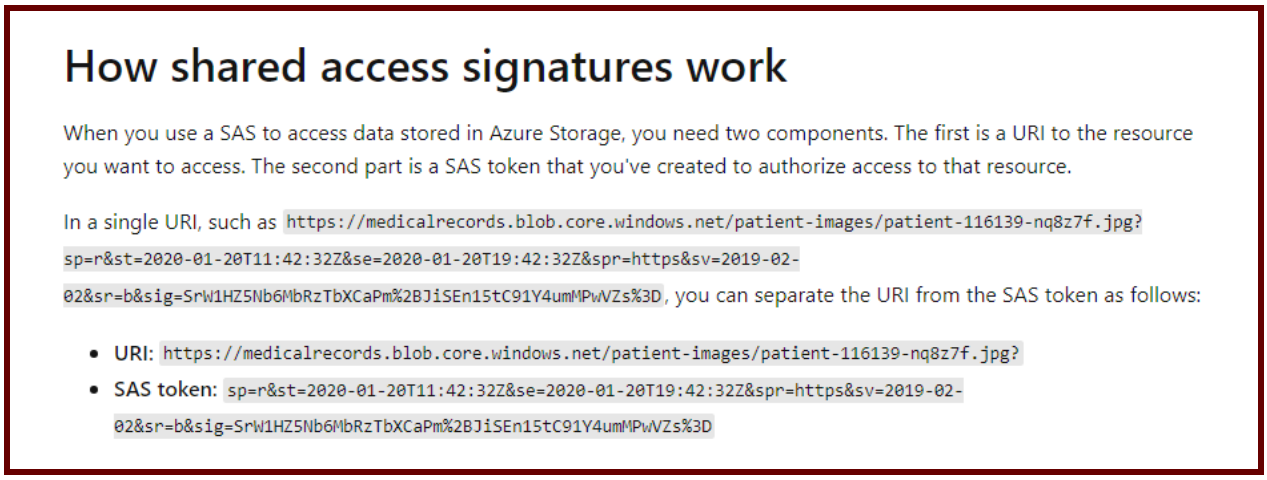
2. A lightweight service authenticates the client as needed and then generates a SAS. Once the client application receives the SAS, it can access storage account resources directly. Access permissions are defined by the SAS and for the interval allowed by the SAS. The SAS mitigates the need for routing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



Source: Microsoft Documentation (Choose when to use shared access signatures)

Types of shared access signatures

- **User delegation SAS:** A user delegation SAS is secured with Microsoft Entra credentials and also by the permissions specified for the SAS. A user delegation SAS applies to Blob storage only.
- **Service SAS:** A service SAS is secured with the storage account key. A service SAS delegates access to a resource in the following Azure Storage services: Blob storage, Queue storage, Table storage, or Azure Files.
- **Account SAS**: The account is secured with a SAS storage account key. A SAS account provides access to resources in one or more storage services. All operations available through Service or User Delegation SAS are also available through Account SAS.



How shared access signatures work

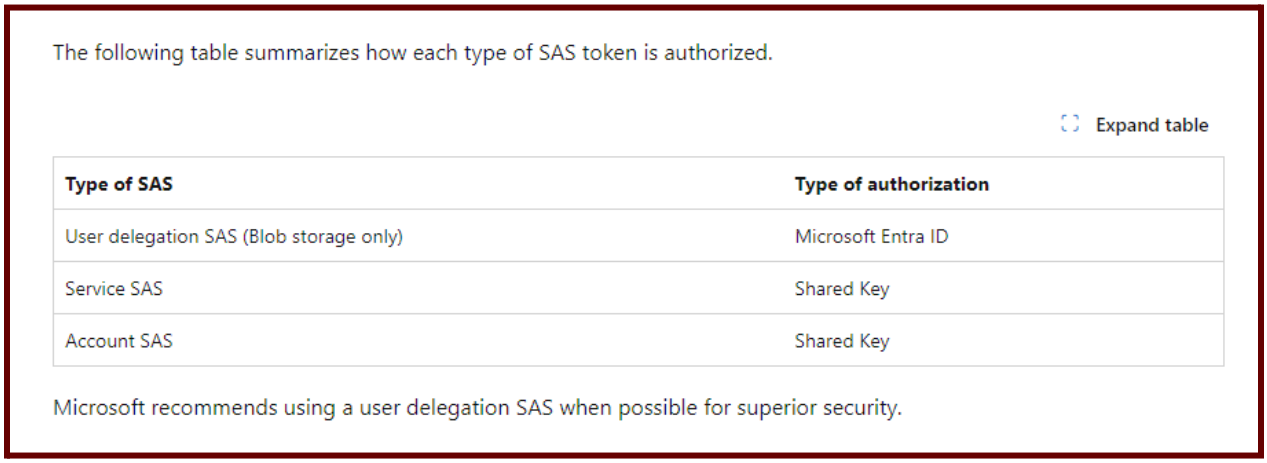
When you use a SAS to access data stored in Azure Storage, you need two components. The first is a URI to the resource you want to access. The second part is a SAS token that you've created to authorize access to that resource.

In a single URI, such as `https://medicalrecords.blob.core.windows.net/patient-images/patient-116139-nq8z7f.jpg?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`, you can separate the URI from the SAS token as follows:

- URI: `https://medicalrecords.blob.core.windows.net/patient-images/patient-116139-nq8z7f.jpg?`
- SAS token: `sp=r&st=2020-01-20T11:42:32Z&se=2020-01-20T19:42:32Z&spr=https&sv=2019-02-02&sr=b&sig=SrW1HZ5Nb6MbRzTbXCaPm%2BJiSEn15tC91Y4umMPwVZs%3D`

Source: Microsoft Documentation (Discover shared access signatures)

Azure SAS token Authorization



The following table summarizes how each type of SAS token is authorized.

Expand table

Type of SAS	Type of authorization
User delegation SAS (Blob storage only)	Microsoft Entra ID
Service SAS	Shared Key
Account SAS	Shared Key

Microsoft recommends using a user delegation SAS when possible for superior security.

Whizlabs Hands-on-labs Links

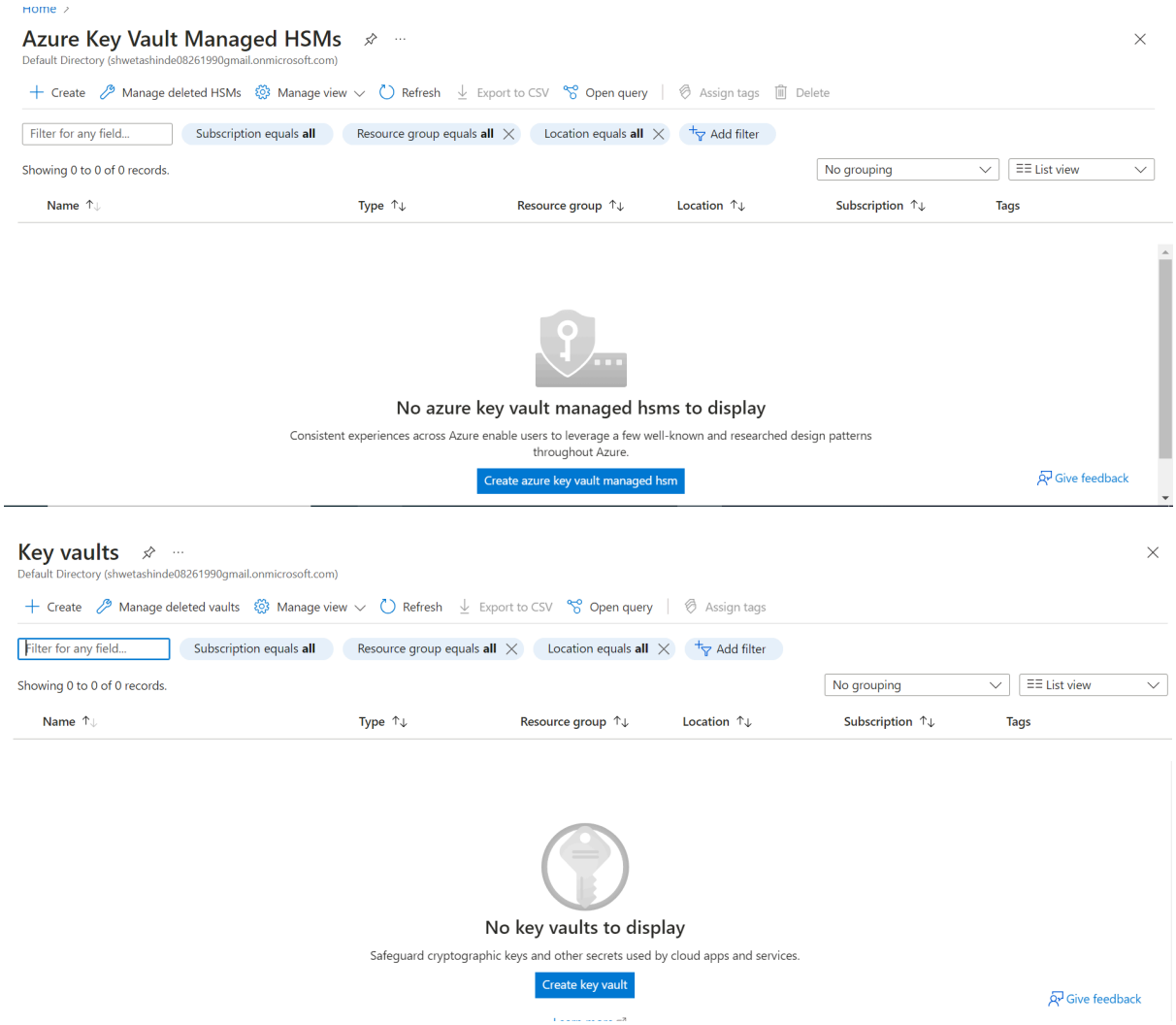
- Working with Shared Access Signatures (whizlabs.com)

References Links:

- Grant limited access to data with shared access signatures (SAS) - Azure Storage
- Create shared access signature (SAS) tokens for your storage containers and blobs Implement shared access signatures - Training | Microsoft Learn

Azure Key Vault

Managed hardware security module (HSM) pools and vaults are the two types of containers that the Azure Key Vault service supports. Software, HSM-backed keys, secrets, and certificates can all be stored in vaults. Only HSM-backed keys are supported by managed HSM pools.



Azure Key Vault assists in resolving the following issues:

- **Secrets Management:** Tokens, passwords, certificates, API keys, and other secrets may be safely stored and strictly controlled with Azure Key Vault.
- **Key management:** Another option for key management is Azure Key Vault. Creating and managing the encryption keys that are used to secure your data is made simple with Azure Key Vault.

- **Certificate Management**: Public and private Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates for usage with Azure and your internal connected resources can be conveniently provisioned, managed, and deployed with Azure Key Vault.

There are two service tiers for Azure Key Vault: Standard, which uses a software key for encryption, and Premium, which has keys safeguarded by a hardware security module (HSM). Visit the Azure Key Vault price page to learn how the Standard and Premium tiers differ from one another.

Benefits

- **Centralized application secrets:** You can manage the dissemination of your application secrets by centrally storing them in Azure Key Vault. For instance, you can safely store the connection string in Key Vault rather than in the app's code. URIs allow your apps to safely retrieve the data they require. Specific versions of a secret can be retrieved by the applications using these URIs.
- **Safely keep secrets and keys:** Before a caller (user or program) may get entry to a key vault, they must first be properly authenticated and authorized. Microsoft Entra ID is used for authentication. Key Vault access policy or Azure role-based access control (Azure RBAC) can be used for authorization. Key vault access policy is used when trying to access data stored in a vault, and Azure RBAC is utilized for managing the vaults. With the Azure Key Vault Premium tier, hardware security modules (HSMs) provide hardware protection for Azure Key Vaults in addition to software protection.
- **Monitor access and use**: You can keep an eye on things by turning on logging for your vaults. You have authority over your logs; you can remove records that you no longer require or secure them by limiting access. Configuring Azure Key Vault allows you to:

    i. To a storage account, archive.
    ii. To an event hub, stream.
    iii. Logs should be sent to Azure Monitor logs.

    - **Simplified management of application secrets**: High availability, life cycle compliance, and security are requirements for security information. Meeting these needs is made easier with Azure Key Vault by:
        - Eliminating the Requirement for Internal Hardware Security Module Knowledge adjusting quickly to meet sudden spikes in the use of your company.
        - replicating your Key Vault's contents both to a secondary region and within a region. High availability is ensured by data replication, which eliminates the requirement for administrator intervention to start the failover.
        - offering normal PowerShell, Azure CLI, and portal administration options.

- automating processes like enrolment and renewal for certifications that you buy from public certification authorities.

Authentication

You must authenticate with Key Vault before you can do any operations on it. To authenticate to Key Vault, there are three options:

- **Managed identities for Azure resources:** You can give your virtual machine an identity that allows it to access Key Vault when you install an application on it. Other Azure resources can also have identities assigned to them. One advantage of this strategy is that the app or service does not control how the initial secret is rotated. Azure rotates the identity's service primary client secret automatically. This method is advised as a best practice.
- **Service principal and certificate:** Key Vault access is possible with the usage of a service principal and certificate that go hand in hand. Since the developer or owner of the program must rotate the certificate, we do not advise using this method.
- **Service principal and secret:** We do not advise using a service principal and secret for Key Vault authentication, even though you are able to. Rotating the bootstrap secret automatically to gain access to Key Vault is a challenging task.

Encryption of data in transit

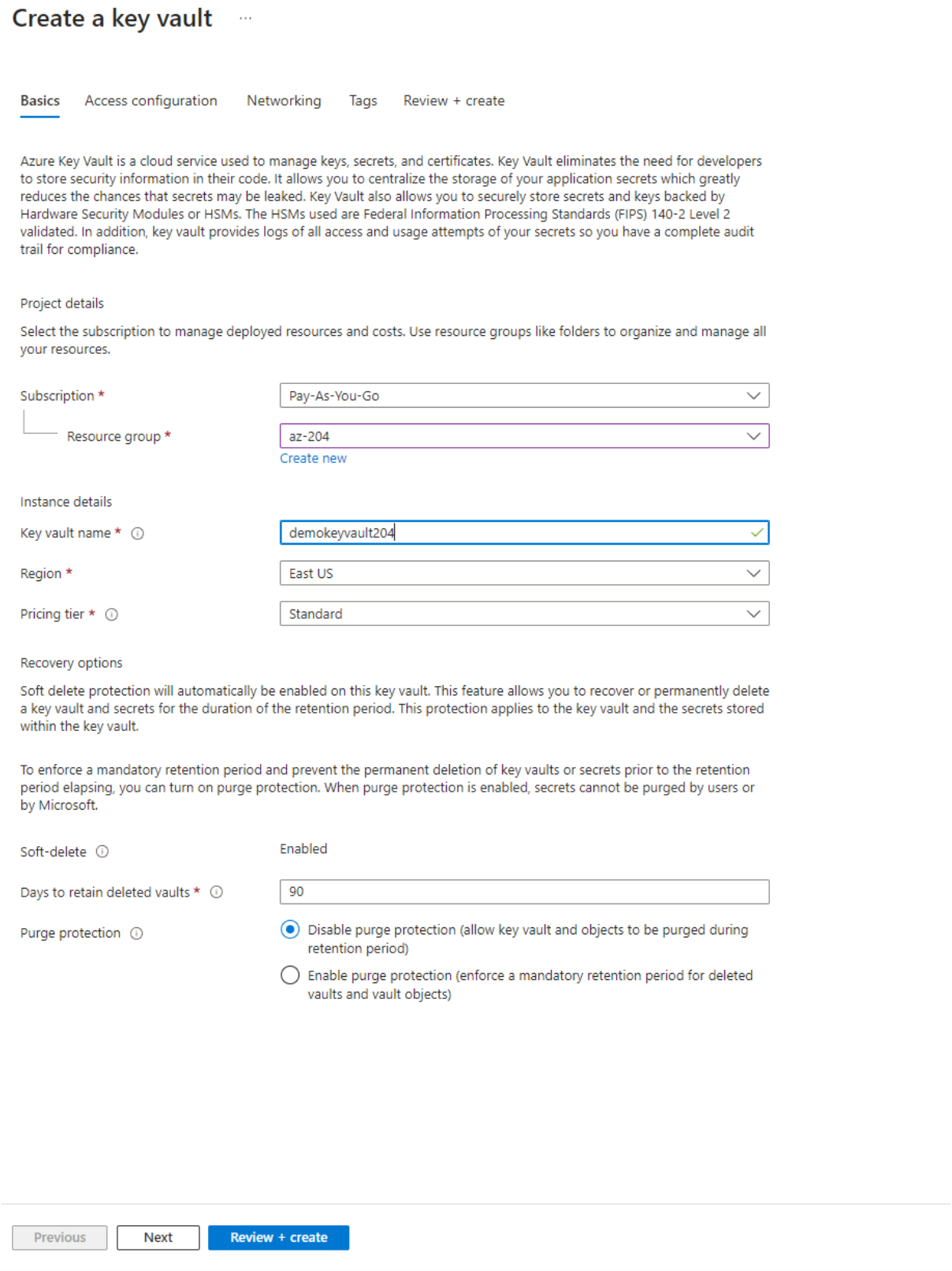
When data is transferred between Azure Key Vault and clients, it is protected by the Transport Layer Security (TLS) protocol, which is enforced by Azure Key Vault. Azure Key Vault and clients negotiate a TLS connection. TLS offers robust authentication, easy deployment and use, interoperability, flexible algorithms, message privacy and integrity (allowing detection of message tampering, interception, and forging), and strong authentication.

Using distinct keys, Perfect Forward Secrecy (PFS) secures connections between client systems of customers and Microsoft cloud services. RSA-based 2,048-bit encryption key lengths are also used for connections. It is more difficult for someone to intercept and access data that is in transit thanks to this combination.

Securing App configuration using Azure key vault

1. In the Azure portal, select the Create a resource option located in the upper-left corner:
2. Type "Key Vault" into the search bar, then choose it from the drop-down menu.
3. Choose Key vaults from the results list on the left.
4. Go to Key Vaults and choose Add.
5. Enter the following details in the "Create key vault" section on the right:

   Click on Subscription to select a desired subscription.
   Choose Create new and enter the name of the resource group, or enter the name of an existing resource group under Resource Group.
   A distinct name must be included in the Key Vault name.
   Select a location from the drop-down list labelled Region.



6. Maintain the default settings for the other Create key vault choices.
7. Click Review+create

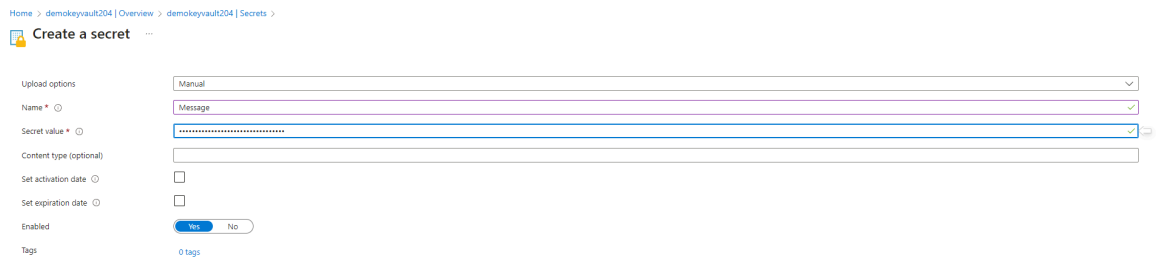
8. The info you supplied will be verified by the system and shown. Select "Create."

   Right now, the only account that is allowed to access this new vault is your Azure account.

Add a secret to key Vault

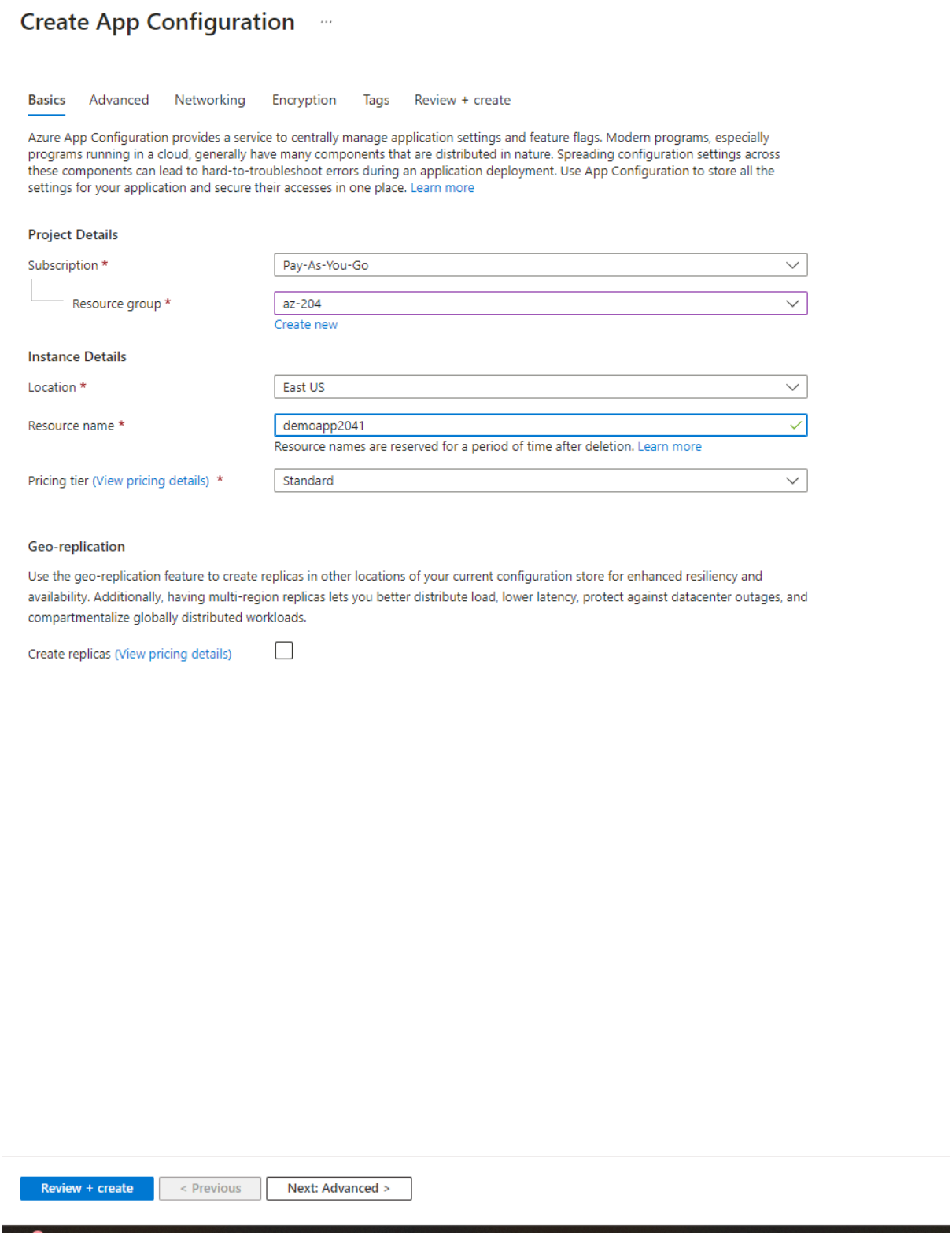
All it takes is a couple more steps to add a secret to the vault. Add a message that you may use to test Key Vault retrieval in this scenario. You store the value "Az- 204 Key Vault testing - Hello" in the message, which goes by the name Message.

1. Choose Secrets from the Key Vault properties pages.
2. Click on Generate or Import.
3. The following settings should be entered in the Create a secret pane:
   - Upload choices: Put in Manual.
   - Enter your message here.
   - Value: Az- 204 Key Vault testing - Hello.
4. Maintain the default settings for the other Create a hidden property.
5. Choose "Create."



Add a Key Vault reference to App Configuration

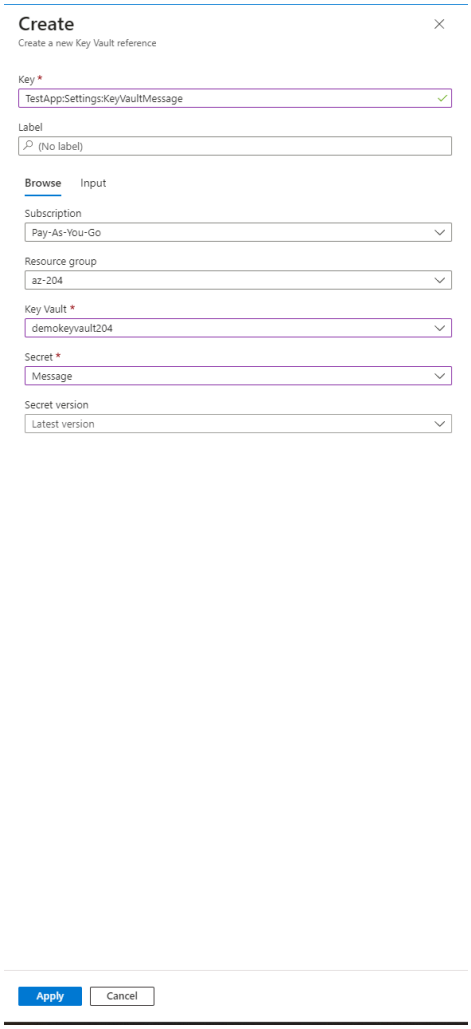
1. Open the Azure portal and log in. search for App Configuration and click on create. Select the appropriate name, leave default setting as it is and click on Review+cerate.



2. Go to Configuration Explorer and select.

3. Once you've chosen + Create > Key value, enter the following values:

   - **Key**: Click on Test:Settings:KeyVaultMessage.
   - **Label**: Enter a blank value here.
   - **Subscription, Resource Group, and Key vault**: Input the values that match the ones in the key vault you made in the preceding stage.
   - **Secret**: Click on the secret you made in the previous phase, called Message.



Adding reference to secret in code

1. Run the following command to create an ASP.NET Core web app in a new DemoAppConfig folder:

```
C:\Users\wildc>dotnet new webapp --output DemoAppConfig --framework net6.0
The template "ASP.NET Core Web App" was created successfully.
This template contains technologies from parties other than Microsoft, see https://aka.ms/aspnetcore/6.0-third-party-not
ices for details.

Processing post-creation actions...
Restoring C:\Users\wildc\DemoAppConfig\DemoAppConfig.csproj:
  Determining projects to restore...
  Restored C:\Users\wildc\DemoAppConfig\DemoAppConfig.csproj (in 71 ms).
Restore succeeded.
```

2. In order to add a Microsoft.Azure.AppConfiguration.AspNetCore NuGet package reference, navigate to the project's TestAppConfig directory and execute the following command:

```
C:\Users\wildc>cd DemoAppconfig

C:\Users\wildc\DemoAppConfig>dotnet add package Microsoft.Azure.AppConfiguration.AspNetCore
  Determining projects to restore...
  Writing C:\Users\wildc\AppData\Local\Temp\tmp1CFB.tmp
info : X.509 certificate chain validation will use the default trust store selected by .NET for code signing.
info : X.509 certificate chain validation will use the default trust store selected by .NET for timestamping.
info : Adding PackageReference for package 'Microsoft.Azure.AppConfiguration.AspNetCore' into project 'C:\Users\wildc\DemoAppConfig\DemoAppConfig.csproj'.
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/microsoft.azure.appconfiguration.aspnetcore/index.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/microsoft.azure.appconfiguration.aspnetcore/index.json 1086ms
info : Restoring packages for C:\Users\wildc\DemoAppConfig\DemoAppConfig.csproj...
info :   GET https://api.nuget.org/v3-flatcontainer/microsoft.azure.appconfiguration.aspnetcore/index.json
info :   OK https://api.nuget.org/v3-flatcontainer/microsoft.azure.appconfiguration.aspnetcore/index.json 255ms
info :   GET https://api.nuget.org/v3-flatcontainer/microsoft.azure.appconfiguration.aspnetcore/6.1.1/microsoft.azure.appconfiguration.aspnetcore.6.1.1.nupkg
info :   OK https://api.nuget.org/v3-flatcontainer/microsoft.azure.appconfiguration.aspnetcore/6.1.1/microsoft.azure.appconfiguration.aspnetcore.6.1.1.nupkg 13ms
info :   GET https://api.nuget.org/v3-flatcontainer/microsoft.extensions.configuration/index.json
info :   OK https://api.nuget.org/v3-flatcontainer/microsoft.extensions.configuration/index.json 248ms
info :   GET https://api.nuget.org/v3-flatcontainer/microsoft.extensions.configuration/3.1.18/microsoft.extensions.configuration.3.1.18.nupkg
info :   OK https://api.nuget.org/v3-flatcontainer/microsoft.extensions.configuration/3.1.18/microsoft.extensions.configuration.3.1.18.nupkg 7ms
info :   GET https://api.nuget.org/v3-flatcontainer/microsoft.extensions.configuration.abstractions/index.json
info :   OK https://api.nuget.org/v3-flatcontainer/microsoft.extensions.configuration.abstractions/index.json 251ms
info :   GET https://api.nuget.org/v3-flatcontainer/microsoft.extensions.configuration.abstractions/3.1.18/microsoft.extensions.configuration.abstractions.3.1.18.nupkg
info :   OK https://api.nuget.org/v3-flatcontainer/microsoft.extensions.configuration.abstractions/3.1.18/microsoft.extensions.configuration.abstractions.3.1.18.nupkg 7ms
info : Installed Microsoft.Extensions.Configuration 3.1.18 from https://api.nuget.org/v3/index.json with content hash G2W2Q0lMv76ZswL6XWJy8PbAu1uvbIKGmKkZEhy7irWfqQWFT8PKQ68cEpotg6e2rc15E5/LQDgvCpXeWFtA==.
info : Installed Microsoft.Extensions.Configuration.Abstractions 3.1.18 from https://api.nuget.org/v3/index.json with content hash ID1/qQDGowofEfOoNcEkerRDy3x5FMguDxDEslqDVROaI/qLa/oWCqUljIDWMuM1GUDNhucSrTMZ8/k9VyOHAg==.
info : Installed Microsoft.Azure.AppConfiguration.AspNetCore 6.1.1 from https://api.nuget.org/v3/index.json with content hash 1nMU7y90Dyj366ASoOIaUCZgXqoNmxwTcTTqmQbNPFd6GgHRN2MhiOR2WU4uk84pi+5fMVPCa+nMm4WuNFIGmQ==.
info : Package 'Microsoft.Azure.AppConfiguration.AspNetCore' is compatible with all the specified frameworks in project 'C:\Users\wildc\DemoAppConfig\DemoAppConfig.csproj'.
info : PackageReference for package 'Microsoft.Azure.AppConfiguration.AspNetCore' version '6.1.1' added to file 'C:\Users\wildc\DemoAppConfig\DemoAppConfig.csproj'.
info : Writing assets file to disk. Path: C:\Users\wildc\DemoAppConfig\obj\project.assets.json
log  : Restored C:\Users\wildc\DemoAppConfig\DemoAppConfig.csproj (in 1.13 sec).
```

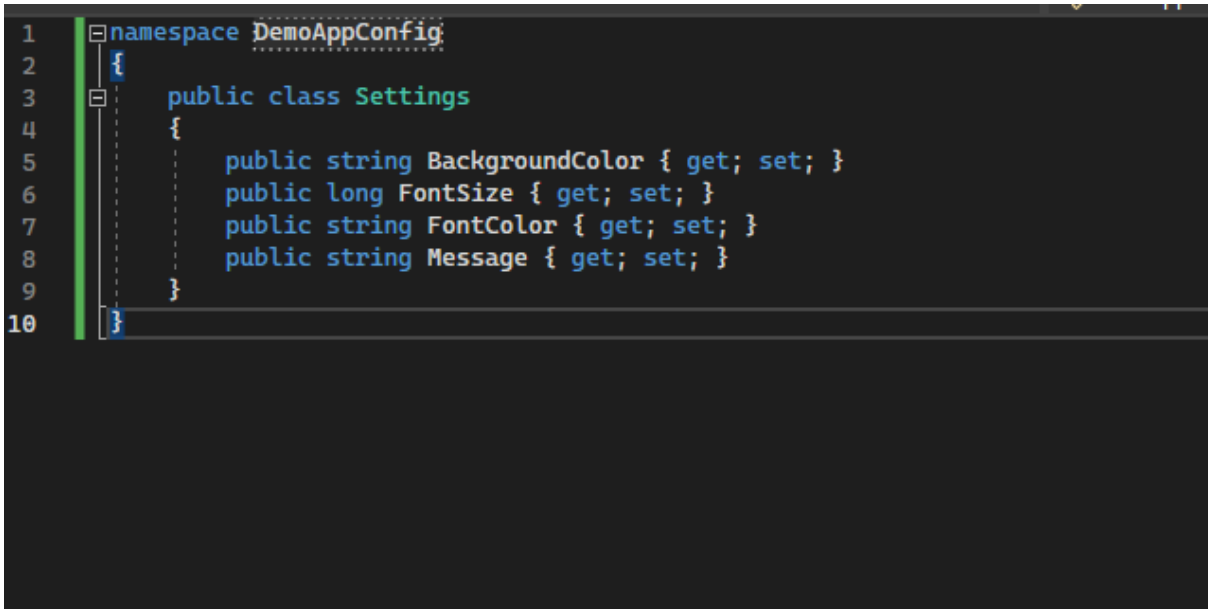
3. Execute the subsequent command. The connection string for your App Configuration store is stored in a secret called ConnectionStrings:AppConfig, which is stored by the command using Secret Manager. The connection string from your App Configuration store should be used in place of the placeholder <your_connection_string>. The connection string is located in the Azure portal's App Configuration store under Access Keys.

```
C:\Users\wildc\DemoAppConfig>dotnet user-secrets init
Set UserSecretsId to '78060706-3192-449b-b425-6e6a02f8ba5c' for MSBuild project 'C:\Users\wildc\DemoAppConfig\DemoAppConfig.csproj'.

C:\Users\wildc\DemoAppConfig>dotnet user-secrets set ConnectionStrings:AppConfig "Endpoint=https://demoapp204.azconfig.io;Id=vKlK;Secret=m15duBVKqAeh32+TMgWlGl9I5Xg2oGBj18FC4FT8qAc="
Successfully saved ConnectionStrings:AppConfig = Endpoint=https://demoapp204.azconfig.io;Id=vKlK;Secret=m15duBVKqAeh32+TMgWlGl9I5Xg2oGBj18FC4FT8qAc= to the secret store.
```

4. To help avoid unintentionally sharing secrets within source code, Secret Manager saves the secret outside of your project tree. It is exclusively utilized for local web app testing. Use the application settings, connection strings, or environment variables to store the connection string when the app is deployed to Azure, such as App Service. Alternatively,

you can connect to App Configuration using managed identities or any other Microsoft Entra identity to completely skip connection strings.

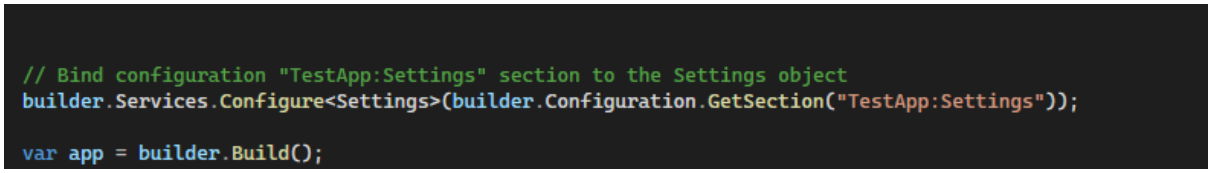
Add a Settings.cs file at the root of your project directory. It defines a strongly typed Settings class for the configuration you're going to use. Replace the namespace with the name of your project.



```
namespace DemoAppConfig
{
    public class Settings
    {
        public string BackgroundColor { get; set; }
        public long FontSize { get; set; }
        public string FontColor { get; set; }
        public string Message { get; set; }
    }
}
```

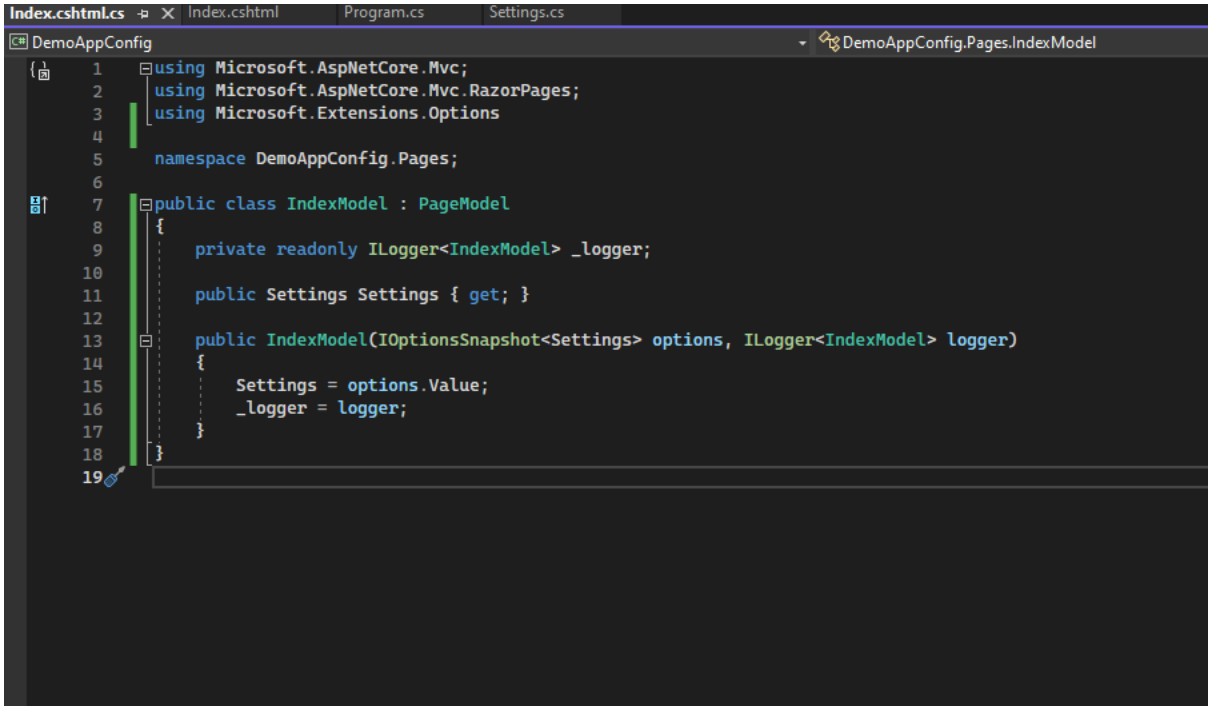
5. Bind the TestApp:Settings section in configuration to the Settings object.

Update Program.cs with the following code and add the DemoAppConfig namespace at the beginning of the file.

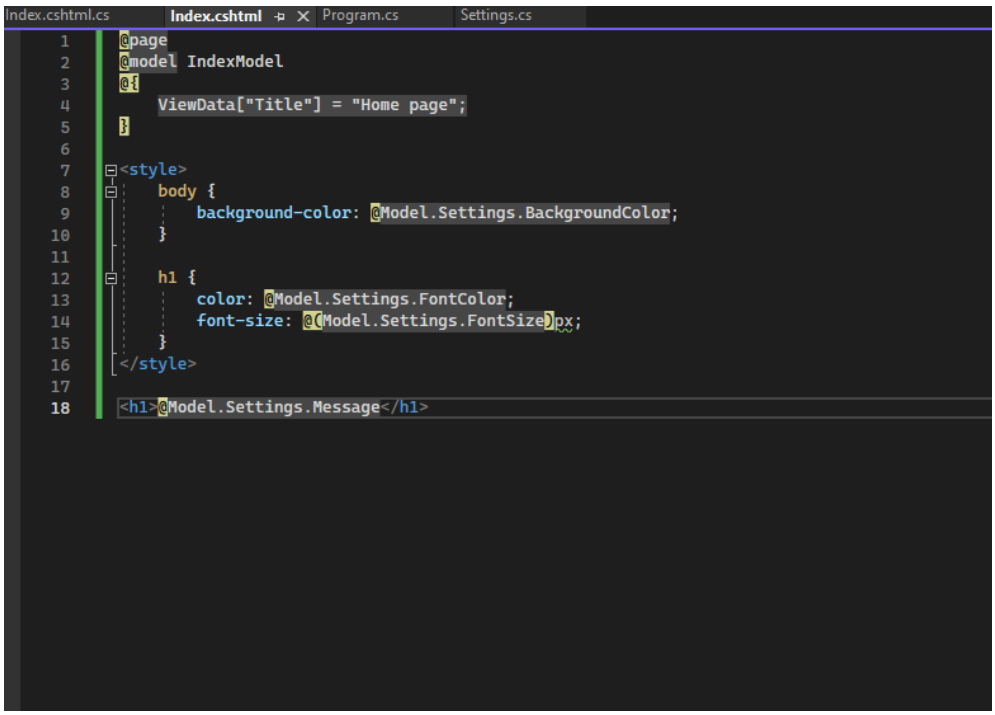


```
// Bind configuration "TestApp:Settings" section to the Settings object
builder.Services.Configure<Settings>(builder.Configuration.GetSection("TestApp:Settings"));

var app = builder.Build();
```

6. Open Index.cshtml.cs in the Pages directory, and update the IndexModel class with the following code. Add the using Microsoft.Extensions.Options namespace at the beginning of the file, if it's not already there.

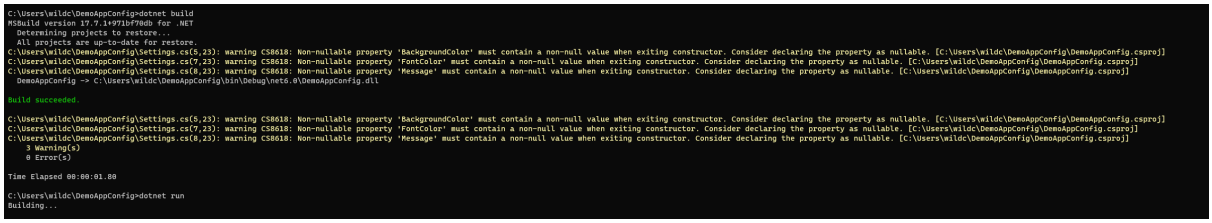


7. Open Index.cshtml in the Pages directory, and update the content with the following code.

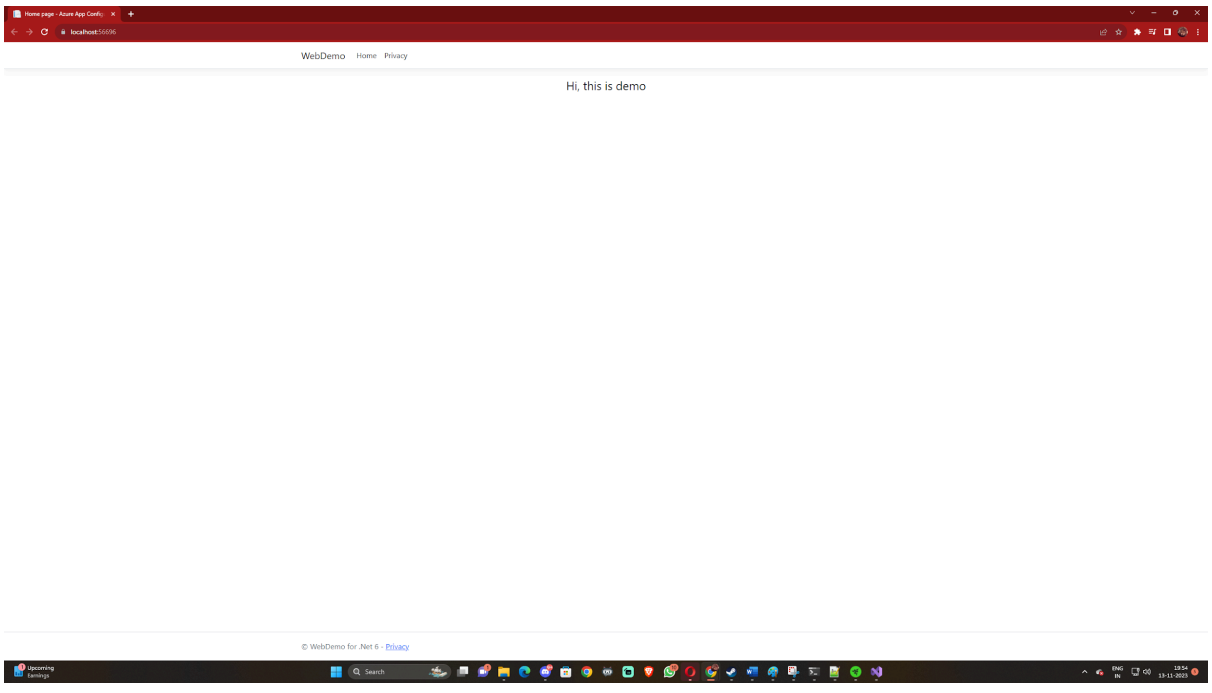


8. To build the app using the .NET Core CLI, navigate to the root directory of your project. Run the following command in the command shell:

To launch the web application locally when the build has successfully finished, type the following command:



9. You will be able to see below output.



Azure Cache for Redis

One popular method for enhancing a system's scalability and performance is caching. In order to achieve this, frequently accessed data is temporarily copied to quick storage that is near to the program. Caching can greatly speed up client application response times by supplying data faster if the fast data storage is situated closer to the application than the original source.

Based on the Redis program, Azure Cache for Redis offers an in-memory data store. Redis enhances an application's scalability and performance when it significantly utilizes backend data stores. Because frequently used data may be swiftly written to and read from the server memory, it can handle high volumes of application requests. Redis provides modern applications with a vital low-latency and high-throughput data storage solution.

Redis Labs' (Redis Enterprise) commercial offering and the open-source Redis (OSS Redis) are both available as managed services through Azure Cache for Redis. Complete Redis API compatibility and safe, dedicated Redis server instances are offered. Microsoft runs the service, which can be accessed by any application running on Azure or on another platform.

Azure Cache for Redis enhances the performance of applications by catering to standard application architectural principles. Among the most typical are the subsequent patterns:

1. Data Cache - Frequently, databases are too big to load straight into a cache. The cache-aside pattern is widely used to put data into the cache only when it's needed. The cache, which is subsequently shared with additional clients, can be updated by the system when modifications are made to the data.

2. Content Cache - A lot of websites are created using templates that contain static elements like banners, footers, and headers. These constants shouldn't be altered frequently. When using an in-memory cache instead of backend datastores, static content can be accessed more quickly.

3. Session store - Shopping carts and other user history information that a web application might link to user cookies are frequent uses for this approach. When a cookie is passed through and validated with each request, storing too much in it can negatively impact performance. A common method queries the data in a database using the cookie as a key. It is quicker to associate data with a user using an in-memory cache, such as Azure Cache for Redis, than it is to interface with a whole relational database

4. Job and message queuing - When the activities involved in a request take a while to complete, applications frequently add tasks to a queue. Extended operations are queued for sequential processing, frequently by a different server. Task queue is the term for this kind of job deferral.

5. Distributed transactions -Applications can occasionally need to issue many instructions to a backend datastore in order to complete an action in one go. Either every order must be successful, or everything must be rolled back to the original state. It is possible to run many commands as a single transaction with Azure Cache for Redis.

Cache for Redis available service tiers

Tier	Description
Basic	a single virtual machine hosting an OSS Redis cache. This tier is best for noncritical and development/test workloads and has no service-level agreements (SLAs).
Standard	an OSS Redis cache that is replicated over two virtual machines.
Premium	speedy Redis caches for OSS. More features, reduced latency, increased throughput, and improved availability are all offered by this tier. VMs running Premium caches are more powerful than those running Basic or Standard caches.
Enterprise	Redis Enterprise software from Redis Labs powers high-performance caches. Redis modules like RediSearch, RedisBloom, and Redis Timeseries are supported on this tier. It provides even greater availability than the Premium tier as well.
Enterprise Flash	Redis Enterprise software from Redis Labs powers affordable big caches. This layer expands Redis data storage into virtual machine (VM) non-volatile memory, which is less expensive than DRAM. The total cost of memory per GB is decreased.

Expiration Policies:

The following are various Azure Redis Cache expiration policies:

- volatile-lru: The default setting that uses a "expire set" to remove the keys that haven't been utilized in a while.
- allkeys-lru: use the least recently used (LRU) method to remove any key.
- unstable-haphazard: removes a key at random from those that have a "expire set"
- Ten-minute break: There is a 10-minute timeout for inactive connections in Azure Cache for Redis.

The portal's settings allow you to modify the expiration policy.

A key assigned a time-out that has elapsed is immediately removed by Azure Cache for Redis.

The *STORE commands SET, SETEX, GETSET, and others can be used to set time-out values.

The key will not be given back to anyone who asks for it after the expiration time has passed. After it expires, though, it might not be physically erased from memory for a while.

Implementing Azure Cache for Redis

Using the Azure portal, Azure CLI, or Azure PowerShell, you can establish a Redis cache. To correctly set the cache for your needs, you must choose a number of parameters.

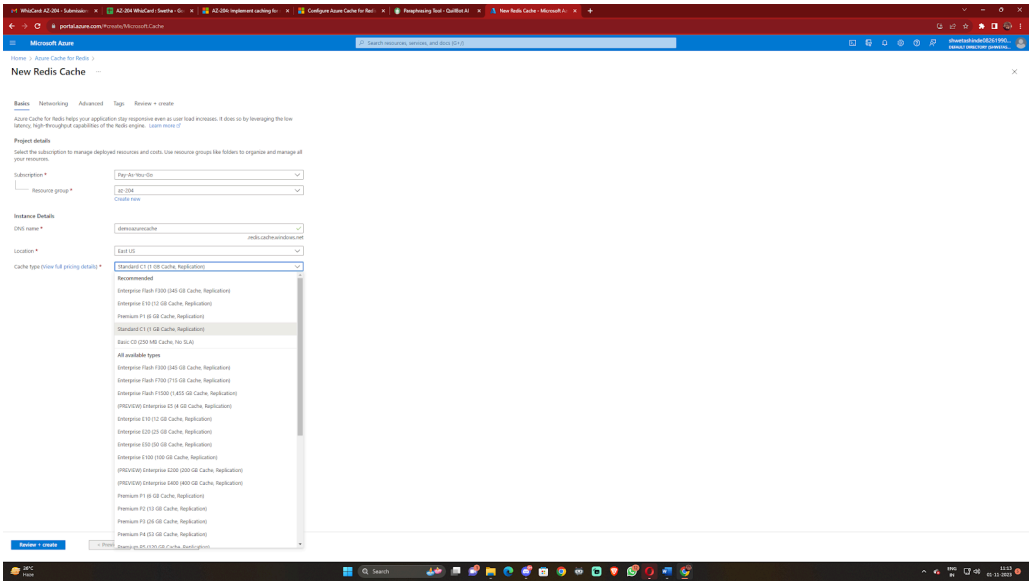
1. Name - The name of the Redis cache must be globally unique. Since the name is used to create a public-facing URL for connecting to and interacting with the service, it must be unique within Azure.

   The name must consist of letters, numbers, and the '-' character and be between one and sixty-three characters long. The '-' character is not allowed to begin or finish the cache name, and consecutive '-' characters are invalid.

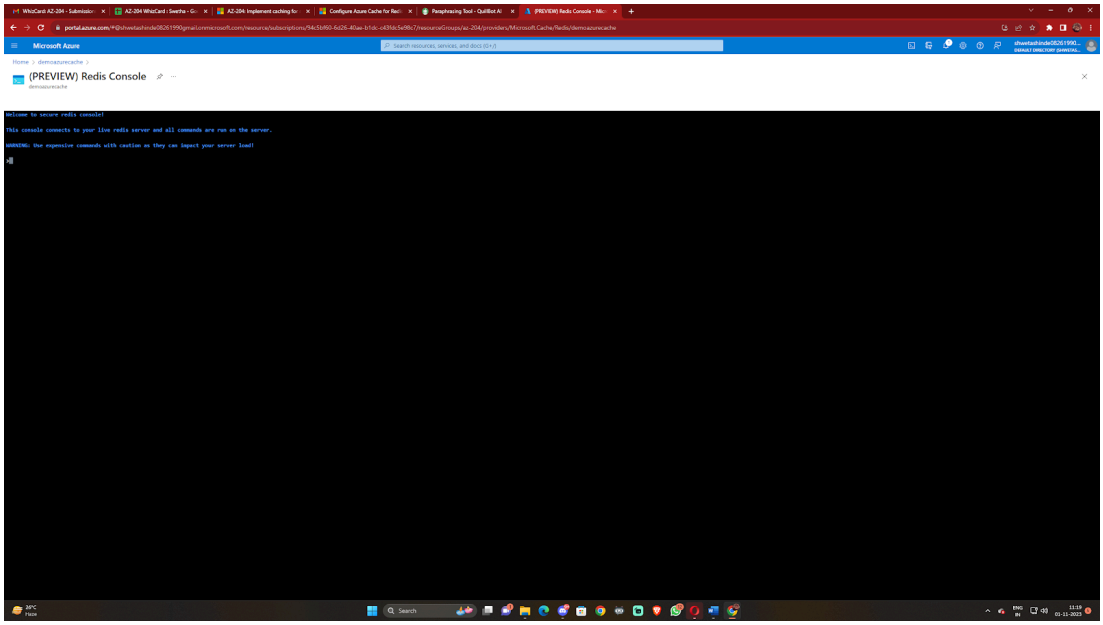
2. Location - Your application and cache instance should always be located in the same area. Reliability can be greatly lowered and latency can rise dramatically when connecting to a cache located elsewhere. Choose a location close to the application that is using the data if you are connecting to the cache from outside of Azure.

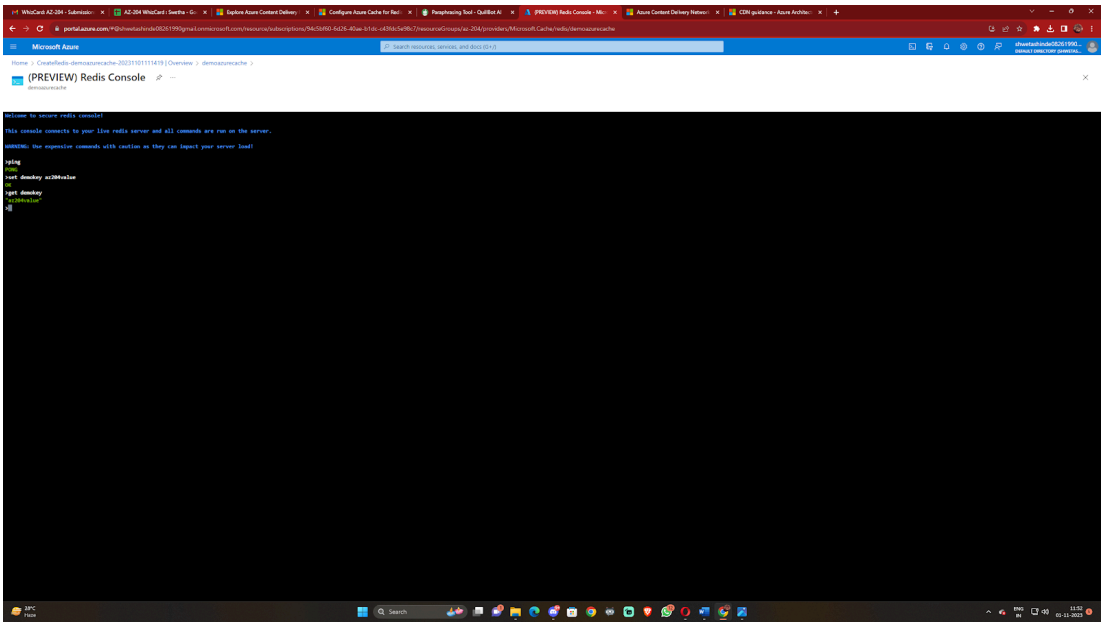
3. Cache type - The cache's size, functionality, and performance are all determined by the tier. Visit Azure Cache for Redis pricing for further details.

4. Clustering Support - You can use clustering with the Premium, Enterprise, and Enterprise Flash tiers to divide your dataset among several nodes automatically. You can designate up to ten shards in order to apply clustering. The amount spent is equal to the initial node's cost times the number of shards.



To interact with Redis instance you can use console from azure portal.





Giving values an expiration time

Because it enables us to retain frequently used values in memory, hurting is significant. But we also need a mechanism to let values expire when they become stale. This is accomplished in Redis by giving a key a time to live (TTL).

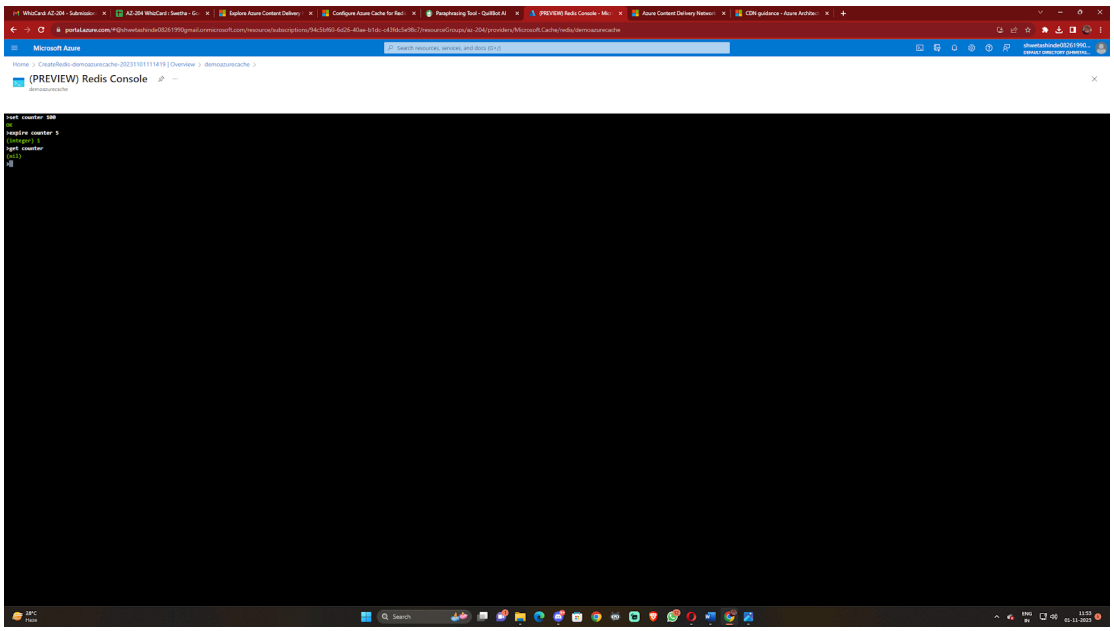
The key is immediately erased after the TTL expires, just like if the DEL command had been sent. Notes about TTL expirations are provided here.

Expirations can be precisely set in milliseconds or seconds.

There is always a 1 millisecond resolution for expiry times.

Redis remembers the date when a key expires, thus when your server is stopped, time is virtually passing while information about expires is replicated and preserved on disk.

Below is the example of how to set expiration:



Azure Content Delivery Network (Azure CDN)

A dispersed network of servers called a content delivery network (CDN) is capable of effectively delivering web material to users. To reduce latency, CDNs keep cached material on edge servers near end users.

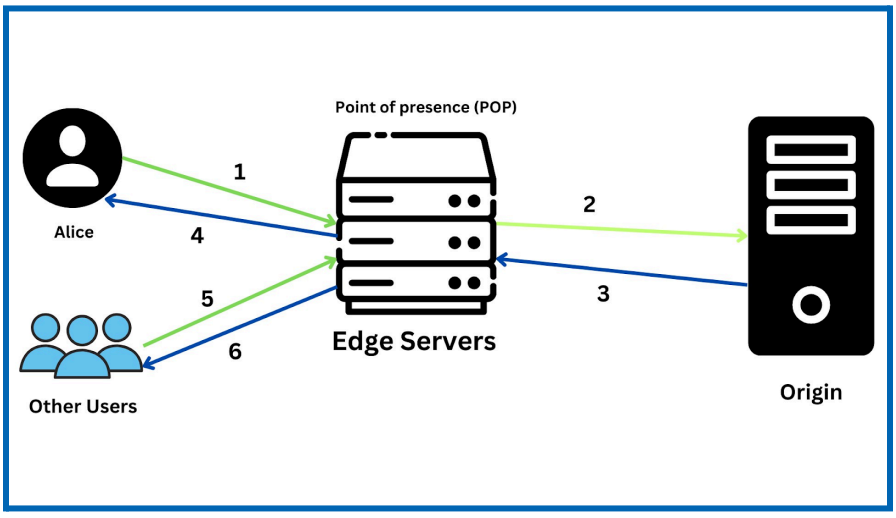
CDNs are commonly employed for the delivery of static content, including HTML pages, style sheets, documents, graphics, and client-side scripts. Regardless of a user's location in respect to the data centre hosting the application, the main benefits of utilizing a CDN are reduced latency and faster content delivery. Because the web application does not have to process requests for the content that is hosted in the CDN, CDNs can also aid in reducing strain on the application.

With the help of Azure Content Delivery Network (CDN), developers can quickly and efficiently provide high-bandwidth content to users worldwide by caching their material at carefully chosen physical nodes all around the globe. Moreover, dynamic content—which cannot be cached—can be expedited by Azure CDN through the use of CDN POPs and different network optimizations. Using route optimization to avoid Border Gateway Protocol (BGP), for instance.

When delivering web site assets, Azure CDN offers the following advantages:

- Enhanced user experience and better performance for end users, particularly when utilizing applications that need several round trips to load content.
- Large scaling is necessary to better manage sudden, high loads, like those that occur during a product introduction.
- In order to reduce the amount of traffic transmitted to the origin server, user requests are distributed and content is served straight from edge servers.

Azure Content Delivery Network working



1. When user1, uses a URL with a unique domain name, such as <endpoint name>, she requests a file, also known as an asset.cloudedge.net. This name may be a custom domain or an endpoint hostname. The best-performing POP location—typically the POP nearest to the user—is the one to which the DNS directs the request.
2. The POP requests the file from the origin server if none of the edge servers in its cache has it. Any publicly available web server, an Azure Web App, an Azure Cloud Service, or an Azure Storage account can act as the origin server.
3. The file is returned to an edge server in the POP by the origin server.
4. The file is cached by an edge server in the POP and then sent back to Alice, the original requester. Until the time-to-live (TTL) indicated by its HTTP headers expires, the file is kept in cache on the edge server in the POP hierarchy. The default TTL in the event that the origin server failed to specify one is seven days.
5. Then, using the same URL that Alice used, other users can request the same file and be sent to the same POP.
6. The POP edge server returns the file straight from the cache if the file's TTL hasn't expired. A quicker, more responsive user experience is the outcome of this approach.

Features: Among the main things that Azure CDN provides are:

- Dynamic site speed up
- CDN cache policies
- support for HTTPS custom domains
- logs from Azure diagnostics
- File condensing
- Geo-restrictions

Use .NET to communicate with Azure Content Delivery Networks

The Azure CDN Library for.NET can be used to automate CDN profile and endpoint generation and management. Using the Visual Studio Package Manager console or the.NET CLI, install Microsoft.Azure.Management.Cdn.Fluent directly. You will find code examples explaining common actions in this unit.

1. Create a CDN client

The **CdnManagementClient** class is used to create a client, as demonstrated in the sample below.

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

2. List CDN profiles and endpoints

In order to prevent us from trying to create duplicates, the procedure below first lists every profile and endpoint in our resource group. If it discovers a match for the profile and endpoint names given in our constants, it notes that information for later use.

```
private static void ListProfilesAndEndpoints(CdnManagementClient cdn)
{
    // List all the CDN profiles in this resource group
    var profileList = cdn.Profiles.ListByResourceGroup(resourceGroupName);
    foreach (Profile p in profileList)
    {
        Console.WriteLine("CDN profile {0}", p.Name);
        if (p.Name.Equals(profileName, StringComparison.OrdinalIgnoreCase))
        {
            // Hey, that's the name of the CDN profile we want to create!
            profileAlreadyExists = true;
        }

        //List all the CDN endpoints on this CDN profile
        Console.WriteLine("Endpoints:");
        var endpointList = cdn.Endpoints.ListByProfile(p.Name, resourceGroupName);
        foreach (Endpoint e in endpointList)
        {
            Console.WriteLine("-{0} ({1})", e.Name, e.HostName);
            if (e.Name.Equals(endpointName, StringComparison.OrdinalIgnoreCase))
            {
                // The unique endpoint name already exists.
                endpointAlreadyExists = true;
            }
        }
        Console.WriteLine();
    }
}
```

3. Create CDN profiles and endpoints

An Azure CDN profile can be created as seen in the example below.

```
private static void CreateCdnProfile(CdnManagementClient cdn)
{
    if (profileAlreadyExists)
    {
        //Check to see if the profile already exists
    }
    else
    {
        //Create the new profile
        ProfileCreateParameters profileParms =
            new ProfileCreateParameters() { Location = resourceLocation, Sku = new Sku(SkuName.StandardVerizon) };
        cdn.Profiles.Create(profileName, profileParms, resourceGroupName);
    }
}
```

Once the profile is created, we will create an endpoint

```
private static void CreateCdnEndpoint(CdnManagementClient cdn)
{
    if (endpointAlreadyExists)
    {
        //Check to see if the endpoint already exists
    }
    else
    {
        //Create the new endpoint
        EndpointCreateParameters endpointParms =
            new EndpointCreateParameters()
            {
                Origins = new List<DeepCreatedOrigin>() { new DeepCreatedOrigin("drive", "www.drive.com") },
                IsHttpAllowed = true,
                IsHttpsAllowed = true,
                Location = resourceLocation
            };
        cdn.Endpoints.Create(endpointName, endpointParms, profileName, resourceGroupName);
    }
}
```

4. Purge an endpoint

Eradicating the material from our endpoint is a routine task that we may wish to carry out.

```
private static void PromptPurgeCdnEndpoint(CdnManagementClient cdn)
{
    if (PromptUser(String.Format("Purge CDN endpoint {0}?", endpointName)))
    {
        Console.WriteLine("Purging endpoint. Please wait...");
        cdn.Endpoints.PurgeContent(resourceGroupName, profileName, endpointName, new List<string>() { "/*" });
        Console.WriteLine("Done.");
        Console.WriteLine();
    }
}
```

Azure Front door

The features of these three goods are offered by Azure Front Door Standard/Premium. By integrating intelligent threat protection with the Microsoft global edge network, it provides a more rapid, dependable, and secure contemporary cloud content delivery network. User queries to your hosted apps are managed by Azure Front Door, which is located in the edge locations. Users use the Microsoft global network to connect to your application. User requests are subsequently forwarded to the fastest and most accessible application backend via Azure Front Door.

There are the subsequent Azure Front door tiers available:

- The entry level is Azure Front Door (classic). Current Azure users frequently enhance these functionalities by utilizing Azure Web Application Firewall and Azure Content Delivery Network.
- Azure Front Door Standard is designed to ensure smooth and uninterrupted delivery of content.

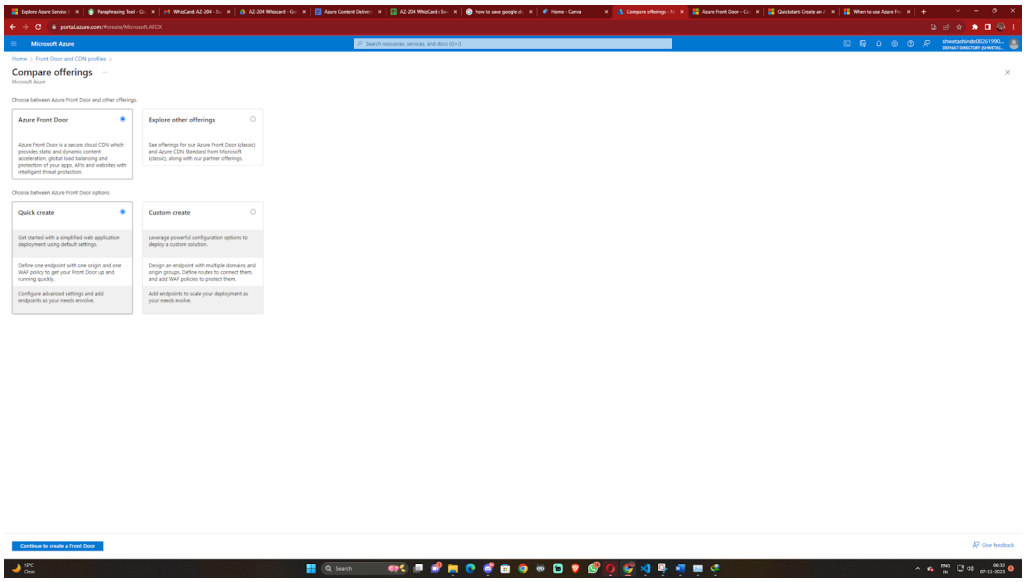
- Azure Front Door Premium is designed with enhanced security in mind.

Create a Azure Front door

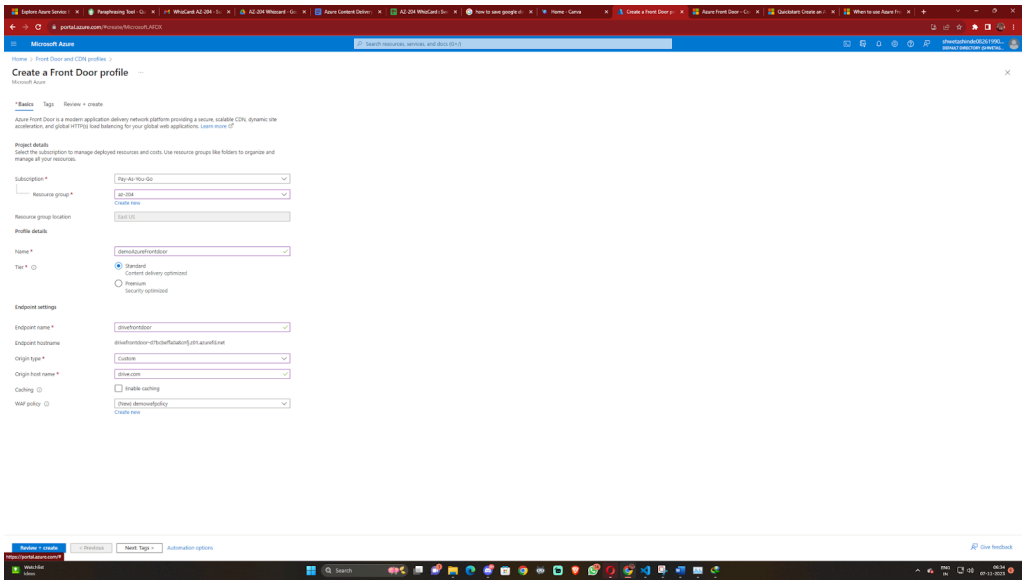
1.Open the Azure portal and log in.

2.Go to the Azure menu or home page, pick + Create a resource button to create a new resource for Front Door and CDN profiles. Next, type CDN profiles and Front Door into the search field, then click Create.

3.Tap Quick creation on the Compare offerings page. Then choose to proceed with creating a front door.



4. Enter the following data for the necessary settings on the Create a Front Door profile page.



- **Subscription** - select your subscription
- **Resource Group** - select already created resource group or else click on create new and create new resource group
- **Name** - give your profile name.In this example it is demoAzureFrontdoor
- **Tier** - Choose the Premium or Standard tier. Optimized content delivery is the standard tier. The Premium tier prioritizes security and expands upon the Standard tier. See Comparison of Tiers.(ref. microsoft learn)
- **Endpoint Name** - Enter globally unique name for your endpoint name
- **Origin type** - Type of originThe resource type for your origin should be chosen. In this case, we choose a custom.
- **Origin host name** - Enter the hostname for your origin
- **Private Link** - If you would like to have a private connection between your Azure Front Door and your origin, enable private link service. Supported services are limited to internal load balancers, Storage Blobs, and App services.
- **Caching** - To cache contents closer to your users utilizing the Microsoft network and edge POPs of Azure Front Door, select this check box.
- **WAF Policy** - To enable this functionality, click Create new or choose an existing WAF policy from the options.

Note - You have to choose an origin from the same subscription in which the Front Door was formed when creating an Azure Front Door profile.

5. Select Review+create and then select create to deploy your azure front door.

Azure API Management Instance

API management offers the essential features—developer interaction, business insights, analytics, security, and protection—to guarantee an API program's success. Every API has one or more actions, and any number of products can incorporate any number of APIs. Developers must subscribe to a product that includes an API in order to utilize it. After that, they can call the API's operation, abide by any applicable usage policies.

API Management Components

An API gateway, a management plane, and a developer portal comprise Azure API Management. By default, Azure hosts and fully manages these components. There are multiple tiers of API Management, with varying capacities and capabilities.

1. The endpoint that acts as the **API gateway** is:

    - Receives API requests and forwards them to the relevant backends
    - Confirms the credentials provided with queries, including the API keys.
    - Enforces rate caps and usage limitations
    - Converts inquiries and answers as outlined in policy declarations
    - Caches replies to reduce stress on backend services and enhance response latency.
    - Emits metrics, traces, and logs for reporting, monitoring, and troubleshooting.

2. The administrative interface where you configure your API program is called the **management plane.** Apply it to:

    - Set up and customize the API Management service parameters.
    - Import or define an API schema.
    - Integrate APIs into goods
    - Configure the APIs with policies like quotas and transformations.
    - Gain knowledge through analytics
    - Manage users

3. The **Developer Portal** is an entirely configurable, automatically created webpage that contains your API documentation. Through the gateway for developers, developers can:

    - Read the API documentation.
    - Use the interactive terminal to contact an API.
    - Register and subscribe to receive API keys.
    - View usage analytics independently
    - Obtain definitions for APIs.

- Manage API keys

Products

Developers are able to access APIs through products. One or more APIs are included in products under API Management, and each product is set up with a title, description, and usage guidelines. Items may be sealed or exposed. Open products can be used without a subscription, whereas protected products require one before they can be utilized.

Groups

Groups are used to control which products are visible to developers. The following immutable system groups are part of API Management:

Administrators: They are responsible for creating the operations, products, and APIs that developers utilize as well as managing instances of the API Management service. Administrators of Azure subscriptions are included in this group.

Developers: Users of your developer site with authentication who create apps with your APIs. Developers can create apps that invoke API operations by gaining access to the developer portal.

Guests: Unauthenticated users of the developer portal. Certain read-only privileges, such as the ability to browse APIs but not call them, may be assigned to them.

Administrators can leverage external groups in related Microsoft Entra tenancies or establish bespoke groups in addition to these system groups.

Policies

A policy is a group of commands that are applied one after the other to an API request or response. Many different rules are available, but popular statements include call rate restriction to limit the number of incoming calls from a developer and format conversion from XML to JSON.

Unless otherwise specified by the policy, policy expressions can be used as text values or attribute values in any API Management policy. Policy expressions serve as the foundation for some policies, including the Control flow and Set variable policies.

Depending on your needs, policies can be enforced at several scopes: global (across all APIs), product, individual API, or API operation.

API Gateways

Proxying API queries, implementing policies, and gathering telemetry are all handled by the service component known as the API Management gateway (sometimes referred to as the data plane or runtime).

In between clients and services is an API gateway. It routes client requests to services in reverse proxy fashion. Additionally, it may carry out a number of cross-cutting functions like rate limitation, SSL termination, and authentication. Clients must submit requests directly to back-end services if a gateway is not deployed. However, exposing services to customers directly may have the following issues:

- Complicated client code may be the outcome. The client has to manage failures robustly and monitor several endpoints.
- Between the client and the backend, coupling is created. The client must understand how each service is broken down. This makes it more difficult to retain clients and to restructure services.
- Calls to various services may be necessary for a single operation.
- Every service that interacts with the public must manage issues like SSL, client rate limitation, and authentication.
- A client-friendly protocol like HTTP or WebSocket must be exposed by services. This reduces the variety of communication protocols available.
- Publicly accessible services need to be protected from potential attacks by hardening them.

By severing the client from the service, a gateway aids in resolving these problems.

Types of Gateways

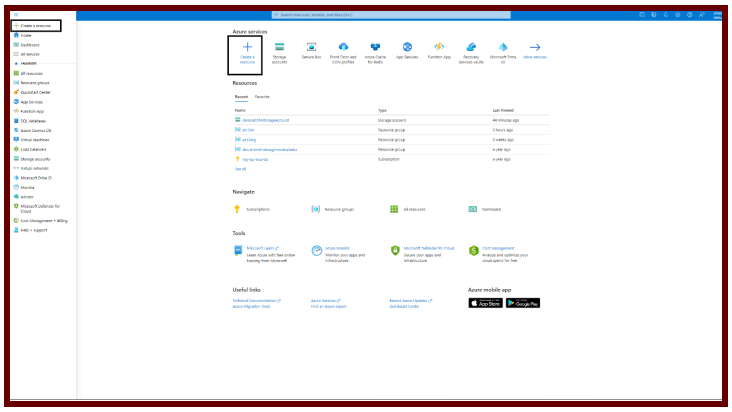
Both managed and self-hosted gateways are provided by API Management:

Managed: For each API Management instance across all service tiers in Azure, the managed gateway is the default gateway component. Wherever the backends hosting the APIs are located, all API traffic is routed through Azure using the managed gateway.

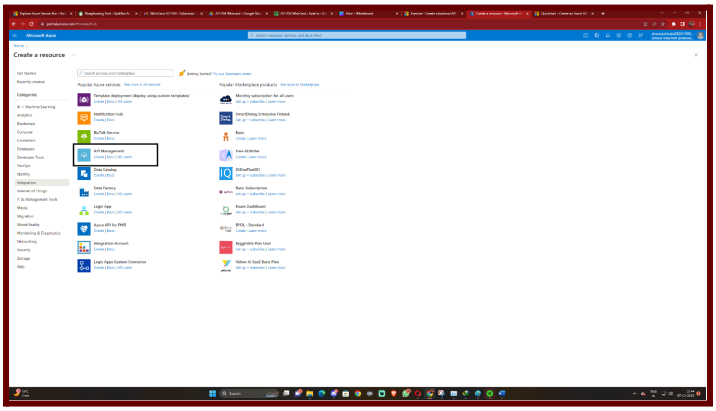
Self-hosted: A containerized variant of the standard managed gateway, the self-hosted gateway is an optional feature. When running the gateways off of Azure in the same settings as the API backends, it's helpful in hybrid and multi cloud scenarios. Customers with hybrid IT infrastructure may manage APIs hosted on-premises and across clouds using a single API Management service in Azure thanks to the self-hosted gateway.

Create API

1. Sign into Azure portal
2. Choose the option to Create a resource from the Azure portal menu. On the Azure Home page, you may also choose Create a resource.

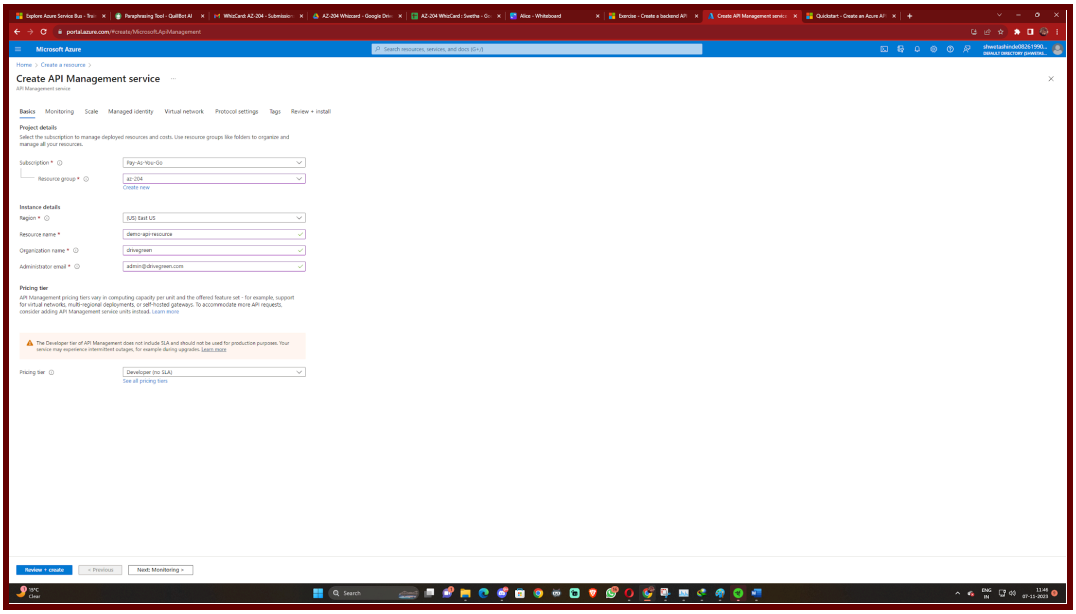


3. Choose Integration > API Management from the Create a resource page.



4. Enter the options in the Create API Management page.

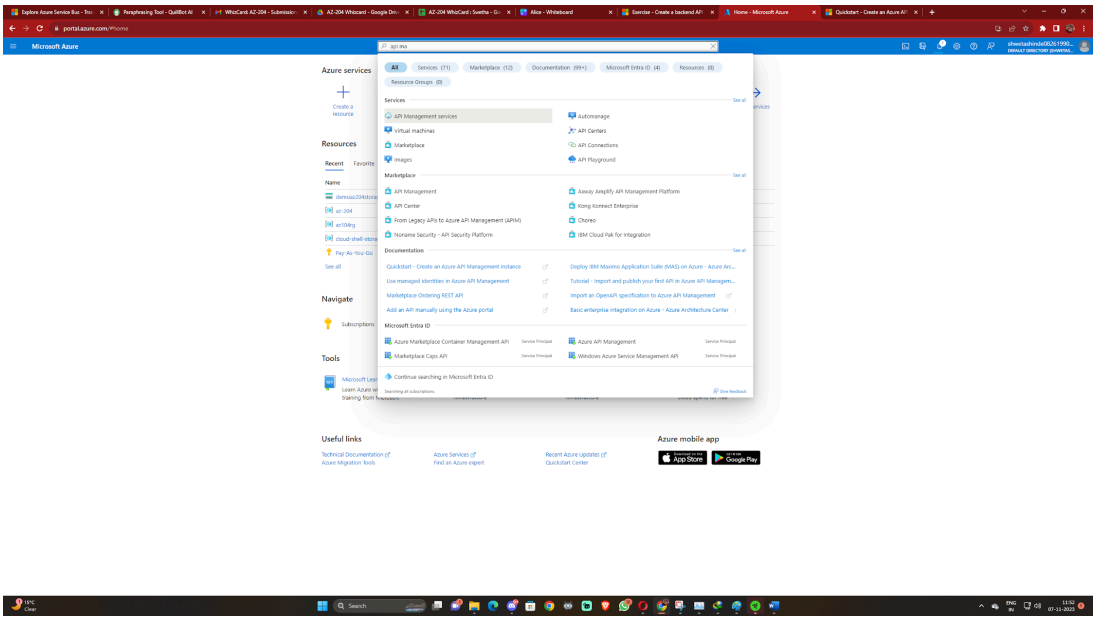
Settings	Description
Resource Name	a distinctive moniker for your service of API management. Later on, the name cannot be altered. Both the service and the associated Azure resource are referred to by the service name. <name>.azure-api.net is the default domain name that is generated using the service name.
Organization Name	The name that your company goes by. This name appears in a number of places, such as the developer portal's title and the sender of email notifications.
Administrator Email	the email address that API Management will use to send all notifications.
Pricing tier	To test the service, choose the Developer tier. You cannot use this tier in production.



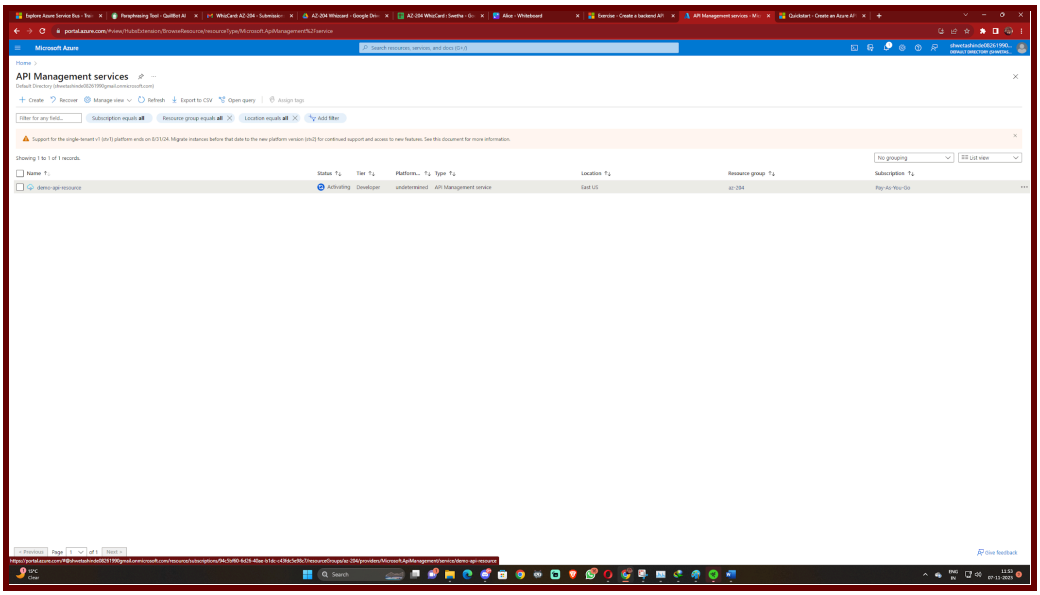
5. Select Review + create

Go to your API Management Instance

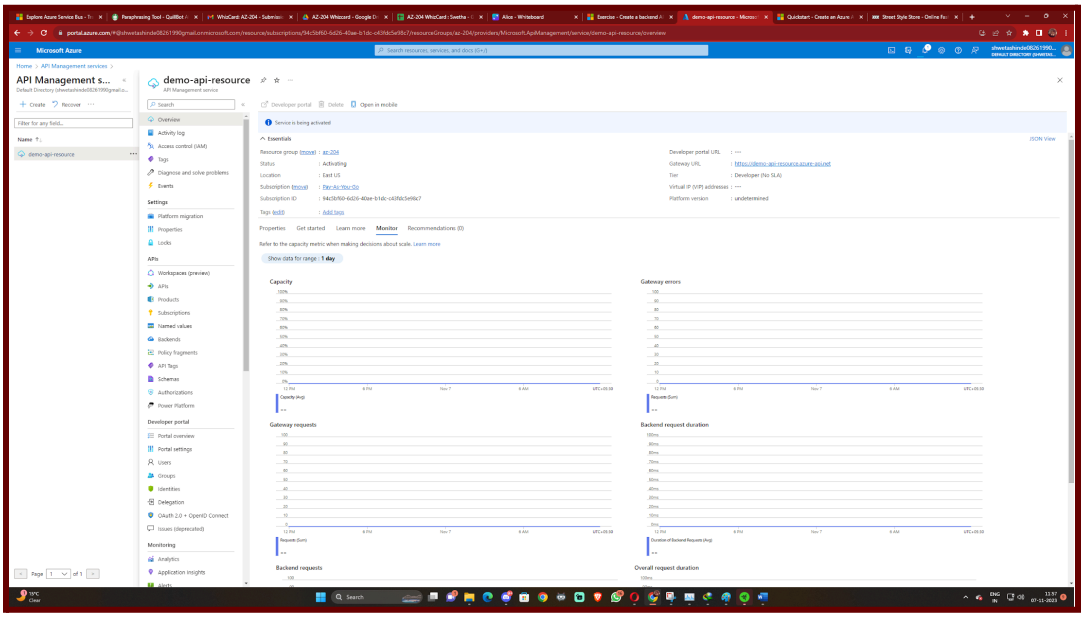
1. Search for and choose API Management services from the Azure portal.



2. Choose your API Management instance from the list of services on the page.

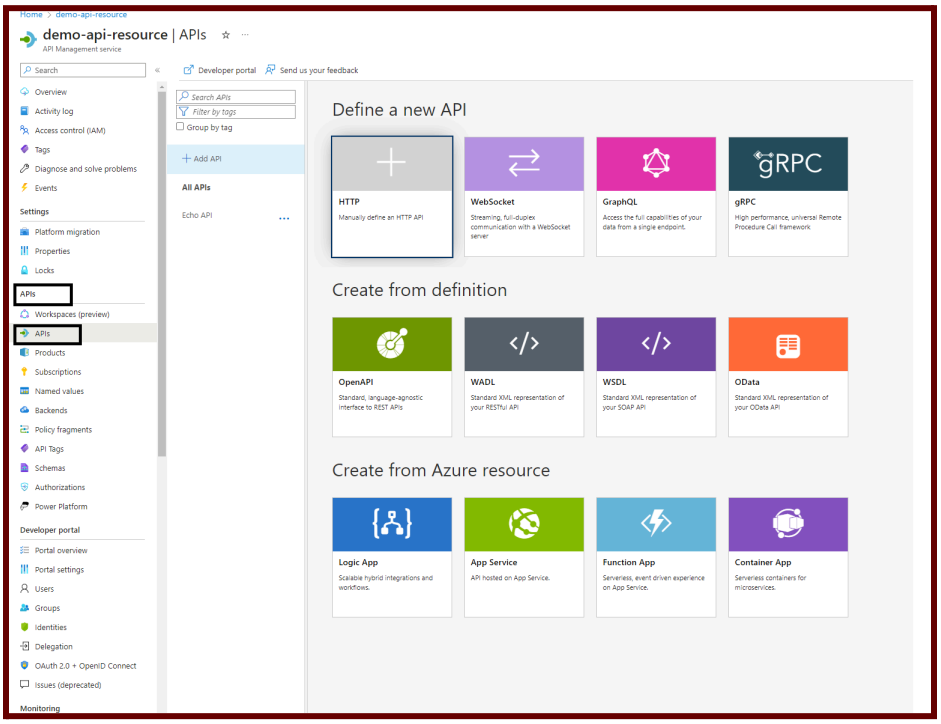


3. Review the properties of your service in the overview page.

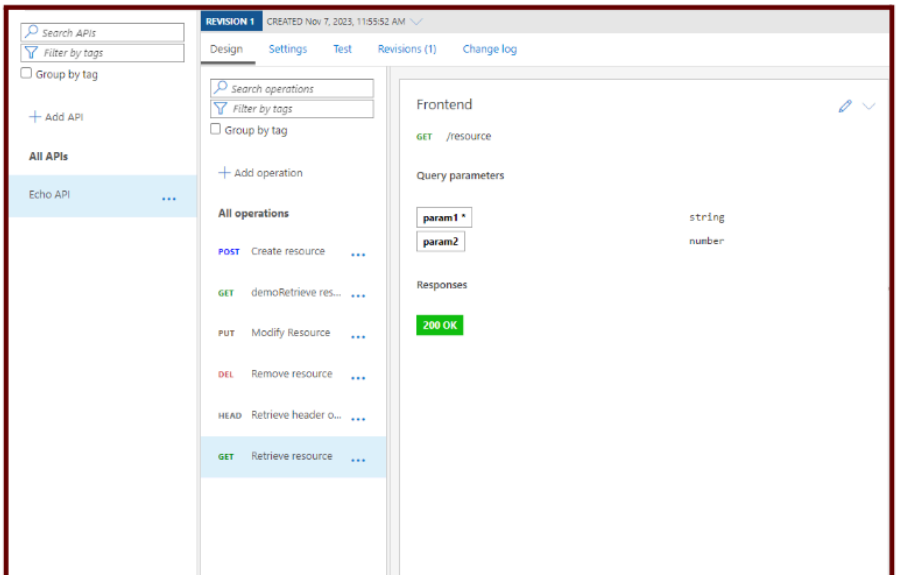


Configure Access to APIs

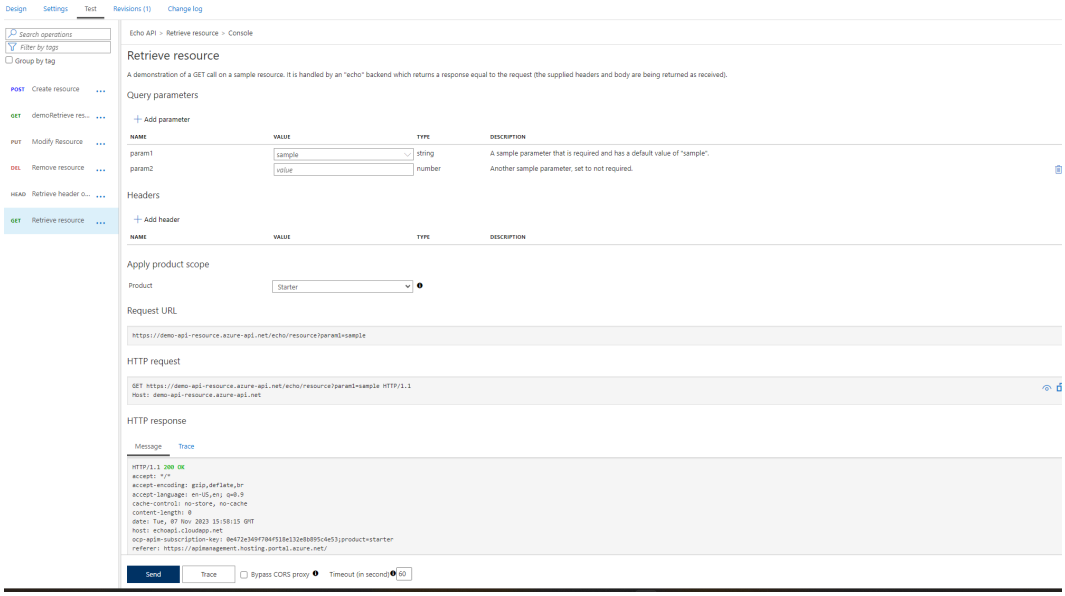
1. Create an API by clicking on APIs under APIs in created API management instance



2. Under that click on Echo API. Here you can create Post, get, del etc. APIs.



3. Now click on test to test the API.



Policies for API

Policies in Azure API Management give the publisher the ability to configure changes to the API's behavior. Statements in a Policy are executed one after the other in response to an API request or response.

The gateway, which stands between the managed API and the API consumer, is where policies are applied. All queries are received by the gateway, which typically sends them to the underlying API unchanged. Changes to a policy, however, may be applied to both the outgoing response and the incoming request. Unless otherwise specified by the policy, policy expressions can be used as text values or attribute values in any API Management policy.

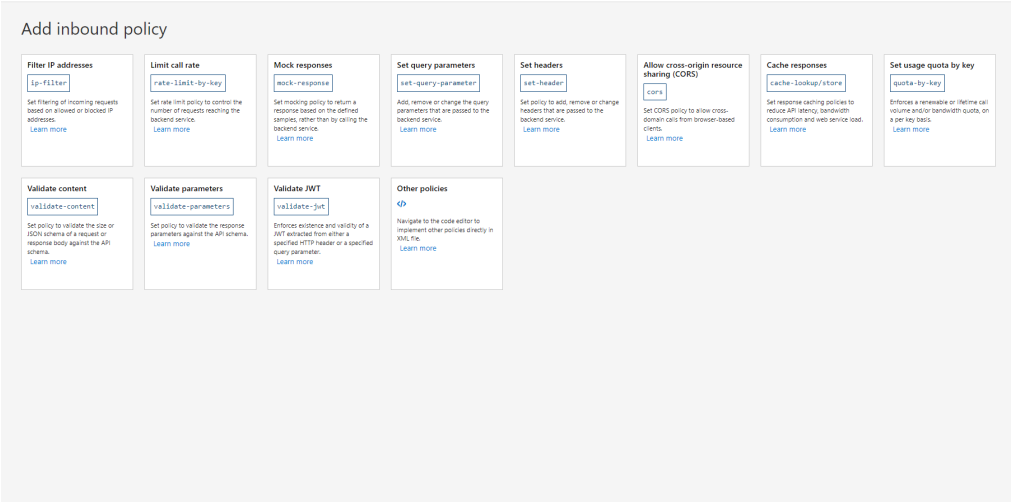
Policy Configuration

A series of incoming and outgoing assertions are described in the policy definition, which is a straightforward XML document. You can directly edit the XML in the definition window.

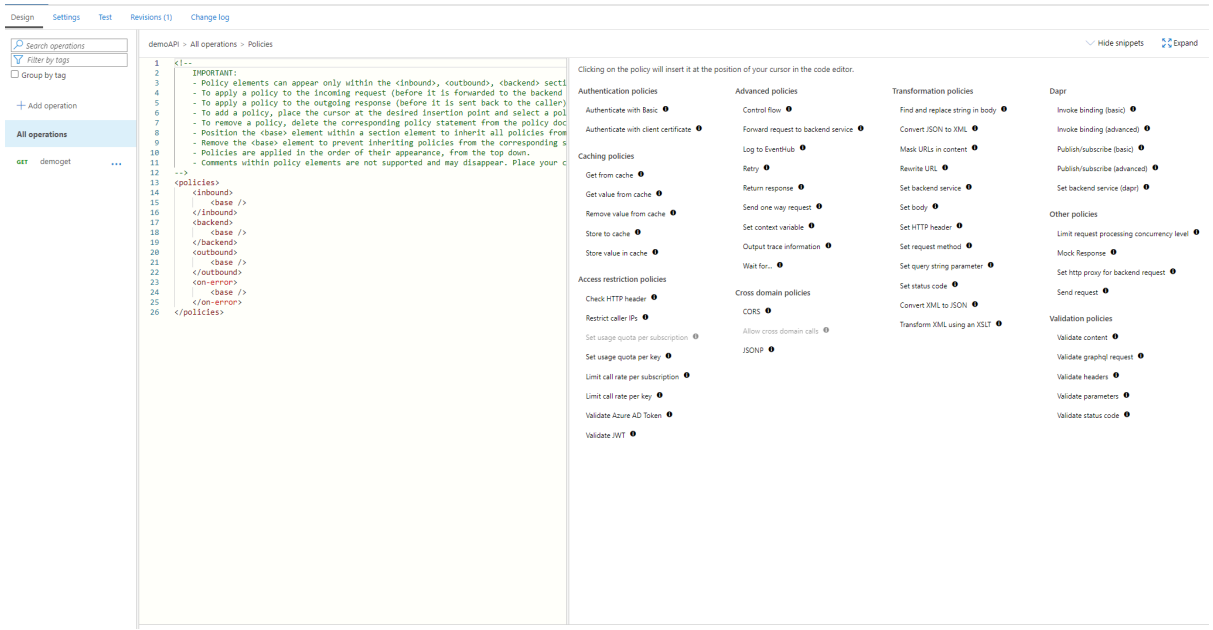
There are four categories in the configuration: inbound, backend, outgoing, and on-error. A request and a response follow the sequence in which the designated policy statements are carried out.

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

1. We can create Inbound, outbound and on -error policies for API.
2. Under Inbound processing click on Add policies you will get different policies that are available.



3. We can implement policies under xml directly also.



4. For this example, we will implement a set status code policy. Save the changes.

```
<!--
    IMPORTANT:
    - Policy elements can appear only within the <inbound>, <outbound>, <backend> section elements.
    - To apply a policy to the incoming request (before it is forwarded to the backend service), place a corresponding policy element within the <inbound> section element.
    - To apply a policy to the outgoing response (before it is sent back to the caller), place a corresponding policy element within the <outbound> section element.
    - To add a policy, place the cursor at the desired insertion point and select a policy from the sidebar.
    - To remove a policy, delete the corresponding policy statement from the policy document.
    - Position the <base> element within a section element to inherit all policies from the corresponding section element in the enclosing scope.
    - Remove the <base> element to prevent inheriting policies from the corresponding section element in the enclosing scope.
    - Policies are applied in the order of their appearance, from the top down.
    - Comments within policy elements are not supported and may disappear. Place your comments between policy elements or at a higher level scope.
-->
<policies>
    <inbound>
        <base />
    </inbound>
    <backend>
        <base />
    </backend>
    <outbound>
        <set-status code="300" reason="Good" />
        <base />
    </outbound>
    <on-error>
        <base />
    </on-error>
</policies>
```

Azure Event Grid and Hub

A. Azure Event Grid

You can utilize Azure Event Grid, a serverless event broker, to integrate apps with events. Event Grid delivers events to subscriber destinations, which can be any endpoint with network access that Event Grid has, or applications, Azure services, etc. These events may originate from other apps, SaaS services, or Azure services. Events are released by publishers, but they have no say in how they are handled. Selecting which events to handle is up to the subscribers.

Developing applications using event-based architectures is made simple using Event Grid. Storage blobs and resource groups are two Azure services that Event Grid supports natively. Moreover, Event Grid supports specific subjects for your own events.

Filters can be used to multicast events to numerous endpoints, route particular events to distinct endpoints, and ensure the dependable delivery of your events.

The following image shows how Event Grid connects sources and handlers, and isn't a comprehensive list of supported integrations.

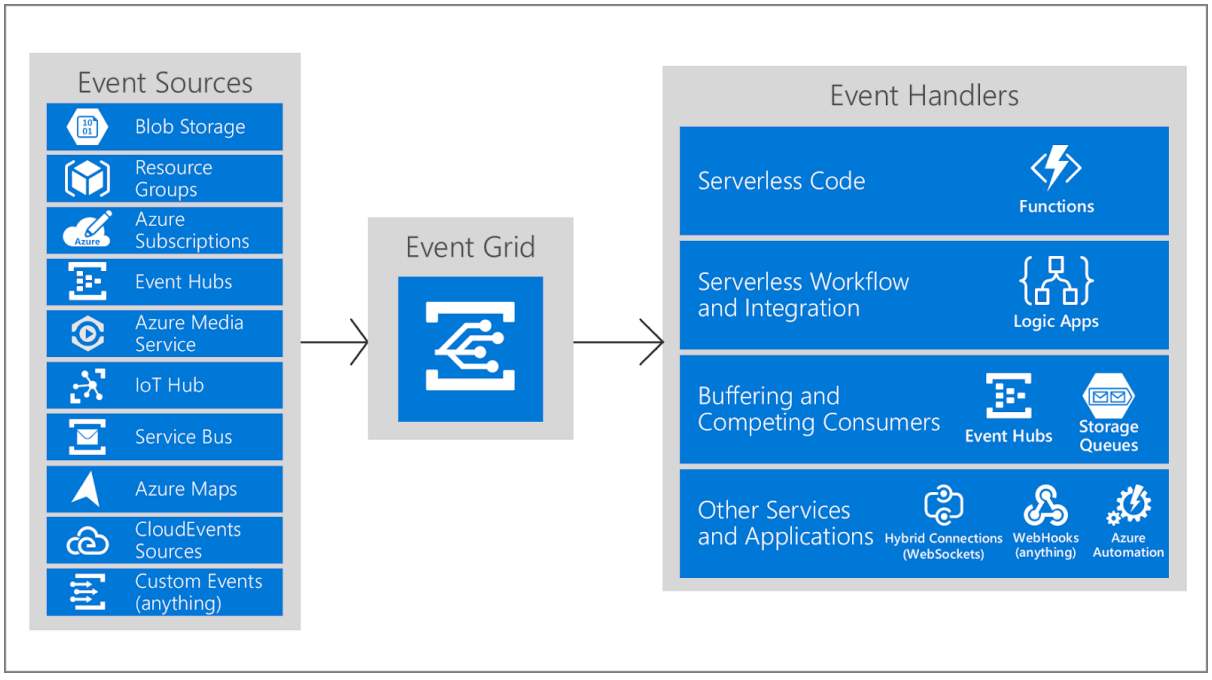


Image ref - https://learn.microsoft.com/en-us/training/modules

There are five concepts in Azure Event Grid you need to understand to help you get started:

- **Events -** What happened.

  The smallest piece of data that completely captures a systemic event is called an event. Every event contains certain common details, such as its origin, the time it happened, and its unique identification. Additionally, each event has unique information that is only pertinent to that particular kind of event. An event pertaining to the creation of a new file in Azure Storage, for instance, contains information about the file, including its lastTimeModified value. Or, the URL of the capture file is present in an Event Hubs event.

  A service level agreement (SLA) for general availability (GA) covers an event up to 64 KB in size. Currently in preview is support for events up to 1 MB in size. Events larger than 64 KB incur fees.

- **Event sources -** Where the event took place.

  Where an event occurs is known as the event source. There are one or more event categories associated with each event source. For instance, the event source for blob-created events is Azure Storage. For events created by devices, the event source is IoT Hub. For custom events that you define, your application serves as the event source. Sending events to Event Grid is the responsibility of event sources.

- **Topics -** The endpoint where publishers send events.

  An endpoint for sending events from the source is provided by the Event Grid topic. The topic for the Event Grid is created by the publisher, who also determines if an event source requires one topic or multiple topics. A group of connected occurrences are grouped together under a theme. Subscribers choose which topics to subscribe to in order to react to specific kinds of events.

  Azure services offer built-in topics known as system topics. Because the publisher controls the topics, you cannot see system topics in your Azure subscription, but you can subscribe to them. You must enter details about the resource you wish to receive events from in order to subscribe. You can sign up for its events as long as you have access to the resource. Third-party and application topics are examples of custom topics. You see a custom subject in your subscription when you create it or are granted access to it.

- **Event subscriptions** - The endpoint or built-in mechanism to route events, sometimes to more than one handler. Subscriptions are also used by handlers to intelligently filter incoming events.

  You can choose the events on a particular topic you want to receive from Event Grid by subscribing. An endpoint for handling the event is provided by you at the time of subscription creation. The events that are sent to the endpoint can be filtered. You can apply a subject pattern or event type filter. If an event subscription is only required temporarily and you don't want to bother about maintaining it, set an expiration date.

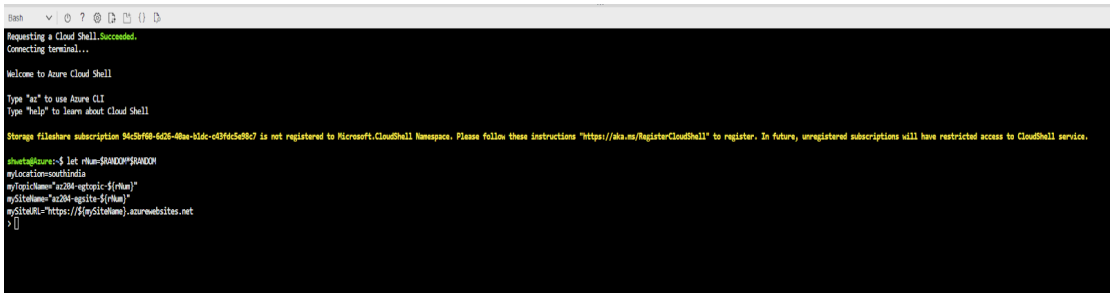
- **Event handlers** - The app or service reacting to the event.

  An event handler is where the event is sent from an Event Grid perspective. In order to process the event, the handler does some further actions. Several handler types are supported by Event Grid. As the handler, you can use your own webhook or one that is supported by Azure. Event Grid uses various methods to ensure that the event is delivered, depending on the type of handler. The event is retried for HTTP webhook event handlers until the handler returns a status code of 200 – OK. Events for Azure Storage Queue are repeated until the message push into the queue is successfully processed by the Queue service.

Implementation

1. Login to azure portal. Launch the Cloud Shell:
2. Select Bash as the shell.
3. Run the following commands to create the variables. Replace <myLocation> with a region near you.

```
let rNum=$RANDOM*$RANDOM
myLocation=<myLocation>
myTopicName="az204-egtopic-${rNum}"
mySiteName="az204-egsite-${rNum}"
mySiteURL="https://${mySiteName}.azurewebsites.net"
```



4. Create a resource group for the new resources you're creating.

```
shweta@Azure:~$ az group create --name az204-evgrid-rg --location $myLocation
{
  "id": "/subscriptions/94c5bf60-6d26-40ae-b1dc-c43fdc5e98c7/resourceGroups/az204-evgrid-rg",
  "location": "southindia",
  "managedBy": null,
  "name": "az204-evgrid-rg",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": "Microsoft.Resources/resourceGroups"
}
```

5. Register the Event Grid resource provider by using the az provider register command.
   az provider register --namespace Microsoft.EventGrid
   It can take a few minutes for the registration to complete. To check the status run the following command.
   az provider show --namespace Microsoft.EventGrid --query "registrationState"

```
shweta@Azure:~$ az provider register --namespace Microsoft.EventGrid
shweta@Azure:~$ az provider show --namespace Microsoft.EventGrid --query "registrationState"
"Registered"
```

6. Create a custom topic by using the az eventgrid topic create command. The name must be unique because it's part of the DNS entry.

```
az eventgrid topic create --name $myTopicName \
  --location $myLocation \
  --resource-group az204-evgrid-rg
```

7. Copy the following command, specify a name for the web app (Event Grid Viewer sample), and press ENTER to run the command. Replace <your-site-name> with a unique name for your web app. The web app name must be unique because it's part of the DNS entry.

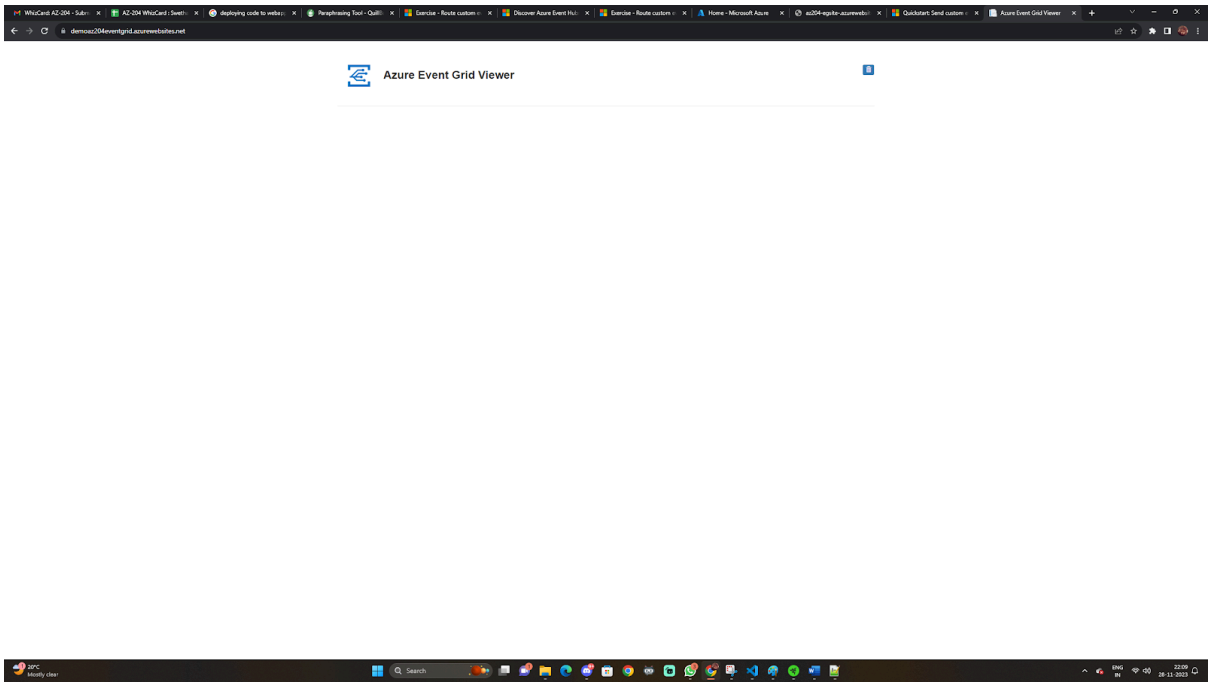
```
sitename=<your-site-name>
```

8. Run the az deployment group create to deploy the web app using an Azure Resource Manager template.

```
az deployment group create
--resource-group az204-evgrid-rg --template-uri
"https://raw.githubusercontent.com/Azure-Samples/azure-event-grid-viewer
/master/azuredeploy.json" --parameters siteName=$sitename
hostingPlanName=viewerhost
```

```
shweta@Azure:~$ az deployment group create -g az204-evgrid-rg  --template-uri "https://raw.githubusercontent.com/Azure-Samples/azure-event-grid-viewer/master/azuredeploy.json" --parameters siteName=$sitename hostingPlanName=viewerhost
{
  "id": "/subscriptions/94c5bf60-6d26-40ae-b1dc-c43fdc5e98c7/resourceGroups/az204-evgrid-rg/providers/Microsoft.Resources/deployments/azuredeploy",
  "location": null,
  "name": "azuredeploy",
  "properties": {
    "correlationId": "86d51292-6b84-4132-92a9-8532772a4cc3",
    "debugSetting": null,
    "dependencies": [
      {
        "dependsOn": [
          {
            "id": "/subscriptions/94c5bf60-6d26-40ae-b1dc-c43fdc5e98c7/resourceGroups/az204-evgrid-rg/providers/Microsoft.Web/serverfarms/viewerhost",
            "resourceGroup": "az204-evgrid-rg",
            "resourceName": "viewerhost",
            "resourceType": "Microsoft.Web/serverfarms"
          }
        ],
        "id": "/subscriptions/94c5bf60-6d26-40ae-b1dc-c43fdc5e98c7/resourceGroups/az204-evgrid-rg/providers/Microsoft.Web/sites/demoaz204eventgrid",
        "resourceGroup": "az204-evgrid-rg",
        "resourceName": "demoaz204eventgrid",
        "resourceType": "Microsoft.Web/sites"
      },
      {
        "dependsOn": [
          {
            "id": "/subscriptions/94c5bf60-6d26-40ae-b1dc-c43fdc5e98c7/resourceGroups/az204-evgrid-rg/providers/Microsoft.Web/Sites/demoaz204eventgrid",
```

The deployment may take a few minutes to complete. After the deployment has succeeded, view your web app to make sure it's running. In a web browser, navigate to: https://<your-site-name>.azurewebsites.net



9. You subscribe to an Event Grid topic to tell Event Grid which events you want to track and where to send those events. The following example subscribes to the custom topic you created, and passes the URL from your web app as the endpoint for event notification.

   The endpoint for your web app must include the suffix /api/updates/.

```
shweta@Azure:~$ az eventgrid event-subscription create --source-resource-id "/subscriptions/94c5bf60-6d26-40ae-b1dc-c43fdc5e98c7/resourceGroups/gridResourceG
roup/providers/Microsoft.EventGrid/topics/$topicname" --name demoViewerSub --endpoint $endpoint
If you are creating an event subscription from a topic that has "Azure" as the value for its "kind" property, you must validate your webhook endpoint by foll
owing the steps described in https://aka.ms/eg-webhook-endpoint-validation.
| Running ..
```

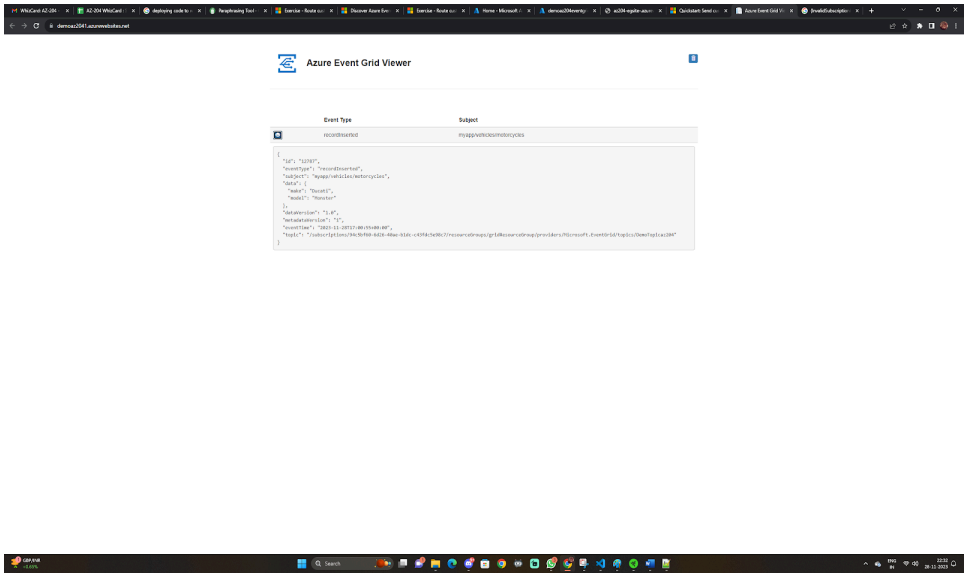
10. Let's trigger an event to see how Event Grid distributes the message to your endpoint. First, let's get the URL and key for the custom topic.

    To simplify this article, you use sample event data to send to the custom topic. Typically, an application or Azure service would send the event data. The following example creates sample event data:
    event='[{"id": "'"$RANDOM"'", "eventType": "recordInserted", "subject": "myapp/vehicles/motorcycles", "eventTime": "'`date +%Y-%m-%dT%H:%M:%S%z`'", "data":{ "make": "Ducati", "model": "Monster"},"dataVersion": "1.0"}]'

11. The data element of the JSON is the payload of your event. Any well-formed JSON can go in this field. You can also use the subject field for advanced routing and filtering.

    CURL is a utility that sends HTTP requests. In this article, use CURL to send the event to the topic.
    curl -X POST -H "aeg-sas-key: $key" -d "$event" $endpoint

12. You've triggered the event, and Event Grid sent the message to the endpoint you configured when subscribing. View your web app to see the event you just sent.



13. After all is done clean up the resources

    az group delete --name gridResourceGroup

B. Azure Event hub

The "front door" of an event pipeline, also known as an event ingestor in solution architectures, is represented by Azure Event Hubs. To separate the creation of an event stream from the consumption of such events, an event ingestor is a part or service that stands in between event publishers and event consumers. By offering a uniform streaming platform with a time retention buffer, Event Hubs helps to separate the needs of event producers and attendees.

The following are the essential elements of Event Hubs:

- The main way that developers interact with the Event Hubs client library is through an Event Hubs client. A variety of Event Hubs clients exist, each focused on a particular use case, such publishing or attending events.
- As part of an embedded device solution, a mobile device application, a game title running on a console or other device, some client or server based business solution, or a web site, an Event Hubs producer is a type of client. They provide telemetry data, diagnostics information, usage logs, or other log data.
- A client that reads data from the Event Hubs and permits processing is known as an Event Hubs consumer. Processing can entail filtering, complicated computing, and aggregation. Distribution or storage of the information in its raw or modified form may also be a part of processing. Customers of Event Hubs are frequently powerful, large-scale platform infrastructure components, such as Apache Spark and Azure Stream Analytics, that have analytics built in.
- An Event Hub's division is a series of events that are held in order. A method of organizing data related to the parallelism needed by event consumers is partitioning. With the partitioned consumer design, which Azure Event Hubs offers, each consumer only reads a certain partition, or subset, of the message stream. Newer events are appended to the end of this sequence as they become available. When an Event Hub is formed, the number of partitions is predetermined and cannot be altered.
- A full Event Hub is seen by a consumer group. Consumer groups allow different applications to separately consume the event stream at their own speed and location, as well as to each have a separate view of the stream. It is advised that there be only one active consumer for each partition and consumer group combination. There can be a maximum of five concurrent readers on a partition per consumer group. All events from a partition are received by each current reader; if many readers are on the same partition, duplicate events will be sent to them.
- Any entity that reads event data from an Event Hub is an event receiver. Users of Event Hubs connect via the AMQP 1.0 session. Events are delivered through a session by the Event Hubs service when they become available. Via the Kafka protocol 1.0 and later, all Kafka consumers establish a connection.

- Processing units, also known as throughput units, are capacity units that are prepurchase and regulate the throughput capacity of Event Hubs.

The following figure shows the Event Hubs stream processing architecture:

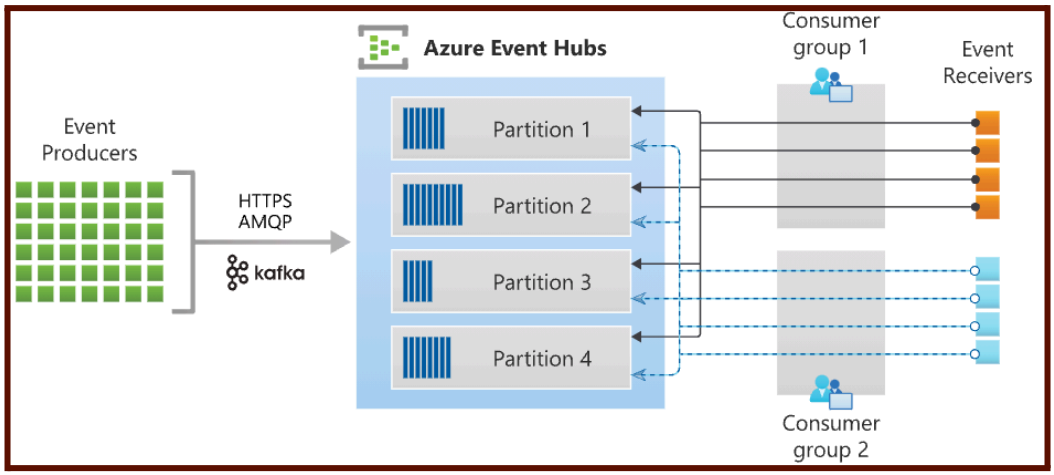
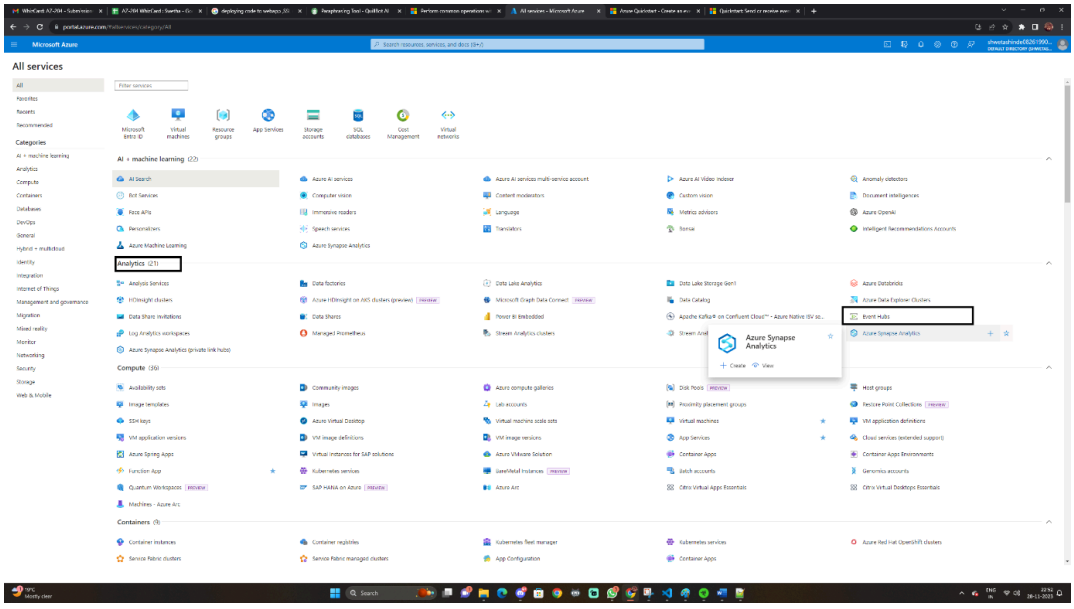


Image ref - https://learn.microsoft.com/en-us/training/modules

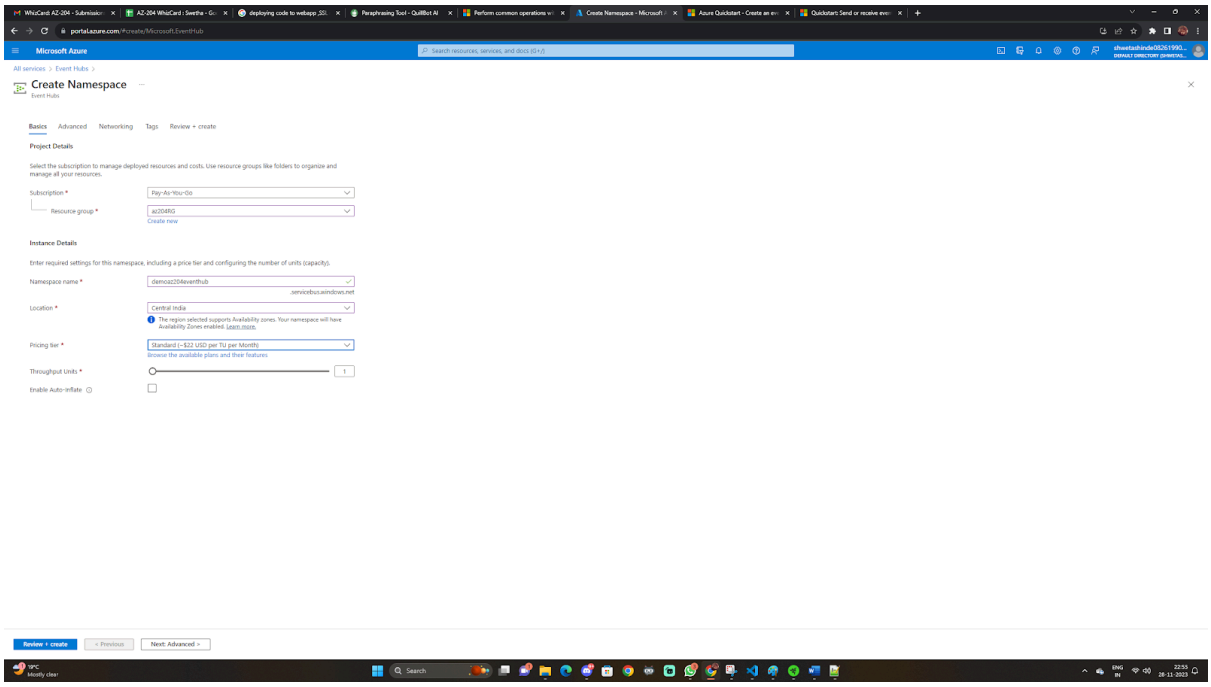
Implementation

1. In the Azure portal, select All services in the left menu, and select star (*) next to Event Hubs in the Analytics category. Confirm that Event Hubs is added to FAVORITES in the left navigation menu.



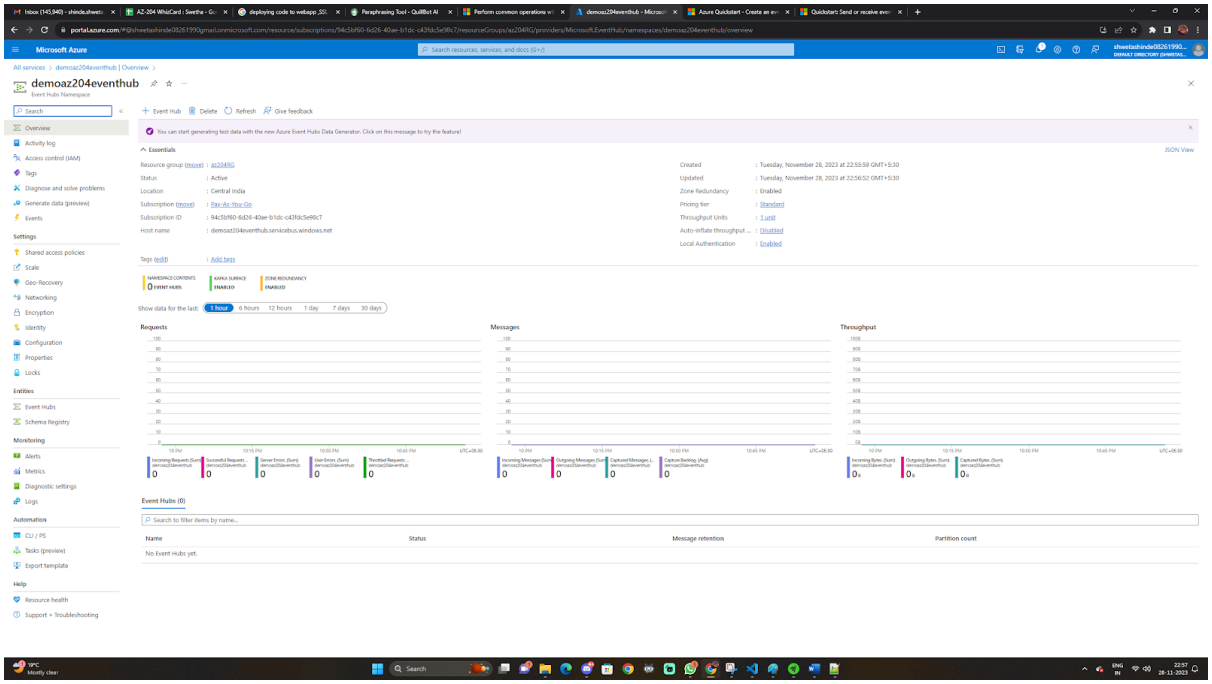
2. Select Event Hubs under FAVORITES in the left navigation menu, and select Create on the toolbar.

3. On the Create namespace page, take the following steps:
   i) Select the subscription in which you want to create the namespace.
   ii) Select the resource group you created in the previous step.
   iii) Enter a name for the namespace. The system immediately checks to see if the name is available.
   iv) Select a location for the namespace.
   v) Choose Basic for the pricing tier. If you plan to use the namespace from Apache Kafka apps, use the Standard tier. The basic tier doesn't support Apache Kafka workloads. To learn about differences between tiers, see Quotas and limits, Event Hubs Premium, and Event Hubs Dedicated articles.
   vi) Leave the throughput units (for standard tier) or processing units (for premium tier) settings as it is. To learn about throughput units or processing units: Event Hubs scalability.
   vii) Select Review + Create at the bottom of the page.

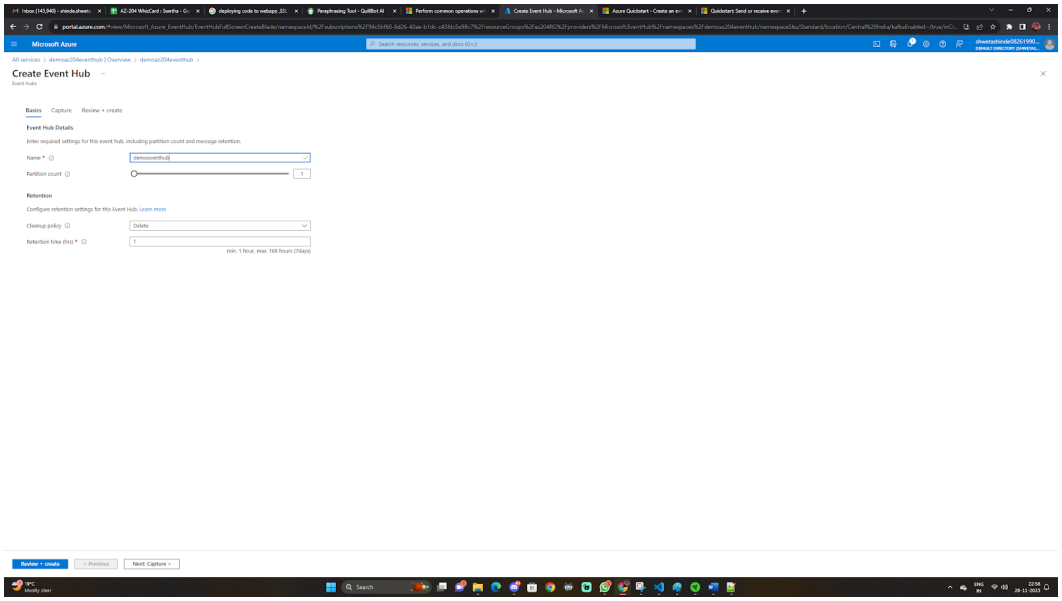


   viii) On the Review + Create page, review the settings, and select Create. Wait for the deployment to complete.

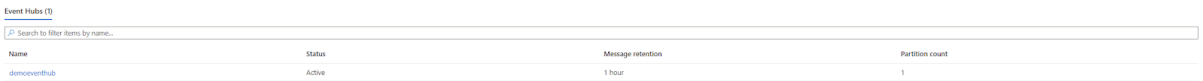
4. Confirm that you see the Event Hubs Namespace page similar to the following example:



5. On the Overview page, select + Event hub on the command bar.
6. Type a name for your event hub, then select Review + create.



7. On the Review + create page, select Create.
8. You can check the status of the event hub creation in alerts. After the event hub is created, you see it in the list of event hubs.



Azure Service Bus and Azure Queue Storage queues

Storage queues and Service Bus queues are the two kinds of queuing mechanisms that Azure supports.

Provision of services Bus queues are a component of a larger Azure messaging system that also includes publish/subscribe, more complex integration patterns, and queuing. Their purpose is to combine different communication protocols, data contracts, trust domains, and network environments with applications or application components.

The Azure Storage infrastructure includes storage queues. You can save a lot of messages with them. Messages can be accessed via HTTP or HTTPS authenticated calls from any location in the world. A queue message may have a maximum size of 64 KB. Millions of messages could be in a queue, depending on the storage account's maximum capacity. A frequent purpose for queues is to generate an asynchronous work backlog.

The feature set of storage queues and service bus queues differs slightly. Depending on what your specific solution requires, you can select one or both of these options.

Solution architects and developers should take these suggestions into consideration when deciding whether queuing technology best serves the needs of a particular solution.

Have a look at leveraging Service Bus queues.

When should a solution architect or developer use Service Bus queues?

- Messages must be received by your solution without requiring polling the queue. You can accomplish this with Service Bus by utilizing one of the TCP-based protocols that Service Bus offers to perform a long-polling receive operation.
- The queue must guarantee a first-in, first-out (FIFO) ordered delivery in order for your method to work.
- The automatic detection of duplicates must be supported by your solution.
- The session ID attribute on the message is used to associate messages with streams for long-running parallel processing in your application. Instead of competing for messages, every node in the consuming application does so for streams in this approach. A consuming node can use transactions to check the application stream state when it receives a stream.
- Transactional behaviour and atomicity are necessary for your solution to function while sending or receiving numerous messages from a queue.
- Although messages can be longer than 64 KB, your program probably won't get close to the 256-KB limit.

Think of utilizing storage queues.

When should a solution architect or developer use storage queues?

- In a queue, your application has to hold more than 80 terabytes of messages.
- The application you are using wants to monitor the status of a message that is in the queue. It is helpful in the event that a worker handling a message crashes. Then, another worker can pick up where the previous worker left off by using that knowledge.
- Server-side logs of every transaction made against your queues are necessary.

Azure Service Bus

A fully managed enterprise integration message broker is Microsoft Azure Service Bus. Applications and services can be separated via Service Bus. Messages are used to communicate data between various services and applications. A message is a data-containing container with metadata decorating it. Any type of information may be included in the data, including structured data encoded in widely used formats like JSON, XML, Apache Avro, and plain text.

Here are a few such messaging scenarios:

- Messaging. Transmit business data, including journals, inventory movements, sales or purchase orders.
- Disconnect the apps. Boost applications' and services' scalability and dependability. Services and clients may not need to be available online at the same time. Subscriptions and subjects. Facilitate 1:n connections between subscribers and publishers.
- Message Sessions. Put into practice workflows that call for message deferral or ordering.

Service Bus Tiers

Service Bus has two tiers: ordinary and premium. Scale, performance, and availability are key client needs that are addressed by the Service Bus Messaging premium tier for mission-critical applications. For production scenarios, the premium tier is advised. These two Service Bus Messaging tiers are intended to support distinct use cases, although having feature sets that are almost identical.

The following table highlights a few significant variations.

Standard	**Premium**
Variable throughput	High throughput
Variable latency	Predictable performance
Pay as you go variable pricing	Fixed pricing
N/A	Ability to scale workload up and down

Message size up to 256kb	Message size up to 100 MB

Service Bus queues, Subscription and topics

Queues, topics, Subscriptions, and rules/actions are the messaging entities that make up the foundation of Service Bus's messaging capabilities.

1. Queues

- First In, First Out (FIFO) message delivery is provided via queues to one or more rival customers. In other words, messages are normally received and processed by receivers in the order that they were added to the queue. Additionally, each message is received and processed by a single message consumer. It is not necessary for producers (senders) and consumers (receivers) to process messages simultaneously because they are kept durably in the queue.
- Load-levelling is a related advantage that allows messages to be sent and received at varying speeds by producers and consumers. Numerous applications exhibit fluctuations in system load over time. On the other hand, processing times are usually constant for each unit of work. When a queue is used as a middleman between message producers and consumers, the consuming application just has to be able to manage average traffic as opposed to peak load.
- There is an intrinsic loose coupling between the components when message producers and consumers are connected via queues. A consumer can be enhanced without affecting the producer because producers and consumers are unaware of one another
- The Azure interface, PowerShell, CLI, and Resource Manager templates can all be used to build queues. Next, use clients to send and receive messages.

2. Topics and Subscriptions

- A queue enables one consumer to process a message at a time. Topics and subscriptions offer a publish and subscribe pattern that facilitates one-to-many communication, unlike queues. Scaling to a large number of recipients is a benefit. Every subscriber registered with the topic has access to every published communication. Depending on the filter criteria used to these subscriptions, a copy of a message sent by the publisher to a subject may reach one or more subscribers. More filters are available for the subscriptions to utilize in order to limit the communications they wish to receive.

- Messages are sent by publishers to topics in the same manner as they are sent to queues. Customers do not, however, get messages straight from the subject. Instead, subscribers to the topic send messages to consumers. A virtual queue that gets copies of

all messages submitted to the topic is what a topic subscription looks like. Customers receive messages from a subscription in the same manner as they do from a queue.

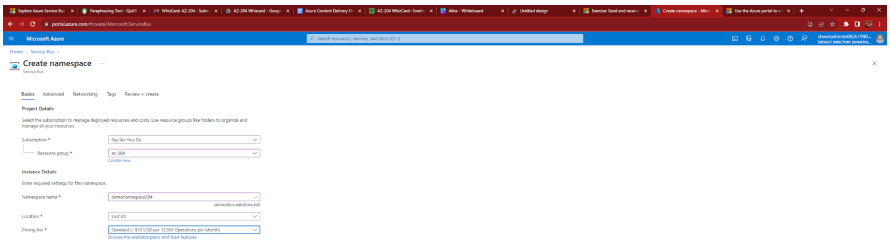
- As mentioned in the preceding section, creating a subject is comparable to creating a queue. Using the Azure portal, PowerShell, CLI, or Resource Manager templates, you can create topics and subscriptions. Next, use clients built in Python, JavaScript, C#, and Java to send messages to a topic and receive messages from subscriptions.

3. Rules and actions

- Messages with certain qualities often require distinct processing methods. You can set up subscriptions to search for messages with required properties and then apply specific adjustments to those properties in order to enable this processing. You can only copy a portion of the messages submitted to the subject to the virtual subscription queue, even though Service Bus subscriptions view all of the messages sent there.
- Subscription filters are used to achieve this filtering. We refer to these changes as filter actions. You can provide a filter expression that modifies the message's attributes when a subscription is created. The properties might be either custom application properties (like Store Name) or system properties (like Label). In this instance, the SQL filter expression is not required. Any filter action defined on a subscription is applied to all of the messages associated with that subscription if no SQL filter expression is used.

Create Azure Service bus

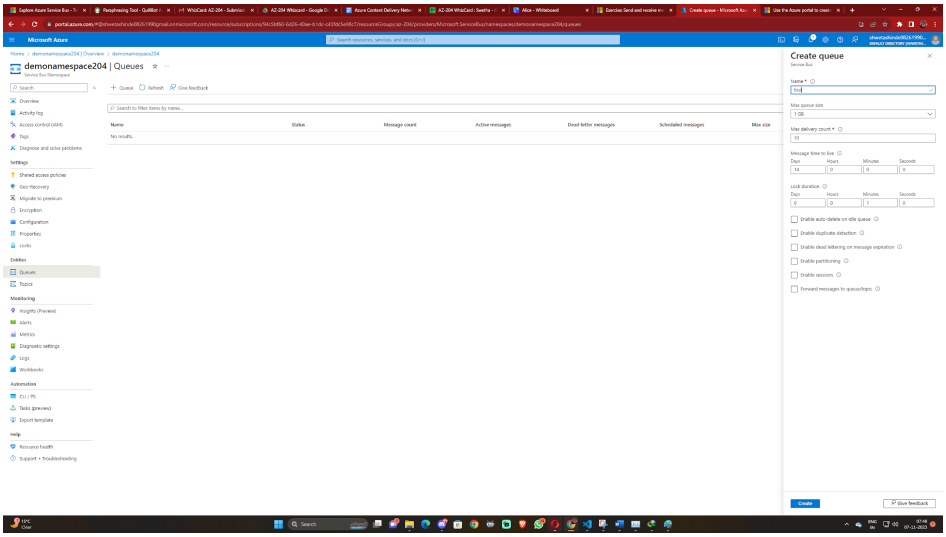
1. Go to azure portal and search **service Bus**
2. Click on create
3. Fill out the necessary details. Namespace name must be uinque



4. Click on Review + create and then click on create.

Create Queue

1. Under Entities click on queue.
2. Click on queue then side window will open. Fill out the necessary details and click on create.



[Introduction to Azure Service Bus, an enterprise message broker - Azure Service Bus | Microsoft Learn]

Azure Queue storage

One service for storing a lot of messages is Azure Queue Storage. Messages can be accessed via HTTP or HTTPS authenticated calls from any location in the world. A queue message may have a maximum size of 64 KB. Millions of messages could be in a queue, depending on the storage account's maximum capacity. A frequent purpose for queues is to generate an asynchronous work backlog.

The Queue service contains the following components:

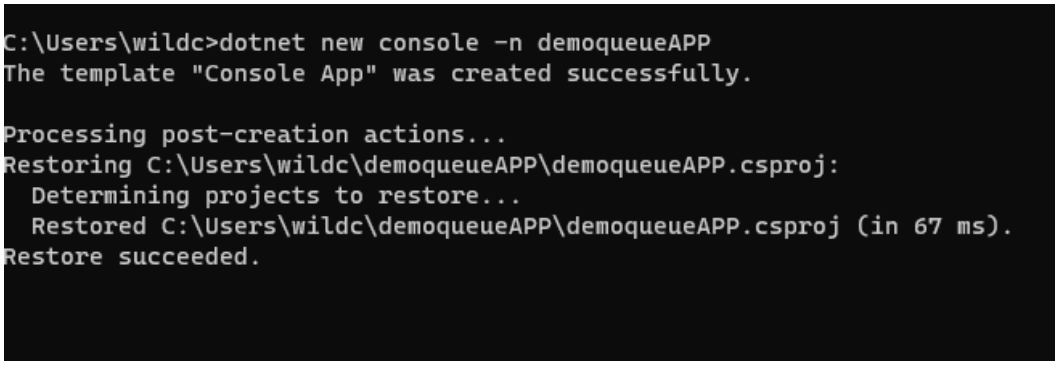
1. **Storage account** - A storage account is required for all forms of access to Azure Storage.
2. **Queue** - A series of messages is contained in a queue. Every message needs to be queued up. Lowercase letters are required in the queue name.
3. **URL Format** - The URL syntax https://<storage account>.queue.core.windows.net/<queue> can be used to access queues. For instance,

the URL https://myaccount.queue.core.windows.net/images-to-download addresses a queue in the diagram above.

4. **Message** - A message up to 64 KB in size, in any format. The maximum time-to-live for versions 2017-07-29 or later can be any positive value or -1, which indicates that the message never expires. The time-to-live is set to seven days by default if this option is left out.

Implementation of queue

1. Create storage account if you don't have already created one
2. Create an app
   - To build a new console app called QueueApp, use the dotnet new command in a console window (such as cmd, PowerShell, or Azure CLI). A basic C# "hello world" project with a single source file called Program.cs is created by running this command.



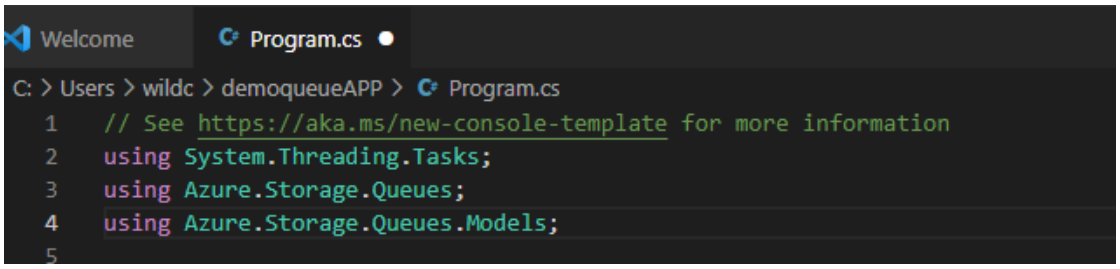
   - To ensure everything is working as it should, switch to the just created demoqueueApp folder and build the app. You will be able to see result as below

3. Add Azure client libraries

```
C:\Users\wildc\demoqueueAPP>dotnet add package Azure.Storage.Queues
  Determining projects to restore...
  Writing C:\Users\wildc\AppData\Local\Temp\tmp9691.tmp
info : X.509 certificate chain validation will use the default trust store selected by .NET for code signing.
info : X.509 certificate chain validation will use the default trust store selected by .NET for timestamping.
info : Adding PackageReference for package 'Azure.Storage.Queues' into project 'C:\Users\wildc\demoqueueAPP\demoqu
.csproj'.
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/azure.storage.queues/index.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/azure.storage.queues/index.json 1770ms
info : Restoring packages for C:\Users\wildc\demoqueueAPP\demoqueueAPP.csproj...
info :   GET https://api.nuget.org/v3-flatcontainer/azure.storage.queues/index.json
info :   OK https://api.nuget.org/v3-flatcontainer/azure.storage.queues/index.json 980ms
info :   GET https://api.nuget.org/v3-flatcontainer/azure.storage.queues/12.17.0/azure.storage.queues.12.17.0.nupk
info :   OK https://api.nuget.org/v3-flatcontainer/azure.storage.queues/12.17.0/azure.storage.queues.12.17.0.nupkg
s
```

4. Add Using statements

- To launch Visual Studio Code in the current directory, type code. from the project directory's command line. Maintain the open command-line window. Later on, you'll need to perform additional commands. Click the Yes button if asked to add C# assets that are needed for debugging and building.
- After using System; line, open the Program.cs source file and add the following namespaces. This application connects to Azure Storage and manipulates queues using types from these namespaces.

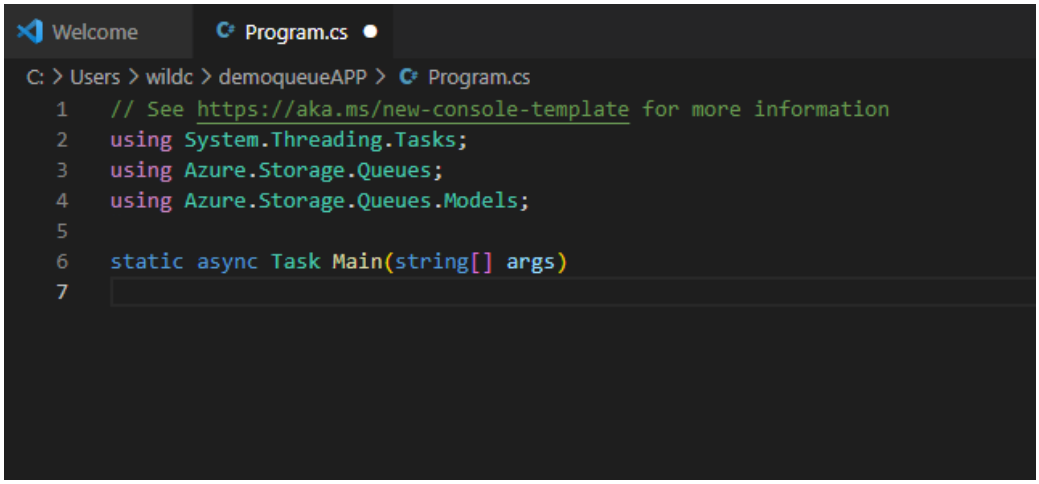


- Save the file.

5. Add support for asynchronous code

The code executes asynchronously since the app makes use of cloud resources.

- Make the Main method run asynchronously by updating it. Put an async Task return value in place of void.



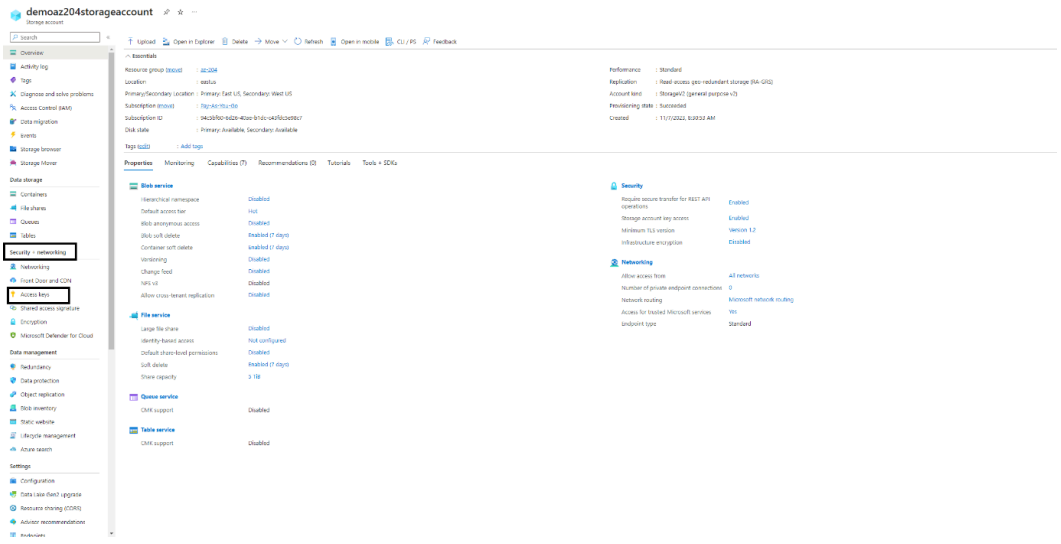
- Save program.cs file

6. Create a queue

   You have to obtain your credentials from the Azure portal before you can make any calls into Azure APIs.

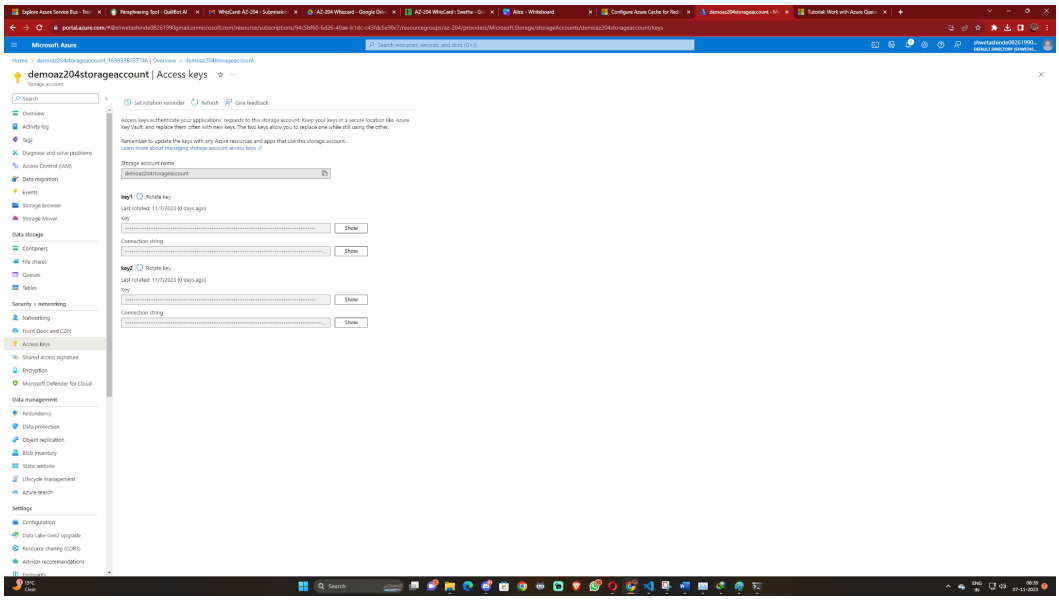
7. Get your credentials from azure portal

   i. Sign into azure portal
   ii. Go to your storage account
   iii. Access keys can be found under Security + networking in the storage account menu pane. The account access keys and the whole connection string for every key are displayed here.



iv. In the Access keys pane select show keys

v. Find the value of the Connection string in the key1 section. To copy the connection string, select the Copy to clipboard icon. In the next part, you will assign the value of the connection string to an environment variable.



8. Configure your storage account connection string

   Once the connection string has been copied, put it into a fresh environment variable on the local computer that is executing the application. Open a console window and follow your operating system's instructions to set the environment variable. Put your real connection string in lieu of <yourconnectionstring>.

```
C:\Users\wildc>setx AZURE_STORAGE_CONNECTION_STRING
cK0fHGPo+AStDxegpg==

SUCCESS: Specified value was saved.
```

   You need to launch a fresh command window in Windows after adding the environment variable.

9. Restart programs

   Restart any open applications that require the environment variable to be read after adding it. For instance, before proceeding, restart your editor or development environment.

10. Add a connection string to program

To allow the program to access the storage account, add the connection string to it.

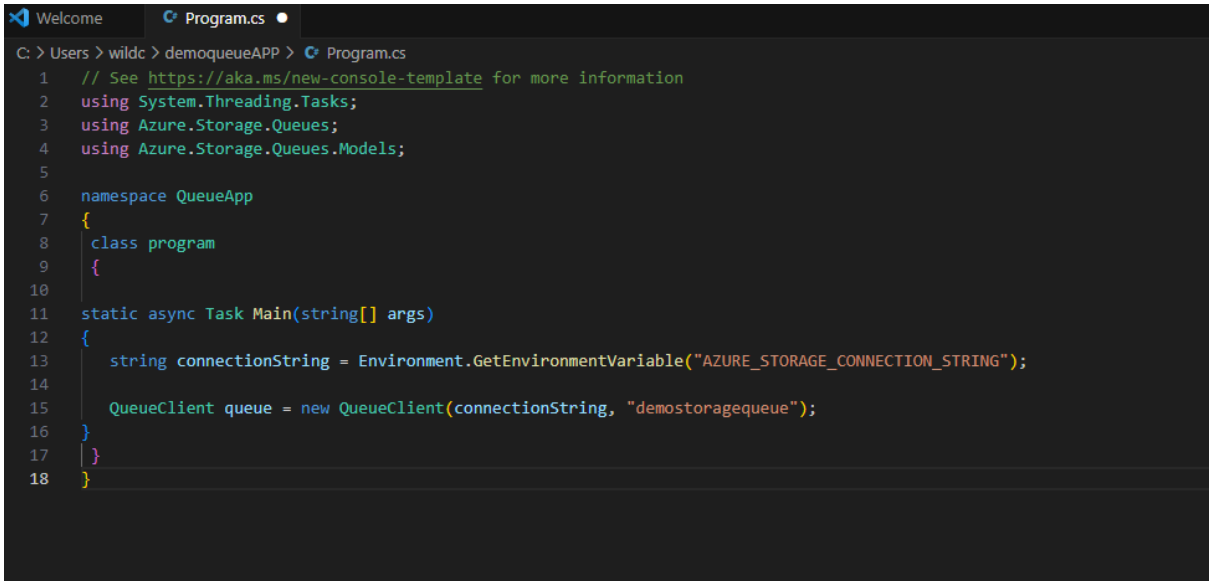
i. Return to the Visual Studio Code window.

ii. To update the console, use the Main method.WriteLine("Hello, World"); code that pulls the connection string from the environment variable is written here.

string connectionString = Environment.GetEnvironmentVariable("AZURE_STORAGE_CONNECTION_STRING");

iii. To establish a queue object, add the following code to Main. This object is then supplied into the send and receive methods.

QueueClient queue = new QueueClient(connectionString, "demostoragequeue");



iv. Save the file.

11. Insert the message into the queue.

To add a message to the queue, create a new method.

i. To your Program class, add the InsertMessageAsync method.

A queue reference is supplied to this method. If the queue does not already exist, it is created by invoking the CreateIfNotExistsAsync function. Next, it calls SendMessageAsync to add the newMessage to the queue.

```
static async Task InsertMessageAsync(QueueClient theQueue, string newMessage)
{
    if (null != await theQueue.CreateIfNotExistsAsync())
    {
        Console.WriteLine("The queue was created.");
    }

    await theQueue.SendMessageAsync(newMessage);
}
```

ii. Optional: By default, a message's maximum time-to-live is seven days. Any positive number can be used as the message time-to-live. The code line that follows appends an eternal message.

Use Timespan.FromSeconds(-1) in your call to SendMessageAsync to add a message that never expires.

```
await theQueue.SendMessageAsync(newMessage, default,
TimeSpan.FromSeconds(-1), default);
```

iii. Save the file.

12. Dequeue the messages

In order to obtain a message from the queue, create a new method. It's crucial to remove the message from the queue as soon as it's successfully received to prevent it from being processed more than once.

i. Incorporate RetrieveNextMessageAsync as a new method into your Program class.
In order to retrieve only the next message in the queue, this method calls ReceiveMessagesAsync and passes 1 as the first parameter. Call DeleteMessageAsync to remove the message from the queue once it has been received.

Messages transmitted to the queue with SDK versions older than v12 are automatically encoded with Base64. With version 12, that feature was eliminated. A message that you obtain using the v12 SDK is not Base64-decoded by default. The contents must be explicitly Base64-decoded by you.

```
static async Task<string> RetrieveNextMessageAsync(QueueClient theQueue)
{
    if (await theQueue.ExistsAsync())
    {
        QueueProperties properties = await theQueue.GetPropertiesAsync();

        if (properties.ApproximateMessagesCount > 0)
        {
            QueueMessage[] retrievedMessage = await theQueue.ReceiveMessagesAsync(1);
            string theMessage = retrievedMessage[0].Body.ToString();
            await theQueue.DeleteMessageAsync(retrievedMessage[0].MessageId, retrievedMessage[0].PopReceipt);
            return theMessage;
        }

        return null;
    }

    return null;
}
```

ii. Save the file

13. Delete an empty queue

At the conclusion of a project, it's great practice to determine whether you still require the materials you produced. If resources are left idle, it might be expensive. Inquire with the user about deleting the queue if it is empty but still remains.

i. Create an extension for the RetrieveNextMessageAsync function that asks to remove the empty queue.

```
static async Task<string> RetrieveNextMessageAsync(QueueClient theQueue)
{
    if (await theQueue.ExistsAsync())
    {
        QueueProperties properties = await theQueue.GetPropertiesAsync();

        if (properties.ApproximateMessagesCount > 0)
        {
            QueueMessage[] retrievedMessage = await theQueue.ReceiveMessagesAsync(1);
            string theMessage = retrievedMessage[0].Body.ToString();
            await theQueue.DeleteMessageAsync(retrievedMessage[0].MessageId, retrievedMessage[0].PopReceipt);
            return theMessage;
        }
        else
        {
            Console.Write("The queue is empty. Attempt to delete it? (Y/N) ");
            string response = Console.ReadLine();

            if (response.ToUpper() == "Y")
            {
                await theQueue.DeleteIfExistsAsync();
                return "The queue was deleted.";
            }
            else
            {
                return "The queue was not deleted.";
            }
        }
    }
    else
    {
        return "The queue does not exist. Add a message to the command line to create the queue and store the message.";
    }
}
```

ii. Save the file.

14. Check for command-line arguments

    If the application receives any command-line parameters, consider them to be messages that need to be put to the queue. To create a string, join the arguments together. By using the InsertMessageAsync function that we previously implemented, you can add this string to the message queue.

    Try a retrieve operation if no command-line inputs are provided. To obtain the subsequent message in the queue, using the RetrieveNextMessageAsync function.

    Finally, before calling Console to end the process, wait for user input.ExamineLine.

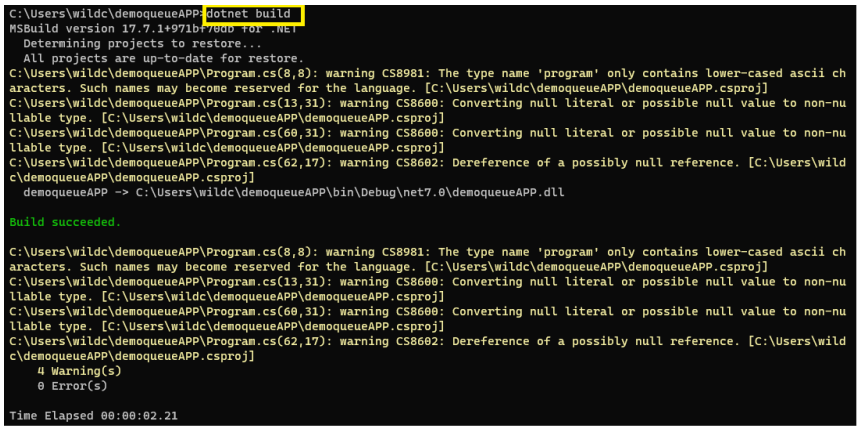
    i. To check for command-line parameters and wait for user input, expand the Main method.

```
static async Task Main(string[] args)
{
    string connectionString = Environment.GetEnvironmentVariable("AZURE_STORAGE_CONNECTION_STRING");

    QueueClient queue = new QueueClient(connectionString, "mystoragequeue");

    if (args.Length > 0)
    {
        string value = String.Join(" ", args);
        await InsertMessageAsync(queue, value);
        Console.WriteLine($"Sent: {value}");
    }
    else
    {
        string value = await RetrieveNextMessageAsync(queue);
        Console.WriteLine($"Received: {value}");
    }

    Console.Write("Press Enter...");
    Console.ReadLine();
}
```

    ii. Save the file.

15. Build and run the app.

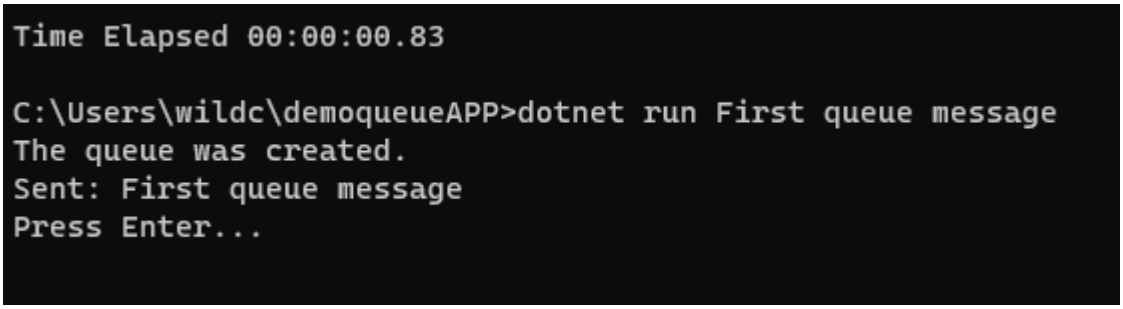
    1. To build the project, execute the following dotnet command from the project directory's command line.



2. Execute the following command to add the first message to the queue once the project has successfully built.

   dotnet run First queue message

   You will see below ouput:



3. Launch the application without any command-line options to obtain and eliminate the initial message in the queue.

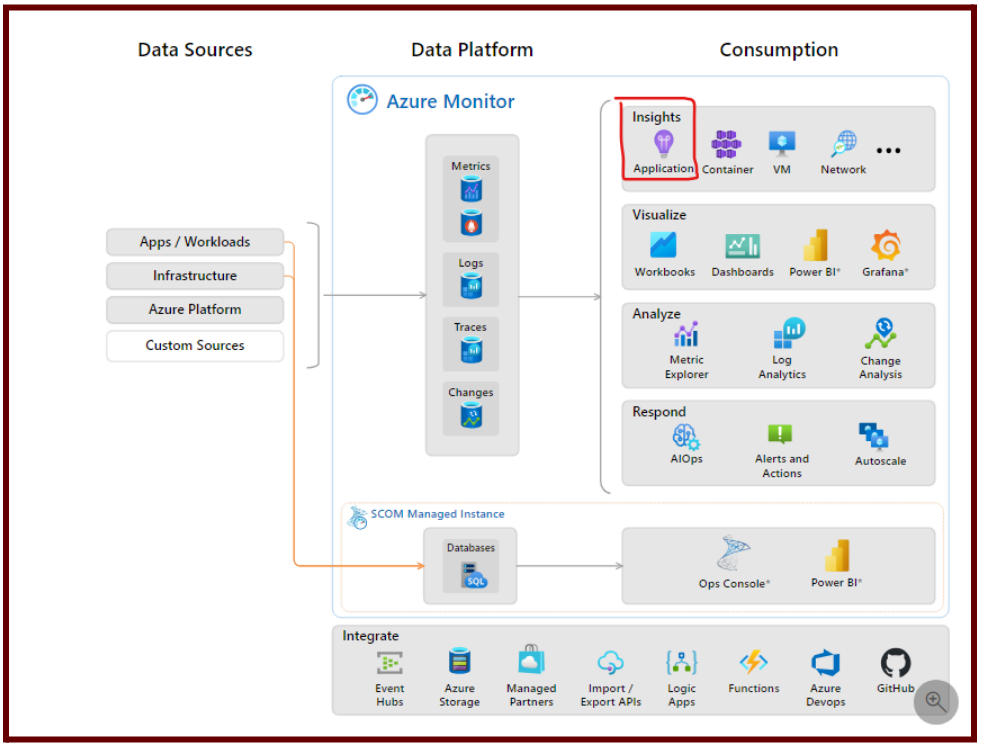
   dotnet run

4. Run the program again until all of the messages are gone. You will receive a notification stating that the queue is empty and an instruction to remove the queue if you run it again.

```
C:\Users\wildc\demoqueueAPP>dotnet run First queue message
The queue was created.
Sent: First queue message
Press Enter...

C:\Users\wildc\demoqueueAPP>dotnet run Second queue message
Sent: Second queue message
Press Enter...

C:\Users\wildc\demoqueueAPP>dotnet run third queue message
Sent: third queue message
Press Enter...

C:\Users\wildc\demoqueueAPP>dotnet run
Received: First queue message
Press Enter...

C:\Users\wildc\demoqueueAPP>dotnet run
Received: Second queue message
Press Enter...

C:\Users\wildc\demoqueueAPP>dotnet run
Received: third queue message
Press Enter...

C:\Users\wildc\demoqueueAPP>dotnet run
The queue is empty. Attempt to delete it? (Y/N) Y
Received: The queue was deleted.
Press Enter...
```

Troubleshoot solutions by using Application Insights

Azure Monitor is a comprehensive monitoring solution for collecting, analyzing and reacting to monitoring data from your cloud and on-premises environments. You can use Azure Monitor to increase the availability and performance of your applications and services. It helps you understand how your applications are performing and allows you to respond to system events both manually and programmatically.



(Source: Microsoft Documentation)

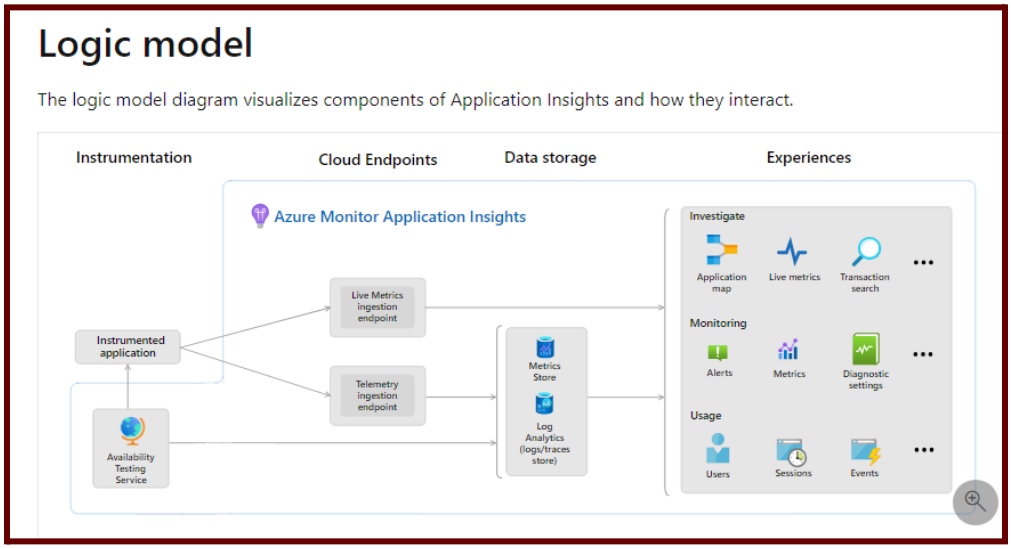
What is Application Insights?

Application Insights is an extension of Azure Monitor and provides Application Performance Monitoring (APM) features. It provides many experiences to improve the performance, reliability and quality of your applications.

APM tools are useful for monitoring applications from development, through test, and into production in the following ways:

- Understand in advance how the application works.
- Review application execution data to determine the cause of the incident.

Application Insights is a feature of Azure Monitor that provides extensible application performance management (APM) and monitoring of live web apps.

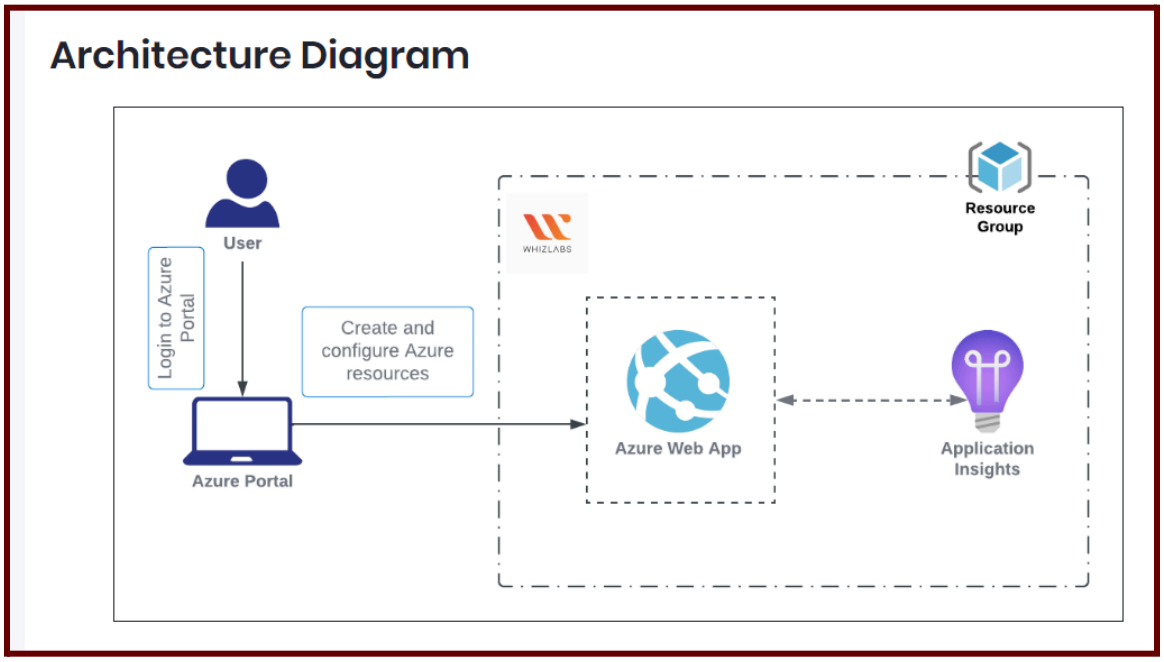


(Source: Microsoft Documentation)

Developers and DevOps professionals can use Application Insights to:

- ➔ Automatically detect performance anomalies.
- ➔ Help identify problems by using powerful analysis tools.
- ➔ See what users actually do with apps.
- ➔ Help continuously improve app performance and usability.

Configure an app or service to use Application Insights:



Application Insights feature overview

Feature	Description
Live Metrics	It observes the activity from the running application in real-time without any impact on the host environment.
Availability	Examine the external endpoint(s) of your applications to test overall availability and responsiveness over time. It's also known as "Synthetic Transaction Monitoring"
GitHub or Azure DevOps integration	Create Azure DevOps or GitHub work items in the context of Application Insights data.
Usage	You can understand which features are popular with users and how users interact with and use your application.
Smart Detection	Automatic failure and abnormality detection through proactive telemetry analysis.
Application Map	A high-level top-down view of the application architecture and at-a-glance visual indications of component health and responsiveness.
Distributed Tracing	Search and visualize the end-to-end flow of a given execution or transaction.

What Application Insights monitors?

Application Insights collects metrics and application telemetry data that describe application activity and health, as well as trace logging data.

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

There are several ways to start monitoring and analyzing app performance:

- **At run time:** Instrument your web app on the server. Ideal for already deployed applications. Prevents any update to the code.
- **At development time:** Add application insights to your code. Allows you to customize telemetry collection and send more telemetry.
- **Instrument your web pages** for page view, AJAX and other client-side telemetry.
- **Analyze mobile app usage** by integrating with Visual Studio App Center.
- **Availability Tests** - Regularly ping your website from our servers.

Application Insights log-based metrics

Application Insights log-based metrics allow you to analyze the health of your monitored apps, create powerful dashboards, and configure alerts.

There are two types of metrics:

- Behind the scenes, Log-based metrics are translated from stored events into Kusto queries
- Standard metrics are stored as pre-aggregated time series.

Standard metrics are pre-aggregated at collection time, they perform better at query time. This makes them the best choice for dashboarding and real-time alerting. Log-based metrics have more dimensions, which makes them a better choice for data analysis and temporal analysis. Use the Namespace selector to switch between log-based and standard metrics in Metrics Explorer.

Interpret and use queries:

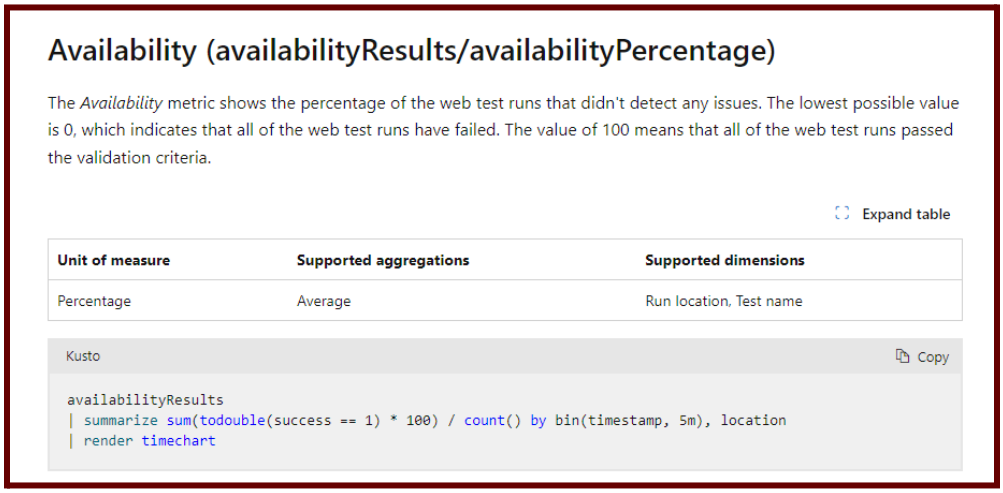
When you plot the same metric in Metrics Explorer, there are no defaults – the query is dynamically adjusted based on your chart settings:

- The selected **Time range** is translated as an additional timestamp... clause to select only events from the selected time range. For example, a chart showing the most recent 24 hours of data in question | Timestamp > ago(24h).

- The chosen/selected **time granularity** is put into the final summary by the ...bin(timestamp, [timegrain]) clause.
- Any **filter** dimensions selected will translate to additional terms.
- The selected **Split chart** size is translated to an additional summary property. For example, if you segment your chart by location and plot using 5-minute time granularity, the summary clause would be summary ... bin(timestamp, 5m), by location.

Availability metrics:

Metrics in the Availability category allow you to see the health of your web application as observed from points around the world. Configure availability tests to start using any metrics from this category.



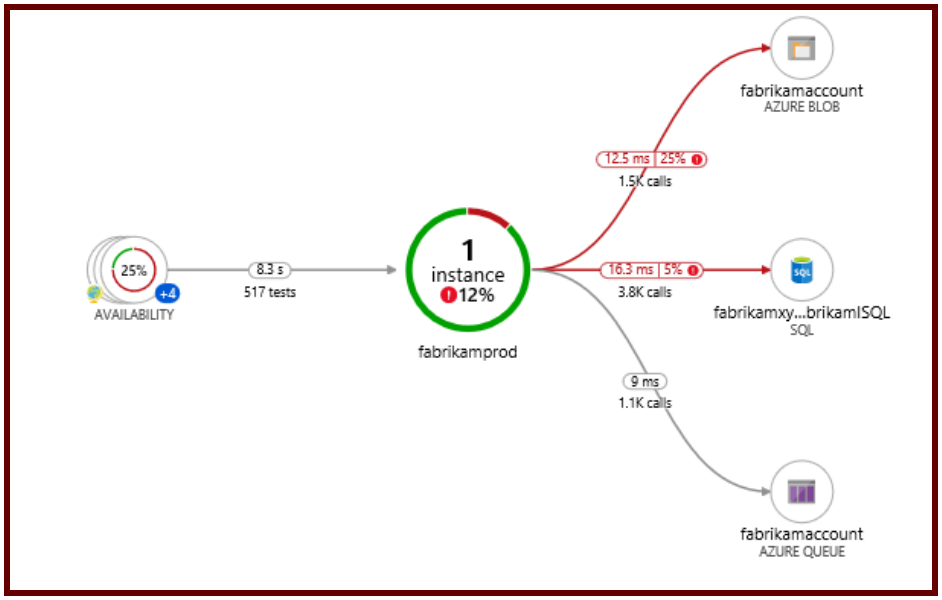
(Source: Microsoft Documentatio)

Troubleshoot app performance by using Application Map

An application map helps you identify performance bottlenecks or failure hotspots across all components of your distributed application. Each node in the map represents an application component or its dependencies; and includes the status of health KPI and alerts.

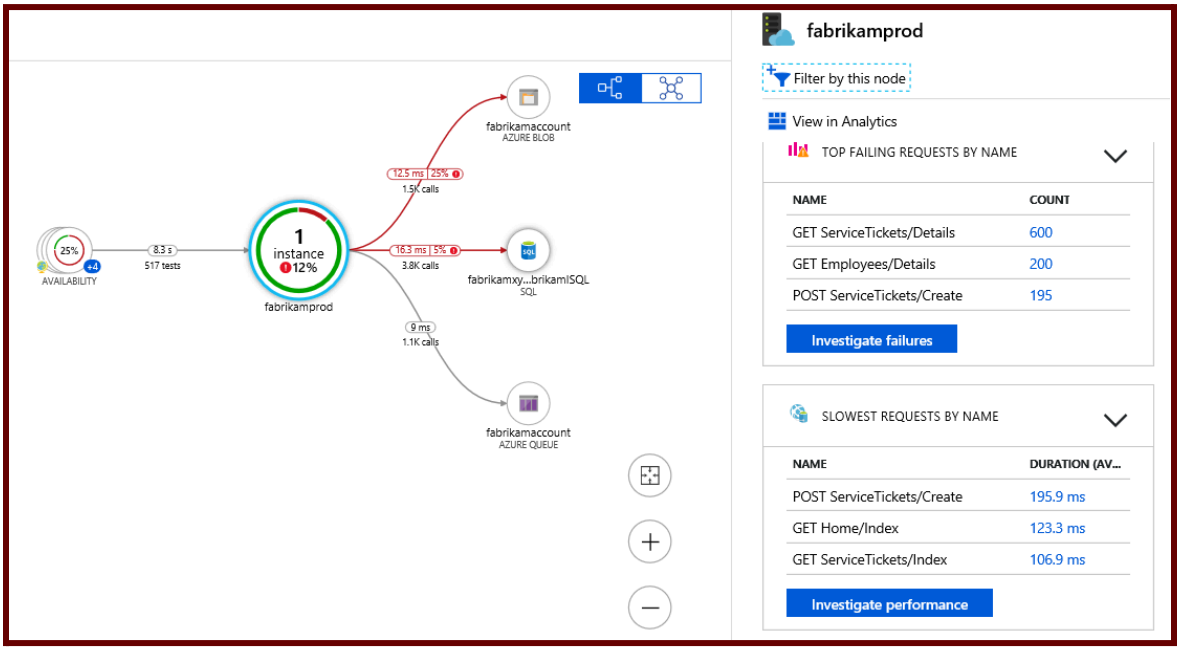
Components are independently executable components of your distributed/microservices application. Developers and operations teams have code-level visibility or access to telemetry generated by these application components.

If all components are roles in the same Application Insights resource, this discovery step is not necessary. The initial load for such an application consists of all its components.



(Source: Microsoft Documentation)

One of the key goals with this experience is to be able to visualize complex topologies with hundreds of components. Click on any component to see related insights and jump to the performance and failure triage experience for that component.



(Source: Microsoft Documentation)

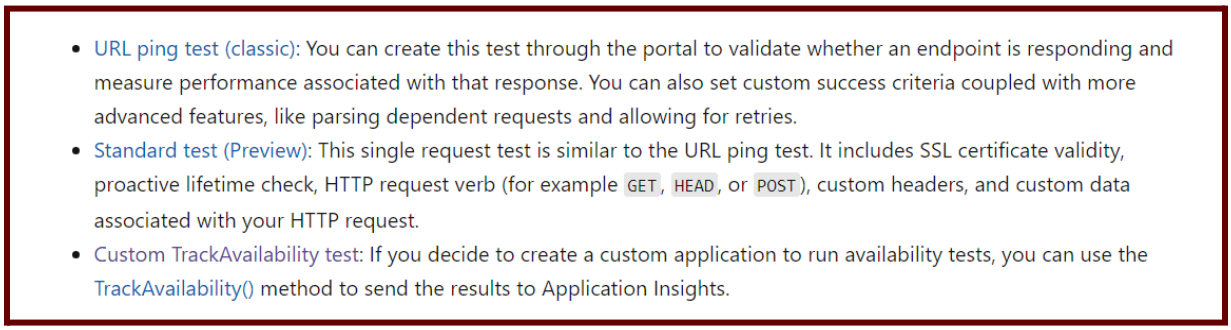
The application uses the map cloud role name property to identify the components of the map. You can manually set or override the cloud role name and change what is displayed in the application map.

Application Insights availability tests

Once you have your web app or website up and running, you can set up recurring tests to monitor availability and responsiveness. Application Insights regularly sends web requests to your application from points around the world. It can alert you if your application is not responding or is responding too slowly.

You can set up availability tests for any HTTP or HTTPS endpoint that is accessible from the public Internet. There is no need to make any changes to the website you are testing. Of course, it doesn't have to be your own site. You can test the availability of a REST API based on your service.

There are four types of availability tests:
- Standard, Custom TrackAvailability, and Classic Tests



- URL ping test (classic): 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.
- Standard test (Preview): 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.
- Custom TrackAvailability test: If you decide to create a custom application to run availability tests, you can use the TrackAvailability() method to send the results to Application Insights.

(Source: Microsoft Documentation)

Whizlabs Hands-on-labs Links

- How to create Application Insights | Azure Labs (whizlabs.com)

References Links:

- Monitor app performance - Training | Microsoft Learn
- Configure monitoring for ASP.NET with Azure Application Insights
- Application Insights overview - Azure Monitor | Microsoft Learn
- Azure Application Insights log-based metrics - Azure Monitor | Microsoft Learn
- Select an availability test - Training | Microsoft Learn
- Review TrackAvailability() test results - Azure Monitor | Microsoft Learn
- Troubleshoot app performance by using Application Map - Training